修 士 論 文

題　目　　コピーアンドペーストを伴うソースコード
変更作業内容のトピックモデルを用いた調査

主任指導教員　　水野　修　教授

指導教員　　崔 恩瀞　助教

京都工芸繊維大学大学院 工芸科学研究科

情報工学専攻

学生番号　　18622039

氏　　名　　廣瀬　早都希

令和2年2月10日提出

様式 2 号

学位論文内容の要旨（和文）

令和 2 年 2 月 10 日

京都工芸繊維大学大学院
工芸科学研究科長　殿

工芸科学研究科　　情報工学専攻
平成 30 年入学
学生番号　　18622039
氏　　名　　廣瀬　早都希　㊞

（主任指導教員　　水野　修　㊞　）

本学学位規則第 4 条に基づき、下記のとおり学位論文内容の要旨を提出いたします。

1.　論文題目

コピーアンドペーストを伴うソースコード変更作業内容のトピックモデルを用いた調査

2.　論文内容の要旨（400 字程度）

ソフトウェア生産性の向上を目的として，ソフトウェア開発者はコピーアンドペーストによる既存のソースコードの再利用を頻繁に行う．しかし，再利用する元のソースコードに不具合が含まれていた場合，そのソースコードの再利用により，不具合の伝搬が発生する可能性がある．この問題を防ぐために，開発者のソースコード再利用作業の支援が必要であり，特に，開発者の作業内容に特化した支援が必要である．そこで，本研究では，開発者のコピーアンドペースト等による再利用がどのような作業中に発生するかを調査した．具体的には，まず，２つのプロジェクト開発履歴を分析して，ソースコードが再利用されたコミットを特定した．その後，特定されたコミットに対してトピック分析を適用し，ソースコードが再利用された作業内容を分析した．調査の結果，新たな機能等の追加の際にソースコードの再利用が頻繁に行われていることが明らかになった．

An analysis of source code modification work with copy and paste using topic model

2020 18622039 *HIROSE Satsuki*

Abstract

Software developers often reuse existing source code by copy and paste during software development in order to improve software productivity. However, if the original source code to be reused contains bugs, there is a possibility that the reused source code causes the propagation of bugs. It is necessary to support developers in the reuse of the source code to prevent this problem. In particular, it is necessary to provide specialized support for the context of each developer's work. In this study, we investigate the context of work when developers reuse existing source code. In more detail, we analyze the histories of two project development and identify the commits in which the source code was reused. We then apply the topic model to the commit messages of the identified commits. Finally, we analyzed the contexts of works when the source code was reused. The result of the investigation shows that the source code was often reused when new functions are added.

目　次

1. 緒言

近年，ソースコードが公開されているオープンソースソフトウェア等の普及によって，再利用できるソースコードの取得がしやすくなった．ソースコードの再利用は，開発効率や信頼性の向上につながるため，ソフトウェア開発でしばしば用いられている．ソースコードを再利用する際には，既存のソースコード片をコピーしペーストした後に、必要に応じてコピー先のコンテキストにより修正を行う．

しかし，ソースコードのコピーアンドペーストはソフトウェアの保守性を脅かす可能性がある [1], [2]．例えば，ソースコードをコピーする際に元のソースコードに不具合が含まれている場合，コピーしたソースコードにも不具合が潜んでいる可能性がある．そのため，不具合が含まれていた場合，元のコード片の修正だけでなく，複製先の全てのコード片の修正も必要となる．ソースコードのコピーを複数回行っていた場合，コピーされた全てのソースコードを修正することになり，困難である．

本研究では，ソフトウェアの保守作業を困難とするコピーアンドペーストによるソフトウェア開発がどのような作業と共に行われるかについて調査を行う．コードクローンが混入しやすい作業の特徴を掴むことで，その作業のコンテキストに基づいた支援ができる．本研究では，まず，コピーアンドペーストによるソースコードの複製を特定するために，コードクローン検出ツール [1], [3], を利用した．コードクローンとは，ソースコード中に存在する互いに類似または一致しているコード片のことである．コードクローンは，コピーアンドペーストによるソースコードの複製や，同一処理の繰り返し部分で発生する．今回はコードクローン可視化ツール CCEvovis[4], [5] を利用することで，ソースコードのコミットで新たにコードクローンが発生したかを調べた．CCEvovis は GitHub (注 1) のコミットから，コードクローンの変更情報を提示する．そこで，CCEvovis で新たにコードクローンが発生したコミットの情報を取得し，コミットのコミットメッセージを取得する．コミットメッセージは変更した作業の内容を記述するものなので，コミットの作業内容を要約しているものだと考えられる．

取得したコミットメッセージを対象とし，トピックモデル LDA[6], [7] を利用してトピック分析を行った．トピックモデルとは，自然言語テキストを分類する統計的

(注 1)：https://github.com

手法の 1 つである．これは，入力データの文章がそれを構成する単語の集合体であるとし，それらの単語は独立して出現するのではなく単語が潜在的に持つトピック (話題) に従い出現するという考えから行われる．トピックとは，文章の主題であり，その文章の中身を抽象化した概念的なものを示している．そのため，トピック分析により，入力データの詳細を確認することなく，その文章が持つ話題を推定することができる．これにより，新たにコードクローンが生成された際の作業内容の傾向を得ることができる．

本研究では，オープンソースプロジェクトである「Apache Ant」(注 2) プロジェクトと「Apache Maven」(注 3) プロジェクトの 2 つのプロジェクトを対象とし，調査を行った．具体的には，新たにコードクローンが生成されるコミットと，新たにコードクローンが生成されていないコミットのそれぞれのコミットメッセージを取得し，トピックの差異を調べた．その結果，ソースコードの変更がされるのは，不具合の修正などの保守作業や機能などの追加が多いが，新たにコードクローンの生成が行われる場合には，新たな機能が追加される場合が多いことが確認できた．

本論文の以降の構成を述べる．2 章では，本論文で使用したツール等の説明を述べ，3 章では，研究にあたり設定した研究設問を示す．4 章では，実際に行った調査について述べ，それに対する考察を 5 章で行う．6 章では，まとめを行う．

(注 2)：https://ant.apache.org
(注 3)：https://maven.apache.org

2. 背景

2.1 コードクローン

2.1.1 概要

コードクローンとはソースコード中に存在する互いに一致または類似したコード片のことである．また，互いにコードクローンとなっているコード片の集合のことをクローンセットという．コードクローンはソフトウェアの保守性を脅かす要因の 1 つと考えられている．しかし，ソフトウェア開発において，ソースコードの再利用が開発効率や信頼性の向上に役立つと言われている．そのため，開発者はコピーアンドペーストによる既存のソースコードの再利用を頻繁に行う．

