修　士　論　文

題　　目　　単語ベクトルによる省略識別子の
復元推定手法に関する研究

主任指導教員　　水野　修　准教授

京都工芸繊維大学大学院 工芸科学研究科

情報工学専攻

学生番号　　13622009

氏　　名　　岡嶋　秀記

平成27年2月10日提出

様式 2 号

学位論文の要旨（和文）

平成 27 年 2 月 10 日

京都工芸繊維大学大学院
工芸科学研究科長　殿

工芸科学研究科　　情報工学専攻
平成 25 年入学
学生番号　　13622009
氏　　名　　岡嶋　秀記　㊞

（主任指導教員　　水野　修　㊞　）

本学学位規則第 4 条に基づき、下記のとおり学位論文内容の要旨を提出いたします。

1.　論文題目

単語ベクトルによる省略識別子の復元推定手法に関する研究

2.　論文内容の要旨（400 字程度）

ソースコードの可読性はソフトウェアを開発・保守する上で重要な要素である．識別子はソースコードを正しく理解するための重要な情報源であるが，コーディング時にプログラマの判断でしばしば省略される場合がある．識別子の過度な省略は初見の読者にとって理解の妨げとなりうる．本報告では省略された識別子の元となった識別子を推定する手法を提案する．近年，深層学習を用いて自然言語を学習し，単語の意味を高次元のベクトルで表せるツール word2vec が注目されている．提案手法では word2vec を用いてソースコードから識別子の意味を表すベクトルに変換し，ベクトル間の距離を比較する事で元の識別子を推定する．オープンソースプロジェクトのリポジトリを用いた事例研究の結果，本手法で省略された識別子を完全な識別子に復元できた．また，完全に失敗した場合でも意味の関係が深い識別子を提示できるため，プログラミング支援のツールとして有効である．

Study on an approach for abbreviated identifier restoration with word vector computation

2015 13622009 *Hideki OKAJIMA*

Abstract

The readability of source code is a crucial factor on software development and maintenance. Identifier is a essential clue to understand the content of source code, but identifiers are often abbreviated to short form by programmers. Unreasonable abbreviation causes hindrance to understanding for readers faced with unfamiliar code. This paper presents an approach for expanding identifiers using a tool "`word2vec`". The `word2vec` is a tool to learn the vector representations of the words in documents by deep learning. In this approach, the candidates of original identifier by transforming source code to the vector representation of identifiers and comparing short identifiers vector with long one are determined. The process on code of open source software repositories is evaluated and it is indicated that the proposed approach is able to successfully expand many abbreviated identifiers. The present approach suggests some similar meaningful identifiers, of abbreviations can't expand correctly. Therefore it is concluded that the present approach is helpful for reading source code.

目　次

1. 緒言

ソフトウェア開発やメンテナンスを高品質かつ効率的に行うためには，ソースコードを正しく理解する事が求められる．ソースコード中の変数や関数などにつける名前を識別子といい，ソースコードの内容を理解するための重要な情報源となる．識別子の表記法はプログラミング言語により異なり，命名規則も組織やプロジェクトにおいて明確に定められている場合もあるものの，多くの場合，プログラマが変数や関数などの役割や属性などを考慮して自由に命名する．このように識別子の命名は自由度が高く，プログラマはしばしば識別子を省略して短い識別子を用いる場合がある．識別子の省略はコーディングの効率を上げるなどの利点がある一方で，他人が書いたコードを読む場合などは識別子の役割が識別子の名前から判断できずコードの可読性を下げる要因となる [1]．本研究では省略された識別子の元の識別子を推定する事で，プログラマに識別子の本来の意味を提示し，生産性向上に寄与することを目指す．

既存の復元手法ではソースコードやドキュメンテーションコメント，辞書などの資料からパターンマッチングを利用して省略された識別子を復元している [4] が，識別子の用途や意味の関連性は考慮されていない．ソースコードはテキストデータであるため，自然言語を計算機で扱う自然言語処理と相性がよく，機械学習がデータマイニングに用いられれいる [10]．近年自然言語から単語の意味を高精度で学習できる手法が開発され [7, 8, 9]，そのオープンソース実装である `word2vec`[6] が公開された．`word2vec` は単語の意味を高次元のベクトルで表せるため，意味の足し引きや距離の計算が可能である．本研究では `word2vec` のこの特徴に着目し，識別子間の意味の距離を比較することで似た意味を持つ識別子を推定し，省略された識別子を省略される前の識別子に復元する．識別子の復元に識別子の意味の概念を導入することで，開発者がソースコードをより直感的で理解の助けとなる手法を提案する．

本稿では「省略されている（可能性のある）識別子」を状況に応じて「省略識別子」または「省略形」と表す．また,「省略識別子の省略元となった（復元されるべき）識別子」を状況に応じて「標準識別子」または「標準形」と表す．

本稿の構成を紹介する．2 章では研究の基礎となる関連研究について述べる．3 章では本稿で提案する省略識別子の復元手法について詳しく説明する．4 章ではオー

プンソースプロジェクトのリポジトリを用いて実際に提案手法を適用した事例研究の内容と考察を述べる．5 章でまとめと今後の課題に付いて述べる．

2. 関連研究

2.1 先行研究

省略識別子を復元する最も単純な手法は，省略形と標準形を対応させた辞書を作ることである [11, 5]．Lawrie ら [5] は復元する単語や語句の辞書を作る事で，約 16%の識別子が復元できたと主張している．しかしながら，この手法ではすべての省略識別子を復元するのは困難である．また,「comp」のように「compare」や「component」など文脈により違う標準形に復元しなければならない場合には対応できない．加えて，手作業で作る辞書では，制作者の知識を越える範囲の識別子をカバーできない．

Hill ら [4] は文脈を考慮した復元手法を提案している．Hill らはソフトウェアリポジトリマイニングの技術を利用して，関数ソースコード，関数のドキュメンテーションコメント，クラスのドキュメンテーションコメントと検索範囲を広げて行くことで Lawrie ら [5] の手法と比べて 57%の向上したと主張している．Hill らの手法では識別子の省略方法を細かく場合分けし，パターンマッチングを用いてその識別子が正しい標準識別子かを判断している．Hill らの手法は識別子のマッチングアルゴリズムが複雑であり，ソースコードおよびコメント内に標準識別子が無ければまったく復元ができない．

