修士学位論文

題目

識別子とその対応するコメントを利用した プログラム理解支援用名詞辞書自動生成手法

指導教員
井上 克郎 教授

報告者
藤木 哲也

平成 23 年 2 月 7 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 22 年度 修士学位論文

識別子とその対応するコメントを利用した
プログラム理解支援用名詞辞書自動生成手法

藤木 哲也

内容梗概

プログラムを理解するために，ソースコードを読むことが広く行われている．作業者がソースコードを読むことを通じて理解を高める際には，ソースコード中の様々な要素を眺める，中でもソースコード中の識別子は重要な手がかりである．作業者は識別子からその識別子が付けられた関数や変数の役割や振る舞いを類推することが可能である．しかし，作業者の知識が十分でない場合は，類推ができないためプログラムを理解すために必要な時間が増加するという問題がある．

そこで本研究では，識別子中の単語の説明を収録した辞書を作業者に提供し，説明文を提示することで識別子中の単語に対する知識を補うことでプログラム理解の支援を目指す．そのために，ソースコード中に記述された識別子とその識別子に対応するコメントを利用することで，識別子中に用いられる名詞の辞書を自動的に生成する手法を提案する．本手法ではまず，事前に収集した Java ソースコードを解析し，識別子とそれに対して記述されたコメントを収集する．次に，収集したコメントに対して自然言語処理で用いられている解析を応用することにより，名詞に対して説明を行っているフレーズを抽出し，それを用いて名詞の説明文を生成する．

また，提案手法をシステムとして実装し，オープンソースソフトウェアを用いて辞書を生成した．生成した辞書を用いた評価実験によって，本手法で生成した辞書がプログラム理解を支援できるかを評価した．評価実験では，被験者にプログラム理解の作業の一部を模した課題を与え，課題への正答率や要した時間を測定した．その結果，生成した辞書を利用することで被験者の正答率が高まり，プログラム理解への手助けとなることことが分かった．

主な用語

プログラム理解 (Program Comprehension)
識別子 (Identifier)

コメント (Comment)
自然言語処理 (Natural Language Process)

目 次

1　はじめに

近年のソフトウェア開発では，ソフトウェアの大規模化，複雑化によってソフトウェア保守に要するコストの増大が問題となっている．ソフトウェア保守を行うためには，対象となるソフトウェアの動作を理解することが必要であり，そのために作業者はプログラム理解を行う．そして，ソフトウェア保守に要するコストのうち，プログラム理解にかかるコストが大きな割合を占めることが知られている [6, 8]．

プログラム理解にとって仕様書や設計書などのドキュメントは有用な手がかりである．ドキュメントにはソースコード中には記述されないソフトウェアとしての機能，製作者の意図，開発の経緯や処理のフローなどが記述されている．しかし，ドキュメントが存在しない場合や不十分な場合には，作業者はソースコードの読解により多くの時間を割くことでプログラム理解を行わなければならない．ソースコードの読解の際，ソースコード中の識別子は重要な手かがかりとなる．識別子名から関数や変数などのプログラム要素の役割を類推し作業者自身のソフトウェア知識と対応づけることで，ソースコードの読解を進める [21, 27]．

識別子名は複数の単語から構成されることが多く，単語の組み合わせにより識別子の役割や振る舞いを表現している．そのため，識別子名に用いられる単語のそれぞれが，識別子のもつ構造や対象など振る舞いの一部を表している．

識別子名から識別子の役割や振る舞いをできない場合，プログラム理解に要する時間が増加するという問題がある．作業者にソフトウェアの開発経験や開発するソフトウェアのアプリケーションドメインの知識が不足している場合，識別子からプログラム要素の役割の類推とソフトウェア知識への対応づけを行うことができない．また，対応づけを行えない場合では行える場合に比べてプログラム理解により多くの時間を必要とする．

作業者が識別子からの類推を行えるようになるためには，実際の様々な事例から学習することが必要である．しかし，事例から学習を行うためには，多くのそして多様なドメインのソフトウェアの開発や保守を経験しなければならない．そのため，学習に要するコストは大きい．

そこで本稿では，プログラム理解支援用の識別子中に出現する名詞の辞書を自動的に生成する手法を提案する．提案手法では，ソースコード中の識別子とそれに対して記述されたコメントから，名詞とその名詞を説明しているフレーズを抽出することで，名詞の説明文を生成する．生成の際には，自然言語処理を応用し，コメントを解析することで説明文の生成を行う．

また，提案手法を実装したシステムを作成し辞書の生成を行った．辞書の生成には，オープンソースソフトウェア (以下，OSS) のソースコードを収集し利用した．

最後に，学生を用いた評価実験を行い，生成した辞書がプログラム理解の支援において有

用かを評価した．評価実験では，被験者にプログラム理解作業の一部を模した課題を与え，課題への正答率を測定した．その結果，辞書を用いることにより被験者の正答率が向上し，生成した辞書のプログラム理解に対して有用であることが分かった．

以降，２節では研究の背景となるプログラム理解などについて説明する．３節では提案手法の詳細について述べ，４節で提案手法の実装及び適用実験について述べる．５節では生成した辞書を用いた評価実験について述べる．６節では関連研究について説明する．最後に７節では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，研究の背景となる知識や問題について述べる

2.1　Java プログラムの標準的な記法

本節では Java プログラミムにおける識別子とコメントの記法について述べる．

2.1.1　プログラム要素と識別子

プログラム要素とはソースコード中の関数や変数などのプログラムを構成する要素である．識別子とは，プログラム中であるプログラム要素を一意に識別するためのものである．識別子は単一の識別子もしくは，複数の識別子をピリオドで連結した形で表現される．

そして，識別子にはソースコードを読む作業者が理解しやすいように識別子名が付けられる．識別子名には分かりやすくその識別子の内容を表しており，かつ簡潔な名前を用いるべきである [15]．そうすることによって，コードの可読性を向上させ，名前の衝突を防ぐことが可能になる．

識別子名は識別子の内容を表現するために複数の単語を連結した複合語として表記されることが多い．行う処理やデータの構造などといった，識別子の特徴を端的に表現した単語を組み合わせることによって表現される．識別子名に記号の羅列や特徴と関係のない単語を用いることは見る者に混乱をもたらすことになる．

識別子名には略語表記された単語が用いられることがある．略語を用いることで，識別子名が長大になりソースコードの可読性が低下することを防ぐ．単語を短縮するルールに明確な規定はなく，慣習，組織の規則や個人の嗜好などに依る．そのため，複数の単語に対して同じ略語が用いられることも起こりうる．

記法　識別子名についてはいくつかの規則が存在する．識別子に用いることができる文字は ASCII のラテン文字の大文字・小文字，数字，アンダースコア (_)，ドルマーク ($) であり，文字数の制限はない．

一般に単語の連結パターンとして以下の 2 つがある．

CamelCase　各単語の先頭の文字を大文字にして連結を行う．

snake_case　各単語の間にアンダースコアを挿入して連結を行う．

クラス名　言語仕様ではその内容を表す名詞や名詞句で，CamelCase で表記し長すぎることのないものと推奨されている．クラス名にはクラスの持つ機能や構造を表す単語が使用されるべきである．

2.1.2　コメント

コメントにはソースコードの可読性を向上させ，作業者にとって理解を助ける働きがある [22]．コメントでは計算式では表現できない仕様や製作者の意図などが自然言語を用いて記述されている．また，コメントが多いソースコードの方がソフトウェアの品質が高い傾向にある [23]．

コメントはソースコード中の任意の行に記述することができるが，一般にコメント内で説明を行っている箇所の付近に記述される．そして，識別子に対するコメントを記述する場合には，その識別子が宣言されている箇所の直前や直後に記述する．また，コメントは以下の 2 つの方法で記述できる．

TraditionalComment /∗ から ∗/で囲まれた部分がコメントとして扱われる．

EndOfLineComment //から行末までがコメントとして扱われる．

また，コメントをそのまま API などのドキュメントとして利用することが行われている．そのようなコメントはドキュメンテーションコメントと呼ばれる．ドキュメンテーションコメントはクラス，インターフェース，コンストラクタ，メソッド，フィールドについて説明し，/∗∗ と ∗/に囲まれる領域に記述する．コメント中には@から始まるいくつかのタグを用いて情報を付与することができる．javadoc を用いることにより Java ソースファイルからから API ドキュメンテーションの HTML 文書を自動的に生成することができる．そのため，ドキュメンテーションコメント中では，HTML タグを用いられることがある．