コードクローンに厳密で普遍的な定義は存在しないが，Bellon[8] らによりコードクローンの違いの度合いで以下の 3 つに分類されている．

タイプ 1

空白とコメント行を除いた，改変なしの完全に一致したコードクローン

タイプ 2

タイプ 1 に加えて，変数名や関数名のみが変更されただけの，構文的に一致したコードクローン

タイプ 3

タイプ 2 に加えて，文の変更，挿入，削除がされた，変更が加えられたコードクローン

2.1.2 コードクローン関連ツール

コードクローン検出ツール

コードクローンの保守性を保つためには，ソフトウェア内の全てのコードクローン情報を管理する必要があるが，非常に手間がかかり現実的に困難である．そこで，コードクローンを自動的に検出するために様々なツールが提供されている．

コードクローンの検出技術 [1] は主に下記の 5 つに分類できる．

行単位のコードクローン検出

ソースコードを行単位で比較し，コードクローンを検出する．この検出技術は他の検出技術と比較して，高速に検出することが可能である．しかし，同じ処理のコードクローンであってもコーディングスタイルの差異によって検出ができなくなるデメリットがある．

字句単位のコードクローン検出

前処理としてソースコードを字句の列に変換し，閾値以上連続して一致している字句の部分列をコードクローンとして検出する．この検出技術はコーディングスタイルに依存せずにコードクローン検出ができ，また他の検出技術より高速に検出することができる．

抽象構文木によるコードクローン検出

前処理としてソースコードに対して構文解析を行い抽象構文木を構築し，抽象構文木上で一致する部分木がコードクローンとして検出される．そのため，コーディングスタイルに依存することはなく，ある関数定義の終わりから次の関数定義の先頭までの類似部分や文の途中からの類似部分などのプログラムの構造を無視した類似部分は検出されない．ただし，抽象構文木を構築するため，コードクローン検出に必要な時間的・空間的コストは高くなる傾向があり，またプログラム全体のコードクローンの割合を算出する場合など，検出されない部分があると不都合な場合には不向きである．

プログラム依存グラフを用いたコードクローン検出

前処理としてソースコードに対して意味解析を行うことで，ソースコードの文や式などの要素間の依存関係を抽出し，要素を頂点，依存関係を有向辺とするプログラム依存グラフを構築する．プログラム依存グラフ上の一致する部分グラフをコードクローンとして検出する．この検出技術では，他の検出技術では検出できないコードクローンも検出することができるが，プログラム依存グラフの構築にかかる計算コストが高いため，大規模なソフトウェアには適していない．

メトリクスを用いたコードクローン検出

プログラムのファイルやクラス，メソッド等のモジュール単位に対してメトリ

クスを計測し，メトリクスの値が一致もしくは近似している度合いを調べることで，そのモジュール単位でのコードクローンを検出できる．この検出技術では，メトリクスの値が近似しているとコードクローンとして検出されるため，2.1.1 節で述べたタイプ 3 のコードクローンも検出できる．しかし．サイズの小さいモジュールの場合，メトリクスの値に差がでないため誤検出の可能性が高まり，サイズの大きいモジュールの場合，モジュール内部の一部が類似していても検出されないことがある．

このようなコードクローン検出技術をコードクローンを扱う状況によって適宜選択する必要がある．

これらの検出技術を用いて，コードクローンの検出を行う様々なツールが提供されている．主なものとして，Nicad[9] や SourcererCC[10], CCFinder[11], CCVolti[12] といったコードクローン検出ツールが存在する．

2.1.3 コードクローン可視化ツール

コードクローンは一般的にソフトウェアの保守を困難にすると言われている．コードクローンに対する主な保守作業として，同時修正と集約が考えられる．同時修正とはクローンセットを一貫して編集することで，例えば，クローンセット中の１つのコード片に不具合があるため修正を行う場合，そのクローンセット中の他のコード片も同時に一貫した修正を行うことである．集約とはクローンセット中のコード片と同様の処理を実装する関数などを作り，各コード片をその関数の呼び出し文に置き換えることである．集約を行うことでソースコード中のコードクローンの数を削減することができ，ソフトウェアの保守につながると言われている．

コードクローンの保守作業を行うためには，ソフトウェア中のコードクローン全体を把握する必要がある．しかし，一般的にコードクローンの検出は膨大な数になりやすく，その中から，同時修正が行われていないコードクローンや新たに発生したコードクローンの変更履歴を人の目で確認することは困難である．そのため，コードクローンの保守作業を効率的に行うツールとして，Clone Notifier[13], [14] や CCEvovis[4], [5] といったコードクローン可視化ツールが提供されている．これらはソフトウェア内にあるクローンセットの情報をまとめて開発者に提供することで，保

守作業の 1 つである集約を行いやすくするものである．開発者は提供されたクローンセット情報から，そのクローンセットが集約可能であるか判断し，集約可能なクローンセットならば実際にそのクローンセットを集約することでソフトウェアの保守性を保つことができる．

Clone Notifier

コードクローンに対する保守作業を効率よく行うツールとして，コードクローン変化管理システム Clone Notifier[13], [14] が提案されている．Clone Notifier はソフトウェアの 2 バージョン間で行われたコードクローンの追加・編集・削除といったコードクローンの変更履歴情報に基づいて，クローンセットを分類し，その変更履歴情報を開発者に提供するものである．このシステムを用いてコードクローンの変更履歴を管理することで，保守作業の対象となるコードクローンの確認コストを削減することができる．

Clone Notifier の処理の流れは次のようになっている．

1. ソフトウェアの旧バージョンと新バージョンのプロジェクトを入力し，コードクローン検出ツールを用いて入力された 2 つのバージョンのコードクローンをそれぞれ検出する．
2. 2 つのバージョンのコードクローンのうち，同ファイル・同位置にあるコードクローンの対応を調査する．
3. 検出されたコードクローンの変更履歴に基づき，各コード片を追加・編集・削除のクローンセットに分類する．
4. 2 バージョン間におけるコードクローン変更履歴情報を過去の変更履歴情報として保存し，開発者にウェブユーザーインターフェースとして提供する．

Clone Notifier は，コードクローン変更履歴に基づいて 2 つのバージョン間のソースコードに含まれる全てのクローンセットを次の 4 つのクローンセットに分類する．

