Vol.2014-SE-183 No.19
2014/3/20

開発メーリングリストマイニングの前処理システムの開発

川島 関夫[1]　亀井 靖高[1]　鵜林 尚靖[1]

概要：オープンソースソフトウェア（OSS）開発プロジェクトのメーリングリストでは，追加機能やバグ修正といったプロジェクトの方針に関するやりとりが多く行われることから，蓄積されたメーリングリストデータを分析し，今後の開発活動に役立つ知見を発見する試み（メーリングリストマイニング）が行われている．しかしながら，メーリングリストデータは，自然言語を含んでいたり，プロジェクトによって運用のされ方が異っているため，解析に用いる前にいくつかの前処理が求められる．本発表では，既存研究で行われている一般的な前処理をまとめ，体系化し，前処理を実施するためのシステムを実装した．Hadoop の開発メーリングリストを対象としてシステムの評価を行った．前処理システムを用いることで，分析データを構成できることが確認できた．

キーワード：オープンソースソフトウェア，開発メーリングリスト，自然言語処理

1. はじめに

複数の開発者によるソフトウェア開発においては，仕様を決定したり，開発者の意思を統一したり，コミュニケーションをとることが必要となる．商用のソフトウェア開発の場合，直接のコミュニケーションが優先されていたり，メールなど，何らかのツールを用いていたとしても，その情報は一般には公開されず，コミュニケーションの様子を観察することはできない．一方，OSS 開発では，コミュニケーションにメーリングリストを用いる場合が多く，コミュニケーションの様子を観察することができる．

また，近年では，Hadoop や OpenStack などをはじめとして，高品質，高機能な OSS の導入事例が数多く報告されており，蓄積されたメーリングリストデータを分析し，今後の開発活動に役立つ知見を発見する試み（メーリングリストマイニング）が行われている [1]．OSS は，複数の人がコミュニケーションをとり，一つのソフトウェアを作り上げていくものであり，メーリングリストマイニングでは，ソースコードの分析だけでは得られないような，ソフトウェア開発に関する新たな知見が期待できる．

しかしながら，メーリングリストは，プロジェクトによって運用の仕方が異なり，分析において問題が発生する場合がある [2]．例えば，開発に関する内容の為のメーリングリストと，使用方法など，ユーザーによる質問を受け付けるメーリングリストを分けて運用している場合もある．しかしながら，一般的に，開発メーリングリストであっても，一般のユーザーも参加することが可能であり，開発者を判別することはできないため，開発者のコミュニケーションを観測することができない．また，メールには，返信をする場合に自動的に返信元メールを引用するなど，送信者が伝えたい文章がどこであるかがわかりづらく，そのまま分析を行うことはできない [2] よって，メーリングリストデータをそのまま分析するのではなく，いくつかの前処理を行いデータを再構成した上で，分析を行う必要がある．

まず，開発メーリングリストマイニングに関する先行研究においてどのような前処理が行われているかを調査した．様々な前処理を実施しているが，そのいくつかは多くの先行研究において共通しており，体系化できる可能性がある．一方，先行研究では，前処理を実施するシステムを，それぞれの研究で行っており，体系立てて前処理を提供しているような研究はまだない．

そこで，一般的な前処理から，必要な前処理だけを選択して実施するようなシステムを実装し，マイニングの高速化，高効率化を目指した．システムは，複数の前処理から必要な前処理だけを選び実施できるようにし，使用者が必要な状態のメールデータを簡単に生成できるような形態としている．そして，開発したシステムを Hadoop 開発メー

[1] 九州大学大学院システム情報科学研究院，福岡市
Graduate School and Faculty of Information Science and Electrical Engineering, Kyushu University, 744 Motooka, Nishi-ku, Fukuoka-shi, 819-0395 Japan