さらに，コメント中に形式的な仕様を記述することで検証を自動的に行うことが可能である．Java Modeling Language(JML）という仕様記述言語を用いて制約条件を記述する．制約条件としてはメソッドの事前条件や事後条件，クラスの不変条件などが記述する．そして，JML を用いることにより，責任の所在を明確にできフトウェアの品質を向上させることが可能である．

クラスへのコメント　クラスへのコメントは外部から見た際の機能，構造や扱うデータが記述される．クラスへのコメントはメソッドやメンバ変数などに対するコメントよりも記述が豊富であることが多い．メソッドではそのメソッドの動作やメンバ変数では保存するデータの内容などについて記述されるのに対し，一方クラスへのコメントではクラスに含まれるそ

れらのメソッドやメンバ変数などを踏まえた上で，クラス全体としてどういった機能を持つのかを記述するからである．

また，クラス名にはクラスの機能を表現する単語が用いられている．そのため，コメントの説明の一部では，そのクラス名に含まれるある単語への説明を行っていると考えられる．

2.2 ソフトウェア保守とプログラム理解

ソフトウェアの保守は，ソフトウェアライフサイクルにおけるコストの大きな部分を占めている [9]．保守では，単なるバグの除去などだけではなく，ソフトウェアの修正，適合，完全化，予防などが行われる [13]．ある程度以上の規模のソフトウェアの場合，ソフトウェアは複数人によって開発保守され，作業者がソフトウェアの全てを理解しておくことは困難である．ソフトウェアを開発に携わった人員が，ソフトウェアの保守まで担当するとは限らず，自身が開発していないソフトウェアの保守を行うことがある．また，ソフトウェアに適切な修正や変更を加えるためには，まずそのソフトウェアのデータの流れや処理手順などについて十分に理解することが不可欠である．そのため，ソフトウェアの保守を行うためにプログラム理解を行うことが必要である．そして，ソフトウェア保守に要するコストのうち，プログラム理解にかかるコストが大きな割合を占めることが知られている．

2.2.1 ソフトウェア保守の手順

ソフトウェア保守では，安定して運用し続けるためにソフトウェアの管理を行い，そのためのコストはソフトウェアライフサイクル全体における多くの部分を占めている．対象のソフトウェアに問題の発生，改善要求や新環境への適用要求が起こった際に以下のプロセスが実行される．

1. 問題把握及び修正分析
2. 修正の実施
3. レビュー及び受け入れ

ソフトウェア保守は，そのソフトウェアが廃棄されない限り行われ続ける．

問題把握，修正分析，修正の実施ではそれぞれプログラム理解が必要とされる．問題把握では問題となっている部分を特定し，修正分析では修正の行う範囲やその影響が与える範囲などを分析し，修正では実際の修正方法を決定し実施する．そのため，それぞれの処理はプログラムの当該領域を把握した後でないと，予想外の副作用が発生したりそもそも処理自体が行えないということになる．

2.2.2 プログラム理解

プログラム理解ではソフトウェアの動作を把握し，どのようにその動作を実現しているかを理解する．ソフトウェアのドキュメント，ソースコード，実行時情報などを利用することでプログラム理解を行う．プログラム理解では自身が知りたい情報に関連するところを探し，そしてその箇所について詳しく調べるという手順を繰り返し行う．

プログラム理解を行うにあたって，機能仕様書やテスト報告書や用語集などのソフトウェアに付随するドキュメントやソフトウェアの開発者へのインタビューは有用な手がかかりである．ドキュメントにはソースコードや実行時情報などからは得られない，ソフトウェア製作の経緯や変遷，製作者の意図，抽象度の高いモデルなどが記述されている．さらに，ソフトウェアの開発に携わった人にインタビューをすることで，知りたい情報に対する明確な解答や説明を得られるかもしれない．

しかし，ドキュメントの利用や開発者へのインタービューは常に行えるとは限らない．ドキュメントはソフトウェアの変化に合わせて適切に変更されていなかったり [16]，そもそも文書化されていなかったり，プロジェクトの途中で適切に引き継がれずに紛失してしまう場合がある．また，開発者を行った人員と保守を行う組織が異なっていたり，開発者が別の組織に移るなどでインタビューを行えない場合がある．そのような場合には，作業者はより多くの時間をソースコードを読むことに費やす必要がある．

プログラム理解のためにソースコードを読む際には識別子が重要な手がかりとなる．ソースコードの読解を行う際に，ソースコード中のコメント，識別子，返り値，関数の呼び出し関係など様々な情報を手がかりを利用する．識別子名からその識別子の持つ振舞いや役割を類推することができる．類推することで，その識別子が知りたい情報に関連するかどうかの判断を行うことや，ある程度の予測をもってソースコードを読み進めることが可能になる．

作業者の知識や経験が不足していて，識別子からの類推を適切に行えない場合にはプログラム理解に要する時間が増加するという問題がある．ソースコード中の知りたい情報と関連の低い箇所を読むことに時間を費やしてしまうことや，誤った予測をもったまま読み進めるなどをした結果，手戻りが発生してしまうことがあるからである．

さらに，識別子からの適切な類推を行えるようになるための学習コストは高いという問題がある．その理由として２つのことが考えられる，１つは識別子の学習には専用の教材や体系だった学習方法はなく，多くのソフトウェア開発に携わることで実際の事例から学び取る必要があること．もう１つは，識別子中の単語には特定のドメインのソフトウェア中のみで使用される単語が存在するため，多様なドメインのソフトウェアに触れることが必要であるからである．一般的に識別子に関する学習はソースコード，仕様書，用語集や情報科学に関する教科書などのドキュメントから行われる．ソースコードからの学習では識別子とそれに

対するコメントや識別子に関連する処理などから，識別子名とそれから類推される内容の対応付けを学ぶ.

3　辞書生成手法

２節で述べた問題に対して，本研究では，識別子中に出現する名詞の説明を集めた辞書を生成し，その辞書を作業者に提供することで作業者の不足している識別子への知識を補う．手作業で辞書を作成することは手間が大きいので自動的な生成を行う．自動的な生成のためにソースコード中の識別子とコメントを利用し説明文を作成する．

コメント中のある名詞に対する説明に相当するフレーズを利用することで説明文を作成する．ある単語を含む識別子に対するコメントを収集した際に，そのコメントの集合中で頻出するフレーズはその名詞に対する説明を行っていると考えられる．本手法ではその頻出するフレーズを抽出し，そのフレーズを加工することで説明文を生成する．

3.1　提案手法の概要

本節では提案する辞書生成手法について述べる．提案手法の概要を図１に示す．本手法では，ソースコード中に記述されたコメントを利用することで名詞に対する説明文の生成を実現する．入力として Java 言語で記述されたソースコード集合を与え，出力として識別子中に出現する名詞とそれに対する説明文との組の集合からなる辞書を生成する．

本手法では，まず，ソースコードを解析し，識別子とその識別子に対応するコメントの組を取得する．そして，識別子とコメントの組から，名詞と名詞に関する文の組を作成する．ここで名詞に関する文とは，コメント中から抽出した名詞に対する記述を含んでいるかもしれない文とする．次いで，名詞に関する文の集合中に頻出する表現をその名詞への説明とし，そのような説明を抽出し加工することで名詞に対する説明文を生成する．

処理手順を以下に示す．

1. ソースコードから識別子の宣言とそれに対応するコメントを抽出
2. 識別子名から名詞を抽出
3. コメントの整形
4. 名詞に関する文を解析し，頻出する表現を抽出する
5. 得られた表現から説明文を作成する

なお，識別子としてクラス名を用いる．クラス名には多様な名詞が出現し，また説明文を生成するためのクラスに対するコメントが豊富に利用できるからである．

以下，提案手法の詳細について述べる．

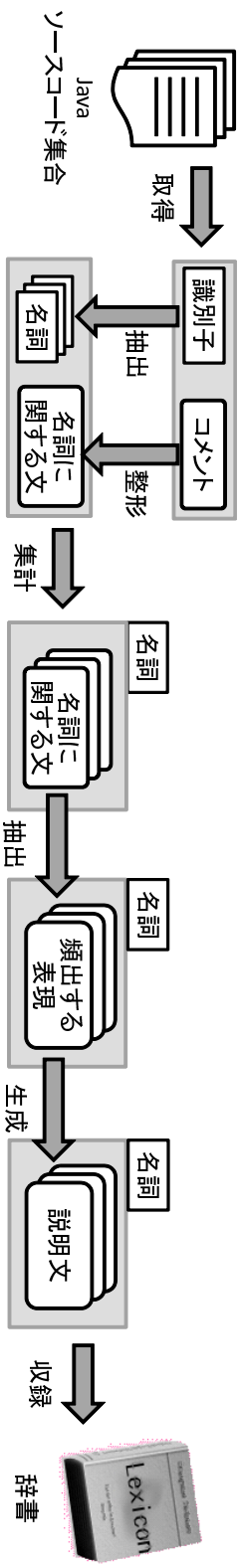
Java
ソースコード集合
取得
識別子
コメント
抽出
整形
名詞
名詞に関する文
集計
名詞
名詞に関する文
抽出
名詞
頻出する表現
生成
名詞
説明文
収録
Lexicon
辞書

