Vol.2014-SE-183 No.15
2014/3/20

キーフレーズ抽出によるコミットと不具合報告チケットの自動リンク

河居 寛樹[1,a)]　上野 秀剛[1,b)]

概要：本研究の目的は不具合報告チケットに対応したコミットを特定し，開発者に提示する手法の提案である．提案手法は形態素解析と N-gram を組み合わせ，連続した形態素をフレーズとして生成する．フレーズを元に，不具合報告チケットの文章に類似したソースコードのコミットコメントを探し，開発者に提示する．提案手法をオープンソースソフトウェアプロジェクトのリポジトリに適用し，評価を行う．実験の結果，提案手法は 75.9%のバグ報告に対して正しいソースコードを推薦できた．

1. はじめに

Firefox や Linux に代表されるオープンソースソフトウェア (Open Source Software: OSS) の発展や，外部組織にソフトウェア開発作業の一部を委託する外注の増加に伴い，複数の国や地域に点在した開発者による分散開発が増えている．複数人によるソフトウェア開発では，開発者間のコミュニケーションが重要とされており，開発者が直接顔を合わせて行うオフラインミーティングや電話会議が良く用いられる．しかし，分散開発においてはこれらの手段は多大な移動時間・費用が必要なことに加え，会議参加者全員のスケジュールを調整し，同じ時間を共有する必要がある．そのため，開発者間でのコミュニケーションの手段としてオンラインによる非同期な情報交換がよく用いられている．ソフトウェア開発における代表的なオンライン非同期コミュニケーションツールとして，メーリングリスト (Mailing List: ML) やバグ管理システム (Bug Tracking System: BTS)，ソースコードのバージョン管理システム (Version Control System: VCS) が用いられている [1][2]．本論文では以降，これらのシステムを開発支援システムと呼ぶ．

これらの支援システムは検出したバグの症状や再現手順，修正担当者の割り当て，変更後のソースコードを記録する．開発者は不具合修正や機能拡張を行う際に，それまでにどのような変更が，なぜ，誰によって，いつ行われたのかを理解するために複数の開発支援システムを同時に参照し，開発のコンテキストを理解する．

大規模な OSS プロジェクトでは，一日に 300 件ものバグが報告されることもあり [3]，これらのバグを修正したソースコードの変更履歴は膨大な数となる．また，BTS のバグ報告と VCS のソースコードはリンク付けされない場合が多く存在する [4]．そのため，複数の開発支援システムに分散した 1 つのコンテキストに属する情報を参照するのは容易ではなくなる．たとえば，BTS に記録されたある 1 つの不具合について，ML 上で議論されたメールを探すためには，検索に用いる単語や議論された時期，関係者などそのコンテキストの特徴を表す情報を理解している必要がある．このような情報は一般に開発支援システムには保存されないため，コンテキストを理解していない開発者や過去のコンテキストの参照が必要な開発者にとっては検索が困難である．

本研究では，閲覧中の情報と同じコンテキストに属する情報を検索する労力を削減するために，バグ報告に関係するソースコードを推薦する手法を提案する．提案手法は，バグ報告コメントやコミットコメントには一連のコンテキストを表すフレーズが含まれていると仮定し，形態素解析と N-gram を用いて，バグ報告と同じフレーズの含まれるコミットコメントがついたソースコードを推薦する．

2. 開発支援システム

開発支援システムとは，開発に関する履歴を記録するシステムのことである．多くの開発プロジェクトでは複数の開発支援システムを同時に利用する．例えば，あるバグが発見され，除去されるまでには，1)BTS に発見されたバグが報告され，2)ML で原因箇所の特定と修正方法につい

[1] 奈良工業高等専門学校 NNCT, 22 Yatatyou,Yamatokoriyamashi, Nara 639–1080, Japan
[a)] h-kawai@info.nara-k.ac.jp
[b)] uwano@info.nara-k.ac.jp

ての議論が行われ，3) 更新されたソースコードが VCS に記録される．個々の開発プロジェクトは彼らの開発形態に適した支援システムを開発・選択し，運用する．一方で，OSS プロジェクトなど比較的規模の小さなプロジェクトでは，管理費用が掛からず，管理者の確保が不要な Google code や Source Forge といったレンタルサービスが多く利用される．