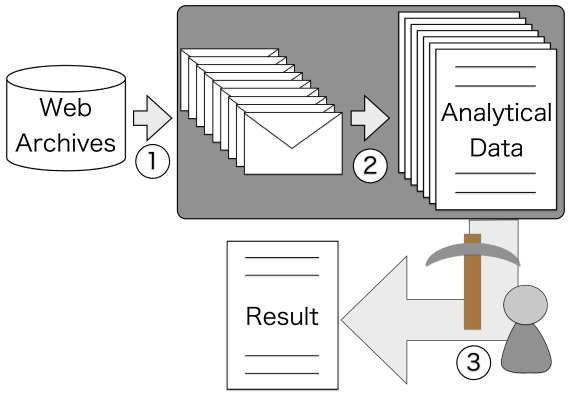

図 1　開発メーリングリストマイニングモデル

リングリストを用いて，システムの評価を行った．

以降，第 2 章では，一般的な開発メーリングリストマイニングの手順を述べ，関連研究をまとめる．第 3 章では，これまでの，メーリングリストマイニングの研究で対処されてきた前処理についてまとめる．第 4 章では，設計したシステムの概要を説明する．第 5 章では，システムの評価を行う．最後に，第 6 章で本稿のまとめと，今後の課題について述べる．

2. 背景と関連研究

2.1 開発メーリングリストマイニング

開発メーリングリストマイニングの一般的な手順を図 1 に示す．メーリングリストデータは，プロジェクトへの新規参加者やユーザーがこれまでの開発についての情報を取得できるように，Web アーカイブスと呼ばれる Web ページを通じてインターネット上で公開されており，データを取得することができる．処理の流れは，下記の通りである．

Step1　Web アーカイブスからメーリングリストデータを取得する

Step2　メーリングリストデータに前処理を施し，後の分析に適した分析データを構成する

Step3　分析データに対して，ツールや人手によって分析する

Step2 における前処理に関して調査した結果，前処理の種類は研究により様々であるが，その多くは共通しており，体系化できる可能性がある．

2.2 関連研究

Miler[3] は，メーリングリストにおけるコミュニケーションを分析するためのツールである．Miler は，メーリングリストのメールとソースコードの関連づけを行うことができる．MarkMail は OSS プロジェクトの開発メーリングリストを公開している Web サービスである．Miler では，主に，6 つの機能を提供している．

(1) メールを読み込む
(2) メールとソースコードのクラスを手動で関連づける Web アプリケーション
(3) メールとソースコードのクラスを自動で関連づけるシステム
(4) スタックトレース，パッチ，ソースコードの特定
(5) メトリクスの算出
(6) メトリクスを XML/CSV 形式で出力

(1)MarkMail のメール，または，Mbox 形式のメールデータを入力とし，メールを読み込み利用する機能がある．

また，メールがソースコードのどの部分に関連しているのかの情報を付加する機能として，(2)Web アプリケーション上で手動操作で関連づけを行う機能と，(3) 自動で関連づけを行う機能がある．

また，(4) メールデータ内に含まれるでは，内容をに関する重要な情報を特定する機能がある．特定できる情報は，3 つあり，エラーメッセージといったプログラムの実行過程の記録であるスタックトレースや，バグや機能変更のためにプログラムに変更を加える場合の差分を表すパッチ，ソースコードが特定できる．

そして，(5) いくつかのメトリクスを算出して出力できる．メトリクスは，メールの送信者の数，本文の行数，ソースコードを含んだメールの件数，Popularity である．Popularity は，ソースコードのあるクラスが，メーリングリストにおいてどれだけ話題になっているかなどの情報から算出されるメトリクスである．

最後に，(6) これらのメトリクスを XML 形式，あるいは，CSV 形式で出力することができる．

Miler を用いることで，メーリングリストをソースコードとつなぎ合わせることができ，ソフトウェアの分析において，最終的な成果物であるソースコードだけではなく，それに至までのコミュニケーションである会話の情報を含めた，違った分析ができることが期待できる．

