単語ベクトルを用いた省略識別子の復元手法

岡嶋 秀記
京都工芸繊維大学 大学院工芸科学研究科 情報工学専攻
h-okajima@se.is.kit.ac.jp

河端 駿也
京都工芸繊維大学 大学院工芸科学研究科 情報工学専攻
s-kawabata@se.is.kit.ac.jp

水野 修
京都工芸繊維大学 大学院工芸科学研究科 情報工学部門
o-mizuno@kit.ac.jp

あらまし

ソフトウェアのソースコードの読みやすさはソースコードを保守する上でその重要な要因である．コーディングする際に単語を適切に省略することでコードを簡潔に書ける場合がある．その一方で，略語を多用すると読みにくいコードになってしまう可能性がある．そこで，ソースコードを効率よく理解するために略語を復元する技術が求められる．本稿では，「*word2vec*」というツールを用いて略語を元の単語に復元する手法を提案する．*word2vec* は文章を入力として，単語の意味を表す高次元のベクトルを作成するツールである．単語の意味をベクトルとして扱えるため，単語間の意味の距離や加減の計算が可能になる．我々はソースコードから抽出した識別子の連続を文章と見なし *word2vec* を適用する．得られた識別子のベクトルを用いて，ソースコード中の省略されている識別子と省略される前の識別子の候補との意味の距離を計算する．この識別子間の距離を比較することでより正確な省略される前の識別子の候補を推定する．本手法を用いて省略された識別子から省略される前の識別子を復元できることを示す．

1 はじめに

ソフトウェアを開発する際に開発者が既存のソースコードを読み，処理内容を理解することは必要不可欠である．効率よくソースコードを理解するためには，可読性の高いソースコードを書くことが求められる．しかし，省略された識別子（以下，省略識別子とする）が本来どのような意味だったのか読者が理解できなかった場合は省略識別子がソースコードの可読性を下げる要因となる．よってソースコードを十分に理解するために省略識別子を復元する技術が求められる．

いくつかの省略識別子を復元する研究がなされている [2,3]．本稿では「word2vec」[1] を用いて省略識別子から本来の識別子（以下，標準識別子とする）へ復元する実用的な手法を提案する．word2vec は文書から各単語の意味を表すベクトル（以下，単語ベクトルとする）を作るためのツールである．単語の意味をベクトルとして表現することで，各単語の意味の加算，減算および距離の計算が可能になる．我々は word2vec を用いてソースコードリポジトリ内の識別子をベクトルとして表現し，単語ベクトルの距離を比較することで標準識別子の候補を算出する．

我々は提案手法のケーススタディとして，Apache ANT および Apache MINA の 2 個のオープンソースプロジェクトのソースコードデータを用いた．まず，ソースコードリポジトリから特定のリビジョンのソースコードを取得する．次に，各ソースコードから条件に従い識別子を抽出する．そして，抽出された識別子の連続を文書と見なし，word2vec を適用して識別子のコーパスを作成する．この過程により，ソースコード内の識別子の高次元ベクトルを構築し，単語ベクトルを計算するための準備が整う．そして，このコーパスを利用して省略識別子に距離の近い単語を探索することで，標準識別子の復元候補を列挙する．ケーススタディの結果から，本手法によって多くの省略識別子を復元できることを明らかにした．

その際，識別子の文字数を元にすべての識別子の中から省略識別子の候補を絞った．そして，提案する復元手順を適用することで，省略識別子に対応する標準識別子の候補が得られた．オープンソースソフトウェアを利用したケーススタディの結果，word2vec を用いる提案手法が省略識別子に対応する標準識別子の推定に十分な精度を持っていることを確認した．

2 関連研究

過去にも省略識別子を元の単語に復元する研究 [2, 3] がなされている．Lawrie ら [3] は復元する単語や語句などの候補のリストを作成し，ソースコード内で出現する各省略識別子に 2 段階の拡張を行うことで省略識別子を復元する手法を提案している．

Hill ら [2] はソフトウェアリポジトリマイニング技術を用いて省略識別から標準識別子を自動生成する手法を提案している．Hill らの手法は Lawrie らの手法と比較して 57%向上した．彼らの手法はソースコードだけでなくソフトウェアの Java Documentation Comments を使用し，省略識別子の正確な推定を達成した．彼らの手法はソフトウェアに関わる様々な種類の資料を必要とすが，我々はソフトウェアのソースコードのみから省略識別子の候補を推定することを目指す．

3 省略識別子の復元手法

3.1 研究設問

本研究の基本となるアイデアは単語のベクトル表現を用いて短い識別子（省略識別子）と長い識別子（標準識別子）の間の距離を計算することである．我々は省略識別子とその元となった標準識別子の間の意味の距離は近いと考えている．我々の仮説が正しい場合，省略識別子の復元は意味の近い標準識別子を求めることで達成できる．そこで本研究では以下の研究設問を設定する．