図 1: 提案手法の概要

種類	識別子名	切り分けの結果
キャメルケース	CamelCase	Camel, Case
スネークケース	snake_case	snake, case
連続する大文字	ASTNode	AST, Node
数字を含む	Dvi2Pdf	Dvi, Pdf

表 1: 識別子名の切り分けの例

3.2 ソースコードの解析

ソースコードから識別子とコメントを取得する．

まず，ソースコードを構文解析することでクラスを宣言している箇所を探す．そして，クラス宣言の直前に記述されたコメントをその識別子に対応するコメントとする．ここで，対応づけするコメントはブロックコメントもしくは連続するラインコメントである．Java では 1 ファイルに複数個のクラス宣言が行われる場合があるがトップレベルクラスのみを利用し，内部クラスと匿名クラスは利用しない．内部クラスには一般的にコメントが記述される量が少なく，匿名クラスには識別子名が存在しないので識別子とコメントの対応がとれないため，内部クラスと匿名クラスは対象としない．

3.3 名詞の抽出

ここでは，識別子名に含まれている名詞の抽出を行う．まず識別子名を単語に切り分け，次いでそれぞれの単語が辞書に載せるかの判定を行う．

2 節で述べたように Java の命名規則として用いられるキャメルケースとスネークケースのスタイルにしたがって単語に切り分ける．また，連続する大文字は最後の 1 文字を除いた文字列を一つの単語としてみなす．さらに，識別子名中に出現する数字とドルマーク ($) も単語の区切りとしてみなす．単語の切り分けの例を表 1 に示す．

本手法では，説明文を生成する対象の単語は名詞であるため，それぞれの単語について品詞付けを行う．各単語に対して POS Tagger(品詞解析器) を用いることで，それぞれの単語の品詞を調べる．その結果が名詞であるもの及び判定不能であったものを識別子名に含まれた名詞として抽出する．同じ綴りで複数の品詞として用いられる単語については，その単語が品詞として名詞となる場合があるなら抽出を行う．また，品詞解析器は一般的に内部に自然言語用の辞書しか持たないために，ソフトウェア中では一般的な名詞でも，それが自然言語で用いられていないなら判定できない場合がある．

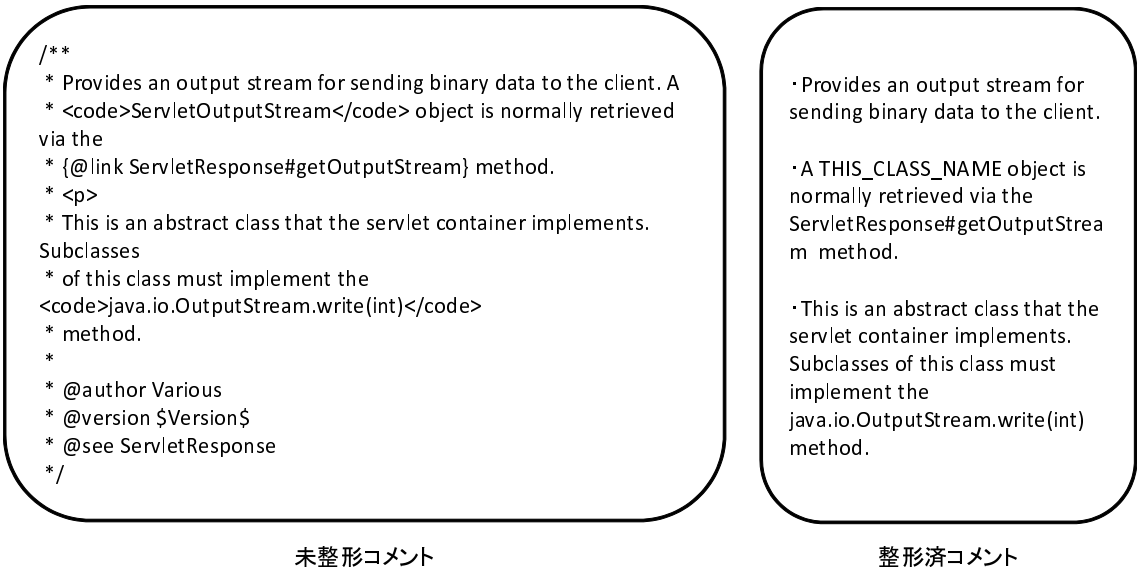

図 2: コメントの整形の例

品詞解析器　品詞の判定には，品詞解析器として Stanford Log-linear Part-Of-Speech Tagger[26] を利用した．入力として英単語を与えると，出力として Penn Treebank[18] に則ったその単語の品詞を返す．また，内蔵した辞書に定義されていない単語はすべて普通名詞と判定される．

今回，次の品詞を名詞として扱い説明文の生成を行った，NN（普通名詞），NNS（普通名詞複数形），NNP（固有名詞），NNPS（固有名詞複数形），PRP（人称代名詞），UH（間投詞），VBG（現在分子）．

3.4　コメントの整形

コメント中の自然言語以外の記述の除去や文の区切りを明確にする．コメントの整形の例を図 2 に示す．

まず，コメントから以下の要素を除去する．

コメント区切り文字　//，/*，*，*/

HTML タグ　<> または<> で囲まれる領域

アノテーション　スタンドアロンタグ（そのタグのみ除去)，インラインタグ（その 1 行を削除）

次に，コメントの文と文の区切りを明確にし，1 文ごとに分割する．最初に，コメント全

体に対してピリオド・セミコロン・クエスチョンマーク・エクスクラメーションマークを文の区切り文字として改行を挿入する．そして，各行の終端が文の区切り文字でなく，次の行が小文字のアルファベットもしくは識別子名から始まるならそれらの行を連結する．

最後に，文中の特定の文字列を置換する．コメントと対応する識別子名が文中に出現する場合，その識別子名を”THIS_CLASS_NAME” というメタ文字に置換する．また，文中の数値文字参照や文字実体参照をその文字や記号自身に置換する．

3.5 名詞との対応づけ

識別子から抽出された名詞と，その識別子に対応するコメントを整形し得られた各文を名詞に関する文として対応づける．また，その識別子が出現したソースコードを含むプロジェクト名を得られた名詞に関する文の出自として関連づける．もしプロジェクトとしてソースコードが纏められていない場合には，ソースコードのパッケージ名を出自として用いる．ソースコードがデフォルトパッケージで記述されていた場合には，名詞に関する文の出自は関連づけない．

3.6 名詞に関する文の集計

名詞とそれに対応付けられた名詞に関する文を，名詞ごとに集計を行う．名詞の大文字小文字は区別せずに扱い，略語は短縮する前の単語とは別の単語として扱う．例えば”Length” と”length” はまとめて集計を行うが，その略語である”Leng” や”Len” はそれぞれ別の単語として集計を行う．

名詞への説明が明らかに含まれていない文などをフィルタリングすることで除去する．フィルタリングは 1 文毎に対して行う．フィルタリングする項目を以下に示す．

英語以外の文 文中に ASCII 文字以外の文字が使用されている文．本手法では対象言語を英語としているので，ASCII 文字以外の文字を含む文は英語以外の言語で記述されていることが多い．