3. 問題と前処理

3.1 先行研究における問題と前処理

Bird ら [1] は，開発者のメールの送信頻度とソースコードの変更頻度には相関があることを発見した．また，メーリングリストでの会話は，開発者を介さずに参加者どうしで会話をすることはほとんどなく，開発者がメーリングリストで重要な役割を果たしていることを明らかにした．Bird らは，参加者が複数のメールアドレス，氏名を用いている場合に，分析結果に誤りが含まれるとして，別名を解決する前処理を行った．また，メーリングリストの情報には，開発者であるか否かの情報は含まれておらず，開発者を特定する前処理を行った．

また，Guzzi ら [4] は，メーリングリストにおいての会話は，どのようなテーマが多いのかを分析し，メーリングリストの役割を明らかにした．Guzzi らは，まず，複数の

メールアドレス・氏名の問題に対する前処理，及び，開発者を特定する前処理 [1] を行っている．更に，また，メールの内容についての分析を行う場合，引用部分などの機械的に生成される文章が解析のノイズとなってしまうとして，前処理おいてその部分の削除を行っている．メーリングリストにおいての会話は，どのようなテーマが多いのかを分析し，メーリングリストの役割を明らかにした..

また，Rigby らは，メーリングリストでの開発者の言葉遣いの特徴について分析した．前処理として，複数のメールアドレス・氏名の問題に対する前処理，及び，開発者を特定する前処理 [1] が必要になる．更に，メールにおいて，開発者が使用した単語をカウントする処理を行っている．

3.2 別名の問題

メーリングリストのユーザーについての分析を行う場合，一人のユーザーが複数のアドレス，氏名を用いる問題を考慮する必要がある．メーリングリストに送信するメールアドレスは，人により一定であるとは言えない．

例えば，リストに登録してあるアドレスから，外出先で確認できるアドレスへ転送設定をしている場合などに，別アドレスから返信することが考えられる．また，長い期間のデータであれば，メールアドレスを変更することも考えられる．つまり，一人の人物が複数のアドレスを用いていることがあり，同じメールアドレスのものだけを同一人物とした分析結果は誤りを含む可能性がある．

一方で，送信者の情報として，氏名が含まれている．しかし，例えば，Andrew という名前の人物は Andy と名乗る可能性もある．他にも，通称を用いる可能性もある．つまり，アドレスを用いる場合でも，氏名を用いる場合でも，ユーザー一人一人を判定する事は難しい．

そこで，一般的に用いられる前処理は，アドレスと氏名の組を用いて，レーベンシュタイン距離によって類似度を決定し，分類する．類似度は，アドレスのみ，氏名のみ，アドレスと氏名両方の三手法によって決定し，どれか一つでも類似であれば，同一人物と決定する．

まず，アドレス同士の比較では，アドレスの “@” より前のローカル部分を同士で距離が近ければ，同一とする．

次に，氏名同士の比較では，氏名において，名字と名前の組の距離と，ミドルネームや称号（Jr，II など）等を含めたフルネームの両方の和により，距離が近ければ同一とする．

最後に，アドレスと氏名の両方での比較は，一方の名字と名前の両方がもう一方のアドレスのローカル部分に含まれている場合，同一とし，名字のイニシャルと名前全体，あるいは，名字全体と名前のイニシャルのどちらかが，ローカル部分に含まれている場合，同一とする．最終的に三手法の和によって同一人物をまとめる．

```
$ git log
commit 8843cb6d16bebe095993d6252635fd63cbe50d0d
Author: Chris Yan <chrisy@gee.mail.com>
Date:   Tue Dec 24 21:34:22 2013 +1030

    Updated openstack/openstack
```