Stable **クローンセット**

コード片が 2 つのバージョン間にわたって存在し，変更がないクローンセットを示す．

Deleted **クローンセット**

コード片が新バージョンになく，旧バージョンのみに存在するクローンセットを示す．つまり，新バージョンで集約などによって削除されたクローンセットを意味する．したがって，開発者はDeletedクローンセットに分類されたクローンセットを確認することによって，集約が行われたコードクローンを確認することが可能である．

Changed クローンセット

コード片が2つのバージョン間にわたって存在し，変更されたクローンセットを示す．したがって，開発者はChangedクローンセットを確認することで，一貫した修正がされているかを確認することができ，一貫した修正がされていないならば同時修正を検討する必要がある．

New クローンセット

コード片が旧バージョンになく，新バージョンのみに存在するクローンセットを示す．つまり，新バージョンで新たに追加されたコードクローンが存在することを意味する．したがって，開発者はコードクローンの数を削減するために，Newクローンセットに分類されたクローンセットを確認し，集約を検討することができる．

Clone Notifierは検出結果を各クローンセット数の表形式で開発者に提供する．しかし，表形式での結果の提供では各クローンセット数の大小や比率を直観的に知ることができない．また，Clone Notifierでは，2つのバージョン間のコードクローンの変更履歴を調査することはできるが，過去から現在にわたる複数のバージョン間のコードクローン変更履歴情報の比較が困難である．Clone Notifierで複数のバージョン間のコードクローン変更履歴情報を比較する場合，2バージョンずつ調査を行い，開発者が各コードクローン変更履歴情報を逐次参照し比較する必要がある．この複数のコードクローン変更履歴情報の比較を表形式で行うと，過去から現在にわたって，各クローンセット数がどのくらい増減するかというクローンセット数の推移を比較することが困難である．このため，Clone Notifierで得られる表形式での結果は複数のコードクローン変更履歴情報の比較に適していない．

CCEvovis

CCEvovis はコードクローン変更履歴可視化システムであり，コードクローン変更管理システム Clone Notifier を拡張したものである．CCEvovis は，2 バージョン間のみしか検出できない Clone Notifier を拡張し，過去から現在にわたるコードクローンの変更履歴全体を分析できるようにしたもので，任意の隣接するバージョン間のクローンセットに関する統計データを集計することで，コードクローン変更履歴全体を可視化するシステムである．

CCEvovis では GitHub リポジトリを与えることで任意の期間のコードクローンの変更履歴を取得することができる．CCEvovis の流れ (図 2.1) を次に示す．

1. 検出したいプロジェクトの GitHub リポジトリと検出したい期間と日時間隔を与える設定ファイルを作成する．
2. 検出期間内のプロジェクトの情報を git リポジトリから取得する．
3. 取得したプロジェクトを時間軸に沿って，旧バージョン・新バージョンのプロジェクトとし，コードクローン検出ツールを用いて，入力された 2 バージョンのコードクローンを検出する．
4. 各コードクローンを変更履歴に基づいて Stable クローンセット, Deleted クローンセット, Changed クローンセット, New クローンセットに分類する．
5. 変更履歴全体を含めたクローンセットの変更履歴情報を可視化して開発者に提供する．