2.2 自然言語処理

人間が日常的に使っている自然言語をコンピュータで処理する技術を自然言語処理 [2] といい，長年研究が行われている．自然言語処理の分野の 1 個に単語の意味の解析がある．同じ文脈で出現する単語は同じ意味を持つという仮説を分布仮説 [12] といい，この性質を利用して自然言語の分布から語句の意味を機械的に推定できる．単語の意味が解析できると，同義語や対義語の関係や未知の単語の意味，異なる言語間での意味の関係などの類推が可能となる．Tomas Mikolov らはニューラルネットワークと skip-gram モデルを用いて自然言語から単語の意味を表すベクトルに変換する `word2vec`[6] というオープンソースのツールを提案した [7, 8, 9]．`word2vec` は従来の手法と比べ高精度な意味解析を可能とし，得られるベクトルには以下のような様々な特徴がある事が分かっている．

- 意味の足し引き，距離の計算が可能.
- 異なる言語でも同じベクトル構造を持つ.

word2vec は現在も盛んに研究が続けられている.

2.3 リポジトリマイニングと機械学習

ソフトウェアリポジトリマイニングとは版管理システムやバグトラッキングシステムなどに蓄えられているソースコードやコメントなどの膨大な情報から統計分析や機械学習，自然言語処理などの分析技術を用いて，プログラミングやバグ検出，テスト.保守などの改善を目指す分野である.. ソフトウェア工学の分野において盛んに研究が勧められている．ソフトウェアリポジトリマイニングのメトリクスに機械学習を用いる研究がなされている [10]．Mizuno ら [10] はソースコードをテキストと見なし，スパムフィルタリングの技術でモジュールの不具合の有無を判定する手法を提案している.

3. 省略識別子の復元推定

3.1 研究設問

本研究の目的を明確にするために以下の研究設問を設定する．

RQ1 ソースコードから識別子のベクトルを学習可能か．

RQ2 識別子のベクトルを用いて省略識別子を復元可能か．

RQ3 ベクトルを用いてどの程度正確に省略識別子を復元可能か．

RQ1 ではソースコードを文章と見なして `word2vec` を適用した場合に，識別子の意味を表すベクトルが学習できるかを調査する．RQ2 では識別子の意味を表すベクトルを比較することにより，省略された識別子を省略される前の識別子に復元することが可能かを調査する．RQ3 では識別子ベクトルの比較を用いた省略識別子の復元手法でどの程度正確に識別子の復元が可能かを調査する．

3.2 省略識別子

識別子の省略は以下のように分類できる [4]．

single-word 省略される元の識別子が一個の単語で構成されている識別子である．省略のパターンは以下の 2 種類がある．

prefixes 単語の頭文字をとって短い形式の識別子とする．例えば「object」を省略する場合，先頭の 3 文字をとって「obj」となる．また，「exception」を「e」と一文字に省略する場合もある．

dropped letters 単語のいくつかの文字を欠落させて短い形式の識別子とする．例えば「message」を「msg」と省略する．

multi-word 省略される元の識別子が複数の単語で構成されている識別子である．single-word の prefixes と dropped letters を組み合わせて短い形式の識別子とする．複数の単語の接続には主に以下の 3 種類がある．

camelCase 単語間の接続部分を大文字にすることで区切っている．

snake_case 複数の単語をアンダースコアで接続した識別子である．

dot.notation 複数の単語をドットで接続した識別子である．

multi-word の中には以上の規則に従わず，複数の単語を全て小文字で接続しているものも存在する．

表 3.1 は以上の分類と省略例をまとめたものである．本研究ではこれら全ての種類を復元の対象とする．

3.3 単語ベクトル

本研究では word2vec というツールを用いてソースコードから識別子の意味を表すベクトルに変換する．word2vec は Mikolov らによって提案された論文のオープンソース実装である．word2vec は巨大なテキストデータから高次元の意味ベクトルを短時間で求められる．得られるベクトルは大きさが 1 で，各成分によって表されるベクトルの向きがその単語の意味を表す．2.2 の項目 1 より，word2vec により得られる単語ベクトルは意味の足し引きができる特徴を有する．

$$\overrightarrow{king} - \overrightarrow{man} + \overrightarrow{woman} = \tag{3.1}$$

例えば式 3.1 は，「“king” のベクトルから “man” のベクトルを引き “woman” のベクトルを足して得られるベクトルの近くには何があるか」という問いを表すが，Mikolov ら [7] は “queen” のベクトルが近くにあると主張している．また，単語の意味をベクトルで表現するため，コサイン類似度を求める事で各単語間の意味の距離を推定できる．距離は-1 から 1 の値をとり，1 に近いほど関係が深く，0 に近いほど関係は薄い．

3.4 学習

以下の手順で識別子の意味ベクトルのコーパスを作成する．

1. ソースコードリポジトリから各リビジョンのソースコードを取得する．
2. ソースコードから識別子を抽出する．

表 3.1　識別子の分類

分類	略し方	識別子	省略の例
single-word	prefixes	object	obj
		attribute	attr
		parameter	param
		organization	org
		exception	e
		integer	i
	dropped letters	event	evt
		message	msg
		source	src
		original	org
multi-word	camelCase	camelCase	cc
	snake_case	snake_case	sc
	dot.notations	dot.notation	dn
			dno
			dnt

3. ユニークな識別子を抽出し ID_{all} とする．

4. word2vec を用いて識別子の意味を学習する．

これらの手順を図 3.1 に示す．次項で各手順の詳しい手順を述べる．

3.4.1 ソースコードの取得

本研究では以下の理由でプロジェクトごとにコーパスを作成する．

- 省略された識別子の省略前の識別子はプロジェクト内にある．
- 同じ省略識別子でもプロジェクトごとに意味が異なる可能性がある．

word2vec の「巨大な訓練データから高速に学習する」という特徴を活かすため，版管理ソフトで複数のリビジョンのソースコードを取得し，利用する．

3.4.2 識別子の抽出

手順 1 で得られたソースコードから「lscp」[13] を用いて識別子を抽出する．lscp は様々な機能が備わったソースコード解析のプリプロセッサであり，ソースコードからコメント，文字列リテラル，識別子などを区別し抽出できる．また，lscp には「camelCase」「snake_case」「dot.notation」などの複数の単語の複合識別子を単語ごとに分割する機能を有する．それぞれの単語が独自の意味を持っていると考えられるため，学習には複合単語を分割した訓練データが必要である．加えて，本研究では multi-word 識別子も復元の対象としているため，これらの複合単語を分割しない訓練データも必要である．よって，本研究では複合単語を分割したデータと分割していないデータの両方を訓練データとする．