図 2　変更履歴の例

3.3 開発者の特定

メーリングリストにおける開発者の行動を分析する場合，通常，メーリングリストデータには開発者か否かの情報は含まれておらず，開発者を特定する事は難しい．そこで，メーリングリストデータと Git などのバージョン管理システムのコミットログの情報を結びつける前処理を行う．バージョン管理システムは，ソースコードの管理などに用いられ，リポジトリと呼ばれるデータベースに，ファイルの各バージョンを保持するシステムである．OSS では，このリポジトリが一般に公開されている．

また，変更履歴を閲覧することができ，変更を行った人（開発者）の情報も含まれている．図 2 は，Git の変更履歴の例である．

変更履歴の Author フィールドには，変更者のアドレスと名前の組が含まれている．そこで，別名の問題と同じアプローチでメーリングリストデータの情報と開発者の情報を結びつけ，メーリングリストにおける開発者を特定する．

3.4 不要部分の削除

メールの内容を分析する場合，内容とは関係ない不要部分が解析のノイズになる可能性がある．

例えば，メールデータには，メールの本文以外にも，メールにより一意なメッセージ ID や送信時間など様々な情報が含まれているが，内容とは関係なく，不要である．分析に関係のない部分をあらかじめ削除する前処理について説明する．

3.4.1 引用部分

メールは返信時に，返信もとメールの本文を自動的に引用している場合がある．分析時に，引用部分も内容として捉えてしまうと，複数回返信が繰り返されたようなメールにおいては，送信者が記述した内容以外の部分が大きくなり，メールの内容の分析のノイズになる．そこで，一般的にメールデータ形式として用いられている，MIME メールの形式として，メールの本文のみを取り出した．

一般的に，引用部分には，引用符 > がつけられることから，引用部分を特定し，削除することができる．

3.4.2 自動送信メール

開発メーリングリスト特有の問題として，プロジェクト管理ツール（Jenkins，JIRA）などからの自動送信メールが多く含まれるという問題がある．プロジェクト管理ツールは，ソフトウェアを自動でビルドし，その結果をメール

で通知するなど，ソフトウェア開発における重要な情報を含んでいる．

一方，メーリングリストの内容を送信者別に分析したい場合，頻繁に送られるこれらのツールからの自動送信メールは，送信者は機械であり，分析の対象とならない場合がある．そこで，自動送信と思われるアドレスを指定することで，不要なメールを削除する前処理を行う．

3.5 スレッド化

メーリングリストにおいて，内容を分析する場合，メールをスレッド化することで，分析の対象を減らすことができる．あるメールに対する返信のメールであれば，スレッドとしてまとめる．全てのメールをスレッド化することで，メール単体の分析から，スレッドにまとまった会話を対象に分析に変更することができる．

スレッドとして分析することで，分析対象を減らすだけではなく，一通のメールだけの内容では，内容を判断できないようなメールであっても，スレッド単位での分析であれば，より細かい内容で分析することができる．

例えば，ある質問者がメーリングリストにおいて，対象ソフトウェアにおいてバグがあるとして質問をしたとする．しかし，返答者は，その質問に対して，それはバグではなく，使用方法の間違いであると返答したとすれば，質問者のメールは単体で見るとバグに関する話題となるが，実際に質問者が知りたかった内容は使用方法についてである．このようなメールは，スレッドでみることでやっと，会話の全容がみえてくる．

メールをスレッド化するために，メールの情報に含まれる “Message-Id” フィールドと “In-Reply-To” フィールド，“References” フィールドの情報を参照する．“Message-Id” フィールドには，メールごとに一意の ID が格納されている．例えば，<CAMkn0zbF@gee.mail.com> のような，ドメインと文字列の組み合わせで，メールの送信時に生成される．また，“In-Reply-To” フィールドには，返信元のメールの Message-Id が格納されている．“References” フィールドには，参照したメール全ての Message-Id が格納されており，返信元だけではなく，返信元の “In-Reply-To” フィールドの情報など，これまで参照した全てのメールの Message-Id を含んでいる．通常，メーラーなどは，この情報を元にメールをスレッド表示している．ここでは，“Message-Id” フィールドと “References” フィールドの情報を用いて，メールを返信元メールと関連づけることができ，メールをスレッド化する．