ライセンス記述 一般的なコメントには出現せずに，主にライセンス記述でのみ使用されるような単語を含む文．

著作権 ⓒ や “copyright” を用いた著作権に対して記述している文．

タイムスタンプ ソースコードの作成された日時や，変更に対する履歴などを記述している文．

コード片 サンプルコードや一部の式などを記述している文

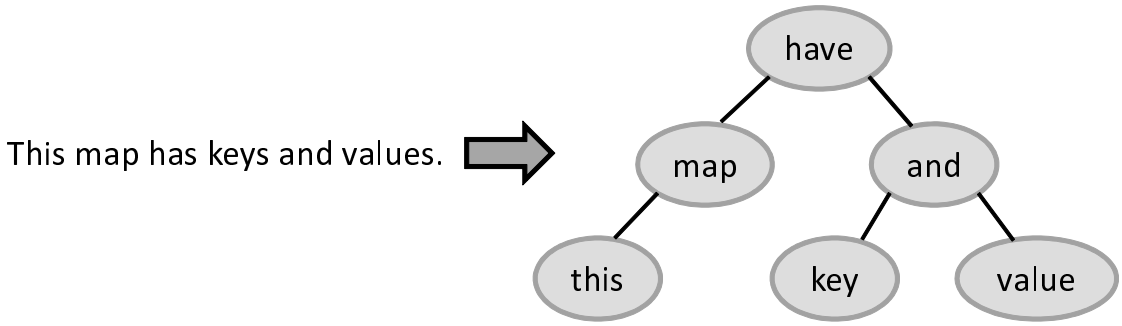

図 3: グラフ化の例

さらに，名詞に関する文の中で重複する文を取り除く．ここで，重複する文というのは，文中の数字と記号と空白を全て除去し，アルファベットをすべて小文字に直した際に，他の文と一致するような文のことする．

3.7 頻出表現の抽出

名詞に対して集めた名詞に関する文の中で頻出する表現を抽出する．頻出する表現を抽出するために，まず，文を解析することで文の構造を利用しグラフ化する．次に，頻出部分グラフと呼ばれるグラフの集合で複数のグラフ中に出現するような部分グラフを抽出する．頻出部分グラフは文の集合中では頻出する表現に相当すると考えられる．

グラフ化 名詞に関する文をグラフに変換する．まず，名詞に関する文を構文解析することにより，文内の各単語間の依存関係を取得する．次に，単語間の依存関係を辺とし，各単語をラベルとする頂点から無向グラフを構築する．頂点には，文中の単語を直接用いるのではなく，単語から活用を取り除いた形を用いる．例えば，”dogs” という名詞は単数形である”dog” で，“”been” という動詞は原形である”be” を用いる．文中に同じ単語が出現する場合には，それらの単語に対して異なる頂点を用いる．単語数が w である文からは，頂点数が w であるグラフを構築される．グラフ化の例を図 3 に示す．

構文解析 英語の構文解析器として Enju を利用した [10, 20]．入力として英語 1 文を与えると，各単語の原型，各単語の品詞，構文構造，述語項構造を出力する．

単語の間の依存関係として述語項構造を用いた．述語項構造とは，文中の構造を述語（主語，目的語）という 3 つ組で表現する．ここでの述語は文としての述語ではなく，各単語を述語としてとらえる．述語-主語と述語-目的語を辺として用いた．

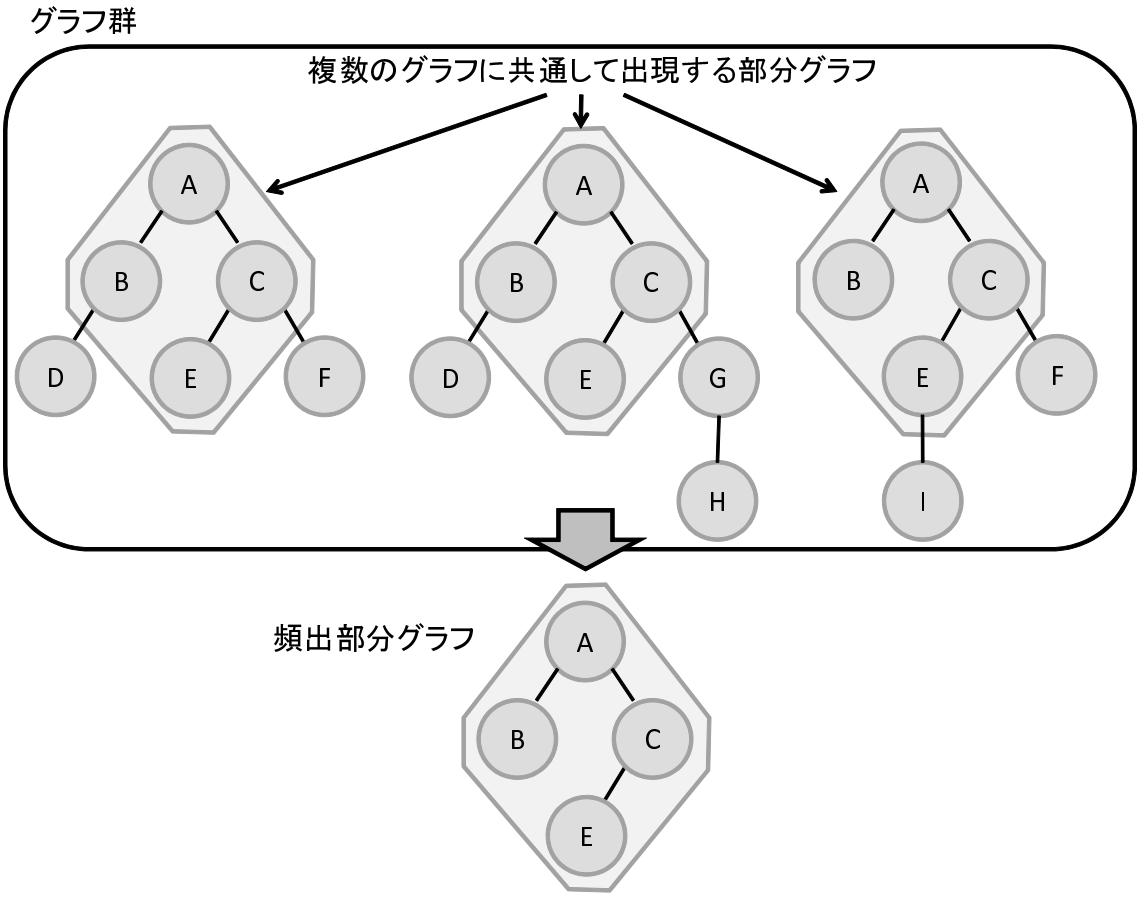

図 4: 頻出グラフマイニングの例

頻出グラフマイニング　グラフ群に頻出するグラフ構造を取得するために頻出グラフマイニング手法を用いる [14] 頻出グラフマニングを用いることで，グラフ群において複数のグラフに部分グラフとして一定回数以上出現するような頻出部分グラフを抽出する．頻出グラフマイニングの例を図４に示す.

抽出するグラフについて述べる．抽出するグラフを G_S とする．ラベル付きグラフ G は $G = (V, E, L, lb)$ で表される．ここで V は頂点の集合，E は辺の集合，L はラベルの集合，lb はラベル付けの関数である．グラフ G の頂点，辺，ラベルの集合をそれぞれ $V(G)$, $E(G)$),$L(G)$) と表す.

以下の条件を満たすような写像 ϕ: $V(G) \to V(G_S)$ が存在するとき.

$$\forall v(v \in V(G_s) \Rightarrow identification(lb(v), lb(\phi(v))))$$

$$identification(LabelA, LabelB) = \begin{cases} true & \text{LabelA と LabelB が等しい} \\ true & \text{LabelA と LabelB が自然言語において類義語の関係にある} \\ fase & \text{その他のとき} \end{cases}$$

さらに，以下を満たすとき，$G_s \subseteq_f G$ と表す.

$$\forall(v_i, v_j)((v_i, v_j) \in E(G_s) \Leftrightarrow (\phi(v_i), \phi(v_j) in E(G))$$

グラフ G_S の支持度 $support(G_S)$ を

$$support(G_S) = \frac{|\ \{G \mid G \in \text{グラフ集合}, G_S \subseteq_f G_S\}\ |}{|\ \text{グラフ集合}\ |}$$

と定義する.

$support(G_S) >$ 最少支持度 (ある定数)　であるような G_S を抽出する.

類義語　類義語の判定には英語の概念辞書である WordNet[19] を利用した．WordNet ではある概念を表す単語を集めたグループと，そのグループ同士の上位下位関係が記述されている．ここで上記の式で用いた”LabelA と LabelB が自然言語において類義語の関係にある”ということは，LabelA と LabelB の両方が属するグループが存在するということである.

3.8　説明文の生成

抽出した頻出表現から説明文の生成を行う．まず，頻出表現はグラフの構造として表現されているので，グラフから自然言語の文への変換を行う．復元を行っただけの文では自然言語の文として不自然なことが多いので，文に対して補完を行う．また，ここで生成された文には名詞の説明文として不適切な文が含まれているため，そのような文をてフィルタリングを行うことで除去する．そして，フィルタリングの結果残った文を説明文とする.