RQ1: word2vec を用いて省略識別子の元となった標準識別子を復元することは可能か．

RQ2: word2vec を用いてどの程度正確に復元が可能か．

RQ1 では，word2vec で省略識別子を標準識別子に拡張できるかを調査する．RQ2 では，オープンソースプロジェクトを用いて提案手法の精度を評価する．

3.2 ソースコード内の省略識別子

図 1 は省略識別子の分類を示している [2]．省略識別子は単一単語および複合単語の 2 種類のカテゴリに分類される．我々の提案手法は単一単語および複合単語の両方の形式の省略識別子の復元を目指す．

省略識別子を復元するためには正解となる標準識別子が必要となる．本研究では，我々の研究室の十分なソフトウェア開発経験のある大学院生が省略識別子が正確に標準識別子に復元できたかを手動で確認した．

3.3 単語ベクトル

本研究では word2vec を用いてソースコード中から 2 個の識別子を選び，意味間の距離を計算する．word2vec は入力としてテキストコーパスを受け取り，出力として単語ベクトルを生成する [1]．word2vec のアルゴリズムは参考文献 [4, 5] に基づいている．word2vec はまず学習用テキストデータから語彙集を構築し，その単語のベクトル表現を学習する．得られた単語ベクトルファイルは学習されたテキストの特徴を表している．word2vec において，2 個のベクトル間の距離は 0 から 1 で表現され，0 に近いほど意味の関係が薄く，1 に近いほど意味の関係が深いことを示す．

3.4 識別子の単語ベクトルコーパスの作成

我々は「lscp」[6] というツールを用いてソースコードから識別子を抽出した．lscp は軽量のソースコードプリプロセッサである．lscp はソースコードファイルから識別子の名前，コメント，文字列リテラルなどを分離し，文字列を操作するために使用できる [6]．我々は lscp を用いて識別子を抽出し，word2vec を用いて単語ベクトルコーパスを作成した．手順を図示したものを図 2 に示す．

本研究では，まず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した．次に取得したソースコードに対して lscp を適用して識別子を抽出した．lscp には「camelCase」「snake_case」「dot.notation」などの単語を分割する方式と分割しない方式がある．我々はソースファイルに両方の方式を適用し，分割された識別子ファイルと分割されていない識別子ファイルの両方を用意した．そして学習するため抽出された識別子に word2vec を適用し，識別子のベクトルコーパスを作成した．さら