4. システム概要

ここでは，3 章で説明した問題に対する前処理を実施するシステムの概要を説明する．図 3 は，設計したシステムの概要図である．Java 言語を用いて，コード行数 1417 行で実装している．JavaSE-1.7 にて，動作を確認している．

表 1 オプション

オプション	処理内容
-f [file name]	入力ファイルを指定
-o [file name]	出力ファイルを指定
-a	別名の問題を解決
-d [file name]	開発者情報を付加
-s	引用部分を削除
-t	スレッド化
-ignore [address]	該当アドレスからのメールを削除

4.1 操作方法

システムの入力は，選択式で任意のオプションを指定する．入力として，メーリングリストデータのパスが必須となり，“-f” オプションを用いて指定する．また，“-o” オプションを用いて，出力ファイルを指定でき，指定がなければ，入力ファイル名の末尾に “-output” を加えたものを出力する．

主な，前処理オプションは，図 3 にある通りである．開発者を特定する “-d” オプションは，変更履歴を保存したファイルのパスを指定する．

システムの実行コマンドを表 1 に示す．実行コマンド：

```
java -jar MLMining.jar [option]
```

オプション：以上のオプションから，必要なものを選び実行する．

4.2 入力データ形式

メーリングリストデータは，Web アーカイブスなどにおいては，年単位で一つのテキストファイルや mbox ファイルとして保存されていることが多い．そこで，入力データ形式を，1 通以上のメールデータが含まれるファイル，もしくは，同様のテキストファイルを Zip 形式で圧縮したものとした．

4.3 抽出データ形式

メーリングリストデータから抽出した，1 通あたりのメッセージデータを保持するクラスのフィールドを図 4 に示す．メーリングリストデータを 1 通 1 通のメールに分割し，ExtractedMessage クラスで保持する．ExtractedMessage クラスでは，前処理に必要な情報をそれぞれフィールドに保持し，生のメールデータは message フィールドに保持している．

Id クラスは，メールから得られる参加者の情報やバージョン管理システムの変更履歴から得られる開発者の情報を保持する為のクラスで，アドレスと氏名を主な入力とし，入力された名前から，名字と名前を判定し保持する．名字