3.4.3 意味ベクトルの学習

手順 2 で得られた識別子の訓練データから word2vec を用いて学習する．得られたコーパスからユニークな識別子を抽出し，ID_{all} とする．

3.5 復元推定

次の手順で省略識別子を復元する．

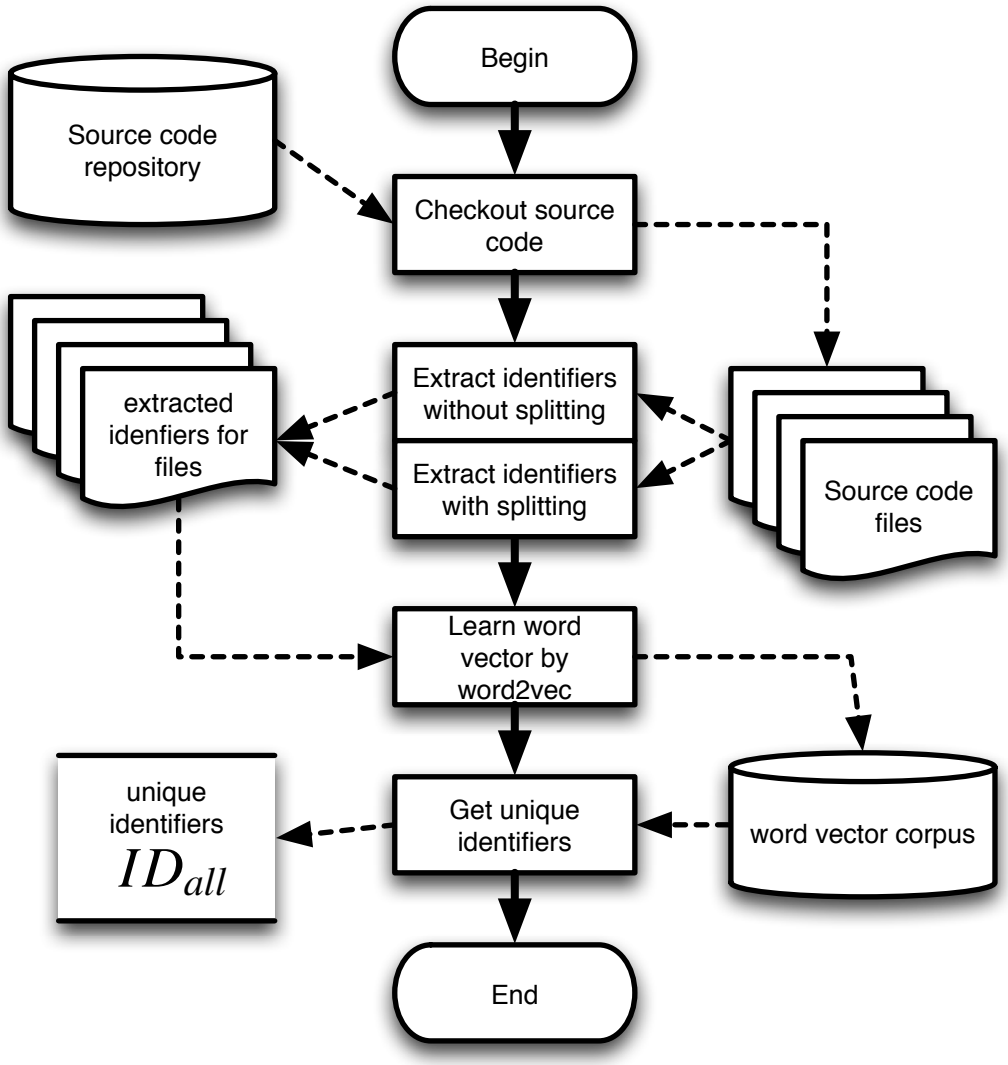

図 3.1　コーパスの学習手順

1. ID_{all} から 3 文字以下の識別子を抽出し，省略識別子の候補 ID_{abbrev} とする．
2. 省略識別子の候補 ID_{abbrev} のそれぞれの識別子に対して，4 文字以上でかつ意味ベクトルの近い識別子を標準識別子の候補 W_{all} とする．
3. 標準識別子の候補 W_{all} から省略識別子の各文字がその順番で出現する識別子を最終的な復元識別子の候補 W_{long} とする．

以上の復元手順について実例を用いて説明する．表 3.2，3.4，3.6 および 3.8 は 4 章で事例研究として用いる Apache ANT の手順 2 の結果であり，識別子「var」「cce」「bcp」および「buf」の W_{all} を示している．これらの表の列「推定標準識別子」は省略識別子の復元候補を示し,「距離」は省略識別子と標準識別子の候補の意味ベクトル間の距離を示し,「順位」は距離に基づいた候補の順位である．

表 3.2 より，省略識別子「var」の正しい標準識別子だと推測される「variable」は第 1 位の候補となった．表 3.6 のより，省略識別子「bcp」の正しい標準識別子だと推測される「xbootclasspath」および「bootclasspath」はそれぞれ第 1 位および第 2 位の候補となった．

表 3.4 より，省略識別子「cce」と最も距離の近い識別子は「cast」である．省略識別子「cce」の正しい標準識別子は「classcastexception」だと推測されるが，表 3.4 では第 2 位である．表 3.8 の識別子「buf」の場合でも同じ状況が確認できた．

次に手順 3 について説明する．省略識別子「cce」の場合には W_{all}^{cce} から「c」「c」「e」の文字がこの順番で含まれる候補のみを抽出する．その結果 W_{long}^{cce} を表 3.5 に示す．正しい標準識別子だと推測される「classcastexception」が第 1 位になっていることが分かる．省略識別子「buf」の場合もまた，正しい標準識別子だと推測される「buffer」が表 3.8 ではランク外だったにもかかわらず，手順 3 を適応することで表 3.9 に示すように第 5 位となった．