CCEvovis では以下の 3 つのコードクローン検出手法を選択し，利用できる．

SourcererCC

大規模なソフトウェアに対して，意味的に処理が類似したコードクローンを高速に検出するツール

CCFinderX

字句解析を用いることで，構文的に一致した字句単位のコードクローンを検出するツール

CCVolti

情報検索技術を用いることで，意味的に処理が類似したブロック単位 (関数単位より小さい粒度) のコードクローンを検出するツール．

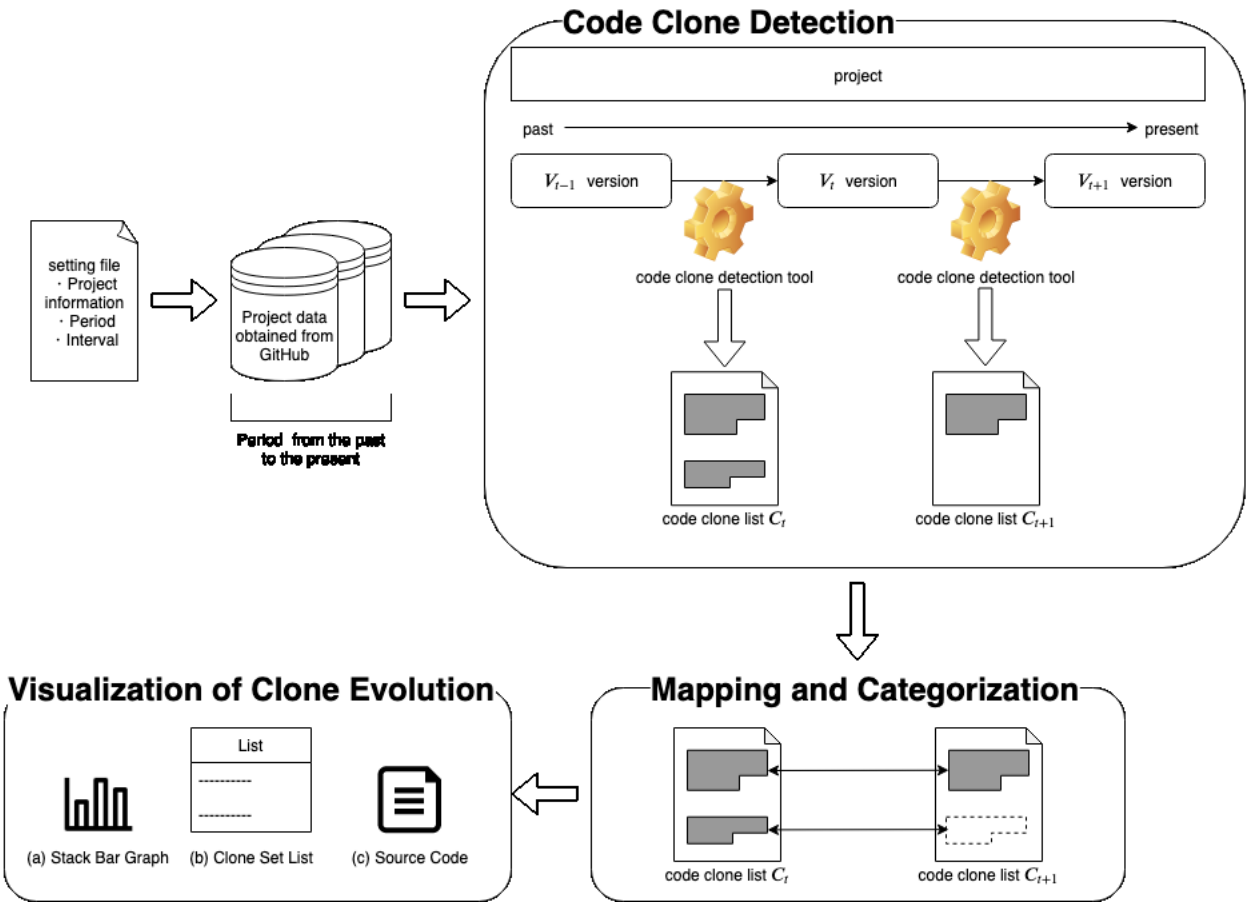


図 2.1　CCEvovis の処理の流れ

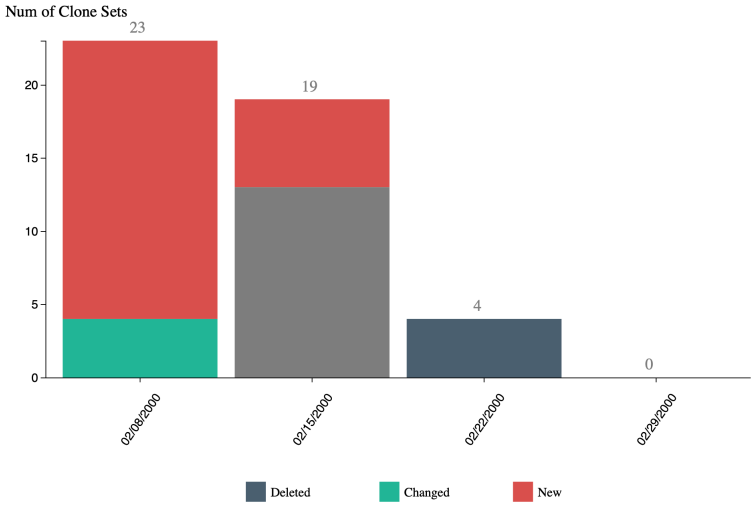
CCEvovis では，コードクローン変更履歴に基づいて，ソースコードに含まれる全てのクローンセットを Clone Notifier と同様の 4 種類のクローンセット (Stable クローンセット, Deleted クローンセット, Changed クローンセット, New クローンセット) に分類する．また，ウェブユーザーインターフェースと CSV ファイルでの情報提供を行う．ウェブユーザーインターフェースは，1. 変更履歴全体を可視化した積み上げ棒グラフページ (図 2.2a)，2. クローンセット一覧ページ (図 2.2b)，3. ソースファイルページ (図 2.2c) の 3 ページで構成される．

変更履歴全体を可視化した積み上げ棒グラフページ

クローンセットの変更履歴全体を含めた変更履歴情報を可視化した積み上げ棒グラフが表示されるページである．積み上げ棒グラフの横軸は調査した日付，縦軸はクローンセット数である．積み上げ棒グラフは上から順に New クローンセット (赤色)，Changed クローンセット (緑色)，Deleted クローンセット (黒色) の順で表示される．Stable クローンセットは，CCEvovis の目的である同時修正や集約の保守作業の支援にあまり関係がなく，提示する必要性が低いため提示していない．また，CCEvovis は 1 日や 1 週間単位と比較的短い期間での調査することを目的としており，短期間の分析においてはクローンセットの大半が Stable クローンセットに分類される．つまり，積み上げ棒グラフの要素のほとんどを Stable クローンセットが占めてしまう可能性が高まり，コードクローンの保守作業の支援に必要である New クローンセット，Changed クローンセット，Deleted クローンセットの情報を開発者が取得しづらくなるため，Stable クローンセットの表示がされていない．積み上げ棒グラフの上の数字は，New クローンセット・Changed クローンセット・Deleted クローンセットの合計の数を示す．積み上げ棒グラフの要素をクリックすると，クリックした要素の分析日のクローンセット一覧ページに移動することができる．

クローンセット一覧ページ

指定した分析日の 2 バージョン間に含まれるクローンセットの一覧が New クローンセット, Changed クローンセット, Deleted クローンセット, Stable クローンセットの順で表形式で提供されるページである．このページでは，各クローンセットに関して，そのクローンセットに属するコードクローンの一覧が表示



(a) 変更履歴全体を可視化した積み上げ棒グラフページ

NEW Clone Set

-

Clone Set ID : 4				
Link	ID	Classification	File name	Location
	4.0	ADDED	src\main\org\apache\tools\ant\DirectoryScanner.java	296.0-334.0
	4.1	ADDED	src\main\org\apache\tools\ant\DirectoryScanner.java	435.0-472.0

Clone Set ID : 5				
Link	ID	Classification	File name	Location
	5.0	ADDED	src\main\org\apache\tools\ant\taskdefs\Zip.java	254.0-258.0
	5.1	ADDED	src\main\org\apache\tools\ant\taskdefs\Zip.java	260.0-264.0

Clone Set ID : 6				
Link	ID	Classification	File name	Location
	6.0	ADDED	src\main\org\apache\tools\ant\DirectoryScanner.java	672.0-679.0
	6.1	ADDED	src\main\org\apache\tools\ant\DirectoryScanner.java	690.0-697.0

Clone Set ID : 7				
Link	ID	Classification	File name	Location
	7.0	ADDED	src\main\org\apache\tools\ant\taskdefs\Javadoc.java	575.0-586.0
	7.1	ADDED	src\main\org\apache\tools\ant\taskdefs\Javadoc.java	588.0-598.0

Clone Set ID : 8				
Link	ID	Classification	File name	Location
	8.0	ADDED	src\main\org\apache\tools\ant\DirectoryScanner.java	533.0-547.0
	8.1	ADDED	src\main\org\apache\tools\ant\DirectoryScanner.java	560.0-574.0

Clone Set ID : 9				
Link	ID	Classification	File name	Location
	9.0	ADDED	src\main\org\apache\tools\ant\taskdefs\Javadoc.java	576.0-585.0

