オープンソースソフトウェアにおけるバグ混入コミットのトピック分析

椋代 凜
京都工芸繊維大学 大学院工芸科学研究科 情報工学専攻
r-mukudai@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

要旨

本研究では, バージョン管理システムにおいて開発者がバグを出したコミット（以下, バグ混入コミット）により変更されたソースコードに対してトピック分析をかけることにより, 特定の開発者のバグ傾向を分析する. 将来的には, 抽出されたバグ傾向を開発者本人に提示することで, バグの改善に役立てることを目指す.

一般的に, 開発者はプログラム理解のためにソースコードの識別子にプログラムの内容や機能に関するキーワードを埋め込むため, ソースコードから識別子を取り出しそれらにトピック分析を適用すれば, プログラムに潜在するトピックが推定可能である. 本研究では, これの対象をバグ混入コミットで変更されたソースコードに限定する. 適用実験には, *GitHub* で公開されている *Java* ネットワークアプリケーションフレームワーク「*Apache MINA*」を用いる. 実験の結果, 開発者によって程度の差はあるものの, 特定のプログラムの機能に属するトピックをバグ混入コミットから抽出することが出来た. また，開発者個人のバグ混入コミットとバグ非混入コミットから抽出されたトピックの違いを分析することで，その開発者がバグを混入する状況の分析が可能となった.

1　はじめに

ソフトウェア開発において，バグが混入する時点や状況を分析することは，以降のバグを未然に防ぐために有効な手段である．特に，ソフトウェア開発者が頻繁にバグを発生させてしまう状況を知ることは，開発者の啓発にもつながるため，開発者の行動に焦点を当てた研究が必要とされている．開発者の開発行動の成果としてはソースコードやコミットメッセージなどが利用でき，テキスト分類手法を適用する研究が進んでいる [4, 8].

自然言語テキストを分類する統計的手法としてトピックモデルがある. これは文章をそれを構成する単語の集合体と見なし, それらの単語は独立に出現するのではなく単語が潜在的に持つトピック（話題）に従って出現するとするものである. トピック分析をすることで, 文章の詳細を確認せずともその文章が持つ話題を知ることが可能になる.

本研究では, 自然言語テキストではなくプログラムのソースコードを対象としてトピック分析を行い，開発者がバグを混入したコミットにおいて、どのようなトピックを扱っていたのか，そしてそれはバグを混入しなかった場合と比べて差異があるのかを分析する.

ソースコード中で用いられる関数名や変数名といった識別子は, それらを構成する単語を理解することでそのプログラムの意味や機能を知ることが可能であり, コメント文と同様にソースコードを理解する重要な情報源である. 一般的に, 開発者が識別子の命名に用いるキーワードはそのプログラムの機能に関する単語であることから, 同じ機能を持ったモジュールに用いられる識別子は一つの文脈のように同じまとまりを持っていると考えられる. 本研究では, このまとまりを一つのトピックとして捉える. そして，特定の開発者が書いたコード群をいくつかのトピックに分類し, 特にその開発者が不具合を多く出すトピックを特定する.

まず, バージョン管理システムである Git を用いて, 適当なオープンソースソフトウェアのリポジトリを取得し, バグ混入コミットによって変更されたファイルだけをそれらのコミットの Author 毎に取り出す. そして, それぞれのファイルから識別子を抽出し, それら識別子にトピック分析を適用する. 先行研究によって，ソースコードの情報からそのソフトウェアに関するトピック情報が抽出