グラフから文への変換　頻出部分グラフから自然言語の文へと変換する．グラフには単語の語順や単語の活用といった情報は含まれていないので，グラフ化を行う前の名詞に関する文の情報を利用する．

グラフ集合からある頻出部分グラフを部分グラフとして持つようなグラフを探索し，そのグラフの生成元となった名詞に関する文を取得する．ここで，ある頻出部分グラフを部分グラフとして持つようなグラフは，頻出部分グラフの性質上グラフ集合中に必ず複数存在する．そこで，利用するグラフを決定する．はじめに，各グラフをラベルの単語を基底として，その単語の出現数を係数として持つようなベクトル v_i で表現する．そして，全てのベクトルの相加平均 $\overline{v}$ を求める．v_i と $\overline{v}$ の類似度が最も高いグラフを利用する．類似度としてコサイン尺度を用いる．コサイン尺度は，２つのベクトルのなす角度を表し，以下の式で定義される．

$$cos(v_i, \overline{v}) = \frac{v_i \cdot \overline{v}}{\mid v_i \mid\mid \overline{v} \mid}$$

名詞に関する文を利用して自然言語の文を生成する．頻出部分グラフに存在する頂点と対応する単語を名詞に関する文からそのままの語順，単語の活用で抜き出し，その単語列を文とする．

文の補完　得られた文に欠けている文法上欠けている要素を補完する．

文の生成に利用した名詞に関する文の述語項構造を利用する．以下の単語の補完を行う．

- 主語があり述語が欠けているなら，述語を補完する
- 目的語があり述語が欠けているなら，述語を補完する
- 述語があり主語もしくは目的語が欠けているなら，欠けている主語もしくは目的語を補完する

単語を追加する場所は名詞に関する文の語順に則り，単語の活用も名詞に関する文に則る．さらに，文に括弧や引用符などが片側のみが出現している場合，対となる要素を補完する．

フィルタリング　補完を行った文に対してフィルタリングを行う．フィルタリングに用いる項目を以下に示す．

・**支持度**　頻出部分グラフの支持度．

・**サイズ**　頻出部分グラフのサイズ (頂点数)．

・**プロジェクト数**　頻出部分グラフを含むグラフの名詞に関する文がいくつのプロジェクトで記述されていたかの合計．

- **語数** 文を構成する単語の数
- **平均単語長** 文内の単語の平均構成文字数
- **最大単語長** 文内の単語の最大の文字数
- **特徴度** 単語の tf(出現頻度) と idf(逆出現頻度) を利用して計算される値
- **特定単語を含む** ある特定の単語含むか
- **数字を含む** 文中に数字を含むか
- **区切り文字の個数** 文中にカンマ，ピリオド，コロン，セミコロンなどをいくつ含むか
- **特定単語の数** ある特定の単語の語数
- **識別子の数** 文中に出現する識別子の数
- **主語を含む** 文に元の文の主語が含まれているか
- **述語を含む** 文に元の文の述語が含まれているか
- **目的の単語を含む** 説明文を生成する対象の名詞が含まれているか
- **主語が目的の単語** 説明文を生成する対象の名詞が主語になっているか
- **括弧** 括弧等が閉じられているか
- **仏語** フランス語によく用いられる’de” や”cest” などを含むか

4　フィルタリング基準の設定と辞書の生成

本節では，フィルタリングの調整と辞書の生成について説明する．

4.1　フィルタリング基準の設定

どのフィルタリングを使用するか，どのパラメーターで使用するかを決定するために被験者を用いた実験を行った．生成した文を被験者に見せ，被験者に定性的に評価してもらう．そして，その評価が良い文を正解集合として，その文をなるべく残すフィルタリング基準を採用する．

実験内容　被験者に生成した文を見せてその文が英語として読めるかとある名詞に対する説明として有用であるかについて定性的な評価を行ってもらった．

フィルタリングを行わずにに文を出力し，その全体から次の条件のもとランダムで選び出した文を用いた．グラフのサイズが 3,4,5,6 以上と支持度が 2，3，4，5 以上 9 以下,10 以上の 2 つの項目で分類した際に，すべての項目がなるべく同数に近くなるように選び出す．

被験者に行った実験は以下の通りである．

1. 被験者に評価してもらう文を提示する
2. 設問 A(表２) に解答してもらう
3. 設問 B(表３) に解答してもらう
4. 文章から得られそうな知識を記述してもらう
5. その文が何の名詞に対して生成されたのかを提示する
6. 設問 C(表４) に解答してもらう
7. 設問 D(表５) に解答してもらう

Java の経験のある被験者 5 名を用いて上記の実験を行った．文を合計 150 個選び出し問題を作成した．回答が一人の被験者のみに依存することを防ぐために，各文あたり 2 名の被験者が問題にあたるように行った．

フィルタリングの決定　実験の結果から英語として読めるものを残すフィルタリングと説明として役立つものを残すフィルタリングを決定した．また，正解集合は以下のように定義した．

表 2: 設問 A

設問	この文は英語として読めますか
選択肢 A	英語として正しく読める
選択肢 B	英語として誤りを含むが読める
選択肢 C	英単語の羅列であり，文として読むことができない （例.”complete done (get intitiated that)”）
選択肢 D	記号などを多く含んでおり，文として成立していない (例.”(byte [ ] , , it) null”)
選択肢 E	英語以外の言語である

表 3: 設問 B

設問	この文から何らかの知識が読み取れそうですか
選択肢 A	知識を得ることができる
選択肢 B	特定のクラスの説明など一般的ではない説明である
選択肢 C	何の知識も得られない

表 4: 設問 C

設問	提示した単語を知っていますか
選択肢 A	単語とその意味を知っている
選択肢 B	単語を見たことはあるが，意味はよく分からない
選択肢 C	知らない

表 5: 設問 D

設問	あなたが説明文を読んで得た知識は提示した単語の理解に役立ちますか 設問 3 で知らないと答えた場合は回答しないでください
選択肢 A	単語に関する知識であり，単語の理解に役立つ
選択肢 B	単語に関する知識であるが， 特定のクラスなどでの知識であり理解には有用ではない
選択肢 C	単語と関係のない知識であり，理解には役立たない
選択肢 D	単語に関して誤った知識を得てしまい，不適切である

表 6: F 値を最大にするときの結果

	英語として読める	説明として役立つ
F 値	0.784	0.5
適合率	0.661	0.5
再現率	0.963	0.5

英語として読める 被験者の 2 人ともが設問 A で，選択肢 A もくしは B を選択した

説明として役立つ 被験者の 2 人の内どちらかが設問 D で，選択肢 A を選択した

それぞれの正解の個数は英文として読めるが 81 個，説明文として役立つが 22 個であった．

評価として，情報検索の分野でよく用いられいる F 値で評価する．F 値とは適合率と再現率の調和平均である．適合率，再現率，F 値を次のように定義した．

$$\text{適合率} = \frac{|\text{フィルタリング後残る文で正解集合に含まれる}|}{|\text{フィルタリング後残る文}|}$$

$$\text{再現率} = \frac{|\text{フィルタリング後残る文で正解集合に含まれる}|}{|\text{正解集合}|}$$

$$F\text{ 値} = \frac{2 \times \text{適合率} \times \text{再現率}}{\text{適合率} + \text{再現率}}$$

事前に定義したフィルタリングの組み合わせで F 値が最も高くなる組み合わせを求めた．その際の F 値，適合率，再現率について表 6 に示す．

それぞれのフィルタリングの組み合わせについて以下に示す．

- **英語として読める** 5≤ 語数 <21，識別子の数 <4, カンマの数 <2, 区切り文字の数 <2, 括弧等が閉じている, フランス語でよく用いられる単語を含まない.
- **説明文として役立つ** 7≤ 語数 <17,2≤ 支持度 <9, プロジェクト数=2, グラフの特徴度が上位 40%を超え 10%以下, 説明文の特徴度が上位 90%を超え 50%以下, 識別子の数 <3,”implements” という単語を含まない,”test” という単語を含まない.