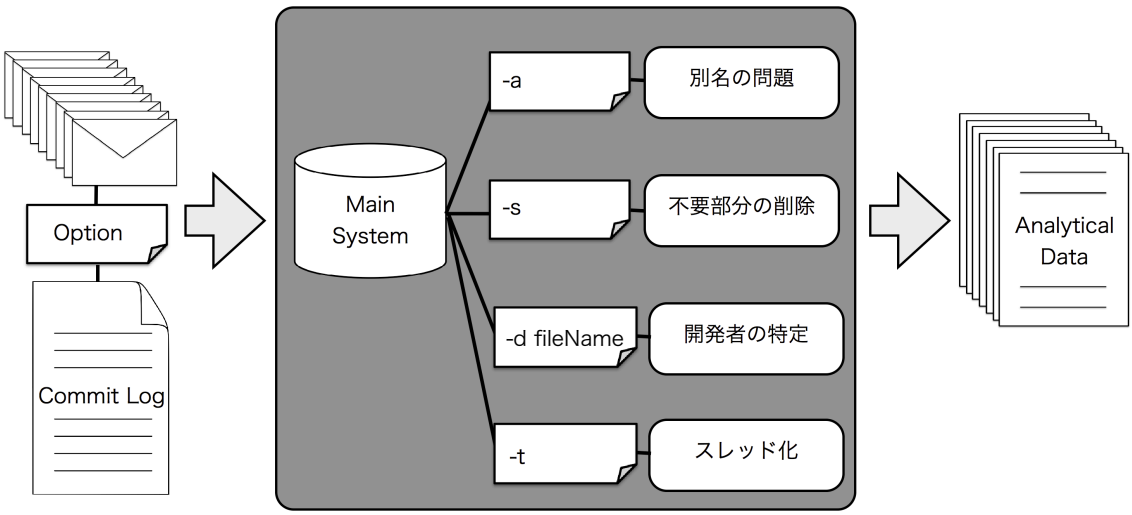

図 3　システム概要図

と名前の判定は，氏名から，Jr や Sr などの，名前に関する主な接尾辞を削除する．また，スペース区切りで 1 文字以下となる部分も削除し，残った部分をそれぞれ，名字と名前としている．

alias フィールドには，“-a” オプションが指定された場合に，その Id と同一人物と思われる Id の組から，1 つを代表とし，保持している．isDeveloper フィールドには，“-d” オプションが指定された場合に，開発者であるか否かを判定した結果を保持しており，オプションの指定がない場合は，false となっている．

```
class ExtractedMessage {
    String from;//送信者のアドレスと氏名
    String subject;//メールの件名
    String date;//送信日時
    String message;//本文
    String messageID;//メールに対して一意なID
    String inReplyTo;//返信元のmessageID
    Id id;//送信者情報を表すオブジェクト
}
class Id {
    String fullName;//フルネーム
    String address;//アドレス
    String firstName;//名前
    String familyName;//名字
    Id alias;//同一人物の別Idの代表
    boolean isDeveloper;//開発者であるかどうか
}
```

図 4　メッセージクラスフィールド

5. システムの評価

5.1 実施内容

システムの評価として，Apache Hadoop の開発メーリングリストに対しての実験を行った．入力は 2014 年 1 月分の Hadoop 開発メーリングリストを用いた．また，Hadoop-common リポジトリの変更履歴を用いて，メーリングリストの情報に開発者の情報を付加する．データセットの情報を表 2 に示す．

5.2 別名の問題の処理結果

別名の問題と開発者の特定の前処理を行った場合の結果を示す．実行には，以下のようなオプションを用いる．

実行オプション：

```
-f [input file] -o [output file] -d [commitlog file] -a
```

別名の問題を解決した場合の効果を表 3 に示す．なお，ここでのアカウントとはアドレスと氏名の組を表す．

表 2　データセット

	メーリングリスト	変更履歴
種類	common-dev[*1]	Hadoop-common[*2]
期間	2014 年 1 月	2009 年 5 月〜2014 年 1 月
量	282 通	8874 件

表 3　参加者データ

総アカウント数	255
参加者の数	25
別アカウントを使っていた人の数	5
最大アカウント数	159

[*1] http://mail-archives.apache.org/mod_mbox/hadoop-common-dev/201401.mbox
[*2] https://github.com/apache/hadoop-common.git

総アカウント数が 255 人存在する中で，実際の参加者は 25 人となった．これは，一部，同一人物でない組を誤って同一人物としてしまっている可能性も考えられる．

そこで，最も大きなアカウント数を使っているときにについて詳しく調べてみた．アカウントの各アドレスと名前の組を見てみると，159 のアカウントのほとんどは，JIRA（バグ報告ツール）からのものであった．対象プロジェクトにおいて，JIRA からメーリングリストへメールが送信されているが，その際，アドレスは JIRA のものであるが，氏名は様々な氏名を用いていることがわかった．これらの氏名が，別の個人のアドレスや氏名と類似している場合に，同一人物としている可能性がある．