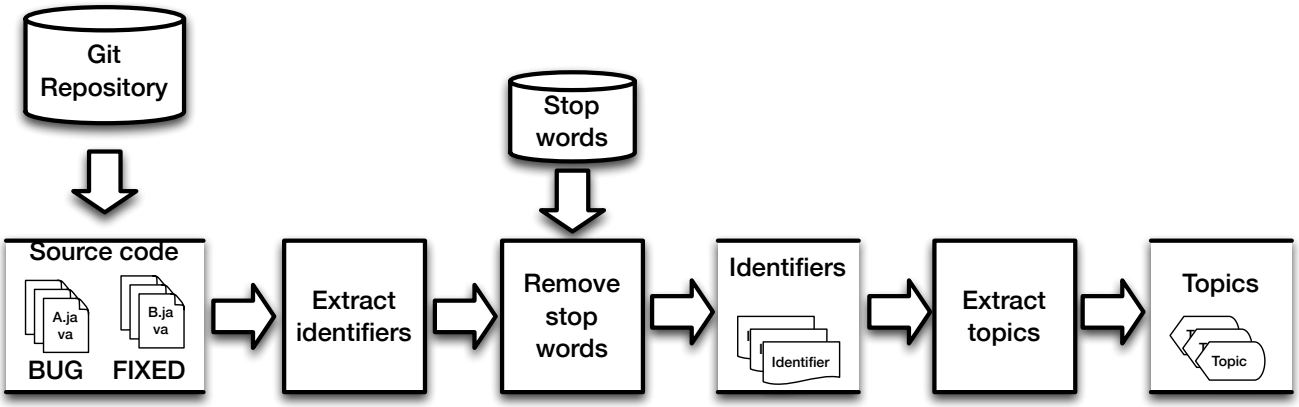


図 1: トピック分析の流れ

できることが知られている [2, 8]．本研究では, 対象のソフトウェア全体が持つトピックも抽出し, それらと各開発者の持つトピックとの関係についても考察する．トピック分析は統計計算ソフト R のパッケージに含まれる LDA(Latent Dirichlet allocation) を用いて行う．一般的に, トピック分析は自然言語テキストを対象とするため, ソースコードを対象とする場合は通常とは異なるストップワードを用いる必要がある．また LDA はトピックの推定は行えるが, 予めトピック数をユーザ側で入力する必要があるため, その推定も別途行った．

オープンソースプロジェクトを利用した適用実験の結果，トピック分析によってバグ混入時と非混入時において，トピックに違いがあること，また各開発者に限定して調べた場合でも，同様に違いがあることを確認した．

本論文の以降の構成を述べる．2 節では，研究に当たって設定した質問を挙げ，問題を明確にする．3 節では，トピック分析の手法について概要を述べる．4 節ではトピック分析を Apache MINA プロジェクトに適用した実験について述べる．5 節ではまとめと今後の課題を述べる．

2 研究設問

本研究では以下の 3 つの研究設問を設定し，これらについて調べる．

RQ1: プロジェクトにおけるバグ混入コミットとバグ非混入コミットの間でトピックの種類に差はあるか．

RQ2: 開発者毎にトピックの傾向は違うか．

RQ3: 個人のバグ混入コミットとバグ非混入コミットの間でトピックの種類に差はあるか．

RQ1 はプロジェクトにおけるコミット作業のうち，バグが混入したものとそうでないものとの間で，トピック情報が異なるかを調べるものである．差があるとすれば，トピック分析によってバグの混入しやすいソースコード特定が可能になる．

RQ2 は開発者個人毎にトピックを抽出し，その傾向の違いを特定することで，個人が得意とする分野を分析する．

RQ3 は開発者個人毎のバグ混入コミットの傾向を分析し，バグが混入したものとそうでないものとの間で，トピック情報が異なるかを調べるものである．ここで差を発見できれば，個人がバグを作り込みやすい場面を特定し，バグの発生を未然に防止することに役立つ．

3 トピック分析

本研究で行うトピック分析の流れを図 1 に示す．以降の節ではこの手順を順を追って説明する．

3.1 コミットに関連するファイルの抽出

バグ混入コミットを抽出して，バグに関わった開発者がどのようなトピックの下で作業をしていたのかを明らかにする．バグ混入コミットの抽出方法は 4.2 節で述べる．

あるコミットが指定されるとそれに関連するファイルは git リポジトリの差分表示によって特定される．そこで，バグ混入コミットに関連づけられるソースコードか

ら識別子を抽出した後，その識別子をさらに単語に分解し，その単語を単位とするトピックモデルを作成する．

3.2 識別子の抽出

次に，ソースコードファイルから識別子だけを抽出する手法について述べる．どの識別子がどのファイルから出現したか（即ち，識別子の出現の頻度）はトピック抽出の結果に影響するので，ファイルと識別子との対応関係は壊さないように抽出する．本研究では以下のツールを用いて抽出を行った．