また，プロジェクトの開発者は複数の開発支援システムに保存された情報を同時に参照する．たとえば，バグ修正を割り当てられた開発者は BTS に記録された症状の説明や，ML で行われた議論を参考にバグの症状を理解し，VCS に記録されたソースコードの変更履歴を元にバグの発生箇所を特定する．また，不具合修正やテスト設計の参考にするために，過去の類似した不具合を BTS や ML から調査する．

このとき，調査している事柄を示す開発コンテキストのつながりを理解していなければならない．図 1 に複数の開発支援システムを用いた開発におけるコンテキストの様子を示す．3 つの長方形は VCS・BTS・ML の各開発支援システム，楕円はそれぞれのシステムに保存された情報，楕円を繋ぐ直線は同一コンテキストに属する情報のつながりを表している．それぞれのコンテキストに含まれる情報は，その種類ごとに異なるシステムに分散して保存されている．ある情報に関係する情報を探したいとき，コンテキストのつながりを元に調査を行う．

このとき，開発時のコンテキストを利用者が理解していなければ検索が困難になる．例えばある不具合について BTS には検出されたバグの症状のみが記録され，ML 上で不具合原因や，修正方法が議論され，VCS に修正履歴が残されたとする．このとき，BTS の閲覧者がこのときの様子を知らず，BTS に ML や VCS へのリンクが記録されていなければ，BTS に記録されたわずかな説明文からキーワードを推測し，ML や VCS 全体を検索する必要がある．これは，時間が掛かり，見落としを引き起こす作業である．一般に，ある話題についてどのような議論がどの開発支援システムを使って行われたのかという，開発のコンテキストは開発支援システムには記録されず，情報同士のリンクも手動に頼っていることが多い．そのため，新規にプロジェクトに参加した開発者はコンテキストの把握ができない．また，以前から参加している開発者であっても，過去のコンテキストについては思い出すのが困難である．

複数システムの併用によるこれらの問題を解消するために，複数のシステムを統合したシステムも存在する [5]．これらの統合システムでは BTS や VCS，ML の他に Wiki やテスト管理システムなどを統合することで，プロジェクトの管理に必要な情報を 1 つのシステム内で管理する．しかし，これらの統合システムには以下の問題がある．

(1) 利用中の開発支援システムに蓄積された開発履歴や，リンクをインポートできない．
(2) レンタルサービスのような外部で提供された開発支援システムの場合，システムの変更ができず，蓄積された情報をそのまま利用できない．
(3) プロジェクトで必要な機能を自由に選択できず，統合システムが提供する機能しか利用できない．

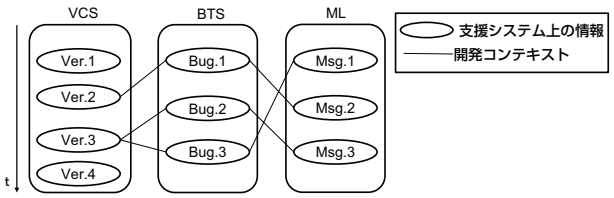

図 1 開発支援システム間のコンテキスト

3. 提案手法

提案手法は，形態素解析を用いてバグ報告文章を形態素に分割し，N-gram を用いて連続した形態素をフレーズとして生成する．同じコンテキストに属してる情報には同じフレーズが含まれる可能性が高いという仮説に基づいて，フレーズがソースコードのコミットコメントに含まれている場合に，そのソースコードがバグ報告と関連があるものとして推薦する．

3.1 形態素解析

形態素解析は，自然言語で書かれた文を言語の中で意味のある最小単位（形態素）に分割し，品詞を特定する手法である．提案手法では，バグ報告コメントを形態素に分割し，次節で説明する N-gram を形態素単位で行うための前処理に用いる．形態素解析システムには，オープンソース形態素解析エンジンの MeCab[*1]を用いる．

3.2 N-gram

N-gram は，文章から N 文字の連続した文字列を切り出す手法である．文章から文字列を切り出す箇所をずらしながら，他の文章から切り出した文字列と比較することで，同じ文字列を含む文章を検出できる．たとえば，“フラグを更新する”という文章に対して N が 3 の N-gram を適用すると“フラグ”，“ラグを”，“グを更”，“を更新”，“更新す”，“新する”の計 6 個の 3 文字の文字列が抽出される．他の文書からも同様に文字列を抽出し，同じ文字列（たとえば“フラグ”）が現れた場合，その文章同士は類似した意味を持つ可能性がある．提案手法では，形態素解析で分割した形態素を N-gram における最小単位とし，連続した N 個の形態素（フレーズ）を抽出するために用いる．