4.2 辞書の生成

システムの入力となるソースコードを収集し，実際に辞書の生成を行った．Java プログラムにおける名詞の一般的な用いられ方を説明する辞書を生成した．

ソースコードの収集 辞書の生成を行うために，OSS のソースコードを収集した．Java の一般的な辞書を作成するために，ソースコードのドメインは特定のアプリケーションドメイ

ンに固定せずに Java 全般で広く収集した．また，品質の高いコメントを得るために，最近に開発保守が行われていたり著名な OSS を対象とした．収集した OSS は以下の通りである．

- SourceForge.Net[2] で公開されているプロジェクトで，2010 年 1 月 1 日から 2010 年 10 月 6 日に少なくとも一回のコミットが行われ，Java のタグが付けられたプロジェクト全て．
- Apache Commons[1] として公開されているコンポーネント全て．

取得したソースコードには複数バージョンが含まれていることがある．しかし，現在開発中のバージョンもしくは最新のメジャーバージョン以外のソースコードのみを使用する．収集したソースコードはソフトウェアプロダクト数 1646 個，Java ファイル数 438369 個であった．

そして，辞書を生成した結果，1575 個の名詞について説明文が生成された．

表 7: プログラミング歴

4 年	5 年	5 年以上 10 年未満	10 年以上
3	6	3	2

表 8: Java プログラミング歴

1 年	2 年以上 3 年未満	4 年以上 6 年未満	10 年以上
2	4	7	1

5 評価実験

本節では提案手法への評価実験について述べる．生成した辞書がプログラム理解にとって有用であるを評価するために被験者を用いた実験を行った．実験ではプログラム理解の作業の一部を模した問題を作成し，被験者にその問題を出題した．そして，辞書を利用した場合と利用しなかった場合で正答率と解答に要した時間に差があるかを調べた．また，辞書は，4.2 節で述べた辞書を利用した

実験に参加した被験者は 14 名であり，全員情報科学の教育を受けており，Java プログラミングの経験がある．また，被験者のプログラミング歴と Java プログラミング歴の人数の分布はそれぞれ表 7 表 8 の通りである．

5.1 実験内容

実施した実験の内容について述べる．プログラム理解においてソースコードの読解を行う際に，ソースコード中の一部のキーワードからその周辺の処理を予測する．そして，その作業を模した問題として，被験者に予測を正しく行えるかどうかを問う問題を被験者に行わせた．具体的には，被験者に識別子 (クラス名) を見せその識別子に関連する処理について予測させる．そして，複数のコード片を見せ，その予測にもっとも相応しいコード片を選択させる．その複数のコード片の中には一つだけ見せた識別子に対応するものが存在する．問題を行う際に被験者には辞書を利用する場合と利用しない場合の問題を行ってもらう．また，被験者には実験中には，JavaAPI のリファレンスを参照することや Web で検索して調べることなど実験外の情報の利用を禁止した．

以下，実験について詳しく説明を行う．

実験の手順　実験は以下の手順で行う．

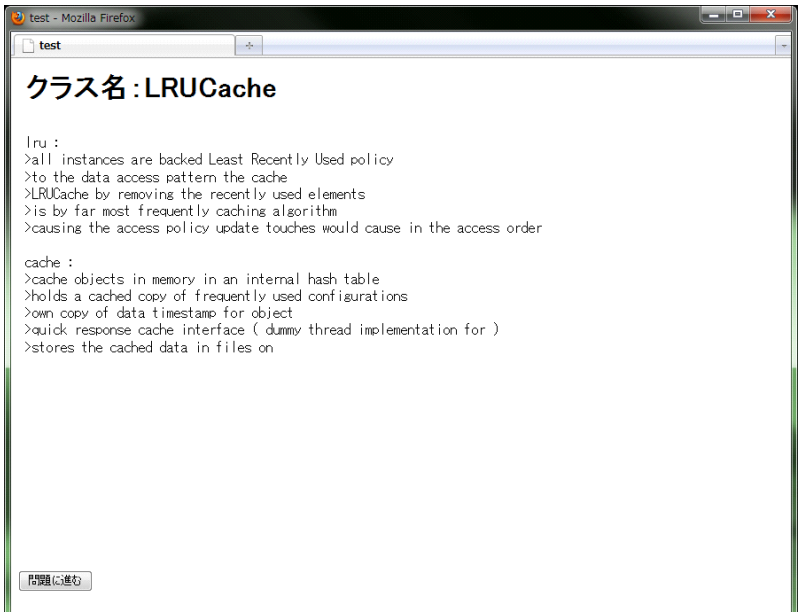

図 5: 実験の例：画面１

1. クラス名を被験者に提示する．
2. クラス名に含まれる名詞で辞書に収録されているならその名詞への説明文を提示する．(説明文利用の場合のみ)
3. そのクラス名から想像されるソースコードを予測してもらう
4. コードを４つ提示する，説明文利用の場合はさらに説明文も提示する．ただし，ここでは複数のコードや説明文とコードを並べて同時に閲覧することはできない．
5. 識別子から予測されるソースコードに近いコードを選択してもらう

実際のソースコードの読解作業は紙面上でなく PC 上で行われるので，Web テストという形でブラウザ上で実験を行った．また，ブラウザ上で実験を行うことで，費やした時間を正確に計測することが可能である．

実際の実験の画面の例について以下に図５，６で示す．手順１～３画面１で，手順４，５が画面２で行う手順である．

5.2 実験の準備

実験に用いた問題の作成法について説明する．

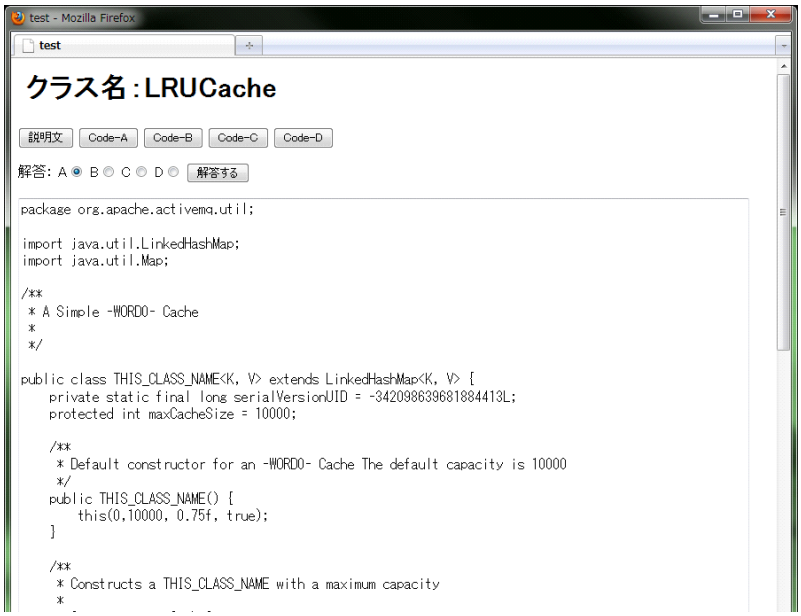

図 6: 実験の例：画面 2

問題に用いるソースコード　今回の実験に用いたコードは 4.2 節の辞書生成のために収集したソースコード集合を利用した．問題に用いたソースコードはいずれも，ソースファイル名を単語に分割した時に 3 単語以下から構成されるようなものかつファイルサイズが 10KB 以下であるソースコードである．1 つの問題に対して 4 つのソースコードを選ぶ．まず，辞書を各名詞に対する説明文の量で降順でソートした際に，上位 500 個の名詞のうちからランダムで一つ選び出す．ソースコード集合中からその名詞をソースファイル名に含むソースコードをランダムに選び出す．そして，そのソースコードを正解となるソースコード，そのソースファイル名を提示する識別子とする．次に，不正解の選択肢で用いるソースコードを選ぶ．提示する識別子に含まれるある単語もしくは，その単語が自然言語上で同位概念か Java Platform Standard Edition 6 の API においてその単語と共通の親クラスを持つような単語を 2 つ選ぶ．ソースコード集合中からそれぞれの単語をソースファイル名に含むソースコードをランダムに選ぶ．選んだ 3 つのクラスとソースファイル名に共通する単語を持たないようなソースコードをソースコード集合から選ぶ．

ソースコードの加工　ソースコードをそのまま被験者に見せると，クラス名が記述されていて問題として成立しないことや，ソースコードによって文量にあまりにも大きな差があるために，ソースコードの 1 部を加工する．

直接答えにつながるであろう記述を隠すために，以下の加工を行う．

表 9: 全体の正答率

	説明文有り	説明文無し
全体の正答率	0.693%	0.621%

- クラス名を”THIS_CLASS_NAME” という単語で隠す
- extends や implements 行っているクラスが一般的なクラスでなく，クラス名と類似している場合に”SUPER_CLASS_NAME” という単語で隠す
- 一般的でないパッケージ名がクラス名中に出現するなど，特定の単語が直接答えに結びつきそうな場合，そのような単語を”WORD0” といった単語で隠す.

各ソースコードで隠す要素の数がなるべく同じくらいになるように加工を行う.