3.2.1 識別子の抽出ツール: lscp

ソースコードからの識別子の抽出には，GitHub で公開されている「lscp」という Perl プログラムを利用した [7]. lscp はソースコードの文章を識別子やコメント，文字列などに区別し抽出することが出来る．また，オプションによって抽出する対象を変更することも出来る．今回はコメント文や文字列は除き，識別子だけを抽出するように設定した．また，識別子は通常複数の単語の組み合わせから構成されている．lscp はこうした単語を分割する機能も有するため，この機能により識別子を単語に分解した．

3.2.2 形態素解析ツール: TreeTagger

抽出した識別子には，英単語の進行形や過去形といった活用形が混じっている．それを形態素解析システム「TreeTagger」を使用することで基本形へと戻す語幹処理を行った [5]. この操作と同時に，ストップワードを利用してトピックとして不要な語を取り除く．なお，ストップワードとしては，トピック抽出ツール mallet [3] が有するストップワードリストを利用した．

3.3 トピック抽出

語幹処理を施した識別子群をテキストとして，LDA に入力する．LDA はトピック数を入力する必要があるが，事前に対象に潜在するトピック数を知っている状況は少ない．事前にトピック数を推定する手法は perplexity や HDP-LDA を利用するなど様々な方法が考案されているが，先行研究である [9] を参考にし，今回は term-score [1] を用いた方法を使用する．

名称	Apache MINA
ユニーク開発者数	20
バグ混入開発者数	12
全コミット数	2014
(内) バグ混入コミット数	688

表 1: 対象としたプロジェクト

3.3.1 Term-score

自然言語処理の分野では，文書内の単語の重要度の尺語として tf-idf がよく用いられる．tf-idf における文書をトピックに置き換えたものが term-score である．term-score はあるトピック内でのある単語の特徴量として，次のように表される．

$$\text{term-score}_{k,v} = \hat{\beta}_{k,v} \log\left(\frac{\hat{\beta}_{k,v}}{(\prod_{j=1}^{K} \hat{\beta}_{j,v})^{\frac{1}{K}}}\right) \quad (1)$$

$\hat{\beta}_{k,v}$: トピック k での語 v の出現確率
K : トピックの総数

右辺の前半部が tf 値に対応し，対数部分が idf 値に対応することで，全てのトピックで頻繁に現れる単語の term-score は低くなる．トピック内の特徴語とその単語の term-score を各次元として対応付けたベクトルを作り，それら同士のコサイン類似度を測定し，内容の似通ったトピックを結合することでトピック数を推定する．

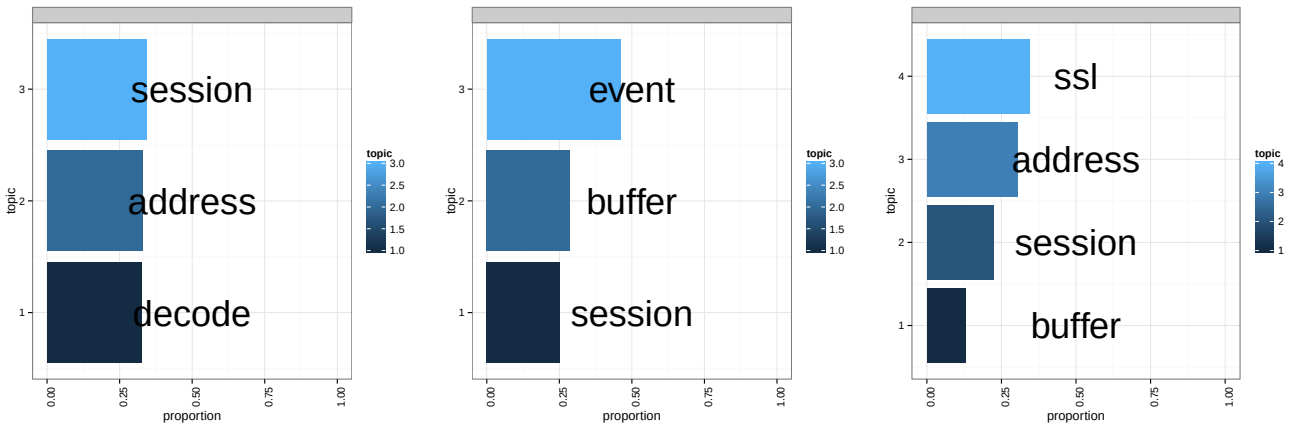
$$\cos(\vec{x_1}, \vec{x_2}) = \frac{\vec{x_1} \cdot \vec{x_2}}{|\vec{x_1}||\vec{x_2}|} \quad (2)$$