3.3 形態素 N-gram

形態素 N-gram は形態素解析と N-gram を組み合わせた

[*1] http://code.google.com/p/mecab/

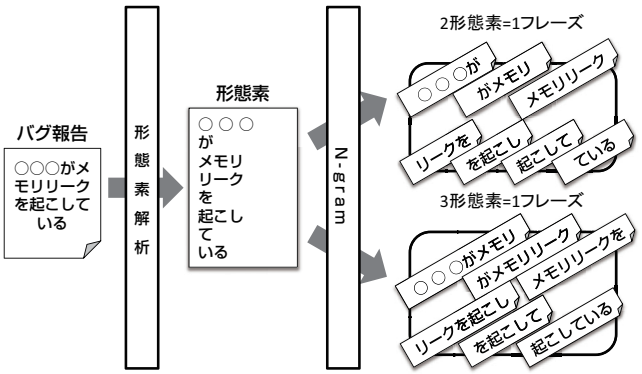

図 2 形態素 N-gram の処理

手法である．形態素解析は文章から形態素を切り出すため，意味を得ることができるが，形態素同士の前後関係を得ることはできない．また，N-gram は文章から連続した文字列を切り出すため，文字列の前後関係は得られるが，文字列の意味を得ることはできない．そこで，この 2 つを組み合わせることにより，文字単位ではなく形態素を単位として，文章に N-gram を適用することで複数単語からなるフレーズや文を抽出できる．バグ報告に対して形態素 N-gram を行う例を図 2 に示す．

図中の“○○のフラグを更新する”という文章の場合，形態素解析を行うことにより，“○○”，“の”，“フラグ”，“を”，“更新する”という 5 つの形態素に分割され，N-gram により 2 形態素を 1 フレーズとして，4 つのフレーズを抽出している．N-gram のみの場合，N が 3 の N-gram によって“フラグ”を抽出できるが“ラグを”といった，元の文章とは意味の異なる文字列が取り出され，異なる文章を推薦してしまう可能性がある．形態素 N-gram では，最小単位を形態素として N-gram でフレーズを求めることで，元の文章と異なる意味を持つ文字列や単語の抽出を抑制できるため，推薦精度が高くなると考えられる．

提案手法は形態素 N-gram を用いることでバグ報告コメントからフレーズを切り出し，コミットコメントに含まれるフレーズと比較することで，バグ報告とソースコードをリンク付ける．

3.4 手順

提案手法の処理手順を図 3 に示す．手順 1 では実際に開発者がバグ報告を読み，ソースコードを探すことを想定し，開発者が閲覧しているバグ報告としてバグ報告を 1 件取り出している．

(1) バグ管理システムからバグ報告を 1 件取り出す
(2) 取り出したバグ報告の文章を形態素 N-gram を適用し，フレーズを抽出する
このとき，形態素 N-gram は文章を読点や’？’，’…’，’・’ など文の終わりや文の先頭を示す記号で区切り，文をまたいだフレーズの抽出は行わない
(3) ソースコードのバージョン管理システムから，すべてのソースコードのコミットコメントに対して，フレーズが含まれているか検索する
(4) コミットコメントがフレーズを含んでいた場合，含まれていた回数をカウントする
(5) より多くのフレーズを含んでいたコミットコメントが上位になるように順位を付ける
(6) 上位数件のコミットコメントに対応したソースコードを，修正候補として開発者に提示する

4. 予備実験

実験を行う前に，バグ報告文に存在するフレーズがソースコードのコミットコメントにどの程度存在するのか予備実験で調査する．予備実験では，すでに修正が完了したバグ報告のコメントに形態素 N-gram を適用し，抽出したフレーズが修正されたソースコードのコミットコメントに存在するか調査する．提案手法の推薦精度はフレーズを抽出するときの形態素 N-gram の N の値によって異なると考えられるため，形態素 N-gram の N の値を 1 から 10 まで変化させて実験を行う．

予備実験の結果を図 4 に示す．図より，形態素 N-gram の N の値を大きくするごとに抽出したフレーズを含むコミットコメントは少なくなっている．そのため，N を 10 以上にして実験を行うメリットは無いと考えられる．