さらに，各ソースコード間の文量の差を小さくするために，以下の要素を削除する場合がある.

- オーバーロードされているメソッドを 1 つを残し他のメソッド.
- アクセサなどのメソッド.
- ライセンス条項，タイムスタンプ，著者情報など，ソースコードの処理に関連しない記述.

説明文の選択　被験者に提示する説明文を決定する．提示する識別子に含まれている名詞に対して，辞書中のその名詞への説明文の中からランダムで 5 文を選択する．非常に類似した説明文が複数選択される場合には，説明文構成する単語がほぼ同じであるものを除くように選択する.

5.3　実験結果

実験を行い得られた結果について示す.

5.3.1　正答率

表 9,10,11 に示すように説明文を利用した場合の方が，利用しない場合に比べて正答率が向上する結果が得られた．被験者の回答が解答となるソースコードである時を正解したとする.

表 10: Java 歴で分けた場合の正答率

	説明文有り	説明文無し
Java 歴 3 年以下	0.667%	0.533%
Java 歴 3 年を超える	0.713%	0.688%

表 11: 問題の正答率で分けた場合の正答率

	説明文有り	説明文無し
正答率 75%以下	0.553%	0.489%
正答率 75%を超える	0.978%	0.891%

表 9 は実施した問題すべてに対する正答率を，表 10 は被験者の Java プログラミングの経験年数が 3 年以下か超えるかで分けた際のそれぞれの正答率を，表 11 は各問題の正答率を調べ正答率が 75%以下か超えるかで分けた際のそれぞれの正答率を示している．示したすべての表において，説明文有りの方が説明文無しより正答率が高くなるという結果が得られた．

辞書の有無によって正答率に差があるかどうかを検定を用いて評価を行った．辞書を利用する際の正答率と辞書を利用しない際の正答率は差がないという帰無仮説を立て，t 検定で帰無仮説を棄却できるかどうかで評価を行った．t 検定の有意水準は 5%で行った．

t 検定を行うには，事前条件として検定の対象の分布が正規分布に従った上で平均が等しい必要がある．今回の実験では，各被験者が説明文有りの問題と説明文無しの問題をどちらについても回答している．そのため，２つの母集団において標本が対になっているので正規分布に従い平均が等しいと仮定できる．また，検定を行った正答率すべてにおいて説明文有りの場合の正答率が説明文無しの正答率を上回っており，説明文により正答率が向上しているのは明らかなので片側検定を行った．

t 検定の結果について表 12 に示す．３つの検定において全て p 値が有意水準を下回ったために帰無仮説を棄却した．そのため，辞書の有無による正答率に有意な差が存在すると言える．

5.3.2 実験における時間

被験者が実験の際に，要した時間について述べる．

被験者が画面１ (図 5.1)，画面２ (図 5.1) において費やした時間をそれぞれ表 13, 表 14 に示す．

表 12: t 検定の結果

	p 値
全体	0.00067816438257421 ($\leq$ 0.05)
Java 歴 $\leq$ 3 年	0.000513 ($\leq$ 0.05)
問題の正答率 $\leq$ 75%	0.00207901201817795 ($\leq$0.05)

表 13: 画面 1 の時間

	説明文有り	説明文無し
滞在時間	74.8	13.1

ここで，画面 1 で計測した時間はこのページを訪れてから，次のページに移るまでの時間である．説明文有りの場合はクラス名を読む時間と説明文を読む時間の合計，説明文無しの場合はクラス名を読む時間だと考えられる．また，画面 2 で計測した時間は，ある説明文もしくはコードを表示するボタンを押してから，説明文か別のコードを表示するボタンを押すか回答の送信ボタンを押すまでである．説明文もしくはコードを表示していた時間がそれぞれを読んでいた時間と考えらえれる．そして，被験者が正解となるコード及び正解以外のコードを読んだ時間でまとめた結果と説明文を見直した時間の結果が表 14 である．

5.4 考察

実験結果から，説明文有りの場合の方が全体の正答率が向上するという結果が示された．このことから辞書を用いることでソースコードを読む際に，識別子から類推される処理とソフトウェアに関する知識と対応づける精度が高くなりプログラム理解に有用であるといえる．また，Java 歴が 3 年以下の被験者のみについて見たときも，正答率が向上するというこ

表 14: 画面 2 の時間

		説明文有り	説明文無し	平均
コードを読んだ時間	正解のコード	49.1	39.9	44.5
	正解以外のコード	104.1	85.3	94.7
	合計	153.3	125.2	139.2
説明文を読んだ時間		19.2		

とが示された．Java プログラムの初学者に対して，本手法で生成した辞書は知識を与えることができることが分かった．さらに，正答率が低い問題においても，正答率が向上した．多くの人が知識を持っていない，もしくは正しく理解していない単語に対する説明を提示することができたと言える．

一方，説明文有りの方が作業時間が長くなる結果が得られた．これは知識の対応付けに要する時間が長くなってしまうということである．しかし，プログラム理解全体の作業時間で見ると，時間が短縮される可能性がある．

プログラム理解において，ソースコードの全体に対して処理を深く追うことで知りたい情報を得るという方法は，膨大な手間を要するために行われることは稀である．まず知りたい情報が記述されていそうな箇所を探しその箇所に対してのみ処理を深く追うことで，効率的にプログラム理解を行う．深く処理を追った箇所が知りたい情報と関連が薄かった場合には，また知りたい情報が記述されていそうな箇所を探さなければならない．ここで，識別子からの予測と知識の対応付けの精度が高いことで，関連度が低い箇所に時間を多く費やすこと未然に防げる．知りたい情報が記述されていそうかどうかの判断に多少の時間を要しても，読まなければならない処理の量を減らすことで，全体として必要な時間が短縮されると予想されるからである．

5.5 内的妥当性

実験で正確にデータを取得することに対する脅威について述べる．

まず，実験における学習効果の影響が考えられる．被験者が問題のパターンを学習することで，後の問題ほど結果がよくなる可能性である．本実験では学習効果による影響がでないように，先に説明文有りの問題を行う被験者と先に説明文無しの問題を行う被験者を同数用意して実験を行っている，さらに被験者毎に出題する問題の順序をランダムにし，出題する問題とそこに用いるコード片を広くランダムに選択することで，問題間で先に行った問題の知見を後の問題に生かしにくい問題を設定したと考えられる．実験の結果としても後半の問題の方が正答率が低いという結果が得られた．

次に，被験者の実験中の疲労やモチベーションによる実験の影響が考えられる．実験を行うことにより，被験者に疲労が蓄積することで被験者のパフォーマンスが低下する可能性がある．本実験では，問題と問題の間で休憩をとることを許可し実験を行った．そのため，被験者がパフォーマンスに影響がでるほど疲労を感じた場合には休憩をとることで，結果への影響は小さいと考えられる．また，実験の課題を明確に正解が存在するクイズ形式で行うことにより，被験者のモチベーションの向上を図った．

5.6 外的妥当性

行った実験設定の一般性に対する脅威について述べる．

1つ目に，行った実験の問題がプログラム理解中で実際に行う作業ではなく，プログラム理解の中で有用性は示せていない可能性がある．しかし，プログラム理解の際に識別子からその処理の内容を予想するという手順は確かに存在し，それを模した課題を設定したことで，プログラム理解において有用であるといえる．

2つ目に被験者の全員が情報科学に対する教育を大学機関で受けており，一般的にソフトウェアエンジニアの多くがそのような教育を受けているとは限らない．しかし，情報科学の教育でも，識別子の命名規則や指し示す概念を体系だって学習しない．そのため，情報科学に対する教育を受けていても被験者自身が実際の事例ベースでしか学習を行うことができなず，被験者自身の経験によるはずである．被験者のプログラミング歴や Java プログラミング歴はある程度分散しているので被験者の一般性は成り立つと考えられる．

6 関連研究

ソースコードを解析することで，知識の集約を行いソフトウェアの特性を調べたり，プログラムに携わる作業者の支援を行う研究が行われている．

Høst ら Java ソースコードを解析することで，メソッド名に用いられる動詞の一般的な用途を説明した辞書を自動的に生成する手法を提案した [11]．また，作業者が識別子へ命名を行うときに不適切であれば警告を示し，適切な命名を支援するツールを作成した [12]．プログラムから識別子を収集し，その中からメソッド名の命名規則を見出し，その命名規則に反する命名を作業者が行おうとした際に警告を表示する．