(b) クローンセット一覧ページ

```
241 +
242 +     /**
243 +      * Sets whether default exclusions should be used or not.
244 +      *
245 +      * @param useDefaultExcludes "true" or "on" when default exclusions should
246 +      *                           be used, "false" or "off" when they
247 +      *                           shouldn't be used.
    -      * Executes the task.
248            */
249 +     public void setDefaultexcludes(String useDefaultExcludes) {
250 +         this.useDefaultExcludes = Project.toBoolean(useDefaultExcludes);
251 +     }
252
253 +     /**
254 +      * Executes the task.
255 +      */
256       public void execute() throws BuildException {
    -
257            // first off, make sure that we've got a srcdir and destdir
258
259 +          if (srcDir == null) {
260 +              throw new BuildException("srcdir attribute must be set!");
261 +          }
262 +          if (!srcDir.exists()) {
263 +              throw new BuildException("srcdir does not exist!");
264 +          }
265 +          if (destDir == null) {
266 +              throw new BuildException("destdir attribute must be set!");
    -          if (srcDir == null || destDir == null ) {
    -              String msg = "srcDir and destDir attributes must be set!";
    -              throw new BuildException(msg);
267            }
```

(c) ソースファイルページ

図 2.2　CCEvovis のウェブユーザーインターフェースでの出力例

される．各コードクローンはコードクローン ID, コードクローンの分類，そのコードクローンが含まれるソースファイル名，及びソースファイル中のコードクローンの位置 (行数) が表示される．また，表の左から 2 行目にあるソースコードのアイコンは，そのコードクローンが含まれるソースファイルページのリンクとなっているため，クリックすると対応するソースファイルページに移動できる．

ソースファイルページ

コードクローンが含まれるソースファイルが表示されるページである．このページでは，必要なコードクローンが見つけやすいように，コードクローンとなっているコード片の部分に黄色の背景色が付いている．また，新バージョンで追加された行の先頭に‘ + ’，新バージョンで削除された行の先頭に ’-‘ が表示されており，New クローンセットや Deleted クローンセット，Changed クローンセットの見分けがつきやすいようになっている．

2.2 トピック分析

(1) 概要

トピック分析とは自然言語テキストを分類する統計的手法の 1 つである．これは文章がそれを構成する単語の集合体であるとし，それらの単語は独立に出現するのではなく，各単語が潜在的に持つトピックに従って出現するものであると考えられている．トピックとは，文章の主題であり，その文章の中身を抽象化した概念的なものを示している．つまり，トピック分析を行うことで，文章の詳細を確認せずにその文章が持つ話題を知ることができる．トピック分析を行う際に必要となる，入力データの文章集合からトピックを確立的に抽出するモデルのことをトピックモデルといい，様々な手法が提案されている．

(2) トピックモデル

トピックモデルとは，入力データが持つ潜在的な意味から得られるトピックを抽出するモデルのことである．潜在的な意味とは，複数の単語の共起性によって得られる

情報のことを示している．トピックモデルにおいて，潜在的という言葉を用いられるのは，入力データに含まれる単語によって共起する単語が異なるため，入力データをトピック分析する時には，入力データ内に実際に現れる健在的共起性だけでなく，入力データ内に現れない隠れた共起性である潜在的共起性の両方を扱うことができるからである．主なトピックモデルとしては，Latent Semantic Index(LSI)[15], Probabilistic Latent Semantic Indexing(PLSI)[16]，Latent Dirichlet Allocation(LDA)[6], [7] 等がある．

Latent Semantic Index(LSI)

Latent Semantic Index(LSI) は，潜在的意味解析と言い，同じ意味をもつ単語を組み分けし，入力データ中の情報量を圧縮して，要点を浮き彫りにさせる手法である．LSI では，前処理として入力データを文章行列として表す必要がある．まず，入力データの文章を形態素解析器等を利用して単語に分割し，文章ベクトルに該当する単語が文章中にある場合はビットとして 1，それ以外は 0 のベクトルを生成する．このような文章ベクトルをまとめたものを文章行列とする．文章ベクトルを生成するには，入力データの全ての文章に存在する単語を全て網羅する必要があるため，文章ベクトルの次元数が大きくなる可能性があり，これは後の計算処理に影響を及ぼす．そこで，この文章ベクトルを意味の近いもの同士でまとめることで，文章ベクトルの次元数を圧縮するという考え方が LSI である．この文章ベクトルの圧縮には，行列分解の手法の一つである特異値分解 (SVD) を利用している．特異値分解は信号処理の分野や統計学等に主に用いられる手法で，これは行と列の形が揃っている正方行列だけでなく，全ての種類の行列を分解することができる．

特異値分解を数式で表すと次のようになる．

$$D = U \sum V^T \tag{2.1}$$

D を m 行 n 列，ランク r の任意の直方行列とすると，3 つの行列 $U, \sum, V$ に分解できるということである．ここで，U は $m \times r$ の列直行行，V は $n \times r$ の列直行行列で，$\sum$ は対角要素に特異値 σ_i を降順に並べた $r \times r$ の対角行列である．この式を重み行列 W を用いて表現すると，次のようになる．

$$D = UW(W = \sum V^T = [\sigma_1 v_1, \sigma_2 v_2, \cdots, \sigma_r v_r)]^T \tag{2.2}$$

特異値分解での次元削減は，特異値σの小さい値を削除すると良い．特異値σが大きければ重要度が高くなるため，特異値σが小さく重要でない軸を削除することがLSIでの次元削減手法である．トピック分析においては，重要な次元をトピック，この重要な次元の数をトピック数として抽出する．

つまり，LSIによる次元圧縮は，入力データから得られた文章ベクトルをそのままトピック分析を行う場合，単語数が多いと似た内容の文章からでも異なる文章ベクトルとして生成される可能性があるため，重要度の低いトピックを削ることで，重要度の高いトピックを厳選することができる．LSIの問題点としては，特異値分解の行列U, Vの値の意味付けが困難であり，負の値を取り得る可能性があることである．

Probabilistic Latent Semantic Indexing(PLSI)

Probabilistic Latent Semantic Indexing(PLSI)は，LSIを確率生成モデルとして考え直したものであり，式で表すと次のようになる．

$$p(d, w) = p(d) \sum_z p(w|z)p(z|d) \tag{2.3}$$

これは文章dの中にある単語wの共起確率を示している．$p(d)$は文章dの出現確率を示し，$p(z|d)$は文章dからトピックzが生成される確率を示している．また，$p(w|z)$は文章d中に出現する単語wのトピックzにおける出現確率を示している．この式中の$p(z|d)$をベイズの定理を用いて展開すると次のようになる．

$$p(d, w) = p(d) \sum_z p(w|z) \frac{p(d|z)p(z)}{p(d)} = \sum_z p(z)p(w|z)p(d|z) \tag{2.4}$$

この式中の$p(z)$はトピックzの存在確率を示し，$p(d|z)$は文章dがトピックzである確率を示す.