つまり，JIRA からのメールは，分析前に取り除いておく，もしくは，JIRA の名前を使った類似の判定は，行わないようにすることが望ましい．

5.3 別名の問題と開発者の特定の効果

別名の問題と開発者の特定の二つの前処理の効果がわかる例を示す．図 5 は，入力データの例である． 図 5 の二つの入力ファイルにおいて，変更履歴ファイルの Author フィールドとメールデータの From フィールドが，それぞれ，アドレスと氏名の組の情報である．変更履歴では，“acmurthy@apache.org” というアドレスであるのに対して，メールデータでは，“acm@hortonworks.com” というアドレスとなっている．また，変更履歴では，“Arun Murthy” という氏名であるのに対して，メールデータでは，“Arun C Murthy” という氏名となっており，アドレスと氏名のどちらも異なる場合の開発者を特定する例である．

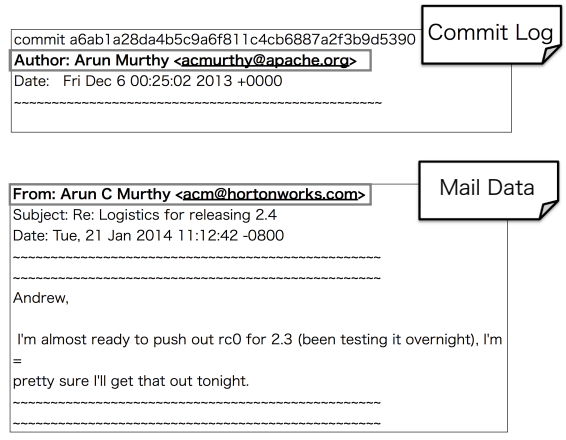

図 5 入力データ例

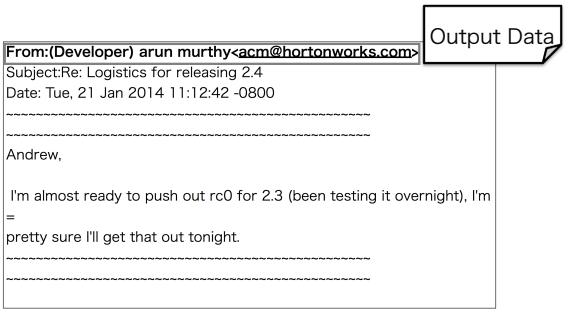

図 6 出力データ例

これらを用いて出力した結果が，図 6 である．出力データでは，氏名の前に開発者であれば，“(Developer)” という目印を付加しており，入力の二つの情報から開発者を特定したことを示した．

5.4 不要部分の削除の効果

引用部分の削除を行った場合の結果を示す．実行には，以下のようなオプションを用いる．

実行オプション：

```
-f [input file] -o [output file] -s
```

メーリングリストデータから，不要な引用部分を削除した場合，一ヶ月のメールの総行数 51579 行のメーリングリストデータから 14833 行の削除を行い，メーリングリストデータを内容の見やすいものへと再構成した．残ったデータを見ると，引用以外の他に，Jenkins や JIRA からの自動送信メールが多く含まれていることがわかった．

そこで，自動送信メールについて詳しく確認してみると，一ヶ月のメール総数 282 通に対し，Jenkins による自動送信メールが，29 通，JIRA による自動送信メールが 137 通であった．

更に，これらの自動送信メールを指定し削除を行った場合の結果を示す．実行には，以下のようなオプションを用いる．

実行オプション：

```
-f [input file] -o [output file] -ignore [address]
```

二つの前処理によって不要部分を取り除いたときの，データ量の変化を表 4 に示す．最終的な出力は 10173 行となり，会話に関係のない多くの不要部分を取り除くことができた．

表 4 データ量

	元データ	引用部分	自動送信メール	出力データ
行数	51579	14833	26573	10173
メール数	282	0	166	116

5.5 スレッド化の必要性

スレッド化を行った場合の結果を示す．実行には，以下のようなオプションを用いる．

実行オプション：

表 5　分析内容に対する前処理

オプション	-a	-d	-s	-t
開発者のメールの頻度 [1]	○	○	×	×
主なトピック [4]	×	×	○	○
開発者が注目するトピック [4]	○	○	○	○