表 3.3 および表 3.7 においても正しい標準識別子だと推測される候補が上位の順位になったことを確認した．

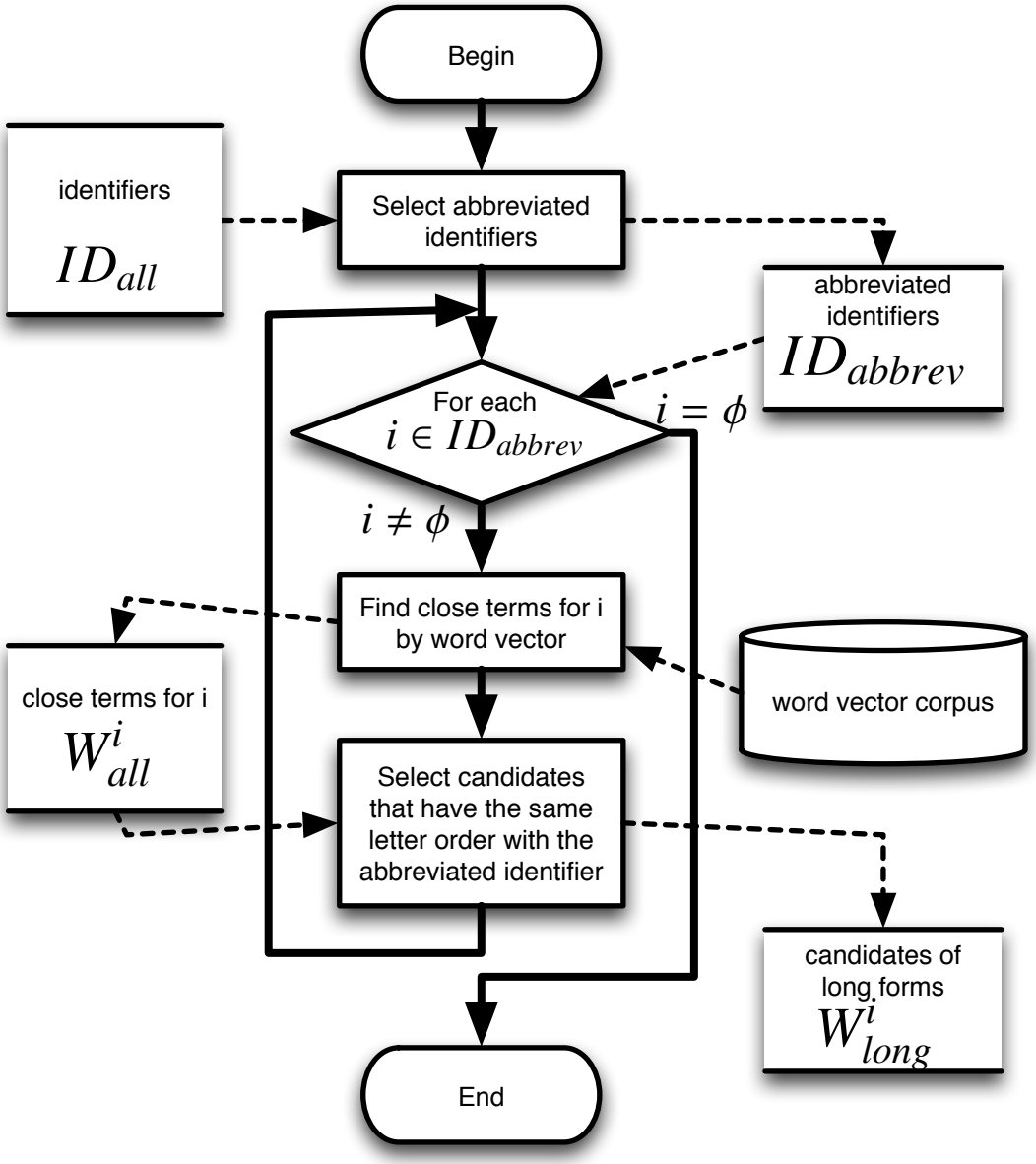

図 3.2　省略識別子の復元手法

表 3.2　識別子「var」と距離が近い識別子 W_{all}^{var}

順位	距離	推定標準識別子
1	0.772	variable
2	0.716	addvariable
3	0.687	cvs_client_port
4	0.678	cvs_passfile
5	0.664	environment
6	0.644	cvs_pserver_port
7	0.639	cvs_rsh
8	0.635	setkey
9	0.533	addenv
10	0.529	xmxm
...	...	...

表 3.3　「var」の文字がその順で含まれる識別子 W_{long}^{var}

順位	距離	推定標準識別子
1	0.772	variable
2	0.716	addvariable
3	0.515	variables
4	0.482	getvariablesvector
5	0.441	getvariables
6	0.365	envvars
7	0.352	getenvironmentvariables
8	0.316	vars
9	0.305	extractvariablepart
10	0.271	getvariable
...	...	...

表 3.4　識別子「cce」と距離が近い識別子 W_{all}^{cce}

順位	距離	推定標準識別子
1	0.597	cast
2	0.571	classcastexception
3	0.507	interesting
4	0.485	caused
5	0.466	cnfe
6	0.437	checkpropertyvalid
7	0.413	innercreateandset
8	0.401	error_loading_caused_exception
9	0.389	error_no_base_exists
10	0.376	ncdfe
...	...	...

表 3.5　「cce」の文字がその順で含まれる識別子 W_{long}^{cce}

順位	距離	推定標準識別子
1	0.571	classcastexception
2	0.437	checkpropertyvalid
3	0.401	error_loading_caused_exception
4	0.345	classnotfoundexception
5	0.341	setcreatemissingpackageinfoclass
6	0.308	testpropcacheinvalid
7	0.275	forcelacheck
8	0.266	setforcelacheck
9	0.266	cachetokens
10	0.259	setcachetokens
...	...	...

表 3.6　識別子「bcp」と距離が近い識別子 W_{all}^{bcp}

順位	距離	推定標準識別子
1	0.736	xbootclasspath
2	0.730	bootclasspath
3	0.720	concatsystembootclasspath
4	0.666	getbootclasspath
5	0.643	calculatebootclasspath
6	0.628	boot
7	0.609	haveclasspath
8	0.600	havebootclasspath
9	0.575	getvmversion
10	0.573	dolinks
...	...	...