Falleri らは識別子の上位下位などの関係を自動的に抽出し，作業者のプログラム理解を支援する手法を提案した [7]．識別子に用いられている単語に対して自然言語処理技術を応用することで，識別子の上位，下位，同位関係を得る．そして，その得た関係を作業者にグラフとして提示することで，複数の識別子の関係を視覚的に見ることでプログラム理解の支援を行った．

Caprile らは C 言語の識別子の命名規則を定義し，関数名に使用される単語の辞書を作成した [5]．また，その辞書を用いて関数への再命名を支援するツールを提案した．

Shepherd らはオブジェクト指向プログラミングにおける識別子名中の名詞と動詞をそれぞれソースコード中のオブジェクトや処理に関連付けて可視化する手法を提案した [25]．さらに後に，識別子やコメント中に出現する動詞と目的語の組から，あるメソッドの関心事を検索するツールを作成した [24]．

Lawrie や Binkley らはプログラム理解支援を目的とし，ソースコード中に出現する自然言語を解析することで，識別子に出現する略語を正式名に拡張するツールを作成した [17]．さらに，識別子の表記のスタイルがよって読み終えるために必要な時間がどう影響するのかや，識別子の長さと記憶に残りやすさの調査を行った [4, 3]．

以上の研究に対し本研究では，Java ソースコードを解析することで，プログラム理解支援のため識別子名に用いられる名詞に対する自然言語による説明文を収録した辞書を自動的に生成する手法を提案した．

7　おわりに

識別子に付けられた名前の意味を知るための体系だった学習法はなく，学習するまでに要するコストは高い．そこで，識別子中に出現する名詞の説明文を集めた辞書を生成する手法を提案した．ソースコード中に記述されたコメントと識別子を利用することで自動的に入力として与えたソフトウェアのドメインにおける辞書を生成することができる．また，提案手法を実装したしシステムを作成し，実際に辞書の生成を行った．さらに，被験者を用いて本手法で生成された辞書の評価実験を行った．その結果，辞書を利用することで利用者に知識を与えることができ，プログラム理解に有用であることが分かった．

今後の課題としては，まず辞書に乗せる説明文の順位付けが挙げられる．現状では一つの名詞あたりに生成される説明文の数が大量であり，利用者がすべての説明文に目を通すことは困難である．そのため，説明文に対して順位付けを行う基準を定義し，上位のものから利用者に提示することで，すばやく利用者に有用な知識を提供できるようになる．次に，辞書を効果的に利用者に提供する方法が挙げられる．統合開発環境などのプラグインとして辞書を組み込むことが考えられる．利用者がソースコードを閲覧している際に，キャレットを識別子にかぶせると統合開発環境内にその識別子に含まれる単語の説明を表示する．それにより，辞書を利用しやすい環境を提供できる．また，複数の単語を組み合わせた複合語の説明文の生成が考えらえる．単語が単独で用いられる時とは異なり，特定の単語と用いられるとには特別な意味を持つ場合がある．そのような説明を収録することで，辞書の有用性を向上させることができる．

謝辞

本研究の全過程を通して，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心より深く感謝致します.

本研究を通して，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准教授に深く感謝申し上げます.

本研究を通して，随時適切な御指導および御助言を賜りました東洋大学総合情報学部早瀬康裕助教に深く御礼申し上げます.

本研究の評価実験へのご協力を頂きました大阪大学情報科学研究科及び基礎工学研究科の友人達に深く感謝致します.

最後に，本研究にあたって，有意義な御指導，御助言および，評価実験へのご協力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ソフトウェア工学講座の皆様に深く感謝致します.

参考文献

[1] Apache Commons. `http://commons.apache.org/`.

[2] SourceForge.Net. `http://sourceforge.net/`.

[3] D. Binkley, M. Davis, D. Lawrie, and C. Morrell. To CamelCase or under_score. In *Proceedings of the 2010 IEEE 18th International Conference on Program Comprehension*, pp. 158–167+, 2009.

[4] D. Binkley, D. Lawrie, S. Maex, and C. Morrell. Identifier length and limited programmer memory. *Sci. Comput. Program.*, Vol. 74, pp. 430–445, May 2009.

[5] B. Caprile and P. Tonella. Restructing program identifier names. In *Proceedings of ICSM 2000*, p. 97, 2000.

[6] T. A. Corbi. Program understanding. In *challenge for the 1990's, IBM Syst. J.*, Vol. 28, pp. 294–306, 1989.

[7] R. J. Falleri, M. Huchard, M. Lafourcade, C. Nebut, V. Prince, and M. Dao. Automatic extraction of a wordnet-like identifier network from software. In *Proceedings of the 2010 IEEE 18th International Conference on Program Comprehension*, ICPC '10, pp. 4–13, Washington, DC, USA, 2010. IEEE Computer Society.

[8] R. K. Fjeldstad and W. T. Hamlen. Application program maintenance study. In *Report to Our Respondents, Proceedings GUIDE 48*, 1983.

[9] N. J. Ford and M. Woodroffe. *Introducing Software Engineering.* Prentice Hall, 1994.

[10] T. Hara, Y. Miyao, and J. Tsujii. Evaluating impact of re-training a lexical disambiguation model on domain adaptation of an hpsg parser. In *Proceedings of the 10th International Conference on Parsing Technologies*, IWPT '07, pp. 11–22, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

[11] E. W. Høst and M. B. Østvold. The programmer's lexicon, volume i: The verbs. In *Proceedings of the Seventh IEEE International Working Conference on Source Code Analysis and Manipulation*, pp. 193–202, Washington, DC, USA, 2007. IEEE Computer Society.

[12] E. W. Høst and M. B. Østvold. Debugging method names. In *Proceedings of the 23rd European Conference on ECOOP 2009 — Object-Oriented Programming*, Genoa, pp. 294–317, Berlin, Heidelberg, 2009. Springer-Verlag.

[13] IEEE Std 1219-1998, IEEE Standard for Software Maintenance, 1998.

[14] A. Inokuchi, T. Washio, and H. Motoda. A general framework for mining frequent subgraphs from labeled graphs. In *Fundamenta Informaticae*, Vol. 66, pp. 53–82, 2005.

[15] G. James, J. Bill, S. Guy, and B. Gilad. *The Java Language Specification*. Addison Welsley, third edition, 2005.

[16] D. F. Jerding, J. T. Stasko, and T. Ball. Visualizing interactions in program executions. In *Proceedings of the 19th international conference on Software engineering*, ICSE '97, pp. 360–370, New York, NY, USA, 1997. ACM.

[17] D. Lawrie, H. Feild, and D. Binkley. Extracting meaning from abbreviated identifiers. In *Proceedings of the Seventh IEEE International Working Conference on Source Code Analysis and Manipulation*, pp. 213–222, Washington, DC, USA, 2007. IEEE Computer Society.

[18] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: the penn treebank. *Comput. Linguist.*, Vol. 19, pp. 313–330, June 1993.

[19] A. G. Miller. Wordnet: a lexical database for english. *Commun. ACM*, Vol. 38, pp. 39–41, November 1995.

[20] Y. Miyao, K. Sagae, and J. Tujii. Towards framework-independent evaluation of deep linguistic parsers. In *Proceedings of the GEAF07 Workshop*, pp. 238–258, 2007.

[21] N. Pennington. Comprehension strategies in programming. In *Eds. Empirical Studies of Programmers: 2nd Workshop*, pp. 100–113, 1987.

[22] S. L. Pfleeger and J. M. Atlee. *Software Engineering: Theory and Practice*. Prentice Hall, fourth edition, 2009.

[23] R. W. Selby. Enabling reuse-based software development of large-scale systems. *IEEE Trans. Softw. Eng.*, Vol. 31, pp. 495–510, June 2005.

[24] D. Shepherd, Z. P. Fry, E. Hill, L. Pollock, and K. Vijay-Shanker. Using natural language program analysis to locate and understand action-oriented concerns. In *Proceedings of the 6th international conference on Aspect-oriented software development*, AOSD '07, pp. 212–224, New York, NY, USA, 2007. ACM.

[25] D. Shepherd, L. Pollock, and K. Vijay-Shanker. Towards supporting on-demand virtual remodularization using program graphs. In *Proceedings of the 5th international conference on Aspect-oriented software development*, AOSD '06, pp. 3–14, New York, NY, USA, 2006. ACM.

[26] K. Toutanova and C. D. Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics - Volume 13*, EMNLP '00, pp. 63–70, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[27] A. von Mayrhauser and A. M. Vans. Identification of dynamic comprehension processesduring large scale maintenance. In *IEEE Trans.Softw. Eng.*, Vol. 22, pp. 424–437, 1996.