-f [input file] -o [output file] -t

処理後のデータを確認すると，例えば，“Will there be a 2.2 patch releases?” というテーマで 6 通のメールが送られており，スレッドにまとめた．送られるメールのいくつかを確認してみると，“Nudge, any thoughts?” という 1 文のみの内容のメールもあった．一通のメールは，会話の一部であり，かならずしも，その内容のみで意味をなすとは限らない．どのような会話が行われているのかを分析する際は，一通のデータではなく，会話をまとめたスレッドが必要となる場合があることが確認できた．

5.6 分析に対する前処理

各前処理のどれを用いるかは，分析の種類によって異なる．本システムでは，分析したい内容に応じて，前処理を選ぶことを想定している．先行研究における分析例と，必要な前処理を表 5 示す．

表 5 の例の様に，一般的な開発メーリングリストの分析における前処理を体系化し，前処理システムとして網羅的に実行できることを示した．しかしながら，前処理の種類が少ないため，今後，システマティックレビュー [5] で，種類を増やしていく．

6. まとめ

本稿では，開発メーリングリストマイニングを行う上で，必要になる前処理をいくつかの先行研究をもとに調査し，それらで共通しているような，一般的な前処理を体系化し，実施するシステム開発を行った．

前処理システムでは，起動時にオプションコマンドとして実施する前処理を選択することができ，使用者は必要な前処理を選ぶことで，マイニングに必要なデータを構成することができる．

開発者の特定と別名の問題の前処理においては，アドレスと氏名のどちらも異なる場合であっても，開発者情報とメールデータをつなげた結果を示し，メール情報に開発者を付加することができることを示した．

また，メーリングリストのデータには，会話の内容を分析する際にノイズとなる，機械的に自動生成される部分がある．それらを削除することで，メーリングリストのデータを整理し，分析に必要な情報を抽出した．

そして，メールのデータは，一言の内容しか含んでいないようなものもある．このようなメールは，他のメールと同時に分析しない限り内容がわからず，スレッド化の意味がある．

体系化することで，これらの複数の前処理を容易に組み合わせることができ，少ない手順で一括して処理を行うシステムを構成した．結果，開発メーリングリストマイニングを行う上での前処理をシステムに任せることができ，マイニングの研究を行う場合に，その用途に応じて必要な処理のいくつかを，一括して行うことできるので，マイニングを行う研究者の負担を軽くすることが期待できる．

今後の課題として，Web 上などから簡単に利用できるユーザーインターフェスを作成することや，更に，文献を調査し，有用な前処理を調べ，追加することで選択できる前処理の幅を広げることが挙げられる．

謝辞　本研究の一部は，日本学術振興会科学研究費補助金（若手 A：課題番号 24680003）による助成を受けた．

参考文献

[1] Bird, C., Gourley, A., Devanbu, P., Gertz, M. and Swaminathan, A.: Mining email social networks, *Proc. the 3rd International Workshop on Mining Software Repositories* (2006).
[2] Bettenburg, N., Shihab, E. and Hassan, A. E.: An empirical study on the risks of using off-the-shelf techniques for processing mailing list data, *Proc. the 25th International Conference on Software Maintenance*, pp. 539–542 (2009).
[3] Bacchelli, A., Lanza, M. and D’Ambros, M.: Miler - a tool infrastructure to analyze mailing lists, *Proc. the 3rd International Workshop on FAMIX and Moose in Reengineering* (2009).
[4] Guzzi, A., Bacchelli, A., Lanza, M., Pinzger, M. and van Deursen, A.: Communication in Open Source Software Development Mailing Lists, *Proc. The 10th Working Conference on Mining Software Repositories*, pp. 277–286 (2013).
[5] Catal, C. and Diri, B.: A systematic review of software fault prediction studies., *Expert Syst. Appl.*, Vol. 36, No. 4, pp. 7346–7354 (2009).