3.4 トピック数の推定

次のステップに基づき，対象のトピック数を推定する．

STEP 1. 意図的に多めにトピック数を設定し，LDA を用いてトピックを抽出する．

STEP 2. 抽出されたトピック間のコサイン類似度をそれぞれ測定する．どのトピックとも類似度が閾値を上回らない（類似度が低い）トピックを「単独トピック」とし，どれか一つでも閾値を上回るトピックが存在するトピックを「類似トピック」とする．



(a) 全コミット　(b) バグ混入コミット　(c) バグ非混入コミット

図 2: プロジェクト MINA におけるトピックの傾向

表 2: 各開発者のバグ混入コミット

開発者	全コミット	バグ混入コミット [1]	バグ関連ファイル [1]
D_1: Alan Cabrera	3	0	–
D_2: Alex Karasulu	49	5	271
D_3: Ashish Paliwal	7	0	–
D_4: Edouard De Oliveira	70	14	78
D_5: Emmanuel Lecharny	386	45	531
D_6: Ersin Er	12	0	–
D_7: Garrett Rooney	1	0	–
D_8: Guillaume Nodet	1	0	–
D_9: Jeff Genender	1	0	–
D_{10}: Jim Jagielski	3	1	1
D_{11}: John E. Conlon	3	0	–
D_{12}: Julien Vermillard	167	35	420
D_{13}: Maarten Bosteels	44	15	52
D_{14}: Mark Webb	62	33	89
D_{15}: Mike Heath	35	17	43
D_{16}: Niklas Gustavsson	10	2	6
D_{17}: Niklas Therning	59	35	149
D_{18}: Peter A Royal Jr	38	22	54
D_{19}: Sangjin Lee	1	0	–
D_{20}: Trustin Lee	1062	464	742

STEP 3. 類似トピックだけを取り出し, それらをノードとして類似度が閾値を上回ったトピックの組にリンクを張ったグラフを考える. その時に完全グラフを構成するノードを一つにまとめる. これを「結合トピック」とし, さらに複数の結合トピックに含まれるトピックを「重複トピック」とする.

STEP 4. 重複トピックは重要性が低いものとして, 各結合トピックから削除する. こうして残った「結合トピック」数と「単独トピック」数の合計を, トピック数とする.