PLSIでは，学習データの尤度が最大となるように$p(z), p(w|z), p(d|z)$のパラメータをEMアルゴリズムによって推定する．EMアルゴリズムは，Expectationステップで期待値を最大化し，Maximumzationステップでその期待値を最大化するようなパラメータの選定をする手法である．

PLSIの特徴としては，文章ごとに複数のトピックを持つ可能性があるため，そのトピック数を事前に示さなければならない．つまり，トピック分析の際に使用するPLSIには，その文章中のトピック数を指定する必要がある．また，PLSIはトピック

ごとに異なる単語の生成分布を持ち得る．PLSI の問題点としては，トピック分布である $p(d|z)$ は学習データの文章 d に直接依存するため新規の文章を自然に扱うことができない点と，推定するパラメータの 1 つの $p(d|z)$ の数は文章数 $\times$ トピック数であり文章数が膨大になると推定すべきパラメータ数も比例して増加するため，学習データに対して簡単に過学習になってしまう点があげられる．

Latent Dirichlet Allocation(LDA)

Latent Dirichlet Allocation(LDA) は，PLSI の問題点を改善したトピックモデルである．PLSI ではトピック分布である $p(z|d)$ が学習データについてのみ定義されていることが問題だったため，$p(z|d)$ 自体を確率変数として生成すると，学習データに依存せずに文章数に応じてパラメータ数が増大することもなくなる．そこで，LDA では，このトピック分布の確率変数をディリクレ分布を用いて生成することで，PLSI の問題点に対処している．

PLSI は文章 d またはトピック z の確率分布を示しているが，LDA はこれを生成するための確率分布をベイズ推定により考える，これにより，LDA は文章 d に直接依存しないトピックの確率分布を推定できるため，PLSI の問題点である学習データにない，新規の文章を扱う事ができるようになる．LDA ではトピック推定の際に文章に依存しないため，文章 d の代わりに文章 d で使用される単語集合 $(w = w_1, w_2, \cdots, w_N)$ を利用する．トピックごとの単語の確率分布と文章ごとのトピックの確率分布は，確率分布の一つであるディレクトリ分布に従うものと仮定し，次のように定義する．ただしトピック数は与えることとする．

$$\text{各トピック毎に単語分布を生成} : \phi \sim p(\phi|\beta) \tag{2.5}$$

$$\text{各文章毎にトピック分布を生成} : \theta \sim p(\theta|\alpha) \tag{2.6}$$

$$\text{単語のトピックを生成} : z \sim p(z|\theta) \tag{2.7}$$

LDA は，これらの確率分布を用いて，各単語のトピックに該当する単語分布を選び，単語を生成する．

$$w \sim p(w|\phi_z) \tag{2.8}$$

このように LDA は，トピック分布を確率変数と生成するので，PLSI の問題点である新規の文章も自然に扱うことができ，また過学習もしづらくなっている．

(3) トピックモデルの可視化

LDAvis

LDAvis[注 4] は LDA の結果を可視化するライブラリで，図 2.3 のように表示される．左側部分は多次元尺度構成法 (MDS) を用いて各トピックを 2 次元の座標系にマッピングして，可視化したものである．図 2.3 の円はそれぞれ各トピックを示し，円の大きさが各トピックに属する単語数の合計を表している．また，円同士の距離は各トピック間の距離を示している．右側部分は頻出単語が表示されており，あるトピックを選択すると対応する左側部分の円が赤く表示され，右側部分に選択したトピックの頻出単語がその出現頻度とその割合が高い順に表示される．この出現頻度はパラメータ λ を変更することで同じトピック内での出現頻度の割合の比率を変えて順位の変更ができる．

$$saliency(term\, w) = freqency(w) * [sum\_t \frac{p(t|w)}{p(t)}]\ for\, topic\, t \tag{2.9}$$

$$relevance(term\, w|topic\, t) = \lambda * p(w|t) + (1-\lambda) * \frac{p(w|t)}{p(w)} \tag{2.10}$$

本研究では多次元尺度構成法の 1 つである Metric Multi-dimensional Scaling(MMDS) により可視化を行っている．MMDS は古典的 MDS の上位集合であり，数量的なデータを扱うため計量 MDS と言われる．比例尺度や間隔尺度等の定量的数値データを対象にして，個体の類似度を計算し最適な配置を算出するため，より厳密な距離関係を処理することに適している．

