一般社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

形態素 N-gram を用いた不具合修正完了ソースコードの特定

河居 寛樹† 上野 秀剛† 伊原 彰紀‡

†奈良工業高等専門学校 〒639-1080 奈良県大和郡山市矢田町 22 番地
‡奈良先端科学技術大学院大学 〒565-0456 奈良県生駒市高山町 8916-5
E-mail: †{h-kawai, uwano} @info.nara-k.ac.jp, ‡akinori-i@is.naist.jp

あらまし 本研究の目的はバグ報告文から修正が完了したソースコードを予測し，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，バグ報告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．評価実験の結果，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．

キーワード 開発コンテキスト，ソースコード推薦，形態素解析，N-gram

Linking Fixed Bug Report to Source Code Using N-gram

Hiroki KAWAI† Hidetake UWANO† and Akinori IHARA‡

†Nara National College of Technology 22 Yata-cho, Nara Yamatokoriyama, Nara, 639-1080 Japan
‡Nara Institute of Science and Technology 8916-5 Takayama-cho, Ikoma, Nara, 565-0456 Japan
E-mail: †{h-kawai, uwano} @info.nara-k.ac.jp, ‡akinori-i@is.naist.jp

Abstract This paper proposes a method to recommend the developer the source code from a bug report. The proposed method combines the N-gram and morphological analysis. It finds the commit log which similar to a comment written in the target bug report. Evaluation experiment applied the proposed method to the repository of open source software projects. The result shows that the proposed method recommended the correct source code for 75.9% of the bug reports.

Keyword Development context, Source code suggestion, Morphological analysis, N-gram

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)がある．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．

一方で，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い．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．我々は予備調査として，すでに修正が完了したバグ報告のコメントと修正されたソースコードのコミットコメントに形態素解析と N-gram を適用し，どれだけ同じフレーズが含まれているか調査した．その結果，5-gram の場合に 58%のコミットコメントにバグ報告コメントから取り出されたフレーズが含まれていた．本稿ではグラム数を変化させたときの推薦精度を調査し，提案手法の有効性を確認する．

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 Googlecode 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，過去に存在した類似した不具合を 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，楕円を繋ぐ点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，点線で示される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 上で不具合原因の突き止めや，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 DSS を使って行われたのかという，開発のコンテキストは DSS 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上記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[1]．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．

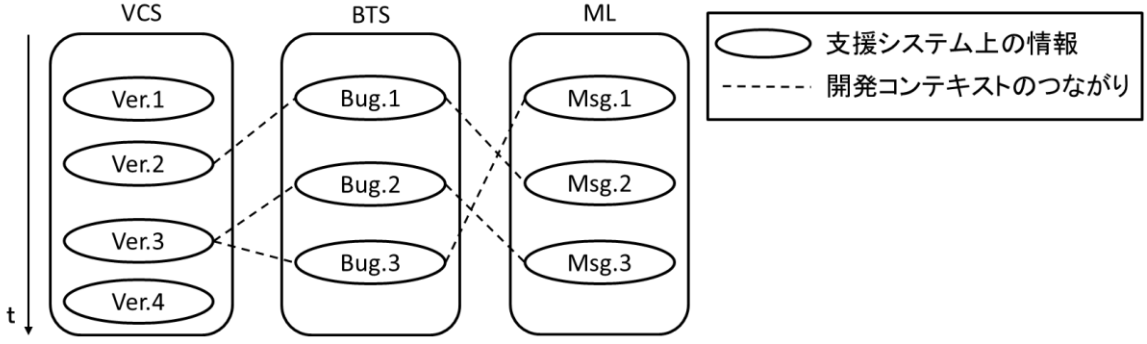

図 1. 開発支援システム上のコミットコメント

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．

3. 提案手法

提案手法は，形態素解析でバグ報告コメントを単語に分割し，N-gram を用いて連続した単語をフレーズとして生成する．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章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単語に分割し，次節で説明する N-gram を単語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単語を N-gram における最小単位とし，連続した N 個の単語（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 を組み合わせた手法である．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．たとえば，“フラグを更新する”という文章の場合，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 を求めることで，元の文章と異なる意味を持つ単語やフレーズの抽出を抑制できるため，推薦制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 2 に示す．

1) バグ報告の文章に形態素 N-gram を適用し，フレーズを抽出する．このとき，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，文をまたいだ抽出は行わない．
2) 抽出したフレーズそれぞれに対して，コミットコメントに出現する回数をカウントする．
3) バグ報告ごとにコミットコメントをカウント数の多い順に順位付けする．
4) 上位のコミットコメントを修正候補として開発者に推薦する．