表 3.7　「bcp」の文字がその順で含まれる識別子 W_{long}^{bcp}

順位	距離	推定標準識別子
1	0.736	xbootclasspath
2	0.730	bootclasspath
3	0.720	concatsystembootclasspath
4	0.666	getbootclasspath
5	0.643	calculatebootclasspath
6	0.600	havebootclasspath
7	0.524	setbootclasspathref
8	0.521	createbootclasspath
9	0.460	setbootclasspath
10	0.392	systembootclasspath
...	...	...

表 3.8　識別子「buf」と距離が近い識別子 W_{all}^{buf}

順位	距離	推定標準識別子
1	0.689	andselect
2	0.625	majorityselect
3	0.620	noneselect
4	0.605	orselect
5	0.602	notselect
6	0.573	noofentries
7	0.573	stringbuffer
8	0.564	tostring
9	0.538	stringbuilder
10	0.528	append
...	...	...

表 3.9　「buf」の文字がその順で含まれる識別子 W_{long}^{buf}

順位	距離	推定標準識別子
1	0.573	stringbuffer
2	0.416	getfilledbuffer
3	0.414	buflen
4	0.338	bufsize
5	0.334	buffer
6	0.334	large_buffer_size
7	0.332	buildfilterchain
8	0.326	getoutputbuffer
9	0.322	testendswithemptybuffer
10	0.309	cleanedbuffer
...	...	...

4. 事例研究

4.1 対象プロジェクト

本研究では Apache ANT，Apache MINA および EC-CUBE の 3 個のオープンソースプロジェクトを利用する．Apache ANT および Apache MINA のソースコードリポジトリは Apache git repositories[14] から入手する．EC-CUBE のソースコードリポジトリは GitHub[15] から入手する．それぞれのプロジェクトの特徴を以下に記す．

Apache ANT ビルドツールソフトウェア

Apache MINA ネットワークアプリケーションフレームワーク

EC-CUBE コンテンツマネージメントシステム

これらのソースコードリポジトリからリリース版のリビジョンをチェックアウトし，そのリビジョンに含まれるすべてのソースコードを学習の対象とする．それぞれのプロジェクトの主なプログラミング言語と使用するリビジョン数を表 4.1 に示す．

4.2 単語ベクトルの学習

3.4 節の手順に従いソースコードから `lscp` を用いて識別子を抽出する．`lscp` の実行例を以下に示す．

```
use lscp;

my $lscp = lscp->new;

# The verbosity of the program. Options: "info", "warn", "error", "fatal"
$lscp->setOption("logLevel", "error");
# The directory containing the input files.
$lscp->setOption("inPath", "input_dir");
# The directory for the output files.
$lscp->setOption("outPath", "output_dir");
# Are the input files source code (1), or regular text files (0)?
$lscp->setOption("isCode", 1);
```

表 4.1　適用プロジェクト

プロジェクト名	言語	使用リビジョン数
Apache ANT	Java	25
Apache MINA	Java	33
EC-CUBE	PHP	32

```
# If isCode = 1, should the program include identifier names?
$lscp->setOption("doIdentifiers", 1);
# If isCode = 1, should the program include string literals?
$lscp->setOption("doStringLiterals",0);
# Should the program create all lower case output?
$lscp->setOption("doLowerCase",1);
# Should the program remove puncuation symbols?
$lscp->setOption("doRemovePunctuation",1);
# Should the program split identifier names,
# such as:camelCase, under_scores, dot.notation?
$lscp->setOption("doTokenize", 0);
# Should the program perform word stemming?
# Note that stemming = 1 implies doLowerCase = 1.
$lscp->setOption("doStemming", 0);
# If isCode&=& 1, should the program include comments?
$lscp->setOption("doComments", 0);
# Should the program remove English stopwords?
$lscp->setOption("doStopwordsEnglish", 0);
# Should the program remove programming language keywords?
$lscp->setOption("doStopwordsKeywords", 1);
# Should the program remove digits [0-9]?
$lscp->setOption("doRemoveDigits", 1);

$lscp->preprocess();
```

ソースコードから識別子のみを抽出するよう設定した．識別子には camelCase などで大文字が使われる事があるが，大文字と小文字の違いで識別子の意味が変わることは稀なので本実験ではすべて小文字に変換した．通常，動詞は活用が変わるとそ含まれる意味が変わる．また，word2vec には動詞の活用などを学習することが可能 [3] なので，本実験ではステミングは行わない．ストップワードが復元の対象である可能性もあるため，この段階では英語のストップリストを排除しない(注 1)．本実験では multi-word の識別子を分割するデータと分割しないデータの両方を用いるため,「doTokenize」のオプションが 0 の場合と 1 の場合の 2 回実行する．それぞれのプ

(注 1)：word2vec の頻出単語を間引くオプションを利用する．

ロジェクトにおける学習に用いた識別子の総数とユニークな識別子の種類数を表 4.2 に示す.

次に得られた識別子のデータから word2vec を用いて単語ベクトルを学習する．学習に用いた word2vec のオプションのサンプルを以下に示す.

```
word2vec -train input.txt -output output.bin -cbow 0 -size 200
    -window 10 -negative 0 -hs 1 -sample 1e-3 -threads 12 -binary 1
```

それぞれのオプションの意味を以下に示す.

-train input.txt 入力ファイルを「input.txt」とする.

-output output.bin 出力ファイルを「output.bin」とする.

-cbow 0 学習のモデルとして CBOW ではなく Skip-gram を使用する.

-size 200 ベクトルの次元数を 200 次元とする.

-window 10 文脈は最大 10 単語とする.

-negatibe 0 ネガティブサンプリングを使用しない.

-hs 1 学習の高速化のために階層的ソフトマックスを使用する.

-sample 1e-3 頻出語をランダムに削除する.

-threads 12 学習を 12 スレッド並列で行う.

-binary 1 出力はバイナリ形式で行う.

学習には MacBook Pro を使用した．MacBook Pro のスペックを以下に示す.