（注 4）：https://cran.r-project.org/web/packages/LDAvis/index.html

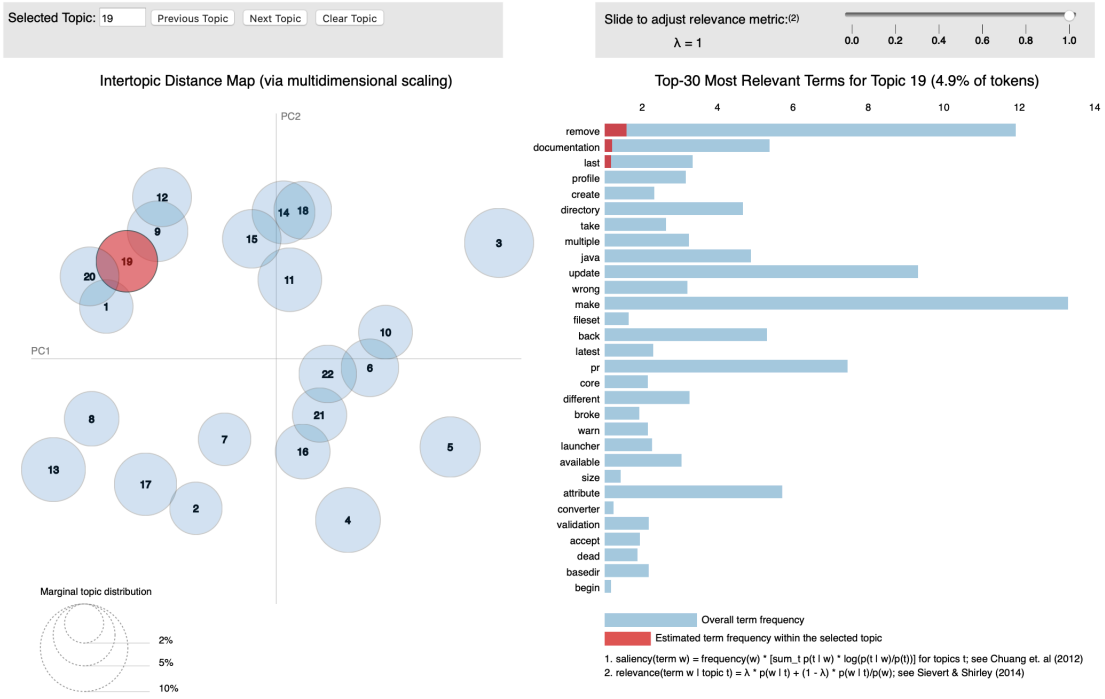


図 2.3　LDAvis の使用例

3. 研究設問

次の 2 つの研究設問を提示する．

RQ1 コードクローンが新たに生成された時の作業にはどのような特徴があるのか．

この研究設問では，コードクローンが新たに生成された時の作業の特徴を調べる．具体的には，コードクローン可視化ツール CCEvovis を用いて新たにコードクローンが生成されたコミットを取得し，そのコミットの作業内容をコミットメッセージのトピック分析を行った結果から，この作業の特徴を推定する．

RQ2 コードクローンが新たに生成された時の作業の特徴とその他の作業の特徴に違いはあるのか．

この研究設問では，RQ1 で得られる特徴と，新たにコードクローンが生成されていない時の作業の特徴を比較する．新たにコードクローンが生成されていない時の作業の特徴を得ることでソースコードが変更される作業全体の傾向を知ることができると考えられる．そのため，新たにコードクローンが生成された時の作業と生成されていない時の作業を比較することで，新たにコードクローンが生成された時の作業の特徴をより深く推定できると考えられる．

4. 調査手順と結果

4.1 対象プロジェクト

本研究ではコードクローン研究で用いられやすいオープンソースソフトウェアである Apache Ant プロジェクトと Apache Maven プロジェクトを対象としている．Apache Ant はソースコードのコンパイルからアプリケーションの配置までを Java の機能を用いて行うビルドツールである．Apache Maven は Java 用のプロジェクト管理ツールで，Apache Ant に代わるものとして作成されたものである．

この 2 つのプロジェクトを対象に CCEvovis を用いてコードクローン情報を取得する．表 4.1 に両プロジェクトから得られるデータをまとめる．

4.2 調査手順

本研究では次のような手順で調査を行った．

ステップ 1．コミットの特定： 新たにコードクローンが発生したコミットとそれ以外のコミットを特定する．

ステップ 2．前処理： ステップ 1 で特定されたコミットメッセージに対して前処理を行う．

ステップ 3．トピック分析： ステップ 2 で前処理されたコミットメッセージからトピックを抽出し，分析する．

ステップ 4．トピックの比較： 新たにコードクローンが生成したコミットとそれ以外のコミットのトピックを比較する．

以降，各ステップの詳細に関して説明する．

ステップ 1. **コミットの特定**

コードクローン可視化ツール CCEvovis を用いて調査対象のプロジェクトに含まれるコードクローン情報を取得する．CCEvovis はプロジェクトの GitHub の URL と検出期間，検出間隔を与えることで，バージョン間のコードクローン情報を検出する．例えば，検出期間を 2019/10/1〜2019/10/31 とし，検出間隔を

表 4.1　対象プロジェクト

	Apache Ant	Apache Maven
検出期間	2000/1/13	2003/9/1
	～2019/11/30	～2019/11/30
総コミット数	14,583	10,634
新たにコードクローンが発生したコミットの数	614(4.2%)	402(3.8%)
新たにコードクローンが発生しなかったコミットの数	13,969(95.8%)	10,232(96.2%)

7 日毎とすると，2019/10/1 と 2019/10/8, 2019/10/8 と 2019/10/15, 2019/10/15 と 2019/10/22, 2019/10/22 と 2019/10/29 のそれぞれのバージョン間のコードクローンを検出する．

今回調査対象となるプロジェクトは java で書かれているため検出言語は java とし，コードクローン検出ツールは SourcererCC[10] を利用する．検出期間は表 4.1 の通りで，検出間隔は 1 日毎とした．

CCEvovis は 2.1.3 節で説明したように，ウェブユーザーインターフェースにより直感的に結果を確認することができるが，今回は csv ファイルによる結果を参照する．2.1.3 節で説明した NEW クローンセットが検出されたコミットを新たにコードクローンが生成された時のコミットとする．また，NEW クローンセットが検出された時以外のコミットを新たにコードクローンが生成された時以外のコミットとする．

ステップ 2. **前処理**

ステップ 1 で特定したコミットから，コミットメッセージを取得し，トピックモデルの入力データとするための前処理を行う．コミットメッセージは変更時点の作業内容を要約していると考えられているため，コミットメッセージから変更時点の作業内容を推察することができる．したがって，本研究ではコミットメッセージを利用してトピック分析をすることで，新たにコードクローンが発生した時の作業内容の特徴を分析する．

得られたコミットメッセージはトピック分析ができるように前処理を行う．まず文章となっているコミットメッセージを単語に区切る．その後，ステミングを行い，記号とストップワードを取り除く．ストップワードとして定義した単語は you，we などの代名詞や and，of などの接続詞や前置詞が含まれ，これによりトピック分析の際に必要とされない単語が除去される．

前処理の終わった入力データから，トピックモデル LDA のモデリングで必要となる辞書とコーパスを作成する．辞書は単語とその単語の ID，その単語の出現回数をまとめたものである．今回のトピック分析では文章をベクトルに変換する手法で一般的に用いられる Bag-of-Words(BoW) を利用する．表 4.2 は辞書の例である．

表 4.2　辞書の例

word ID	単語	出現回数
12	access	9
86	bug	19
14	chang	57
...	...	...