N の値が 5 以下のときに，抽出したフレーズを含むコミットコメントは，修正されたソースコードのコミットコメントの総数の半分以上存在している．これより，修正が完了したバグ報告文から抽出したフレーズをもとに，実際に修正されたソースコードを探しだすことができると考えられる．また，N の値が小さいほど多くのコミットコメントに抽出したフレーズが存在している．これは，N の値が小さいということはフレーズが短いということでもあり，フレーズをもとにソースコードを探しだす際に検索のノイズとなる可能性がある．本実験では，ノイズとなる情報を除去しつつ，高い精度でフレーズを元にソースコードを探し出せる形態素 N-gram の N の値を求める実験を行う．

5. 実験

提案手法を用いた推薦の精度を評価するために実験を行う．実験ではオープンソースソフトウェアの開発プロジェクトに報告されたバグ報告と，ソースコードのコミットコメントに対して提案手法を適用し，精度を求める．予備実験と同様に N を 1 から 10 まで 1 ずつ増やし，それぞれの場合の推薦精度を評価する．また，提案手法の有用性を確認するために TF-IDF を用いた推薦手法と比較する．

5.1 推薦対象

実験に用いるデータは，日本語で記述可能なプログラミ

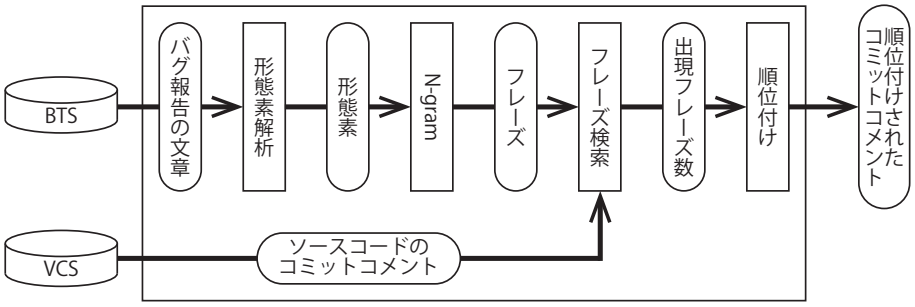

図 3 提案手法の処理手順

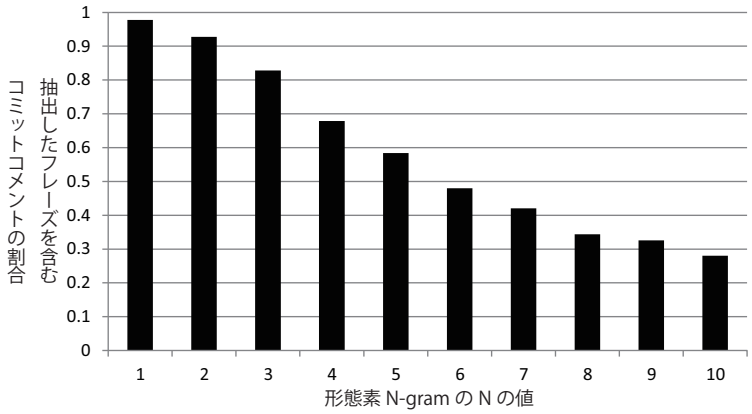

図 4 修正が完了したバグ報告文から抽出したフレーズを含むコミットコメントの割合

ング言語「なでしこ」の開発プロジェクトで 2008 年 10 月から 2010 年 9 月に記録されたバグ報告の文章データと，ソースコードのコミットコメントの文章データである．開発者がバグを修正した際に BTS から VCS へリンク付けしたものを正解集合として，推薦精度を評価する．バグ報告は BTS から VCS へリンクがあるものを用い，ソースコードのコミットコメントは期間中に記録されたすべてを用いる．上記の条件を満たしているデータ数は，バグ報告が 153 件，ソースコードのコミットコメントが 1842 件，開発者によるバグ報告とソースコードのコミットコメントのリンクが存在する組（正解集合）が 158 組である．