プロセッサ 2.9 GHz Intel Core i7

メモリ 8 GB 1600 MHz DDR3

OS OS X 10 9.5

学習に要した時間を表 4.3 に示す．それぞれの項目の意味を以下に示す.

real プログラムの呼び出しから終了までにかかった実時間

user プログラム自体の処理時間（ユーザ CPU 時間）

sys プログラムを処理するために，OS が処理をした時間（システム時間）

表 4.2　プロジェクトに含まれる識別子数

プロジェクト名	識別子の総数	識別子の種類数
Apache ANT	18,938,966	22,543
Apache MINA	8,319,180	7,242
EC-CUBE	47,599,795	18,491

表 4.3　学習時間

プロジェクト名	real	user	sys
Apache ANT	23m22.081s	87m20.872s	0m11.135s
Apache MINA	8m47.096s	31m25.159s	0m4.776s
EC-CUBE	56m47.211s	203m46.975s	0m23.582s

4.3 省略識別子の拡張

本研究では ID_{all} から 3 文字以下の識別子を抽出し，省略された可能性のある識別子 ID_{abbrev} とする．それぞれのプロジェクトの ID_{abbrev} に含まれる識別子の個数を表 4.4 に示す.

4.3 節の手順に従い，ID_{abbrev} に含まれるすべての識別子を復元する．

4.3.1 評価

本研究では 3 文字以下の識別子を省略されている可能性のある識別子だと仮定し，そのすべて対して復元を試みた．推定標準識別子が正解であるかの判定は本研究室のプログラミング経験のある大学院生が手作業で行った．復元候補の第 16 位まで正解であるかの判定をしたが，チェックの手間を省くために識別子間の意味ベクトルの距離が 0.2 未満の候補を機械的に省いている．以下に正解の判定を行った際の基準を示す.

- 候補に時制が異なる動詞があった場合，原型を選択.
- 候補に単数形および複数形の単語があった場合，省略識別子の末尾が「s」の場合は複数形を選択.「s」の有無に関わらず，単数形または複数形しかなければその単語を選択.
- 省略識別子が拡張子の場合，候補にファイル名か説明文があれば選択.
  例: zip → zipfile，txt → default_suffix_text，py → python
- 省略識別子がコマンド名の場合，コマンドと明示されていれば選択.
  例: gcj → setupgcjcommand
- 省略識別子がメタ構文変数の場合は未選択.

4.4 結果および考察

4.4.1 識別子ベクトルの学習

研究設問 1「ソースコードから識別子のベクトルを学習可能か.」について述べる．表 4.5 に Apache ANT のコーパスを用いて

表 4.4　省略識別子とした識別子の数

プロジェクト名　3 文字以下の識別子の種類数	
Apache ANT	971
Apache MINA	456
EC-CUBE	1119

$$\overrightarrow{output} - \overrightarrow{input} + \overrightarrow{pop} = \tag{4.1}$$

の計算をした結果に近いベクトルを持つ識別子を示す．式 4.1 は「“output” から “input” を引いて “pop” を足したものは何か.」，言い換えると「『“input” → “output”』の関係を “pop” に適用したものは何か.」という計算に近いベクトルをもつ識別子を求めている.「input」および「output」はそれぞれ「データの入力」および「データの出力」を意味すると考えられる．つまり「input」と「output」の間には「対」の関係があると言える．一方「pop」はプログラミングにおいて「スタックから要素を取り出す」操作を表す.「pop」の対となる単語は「スタックに要素を追加する」操作を表す「push」である．ソースコードから識別子の意味を正しく学習できているならば式 4.1 の結果 “push” の単語ベクトルが近くに存在するはずである．表 4.5 に示すように「push」が第 1 位に順位付けされており，ソースコードから識別子の意味が学習できている事が分かる．

表 4.5 より，距離は少し遠くなるが第 2 位以下についても「push」と関係の深い単語が算出されている．第 3 位の「stack」は「push」を基本操作とするデータ構造そのものである．第 2 位の「handlers」について，スタックは割り込みハンドラと相性がよく，共に用いられる事が多い．第 4 位の「peek」について，Java にはスタックの先頭の要素を削除せずに取り出す「Stack.Peek」メソッドが存在する．第 5 位の「enelement」はスタックに保存されていた最後の要素を表す識別子だと考えられる．

以上の結果より，研究設問 1 に対する答えは「可能」だと結論づける．また，学習したベクトルを加減算することで識別子の意味の加減算ができることが示された．加えて，近い意味ベクトルを持つ識別子は関係の深い識別子である可能性が高い．

4.4.2 単語ベクトルを用いた復元

研究設問 2「識別子のベクトルを用いて省略識別子を復元可能か.」について述べる．

プロジェクト Apache ANT および Apache MINA の省略識別子を復元した結果の特徴的な識別子を表 4.6 および 4.7 に示す．提案手法では multi-word の復元の精度が優れている．3 文字で 3 文字目が「e」の省略識別子の上位の標準識別子の候補に文字

表 **4.5**　「$\overrightarrow{output}$-$\overrightarrow{input}$+$\overrightarrow{pop}$=」の計算結果

順位	距離	推定標準識別子
1	0.582	push
2	0.471	handlers
3	0.468	stack
4	0.445	peek
5	0.398	endelement
...	...	...

表 4.6　ANT における省略識別子の代表的な復元結果

分類	省略形	推定標準形	順位	距離
single	obj	object	1	0.473
		getobject	2	0.459
	pos	parseposition	1	0.378
		findtargetposition	2	0.371
		position	4	0.348
	len	actuallength	1	0.457
		newlength	3	0.438
	env	newenvironment	1	0.679
		environment	2	0.662
	buf	stringbuffer	1	0.573
		getfilledbuffer	2	0.416
	msg	getmessage	1	0.590
		message	4	0.395
	var	variable	1	0.771
		addvariable	2	0.716
		variables	3	0.515
	fg	foreground	8	0.282
multi	bcp	xbootclasspath	1	0.736
		bootclasspath	2	0.729
	cce	classcastexception	1	0.571
	npe	nullpointerexception	1	0.482
	fne	filenotfoundexception	1	0.387
	nfe	numberformatexception	1	0.857
	nse	nosuchmethodexception	2	0.559
	fc	filterchain	1	0.636
	d	yyyymmdd	9	0.374