4. 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．

提案手法の推薦精度は形態素 N-gram を抽出するときの N の値によって異なると考えられる．本実験では 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．

4.1. 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 145 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 153 組である．

4.2. 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 位，5 位，10 位以内に答えが存在するか確認する．上位 1 位，5

[1] MeCab: http://mecab.sourceforge.net/

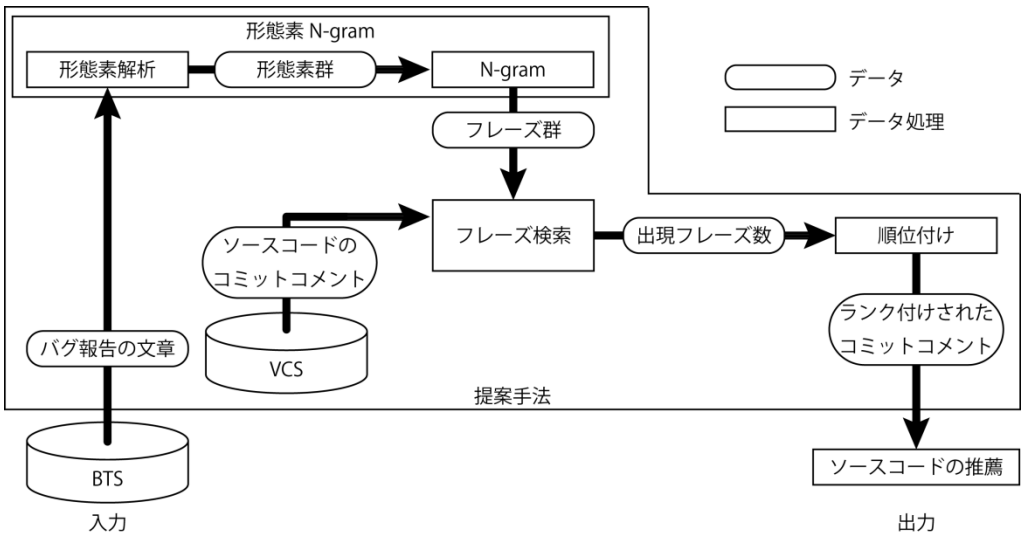

図 2：提案手法の処理手順

位，10 位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，評価を行う．精度は 0～1 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．

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

4.2.1. 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．

4.2.2. 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[2]．TF-IDF 値は式(2)から(4)で計算される．

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

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

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

$tf_{c,n}$ 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[3]や電子メールとソースコードをリンク付ける研究[4]で利用されているオーソドックスな方法なため，TF-IDF と提案手法の比較を行う．TF-IDF を用いた手法の手順を以下に示す．

1) バグ報告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用いて TF-IDF 値を求める
3) 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
4) 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

4.3. 実験手順

実験の手順を以下に示す．

1) なでしこの開発プロジェクトのバグ報告 145 件に対して提案手法を適用する
2) バグ報告 145 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)2)の処理を行う
4) 手法を TF-IDF に変更し，1)2)の処理を行う
5) 3)4)で求めた精度を比較する

5. 結果

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 位，5 位，10 位まで推薦したときの精度を図 3 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 位：0.759，上位 5 位：0.911，上位 10 位：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．

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 位まで推薦した場合と上位 5 位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．

また，TF-IDF を用いた手法の場合，上位 1 件のみに注目した場合の精度は 0.634，上位 5 件の場合は 0.876，上位 10 件の場合は 0.915 となった．N-gram の結果と TF-IDF の結果を比較すると，N が 3 の場合に TF-IDF より高い精度が得られた．上位 1 位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．

6. 考察

N-gram の 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でコミットコメントを推薦できた．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，提案手法は形態素 N-gram 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．そのため，提案手法においてより高い推薦精度があえられたと考えられる．

特に，本実験で最も精度の高かった N が 3 の場合，“ファイルの先頭”や“反転処理の改善”のようなある単語をより詳しく説明するフレーズを抽出できるため，同じ単語を用いた，異なるコンテキストに属する文章を推薦候補から除外することができたと考えられる．一方で，N が 2 以下の場合は，このような効果はほとんど得られないため精度が低かったと考えられる．