5.2 推薦精度の評価

それぞれのバグ報告に対して，提案手法によってランキングされたコミットコメントの上位 1 件，5 件，10 件以内に答えが存在するか確認する．上位 1 件，5 件，10 件以内のそれぞれに答えが 1 つでもあった場合，提案手法により正しくバグ報告とソースコードをリンク付けできたとみなす．これをすべてのバグ報告に対して行い，式 (1) で精度を求めて評価する．精度は 0 から 1 の範囲をとり，値が大きいほど正確にバグ報告と修正されたソースコードをリンク付けしていることを表す．

$$\text{精度} = \frac{\text{上位 n 件内に答えがあるバグ報告数}}{\text{バグ報告の総数}} \quad (1)$$

5.3 提案手法の評価

形態素 N-gram を適用する際の N の値を 1 から 10 まで変化させ，それぞれの精度を求める．N-gram は，N の値が大きいほど長いフレーズを抽出することができるため，文章のコンテキストと関係のない一般的なフレーズの抽出を抑えることができると考えられる．そのため，N の値を大きくするほど同一のコンテキストに属する文章を適切に推薦でき，精度が高くなると考えられる．

5.4 TF-IDF を用いた手法の評価

TF-IDF 法は，ある文章の集合（文章セット）の中に含まれる一つの文章に注目したとき，その文章が文章セットの中でどういった単語 (term) で特徴づけられるか調べる手法である [6]．TF-IDF 値は式 (2) から (4) で計算される．

$$tf_{c,n} = \frac{d_{c,n}}{\sum_{i=1}^{C} d_{i,n}} \quad (2)$$

$$idf_c = log_e \frac{|D|}{|\{d : t_c \in d\}|} \quad (3)$$

$$TF-IDF_{c,n} = tf_{c,n} \cdot idf_c \quad (4)$$

$tf_{c,n}$ は単語 c が文章 n に出現した回数 $d_{c,n}$ を，n の総単語数で割ったもので，値が大きいほど文章 n に単語 c が多く出現していることを表す．idf_c は c が文章セット中のいくつの文章に含まれているかを表す DF(Document Frequency) の逆数の対数である．ここで，$|D|$ は全文章数，

$|\{d : t_c \in d\}|$ は単語 c を含むドキュメント数を表す．

idf の値が大きいほど単語 c が少数の文章にしか出現しないことを表す．各文章における単語の特徴度は式 (4) より求めることができる．TF-IDF 値は文章におけるその単語の特徴度の高さを表している．

TF-IDF は重複したバグを見つける研究 [7] や電子メールとソースコードをリンク付ける研究 [8] で利用されているオーソドックスな方法である．そのため，提案手法の有用性を確認する比較対象として TF-IDF を用いる．TF-IDF を用いた手法の手順を図 5 に示す．

(1) バグ報告の文章に形態素解析を適用する
(2) 形態素解析の結果から名詞のみを抽出する
(3) 形態素（名詞）の TF-IDF 値を計算する
(4) バグ報告の文章から抽出した名詞が，すべてのソースコードのコミットコメントに含まれるか検索する
(5) コミットコメントに含まれる名詞の TL-IDF 値を合計し，コミットコメントの TF-IDF 値とする
(6) TF-IDF 値をもとにバグ報告ごとにコミットコメントを順位付けする
(7) 上位のコミットコメントを修正候補として開発者に提示する

5.5 実験手順

(1) なでしこの開発プロジェクトのバグ報告 153 件に対して提案手法を適用する
(2) バグ報告 153 件それぞれにコミットコメントが推薦された精度を求める
(3) N-gram の値を変更し，1. および 2. の処理を行う
(4) 手法を TF-IDF に変更し，1. および 2. の処理を行う
(5) 3. および 4. で求めた精度を比較する

6. 実験結果と考察

6.1 提案手法と TF-IDF の比較

提案手法と TF-IDF を用いた比較手法の結果を表 1 に示す．

表 1 提案手法と TF-IDF の精度の比較

	N	推薦する件数		
		1 件	5 件	10 件
提案手法	1	0.310	0.367	0.392
	2	0.411	0.506	0.557
	3	0.759	0.911	0.981
	4	0.703	0.842	0.873
	5	0.709	0.835	0.867
	6	0.589	0.646	0.646
	7	0.538	0.576	0.576
	8	0.443	0.475	0.475
	9	0.411	0.443	0.443
	10	0.329	0.354	0.354
TF-IDF	−	0.634	0.876	0.915