表 4.7　MINA における省略識別子の代表的な復元結果

分類	省略形	推定標準形	順位	距離
single	obj	putobject	1	0.315
		expiringobject	2	0.298
		expobject	3	0.262
		testinheritedobjectserialization	4	0.234
	pos	position	1	0.421
		responsepos	3	0.349
		getoverflowposition	4	0.346
	len	destlength	1	0.399
		byte_digest_length	2	0.368
		byte_block_length	3	0.351
		encodeinitialgreetingpacket	4	0.339
	ctx	getcontext	1	0.529
		context	2	0.482
		gss_context	3	0.450
		createcontext	4	0.434
multi	baf	bytearrayfactory	1	0.723
		getbytearrayfactory	2	0.612
		simplebytearrayfactory	3	0.600
	tmf	trustmanagerfactorykeystore	1	0.845
		trustmanagerfactoryalgorithm	2	0.826
		trustmanagerfactoryparameters	3	0.812
		trustmanagerfactoryprovider	4	0.807
	nfe	numberformatexception	1	0.813
	pde	protocoldecoderexception	1	0.614
	pee	protocolencoderexception	1	0.672

列「exception」が含まれているものが複数存在した．これは，word2vec による学習において「exception」の意味を十分に学習できていることを示している．また，識別子は省略されてもソースコードの文脈において同じ役割を果たしていると言える．

「exception」の復元はEC-CUBEでは確認できなかった．Apache ANTおよびApache MINAでのみ出現し，共に高精度であった理由として，

- Apache ANTおよびApache MINAは共に開発元がApacheソフトウェア財団であり，ソースコードの共有が盛んに行われている．
- 開発元が異なるため，コーディング規約などが異なる．

などが考えられる．

一方，EC-CUBEの省略識別子の復元では，時間や時期などに関する省略識別子の復元の精度が良かった．EC-CUBEにおいて月の省略識別子を復元した際の結果を表4.8に示す.「may」を除く(注2)11個中7個の月が第1候補で正しく復元できており，9個の月が第2候補までで正しく復元できている.「sep」および「dec」の正しい標準識別子である「september」および「december」は確認した距離0.2以上には含まれていなかった.「dec」と略されるものは「december」以外に「decorate」や「decode」などが存在し，それらが上位の候補となっていた．識別子「de」の第6候補で「december」が復元できていたため，EC-CUBEのプロジェクトでは「de」が「december」の省略形として使われている可能性が高い.「sep」については正しい標準識別子と考えられる識別子が候補に無く，復元できなかった理由は不明である．

次にEC-CUBEにおいて曜日の省略識別子を復元した際の結果を表4.9に示す．7個すべての省略識別子と期待する推定識別子の意味ベクトルの距離が0.5以上の近い距離を示している．7個中4個の曜日が第1候補で復元できており，5個の曜日が第2候補までで正しく復元できている.「mon」の上位の標準識別子の候補には「month」が含まれるものがあり,「monday」は第6候補まで出現しなかった.「wed」の上位の標準識別子の候補には「weekday」が含まれているものがあり,「wednesday」は第7候補まで出現しなかった．これらより，word2vec の学習で時間や時期などに関する識別子の意味の特徴を学習できていることが分かる．その一方で，意味の距

(注2):「may」は3文字で意味をなしており，省略されていないため．

表 4.8　EC-CUBE における月の省略識別子の復元結果

省略形	推定標準形	順位	距離
jan	january	1	0.765
feb	february	1	0.502
mar	march	2	0.728
apr	april	1	0.531
may	may	-	-
jun	june	1	0.729
jul	july	1	0.703
aug	august	1	0.703
sep	september	-	-
oct	october	2	0.314
nov	november	1	0.543
dec	december	-	-

表 4.9　EC-CUBE における曜日の省略識別子の復元結果

省略形	推定標準形	順位	距離
mon	monday	6	0.530
tue	tuesday	1	0.640
wed	wednesday	7	0.640
thu	thursday	1	0.595
fri	friday	1	0.646
sat	saturday	2	0.631
sun	sunday	1	0.572

離が近い復元の候補が複数ある場合には本手法での復元では期待する標準識別子の候補を上位に順位付けできない可能性がある．

以上で示したように省略された識別子から標準形の識別子に復元する事が可能である．必ずしも第 1 候補で適切な標準識別子が推定できるわけではないが，第 2 候補，第 3 候補と確認していくと適切な標準形だと考えられる識別子が復元できている場合があった．以上から研究設問 2 に対する答えは「可能」だと結論づける．また，復元できていない場合であっても候補として挙げられる識別子は省略識別子と関係が深い識別子なので，ソースコードを理解するための有効な補助ツールとなり得る．

4.4.3 推定の精度

研究設問 3「ベクトルを用いてどの程度正確に省略識別子を復元可能か.」について述べる．表 4.10 に提案手法の精度を示す．第 1 候補で正しい標準識別子を推定できた割合は Apache ANT で 16.2%，Apache MINA で 13.4%という結果になった．第 2 候補，第 3 候補と確認すると正しい標準識別子が見つかる確率が上がっていき，第 16 候補までに約半数の省略識別子が正しく復元できた．`word2vec` は 2013 年に発明された技術で現在も研究がなされている．`word2vec` の精度の向上と共に本手法の精度も向上する事が期待できる．

4.4.4 パターンマッチによる制限

4.3 節の手順 3 で標準識別子の候補を絞るため，省略識別子を校正する文字がその順番で出現する識別子のみを抽出していが，この処理の妥当性について述べる．識別子の省略には表 3.1 に示す分類が存在するが，4.3 節の手順 3 でこの全ての分類に対して正しい標準識別子を排除してしまう可能性はない．

提案手法の新規性は 4.3 節の手順 2 の識別子の意味の距離を比較するところにあるが，手順 3 の処理だけでも識別子が復元できる可能性がある．表 4.11 は提案手法を用いた EC-CUBE における省略識別子が 1 文字の識別子の復元の例を示している．省略識別子が 1 文字のため，手順 3 の処理だけではそれぞれ省略識別子「d」の復元で 6460 個,「y」の復元で 2081 個のという膨大な数の候補にしか絞る事ができない．しかし，表 4.11 に示すように，手順 2 の処理を適用する事により，省略識別子「d」

表 **4.10**　プロジェクトごとの正しく復元できた省略識別子の割合

プロジェクト	省略識別子の種数	第 1 位	第 2 位	第 3 位	第 16 位
Apache ANT	997	16.2%	23.4%	28.7%	48.0%
Apache MINA	460	13.4%	19.7%	24.1%	41.2%

に関しては「date」「day」など,「y」に関しては「year」など正しいと推測できる識別子を，数千の候補の中から適切に候補を絞っている．加えて，推定識別子の上位の候補のほとんどが日程，時期などに深く関係する識別子であり,「d」「y」の役割を推測するには十分な情報を提示できていると言える．