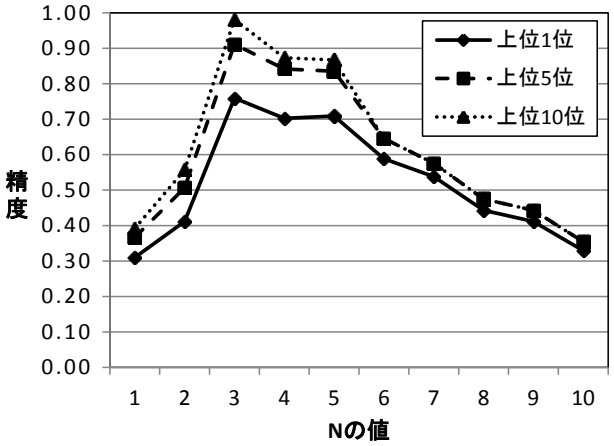

図 3：提案手法の推薦精度

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．提案手法では，文をまたいだ形態素 N-gram の抽出は行っていない．そのため，あるコミットコメントに含まれる文の形態素数がいずれも N より小さい場合に推薦を行えず，精度が下がる可能性がある．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 4 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．これは，実験における，提案手法が抽出できる形態素数 10 を超える文が多数存在していたことを示している．

一方で，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に推薦を行うため，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．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．

推薦する件数を上位 1 から，上位 5 位まで，上位 10 位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メントを 1 位で推薦できなかった場合において，5 位以内，10 位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．

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．

今後，提案手法によって推薦できなかったコミットコメントについて文章を精査し，提案手法が適切に機

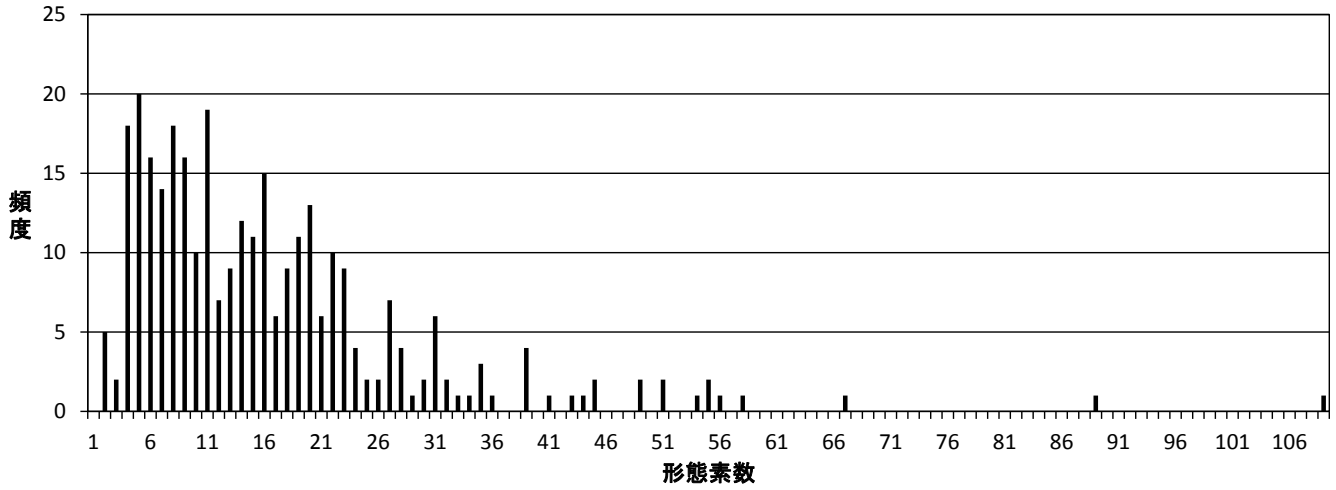

図 4：一文の形態素数

能しないと思われる以下の 2 点について調査する．

- 文書間で現れる形態素 N-gram が類似しているが同じコンテキストに属さない場合
- 同じコンテキストに属している（開発者がリンクしている）文書間で抽出される形態素 N-gram が大きく異なる場合

また，バグ報告コメントに対応した修正が行われた際のコミットコメントには，バグ報告コメントからコピー・アンド・ペーストした文章が含まれている可能性が高い．今後，このような文章がどれだけ存在するのか調査し，推薦精度にどのような影響を与えているか調査する．

文　献

[1] M. Ohira, R. Yokomori, M. Sakai, K. Matsumoto, K. Inoue, and K. Torii, "Empirical Project Monitor: A Tool for Mining Multiple Project Data," In Proc. International Workshop on Mining Software Repositories (MSR2004), pp.42-46, 2004.

[2] G. Salton, and M. J. McGill, "introduction to modern information retrieval," McGraw Hill, 1983.

[3] Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, Siau-Cheng Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," In Proc. the 32nd ACM/IEEE International Conference on Software Engineering, Vol.1, pp.45-54, 2010.

[4] Alberto Bacchelli, Michele Lanza, Romain Robbes, "Linking e-mails and source code artifacts”, In Proc. the 32nd ACM/IEEE International Conference on Software Engineering, Vol.1, pp.375-384, 2010.