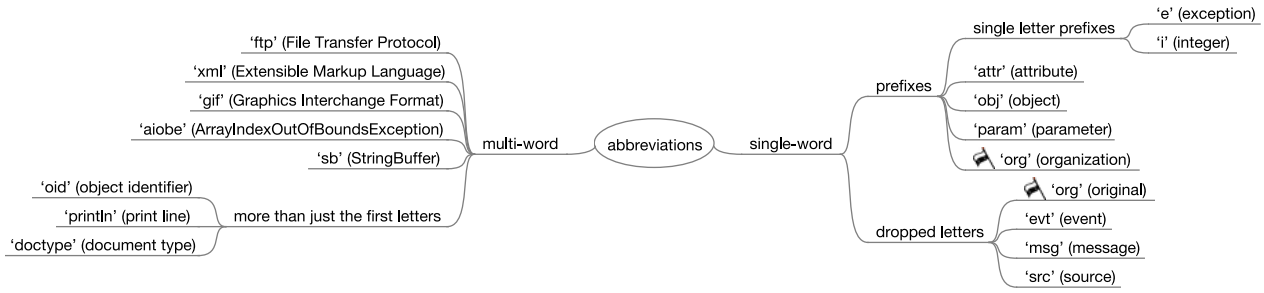

図 1: 省略識別子の分類

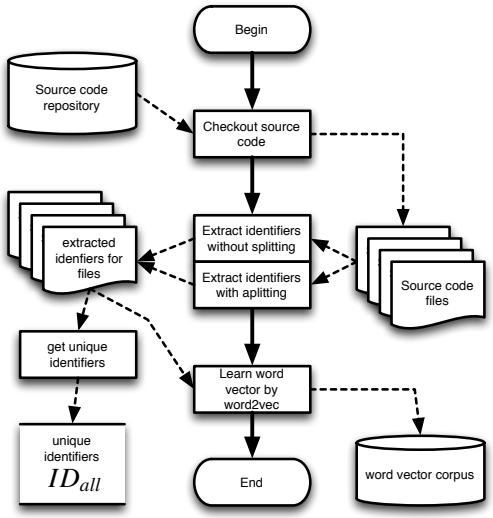

図 2: 識別子の単語ベクトルの作成手順

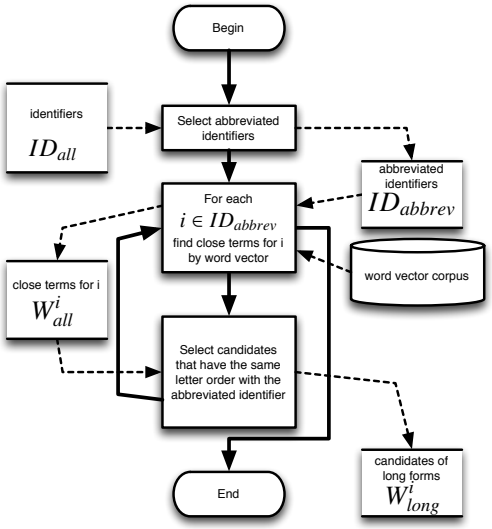

図 3: word2vec による省略識別子の復元手法

に，これらの識別子ファイル内の一意の識別子のリストを作成した．この識別子のセットを ID_{all} として示す．

3.5 復元手順

次の手順で省略拡張子を復元する．(図 3 参照)

1. 4 文字未満の識別子を省略識別子の可能性がある識別子とする．ID_{all} から 4 文字未満の識別子を抽出し，ID_{abbrev} のセットを作成する．
2. 各省略識別子 $i \in ID_{abbrev}$ に対し，ベクトルの距離を計算し，距離が近い識別子を標準識別子の候補 W^i_{all} とする．
3. その際，W^i_{all} から i の文字列の順番で出現する識別子 W^i_{long} のみを抽出する．

復元手順について実例を用いて説明する．表 1，3，5 および 7 は手順 2 の結果である．これらの表は 4 章でケーススタディとして用いる識別子「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単語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 1 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 5 の場合，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 より，省略識別子「cce」と最も距離の近い識別子は「cast」であると分かる．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 では第 2 位である．表 7 の識別子「buf」の場合で

表 1: 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 2: 「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: 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 4: 「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
...	...	...

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 4 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 7 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 8 に示すように第 5 位となった．

表 2 および表 6 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．

4 ケーススタディ

4.1 対象プロジェクト

このケーススタディでは，Apache ANT および Apache MINA の 2 個のオープンソースプロジェクトを利用する．これらのプロジェクトのソースコードリポジトリは Apache git repositories [7] から入手する．それぞれのプロジェクトについて，リポジトリからいくつかのリビジョンを取り出す．ANT では 25 のリビジョンを，MINA では 33 のリビジョンをそれぞれ利用した．

4.2 単語ベクトルの学習

3.4 節の手順に従い，それぞれのプロジェクトのリビジョンのソースコードを `word2vec` で学習し単語ベクトルのコーパスを生成する．この手順で，我々はソースコードの情報しか使用していない．よって特別な辞書やストップワードリストを用意する必要は無い．

識別子を抽出した結果として，ANT および MINA プロジェクトの全識別子 ID_{all} を得る．ANT プロジェクトで 22543 種類，MINA プロジェクトで 7242 種類の識別子が存在した．

4.3 復元

省略識別子の復元にあたり，すべての識別子から省略された可能性のある識別子を選択する．本ケーススタディでは，リポジトリから取得したすべての識別子 ID_{all}

表 5: 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 6: 「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 7: 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 8: 「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
...	...	...

から長さが 4 文字未満の単語をリストアップし，ID_{abbrev} とした．ANT および MINA プロジェクトにおける省略識別子の可能性のある識別子はそれぞれ 971 種類および 456 種類であった．

図 3 の手順に従い，ID_{abbrev} のすべての識別子を復元し，調査した．復元結果の例を表 1 から表 8 に示す．

また，復元結果の一部を表 9 および表 10 に示す．これらの表は各プロジェクトのソースコード中に頻繁に出現する省略識別子の一部 [2] と，我々が発見した興味深い復元結果を示す．これらの表において，列「分類」は図 1 で示した省略識別子の分類と対応している．列「省略形」は省略識別子を,「推定標準形」「距離」「順位」はそれぞれ標準識別子の候補，それと対応する省略識別子の単語ベクトル間の距離および距離を元に導きだした順位を示す．

5 考察

5.1 word2vec を用いた省略識別子の復元

ここでは，RQ1「word2vec を用いて省略識別子の元となった標準識別子を復元することは可能か.」の答えについて述べる．

ケーススタディで示したように，省略識別子を長い形式に復元することは可能である．標準識別子が一意に特定できない場合でも，適切な標準識別子の候補をいくつか提示することが可能である．よって我々は RQ1 の答えは「可能」と結論づける．