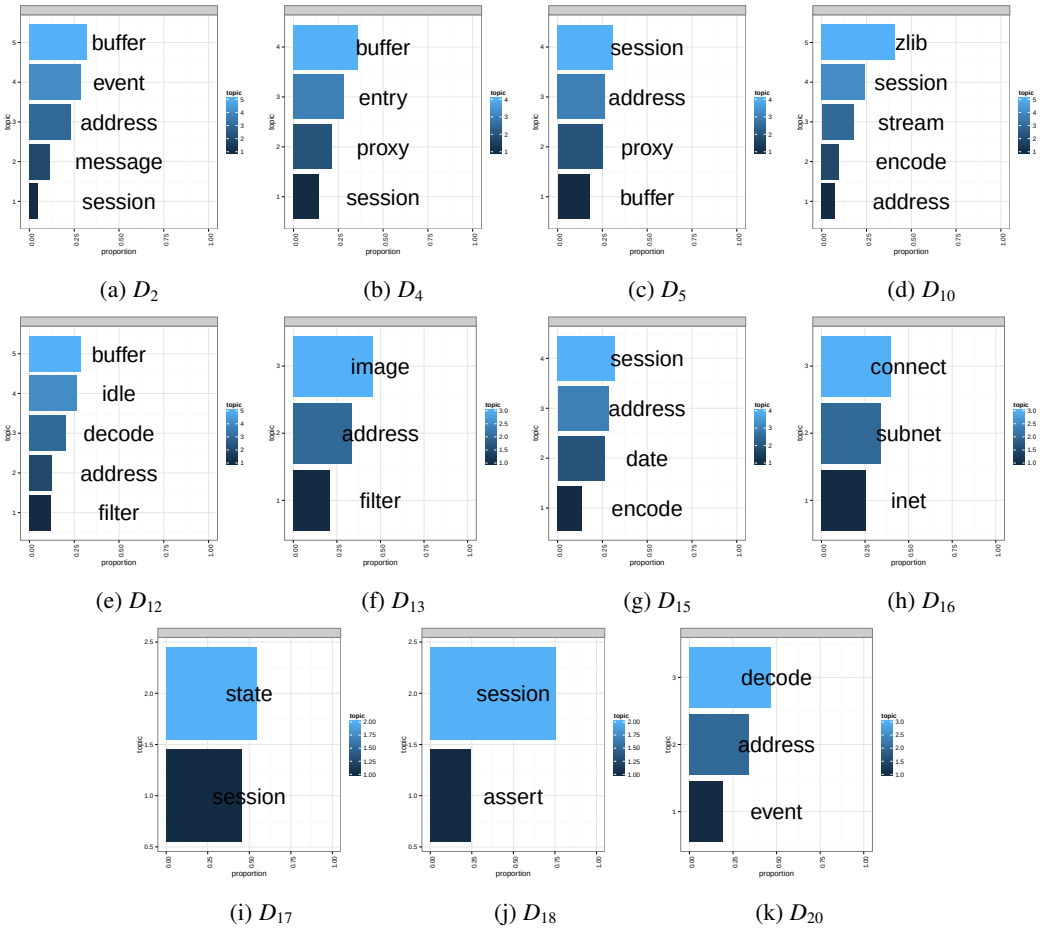
STEP 5. STEP 4 で出したトピック数で，再び STEP 1 から操作を行う. これを繰り返し行い, 平均してこれ以上トピック数が減らなくなった時点のトピック数を, 最終的なトピック数とする.

STEP 3 で用いた閾値は，類似度の乖離に基づいて設定する.

4 適用実験

4.1 対象プロジェクト

本実験で対象としたのは Apache プロジェクトで開発されているソフトウェア「MINA」の開発データである. Apache MINA はネットワークアプリケーションの開発フレームワークであり，ユーザが簡単に高度なネットワークアプリケーションを作成できるようにするための補助ツールである.



(a) D_2 (b) D_4 (c) D_5 (d) D_{10}

(e) D_{12} (f) D_{13} (g) D_{15} (h) D_{16}

(i) D_{17} (j) D_{18} (k) D_{20}

図 3: 開発者毎の全コミットにおけるトピックの傾向

具体的には，ソースコードの git リポジトリとバグデータベース JIRA から抽出したバグが混入した時点などの情報を用いる．

MINA の全開発人数は 20 人で，全コミット数は 2014 個である．そのうちの 688 個がバグ混入コミットである．表 1 に本プロジェクトの特性を挙げる．