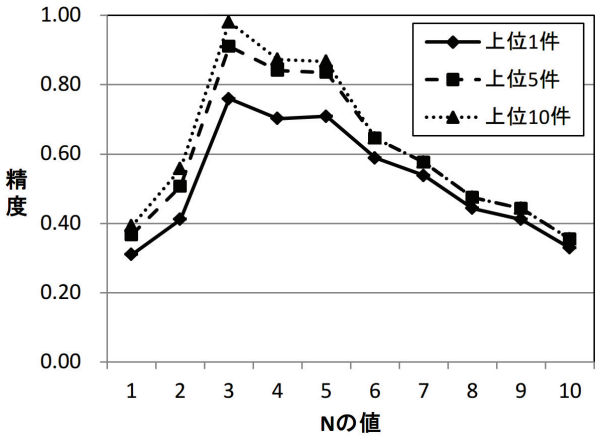

図 6 提案手法の推薦精度

表 1 より，形態素 N-gram の N を 1 から 10 まで変えた結果，N が 3 の場合において，提案手法は TF-IDF を用いた手法よりも高い精度が得られた．上位 1 件のみに注目した場合，N が 3，4，5 の場合において TF-IDF より精度が高かった．

提案手法と TF-IDF を用いた手法は，いずれも 2 つの文書に現れる単語から類似性を推測する点で，似た手法といえる．しかし，TF-IDF を用いた手法は複数の単語が文書中に現れる順序を考慮できない一方で，提案手法はフレーズを抽出することで単語の順序も考慮して文書間の類似性を見ることができる．このため，提案手法においてより高い推薦精度が得られたと考えられる．

3 形態素で構成されるフレーズに注目すると，バグ報告とすべてのコミットコメント両方に存在するフレーズを全部で 984 種類，合計 5552 個抽出していた．出現回数に注目すると“命令を追加”，“問題を修正”，“不具合を修正”のような，どんなバグ報告やコミットコメントにでも出現しそうなものが多く，特にこの 3 種類のフレーズだけで全体のフレーズの 36.9%を占めていた．

3 形態素で構成されるフレーズのうち，正解集合から抽出されたフレーズは 893 種類，合計 1250 個であった．出現回数に注目すると“命令を追加”，“問題を修正”，“不具合を修正”のフレーズが多く出現していたが，正解集合のフレーズに占める割合は 5.44%と低く，1 回しか出現しないユニークなフレーズが 58.2%を占めていた．これより，特定のバグ報告にしか出現しないユニークなフレーズが存在し，提案手法ではユニークなフレーズを検出することで，対応するコミットコメントを特定できた可能性がある．

3 形態素で構成されるフレーズの出現回数を図 7 に示す．3 色に色分けした棒グラフのうち，一番下の色の濃い部分が“命令を追加”，“問題を修正”，“不具合を修正”の 3 つのフレーズの出現回数を表している．一番上の色の薄い部分が 1 回しか出現しないユニークなフレーズの出現回数を表しており，中央の部分はこれ以外のフレーズの出現を表し

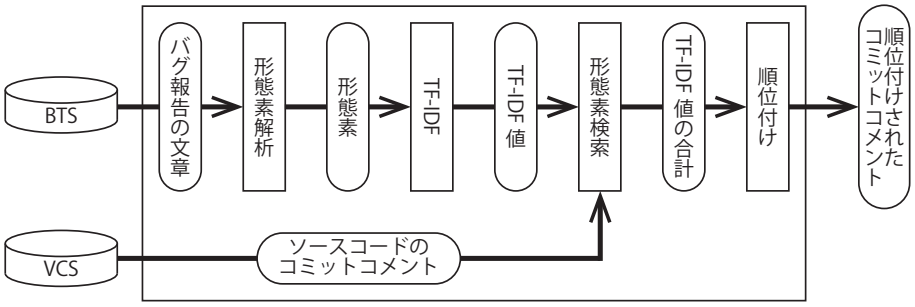

図 5 比較実験の処理手順

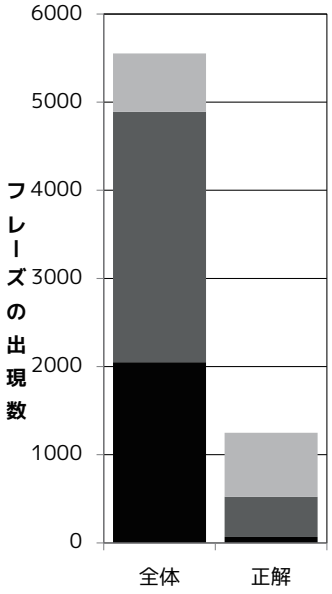

図 7 3 形態素で構成されるフレーズの出現回数

ている．また，全体という項目は合計 5552 回出現したバグ報告とすべてのコミットコメント両方に存在するフレーズであり，正解という項目は合計 1250 回出現した正解集合から抽出されたフレーズを表している．