5.2 復元の精度

次に，RQ2「word2vec を用いてどの程度正確に復元が可能か.」の答えについて述べる．

表 11 に提案手法の精度と，本研究で用いた省略識別子の数を示す．

第 1 位に順位付けられた候補が正しい標準識別子だった割合はそれほど高くないことが分かった．その一方で，

表 11: 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 位の正解率	第 2 位までの正解率	第 3 位までの正解率	第 16 位までの正解率
Apache ANT	997	16.2%	23.4%	28.7%	48.0%
Apache MINA	460	13.4%	19.7%	24.1%	41.2%

表 9: 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 10: 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

意味の距離が近い順にいくつか候補を挙げていくことでその割合は上がり，第 16 位までで約 50%もの標準識別子を推定できた．

5.3 先行研究との比較

表 12 は Hill ら [2] による Java 5 のソースコード中で頻繁に出現する省略識別子を示している．彼女らは 20 個の省略識別子を示しているが，表 12 では 3 文字の省略識別子のみ示している．表 12 の各列は 3 文字の省略識別子，対応する正しい標準識別子，我々の提案手法で算出した標準識別子の順位，省略識別子と標準識別子の距離およびその結果が得られたプロジェクト名をそれぞれ示す．省略識別子「int」「val」および「str」は我々の提案手法では復元できなかった．我々の提案手法ではソースコードのみを利用しているため，ソースコード内に出現しない単語の復元は難しい．

表 12 より，省略識別子「obj」「pos」「var」「env」および「ctx」の復元は距離の順位付けを考慮すると精度が高かったが，その他の省略識別子については顕著には現れなかった．すべての結果の調査として,「add」「get」「is」および「set」の様な一般的によく使われる単語の識別子は復元できなかったことが分かった．以上の理由で，単一識別子の復元は複合識別子の復元に比べて精度が落ちる傾向にあると推察できる．

5.4 復元の特徴

表 9 および表 10 は両方のプロジェクトで得られた特徴的な復元を示している．表 9 および表 10 において，我々は「exception」に関係のある省略識別子をいくつか

表 12: 本手法と AMAP [2] との比較

省略形 [2]	標準形 [2]	順位	距離	プロジェクト
int	integer	–	–	–
obj	object	1	0.473	Apache ANT
pos	position	1	0.422	Apache MINA
len	length	10	0.362	Apache ANT
num	number	12	0.297	Apache ANT
env	environment	2	0.662	Apache ANT
val	value	–	–	–
str	string	–	–	–
buf	buffer	5	0.333	Apache ANT
ctx	context	2	0.483	Apache MINA
msg	message	4	0.395	Apache ANT
var	variable	1	0.771	Apache ANT
arg	argument	16	0.432	Apache ANT

復元できていることが確認できた．exception に関係のある識別子は「npe」や「nfe」などのようにたいていの場合 3 文字の形式で発見された．我々の結果では、そのような省略識別子を復元する精度は非常に高かった．3 文字の単語が複合されてできたキャメルケースの復元の精度は他の形式の復元と比べて高いことが分かった．この特徴は両方のプロジェクトで共通して確認できた．

5.5 制限

これらの観点から，我々の提案手法は節 3.5 において復元の手順 3 で候補となる識別子の形式を制限した．この制限は復元の精度のためであるが，標準識別子の可能性のある識別子を逃す可能性もある．

6 結論

本稿では，`word2vec` を用いて省略された識別子を標準の状態の識別子へ復元する方法を提案した．`word2vec` を用いることによって，ソースコードリポジトリ内の識別子を構成する単語のベクトル表現を作成した．そしてベクトル表現を用いて省略識別子の標準識別子の候補を求めた．我々の提案手法は省略識別子を長い形式に復元できるという結果となった．

今後の課題として，先行研究 [2, 3] との詳細な比較がある．そのためには，我々の提案手法をさらに多くのプロジェクトで適用する必要がある．なお，本稿では Java 言語にのみ適用したが，提案手法はプログラミング言語には依存しない．よって，違う言語に提案手法を適用したり，プロジェクト間を横断して検証することは，興味深い今後の課題である．

謝辞

この研究は日本学術振興会科学研究費補助金基盤研究 (C)24500038 の助成を受けて実施された．

参考文献

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

[2] 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. New York, NY, USA: ACM, 2008, pp. 79–88. [Online]. Available: http://doi.acm.org/10.1145/1370750.1370771

[3] 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. Washington, DC, USA: IEEE Computer Society, 2007, pp. 213–222. [Online]. Available: http://dx.doi.org/10.1109/SCAM.2007.17

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

[5] 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.

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

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