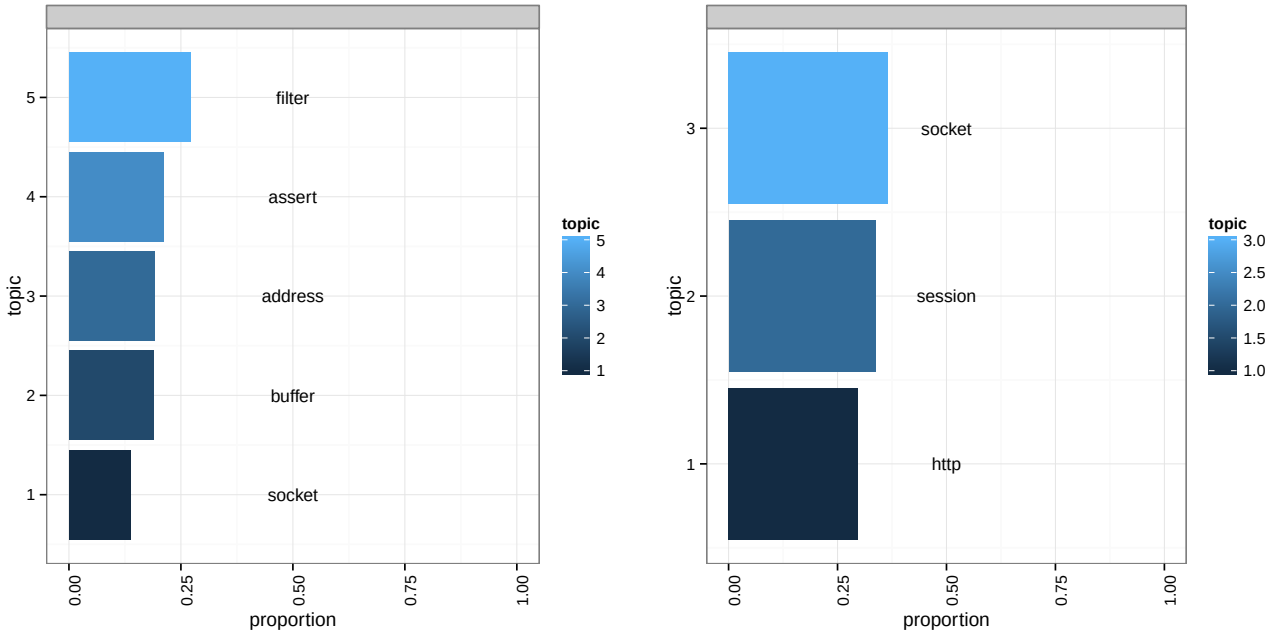
4.2 バグ混入コミットの抽出

バグ混入コミットの抽出には文献 [6] に示されている SZZ アルゴリズムを用いる．SZZ アルゴリズムの概要は以下の通りである．

1. ある不具合 f_i に対し，バージョン管理記録のコメント部分から f_i が修正されたコミットを特定する．
2. f_i の修正の際に実際に変更されたクラス $CL_{FaultFixed}$ を特定し，それ以前の状態は不具合が含まれていないと仮定する．
3. 不具合は発見される前に作り込まれているはずなので，$CL_{FaultFixed}$ のうち，不具合が初めて報告された時点 D_{f_i} から f_i が修正されるまでの間，1 度も変更されなかったクラスを CL_{Faulty} とする．この CL_{Faulty} を作った時点のコミットをバグ混入コミット $CM_{BugInject}$ とする．
4. 全ての不具合に対し，1 から 3 の作業を繰り返す．

この結果, バグ混入コミットの集合 $CM_{BugInject}$ が得られる [1]．

[1] バグ混入コミットに関して，ファイル “pom.xml” に関する不具合は不具合として数えていない．



(a) バグ混入コミット

(b) バグ非混入コミット

図 4: 開発者 D_{20} のコミットにおけるトピックの傾向

```
charset char get set read write
open org net com file size length
index type output input put array
list flag integer min max www src
utf impl val prev util
```

図 6: MINA におけるトピック抽出のストップワード

4.3 バグ混入コミットの取り出し

対象となるプロジェクト Apache MINA のバグ混入コミットで変更されたファイルの総数は 2436 個であり，それに携わった開発者は 12 人である．表 2 に各開発者についての全コミット数，バグ混入コミット数，バグ混入に関連したファイル数を示す．表 2 からは，開発者のコミット数には大きなばらつきがあり，D_{20} が主要な役割を果たしていることが分かる．