4.4.5 先行研究との比較

Hill ら [4] の手法では辞書に載っていない単語からランダムに 250 個の識別子を選び，復元した結果，57%の精度で正しく復元できたとしている．一方，本稿の提案手法では 4.3.1 項に記載した方法で評価した結果，Apache ANT の場合で第 1 候補で正しく復元できた割合は 16%，第 16 候補までに正しく復元できた割合は 48%であった．評価方法が違うため単純に比較は難しいものの，本稿の提案手法は Hill らの手法と比べて精度では劣る傾向にある．一方で本稿の提案手法では，以下のような利点がある．

- 意味の近い標準識別子の候補を複数提示できるため，正しい標準識別子がソースコードで使われていなくても省略識別子の意味が類推できる．
- 正しい標準識別子に復元できた場合でも，その標準識別子を見ただけでは意味が分からないとき，意味の近い識別子から省略識別子の意味を類推できる．
- 復元の候補を複数提示するだけでなく，識別子間の意味の距離も同時に提示する事で，標準識別子候補の信頼度が直感的にわかる．

4.4.6 今後の課題

- `word2vec` は動詞の活用などについても学習ができる [3]．本研究にこの考えを適応すると，省略識別子と標準識別子のベクトルの分布には特徴がある可能性がある．具体的には，省略識別子と標準識別子の差分を表すベクトルが存在し，そのベクトルを省略識別子に加減算することでより正確な省略識別子の復元が可能になる可能性がある．
- 本研究では標準識別子がプロジェクト内にあるものと考え，プロジェクトごとにコーパスを作成した．ここで，`word2vec` には 2.2 節に示すように，異なる言語を学習しても同じ意味ベクトルを持つ特徴がある．これは対象をソースコー

表 4.11　1 文字の識別子 i と距離の近い意味ベクトルを持つ識別子 W^i_{long}

i	d		y	
順位	距離	推定標準識別子	距離	推定標準識別子
1	0.544	date	0.550	maxdays
2	0.533	mdbtime	0.532	weekofyear
3	0.517	mdbtoday	0.490	nextdayofweek
4	0.483	adjustdate	0.425	ybefore
5	0.463	unixtoduration	0.394	getweekninyear
6	0.455	durationtounix	0.388	weektypes
7	0.452	maxdays	0.383	getmaxyears
8	0.450	datemdbstamp	0.375	getdaysinweek
9	0.432	mkduration	0.375	getfirstdayinmonth
10	0.426	isduration	0.375	getfirstdayofweek
11	0.415	getduration	0.367	getdayofweek
12	0.410	tmpday	0.364	getweekdays
13	0.401	dayvalueformat	0.364	calendaryear
14	0.400	datecalc	0.359	sjismulticellforfancytab
15	0.397	mdbstampunix	0.359	tmpyear
16	0.396	eregduration	0.356	bakfontstyl
...	...	...	...	...

ドとした場合，違うプログラミング言語間でも同じ意味を持つ識別子は同じベクトルを持つ可能性が高い．そのため，プロジェクトや言語を横断した解析が可能だと考えられる．

- 本研究の標準識別子を探すスコープはプロジェクト全体である．Hill ら [4] の「候補が見つからなければスコープを順に広げていく」という手法は極めて優秀であり，本稿の提案手法と組み合わせて更なる精度の改善が期待できる．

5. 結言

5.1 まとめ

本稿では，ソースコードから機械学習を用いて識別子の意味ベクトルを学習し，ベクトル間の距離を比較する事で省略された識別子を省略元となった識別子に復元する手法を提案した．オープンソースソフトウェアプロジェクトに対して提案手法を適用し，省略された識別子から省略元となった識別子に復元可能なことを示した．

5.2 今後の課題

今後の課題として以下の項目がある．

- 省略された識別子の意味ベクトルの特徴の解析．
- プロジェクトや言語を横断した解析．
- スコープの概念を取り入れた更なる精度の向上．

謝辞

本研究を行うにあたり，研究課題の設定や研究に対する姿勢，本論文の作成に至るまで，全ての面で丁寧なご指導を頂きました，本学情報工学部門水野修准教授に厚く御礼申し上げます．本論文執筆にあたり貴重な助言を多数頂きました，本学研究生胡軼凡君，本学情報工学専攻河端駿也君，山田晃久君，采野友紀也君，本学情報工学課程藤原剛史君，森啓太君，山本滉明君，Zolbayar Naijandav 君，学生生活を通じて筆者の支えとなった家族や友人に深く感謝致します．

参考文献

[1] D. Boswell and T. Foucher, *The art of readable code.* O'Reilly Media, Inc., 2011.

[2] S. Bird, E. Klein and E. Loper, *Natural Language Processing with Python.* O'Reilly Media, Inc., 2009.

[3] 西尾泰和, *word2vec* による自然言語処理. O'Reilly Media, Inc., 2014.

[4] E. Hill, Z. P. Fry, H. Boyd, G. Sridhara, Y. Novikova, L. Pollock, and K. Vijay-Shanker, "Amap: Automatically mining abbreviation expansions in programs to enhance software maintenance tools," in *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, ser. MSR '08. 2008, pp. 79–88.

[5] 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*, ser. SCAM '07.2007, pp. 213–222.

[6] word2vec – tool for computing continuous distributed representations of words. [Online]. Available: https://code.google.com/p/word2vec/

[7] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[8] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems*, 2013, pp. 3111–3119.

[9] T. Mikolov, W.T. Yih, G. Zweig, "Linguistic Regularities in Continuous Space Word Representations," in *HLT-NAACL*, 2013, pp. 746-751.

[10] O. Mizuno, S. Ikami, "Spam filter based approach for finding fault-prone software modules," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, IEEE Computer Society, 2007, pp. 4.

[11] P. Runeson, M. Alexandersson, O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *Software Engineering, 2007. ICSE 2007. 29th International Conference on*, IEEE, 2007, pp. 499-510.

[12] Zellig S. HARRIS, "Distributional structure," *Word*, 1954.

[13] lscp – lightweight source code preprocesser. [Online]. Available: https://github.com/doofuslarge/lscp

[14] Apache git repositories. [Online]. Available: http://git.apache.org/

[15] GitHub. [Online]. Available: http://github.com/EC-CUBE