正解集合に出現した回数と全体で出現した回数が同じフレーズは特定のバグに関係するコンテキストにしか存在しないフレーズ（キーフレーズ）であり，ソースコードを特定する手がかりとなる．キーフレーズは，正解集合の 57.9%を占め，そのうち 84.3%が 1 回しか出現しないフレーズであった．キーフレーズとして，“２ついっぺんに”，“からプリプロセス宣言”，“階層タグ切り出し”，“命令でアンチエイリアス”などのフレーズが出現していた．これらのフレーズは，特定のバグ報告にしか出現しないユニークなフレーズであり，これらのフレーズをキーとして対応するコミットコメントを特定できていると考えられる．

また，“命令に影響”，“問題とお手本”，“処理の不具合”などのフレーズもキーフレーズとなっていた．これらのフレーズに含まれる“命令”，“問題”，“不具合”といった形態素は，バグ報告やコミットコメントに多数存在するため TF-IDF では一般語として扱われ特徴語にはならない．一方で，提案手法ではフレーズを生成することで“命令に影響”，“問題とお手本”，“処理の不具合”のような一般語として扱われる形態素を含むフレーズをユニークなキーフレーズとして抽出できている．このため，提案手法は TF-IDF を用いた手法に比べて高い推薦精度が得られたと考えられる．

6.2 フレーズを構成する形態素数の変更

形態素 N-gram の N の値を 1 から 10 まで変化させたときの，上位 1 件，5 件，10 件まで推薦したときの精度を図 6 に示す．いずれの順位においても N の値が大きくなるにつれて精度が向上し，N が 3 の時に最も良い精度（上位 1 件：0.759，上位 5 件：0.911，上位 10 件：0.981）だった．それ以降は N が大きくなるにつれ精度が低下した．

N の値を大きくするほど精度が下がる原因として，コミットコメントの 1 文に含まれる形態素の数が少ないことが考えられる．実験で用いたバグ報告の 1 文の形態素数を表したヒストグラムを図 8 に示す．図の横軸は 1 つの文に含まれる形態素の数を，縦軸は頻度を表している．図より，全体の 86.6%のデータが 27 形態素までの範囲に存在しており，形態素数が 10 未満の文は全体の 34.8%であった．提案手法は，バグ報告とコミットコメントから長さ N の同じフレーズを抽出できたときのみ推薦するため，いずれかの形態素数が N より小さいと，適切に推薦できない．すなわち，N=10 の時，携帯素数が 10 未満である 34.8%の文は推薦できず，精度の低下につながっていると考えられる．今後，正解集合の 153 組について，それぞれの取り得る N の最大値を調べ，それを超える N による推薦を分析から除外する事でより適切に評価が行えると考えられる．

6.3 推薦件数の変更

コミットコメントの順位に注目すると，より低い順位まで推薦に含めたときに精度が高い．しかし，上位 10 件まで推薦した場合と上位 5 件まで推薦した場合について，N が 5 以上の時に差は見られなかった．

推薦する件数を上位 1 件から，上位 5 件まで，上位 10 件までに増やすほど精度が向上した．これは，形態素 N-gram の一致数による順位付けにおいて，正解であるコミットコ