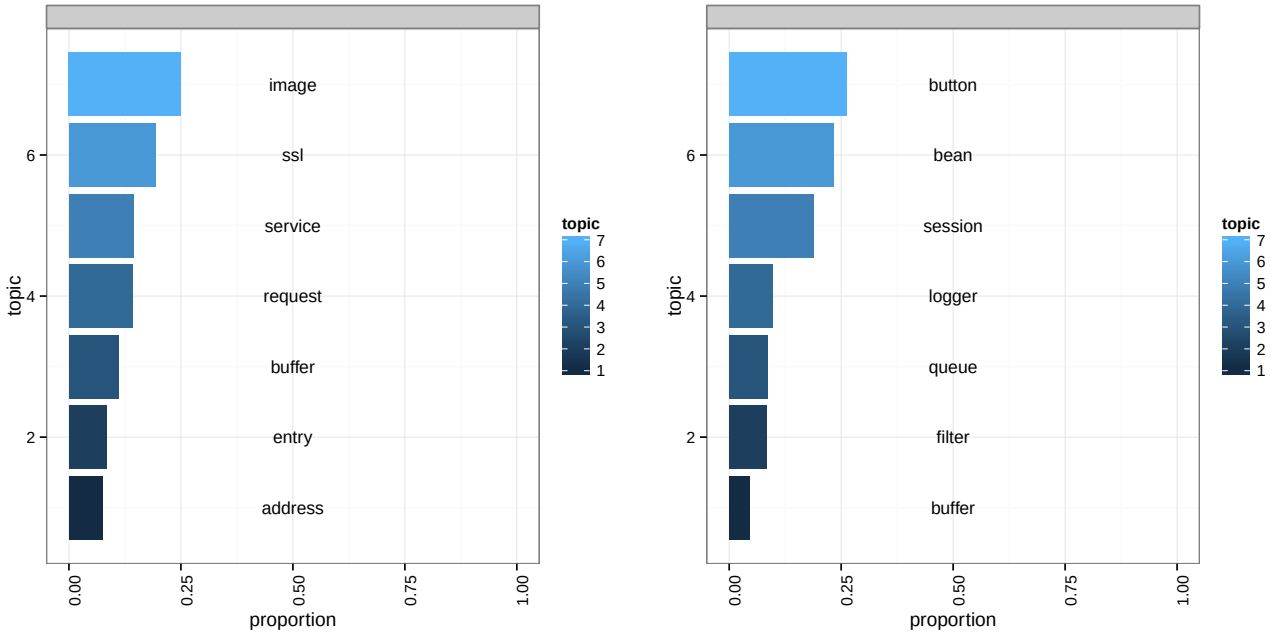
Apache MINA からトピックを抽出するにあたって，図 6 に示す語は追加のストップワードとして扱った．これらの語はあまりにも頻繁に出現し，さらにプログラミング言語中に一般的に存在する語であることから除外した．

4.4 実験結果

4.4.1 プロジェクト全体に対する分析

図 2 は Apache MINA プロジェクトの全体，バグ混入，バグ非混入それぞれのコミットから抽出したトピックの最も重要な語を表したものである．図 2a にプロジェクト全体から抽出したトピックを挙げる．ここで挙げられた “session”, “address”, “decode” の 3 語が，MINA では支配的であることを示している．以降のトピックを示すグラフの横軸は正規化したトピックの出現頻度を示しており，縦軸はトピックの種類を表す．また，グラフに重ねてトピックに含まれる単語を示している．図 2 中のトピックに含まれる単語はネットワークアプリケーションの開発フレームワークという特性を反映し，session や address といった語が多く出現する．

図 2b にプロジェクト MINA におけるバグコミットから抽出した頻出トピックを挙げる．図 2b からは “event”, “buffer”, “session” というトピックが出現し，中でも “event” に関するバグ混入コミットが多いことが分かる．一方，図 2c にプロジェクト MINA におけるバグ非混入コミットから抽出したトピックを示す．このグラフでは，“ssl” が最頻出であり， “address”, “session”, “buffer” と



(a) バグ混入コミット　　(b) バグ非混入コミット

図 5: 開発者 D_{13} のコミットにおけるトピックの傾向

続く．

このことから，バグが混入するコミットでは “event” に関するトピックが多く，逆に “ssl” に関連するトピックが存在したコミットではバグが混入しないことが多いと分かる．

以上より，RQ1 での設問「プロジェクトにおけるバグ混入コミットとバグ非混入コミットの間でトピックの種類に差はあるか.」は「差を発見できる」と結論づける．

4.4.2　開発者全体の特徴分析

ここでは，各開発者の全コミットに対するトピックを抽出し，その結果を図 3 に示す．なお，表 2 に挙げた開発者のうち，バグ混入コミットを作成していない開発者は以降の分析で除外している．

図 3 を調べると多くの開発者は図 2a で挙げるトピックを含む開発をしていることが分かる．開発者間の傾向は類似したものが多いが，D_{13} は唯一 “image” というトピックが見られる．これは，D_{13} が GUI 関連のコードを多く書いていることに起因している．

開発者 D_{10} も “zlib” という語がトピックで出現している．ただし，この D_{10} は 3 コミットしかしていないため，少ないファイルでの特徴が抽出されたものとなっている．