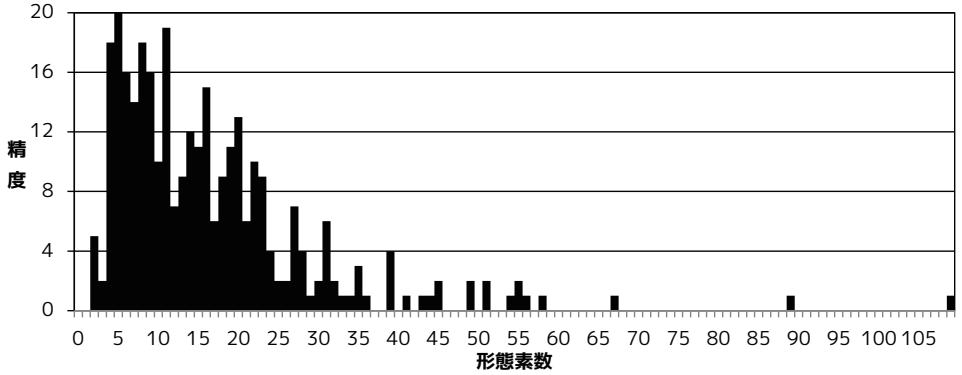

図 8 一文の形態素数

メントを上位 1 件で推薦できなかった場合において，5 件以内，10 件以内に含まれていたことを示す．したがって，提案手法を用いた推薦を行う場合，上位 5 件，または上位 10 件を開発者に提示することで，高い確率で適切なソースコードを推薦できることを示している．開発者に上位 1 件のソースコードとコミットコメントを提示する場合，開発者はその 1 つのみを読んで確認する．しかし，上位 5 件まで，10 件までを提示する場合，開発者は提示されたものを順番に確認し，それぞれが正しいものかを判断する必要がある．本研究の目的は，閲覧中の情報と同じコンテキストに属する情報を検索する労力を削減することであるが，上位 5 件，10 件ものソースコードとコミットコメントをそれぞれ確認する作業は開発者の労力を割くことになる．提案手法では，上位 1 件のときに 75.9%の精度があり，TF-IDF 法と比較して，12.5%精度が高いため，より少ない提示数で適切な情報を開発者に示すことができる．

7. おわりに

本研究ではコンテキストを検索する労力を削減するために，バグ報告に関係するソースコードを推薦する手法を提案した．提案手法は，バグ報告のコメントから形態素 N-gram によりフレーズを抽出し，バグ報告コメントと同じフレーズを多く含んでいるソースコードのコミットコメントを推薦する．提案手法をオープンソースソフトウェアプロジェクトのリポジトリに適用した結果，最大で 75.9%の精度で推薦できた．提案手法を TF-IDF を用いた推薦手法と比較したところ，N が 3 の場合に TF-IDF より高い精度で推薦することができた．

この要因として，特定のバグに関係するコンテキストにしか存在しないキーフレーズを抽出できていたためだと考えられる．今後，本論文では形態素 N-gram のフレーズのみに注目したが，TF-IDF で扱う形態素とくらべてどう違うのかを考察することにより，提案手法の特徴を得られると考えられる．また，形態素数のヒストグラムより，形態素数が 10 未満の文が全体の約 3 分の 1 存在しており，例えば，8 形態素を 1 フレーズとする場合に，7 形態素以下で構成される文は，その文がそのまま不正解になり，精度が下がる原因となっている可能性がある．今後，フレーズを構成する形態素数を変更する場合に，形態素数未満の文を省いて実験を行うことで，より精度が上がると考えられる．この他，実験対象とした，OSS 開発プロジェクトはなでしこプロジェクト一つだったため，他の OSS 開発プロジェクトに提案手法を適用することにより，新たな知見が得られると考えられる．

参考文献

[1] Sengupta, B., Chandra, S. and Sinha, V.: A research agenda for distributed software development, *Proceedings of the 28th international conference on Software engineering*, pp. 731–740 (2006).
[2] Gutwin, C., Penner, R. and Schneider, K.: Group awareness in distributed software development, *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pp. 72–81 (2004).
[3] John, G. H. and Langley, P.: Estimating continuous distributions in Bayesian classifiers, *Proceedings of the 11th conference on Uncertainty in artificial intelligence*, pp. 338–345 (1995).
[4] Bird, C., Bachmann, A., Rahman, F. and Bernstein, A.: LINKSTER: enabling efficient manual inspection and annotation of mined data, *Proceedings of the 18th ACM SIGSOFT international symposium on Foundations of software engineering*, pp. 369–370 (2010).
[5] Ohira, M., Yokomori, R., Sakai, M., ichi Matsumoto, K., Inoue, K. and Torii, K.: Empirical project monitor: A tool for mining multiple project data, *Proceedings of the International Workshop on Mining Software Repositories*, pp. 42–46 (2004).
[6] Salton, G. and McGill, M. J.: *Introduction to Modern Information Retrieval*, McGraw-Hill, Inc. (1986).
[7] Sun, C., Lo, D., Wang, X., Jiang, J. and Khoo, S.-C.: A discriminative model approach for accurate duplicate bug report retrieval, *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pp. 45–54 (2010).
[8] Bacchelli, A., Lanza, M. and Robbes, R.: Linking e-mails and source code artifacts, *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pp. 375–384 (2010).