このように，開発者のトピックを分析することでその開発者が従事している開発をある程度特定できる．さらに抽出したトピックを利用すればより詳細な情報を得ることも期待できる．

以上のことから，RQ2 での設問「開発者毎にトピックの傾向は違うか」は「違う」と結論づけられる．

4.4.3　開発者のバグ傾向分析

上で挙げた 2 人の開発者 D_{20} と D_{13} について，それぞれのバグ混入コミットとバグ非混入コミットを区別した上でトピックを抽出したものを図 4，図 5 に挙げる．

図 4a と図 4b は最も多くのコミットを行った開発者 D_{20} のバグ混入時と，バグ非混入時のトピックをそれぞれ示したものである．この図からは，D_{20} がバグを混入するコミットは “filter”, “assert”, “address” などのトピックがあるときに多いことが分かる．一方でバグ非混入トピックのトピックは “socket”, “session” などであり，こうしたトピックについてはあまりバグを混入しなかったことが分かる．

同様に別の開発者 D_{13} のバグ混入コミットとバグ非混入コミットのトピックを図 5 に示す．

図 5a と図 5b を比較する．図 5a のバグ混入コミットのトピックには “image”, “ssl”, “service” が存在する．これは，先に図 3f で示されたように，GUI 関連の語である “image” の出現頻度が最も高いことを示している．一方で図 5b のバグ非混入コミットにおけるトピックでは “button”, “bean”, “session” となっており，こちらも GUI 関連のトピック “button” が含まれている．すなわち，バグを混入した場面で最も多かったのは GUI の image に関わるものの変更であり，それはバグ非混入のものとは傾向が違うことが分かる．

以上の結果より，RQ3「個人のバグ混入コミットとバグ非混入コミットの間にトピックの種類に差はあるか.」については「差がある」と結論づける．

5 まとめと今後の課題

本報告では，ソフトウェアのソースコードからトピックを抽出する技法を利用し，開発者がバグを混入するコミットでの特徴を抽出することを試みた．適用実験として，Apache MINA プロジェクトからトピックを抽出し，開発者毎の分析を実施した．分析の結果を以下にまとめる．

- プロジェクトにおけるバグ混入コミットとバグ非混入コミットの間でトピックの種類に差は存在する．
- 開発者毎にトピックの傾向は明らかに異なる．
- 個人のバグ混入コミットとバグ非混入コミットの間でのトピックの種類に差が見られる．

この結果から，トピックの分析は開発者個人が混入するバグの分析に有効であろうと考えられる．

今後の課題としては以下のものが挙げられる．

- 多くのプロジェクト，多くの開発者に対する適用
- 複数のプロジェクトに関わる開発者についてのトピック抽出
- 得られたトピック情報をソースコードに還元して注意喚起を促すシステムの開発

参考文献

[1] D. Blei and J. Lafferty. *TOPIC MODELS*. Taylor and Francis, 2009.

[2] G. Maskeri, S. Sarkar, and K. Heafield. Mining business topics in source code using latent dirichlet allocation. In *Proceedings of the 1st India Software Engineering Conference*, ISEC ’08, pp. 113–120, New York, NY, USA, 2008. ACM.

[3] A. K. McCallum. Mallet: A machine learning for language toolkit. http://mallet.cs.umass.edu, 2002.

[4] O. Mizuno and T. Kikuno. Prediction of fault-prone software modules using a generic text discriminator. *IEICE Trans. on Information and Systems*, E91-D(4):888–896, 2008.

[5] H. Schmid. Treetagger - a language independent part-of-speech tagger, Nov. 2013. `http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/`.

[6] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? In *Proceedings of the 2005 International Workshop on Mining Software Repositories*, MSR ’05, pp. 1–5, New York, NY, USA, 2005. ACM.

[7] S. Thomas. lscp, Nov. 2013. `https://github.com/doofuslarge/lscp/`.

[8] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein. Modeling the evolution of topics in source code histories. In *Proceedings of the 8th working conference on mining software repositories*, pp. 173–182. ACM, 2011.

[9] 芹澤, 小林. 文書内のトピック数を考慮したトピック追跡の試み. 言語処理学会年次大会発表論文集, No. 18, pp. 3–32, 2012.