辞書を作成した後，コーパスを作成する．本研究では，各単語の出現回数とその単語の ID から疎ベクトルを生成し，TF-IDF により重み付けしたベクトル表現に変換したコーパスを用いる．TF-IDF を利用することで，文章に含まれる単語の重要度を考慮するため，文章の特徴を捉えやすくなる．

ステップ 3. **トピック分析**

トピック分析を行う際に，トピック数は自動で決まるわけではないため，トピック分析を行う前に適切なトピック数を与える必要がある．今回はトピックモデルの評価指標として広く使われている Perplexity と Coherence[17], [18] を用いてトピック数の検討を行う．Perplexity はトピックモデルの予測性能を示す指標であり，次の式で表される．

$$Perplexity = exp\{-\frac{1}{N}\sum_{i=1}^{N} log_2 p(w_i|\theta)\} \tag{4.1}$$

ここで，N は出現単語数，$p(w_i|\theta)$ は周辺の単語に対するある単語 w_i の発生確率を示す．これは入力データを学習データとテストデータに分け，トピックモデルに従い学習データでパラメータ θ を求める．その後，テストデータを使い $p(w_i|\theta)$ が最大になるようにパラメータを更新する．これは，負の対数尤度であり，一般的にこの値が低い程よいモデルであると言われている．Coherence はトピックの品質を測る指標であり，抽出されたトピックが人にとって解釈しやすいかどうかを示す．これは，学習コーパスからトピックごとの単語間類似度の平均を対数条件付き確率により求める．トピック全体の Coherence が高いと良いトピックモデルであると考えられる．本研究では言語処理でよく使われるアルゴリズムを含むライブラリである Gensim で C_V という Coherence の最適化アルゴリズムを利用している．

トピックモデル LDA に与えるトピック数を検討した後，その値を与えた LDA のモデリングを行う．学習率は与えたコーパスから非対称事前分布を学習した値とした．LDA の学習ができた後，LDAvis により LDA の可視化を行った．

ステップ 4. **トピックの比較**

ステップ 3 で行ったトピック分析の結果に基づき，新たにコードクローンが生成された時のコミットメッセージのトピックとそれ以外の作業が行われたコミットメッセージのトピックを比較する．

4.3 調査結果

A) 新たにコードクローンが生成された時のトピック抽出結果

CCEvovis により NEW コードクローンが検出された時のコミットメッセージからトピック分析を行った結果を示す．トピック数は，図 4.1 により，Perplexity が低く，Coherence が高くなる値である 22 とする．また，図 4.2 はトピック分析の結果を LDAvis により可視化したものである．1 から 22 の番号がついた円がそれぞれトピックを表す．円同士が近いものは似た意味を持つトピックであるので，これらをグループとしてまとめたものを表 4.3 に示す．

これらのグループ毎に各トピックに含まれる単語を表 4.5 から表 4.13 にまとめる．ここで，太字となっている単語はトピックの特徴を表していると考えられる単語である．

B) 新たにコードクローンが生成された時以外のトピック抽出結果

CCEvovis により NEW コードクローンが検出された時以外のコミットメッセージからトピック分析を行った結果を示す．トピック数は，図 4.3 により，Perplexity が低く，Coherence が高くなる値である 20 とする．また，図 4.4 はトピック分析の結果を LDAvis により可視化したものである．1 から 20 の番号がついた円がそれぞれトピックを表す．円同士が近いものは似た意味を持つトピックであるので，これらをグループとしてまとめたものを表 4.4 に示す．

これらのグループ毎に各トピックに含まれる単語を表 4.14 から表 4.25 にまとめる．ここで，太字となっている単語はトピックの特徴を表していると考えられる単語である．

表 4.3　新たにコードクローンが生成された時のトピックグループと属するトピックの一覧

グループ G_1^A	トピック T_1^A，トピック T_9^A，トピック T_{12}^A，トピック T_{19}^A，トピック T_{20}^A
グループ G_2^A	トピック T_{11}^A，トピック T_{14}^A，トピック T_{15}^A，トピック T_{18}^A
グループ G_3^A	トピック T_3^A
グループ G_4^A	トピック T_6^A，トピック T_{10}^A，トピック T_{16}^A，トピック T_{21}^A，トピック T_{22}^A
グループ G_5^A	トピック T_5^A
グループ G_6^A	トピック T_4^A
グループ G_7^A	トピック T_7^A
グループ G_8^A	トピック T_2^A，トピック T_{17}^A
グループ G_9^A	トピック T_8^A，トピック T_{13}^A

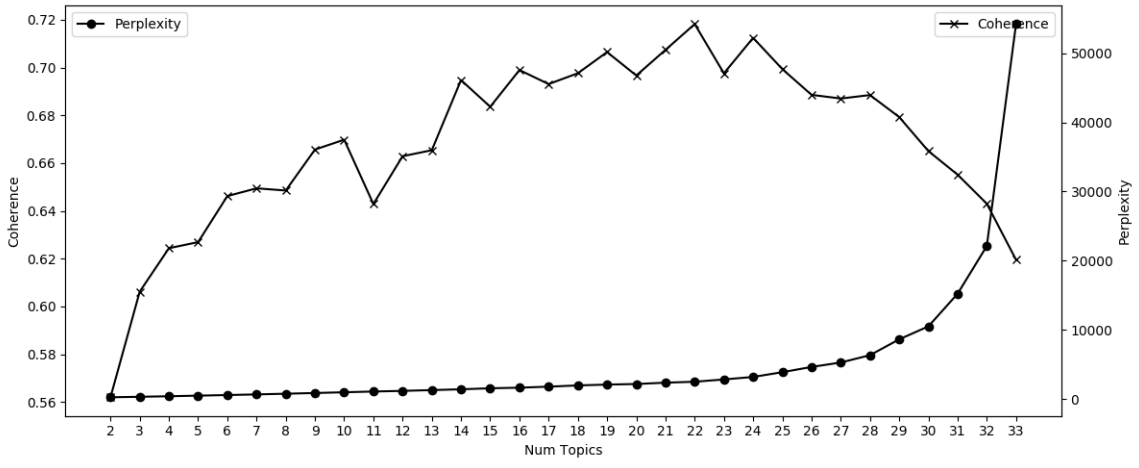


図 4.1　新たにコードクローンが生成された時のトピック数の検討

表 4.4　新たにコードクローンが生成された時以外のトピックグループと属するトピックの一覧

グループ G_1^B	トピック T_1^B，トピック T_{10}^B
グループ G_2^B	トピック T_7^B
グループ G_3^B	トピック T_3^B
グループ G_4^B	トピック T_9^B
グループ G_5^B	トピック T_{17}^B
グループ G_6^B	トピック T_4^B，トピック T_8^B，トピック T_{11}^B，トピック T_{14}^B，トピック T_{19}^B，トピック T_{20}^B
グループ G_7^B	トピック T_{13}^B，トピック T_{18}^B
グループ G_8^B	トピック T_6^B
グループ G_9^B	トピック T_2^B
グループ G_{10}^B	トピック T_5^B
グループ G_{11}^B	トピック T_{12}^B
グループ G_{12}^B	トピック T_{15}^B，トピック T_{16}^B

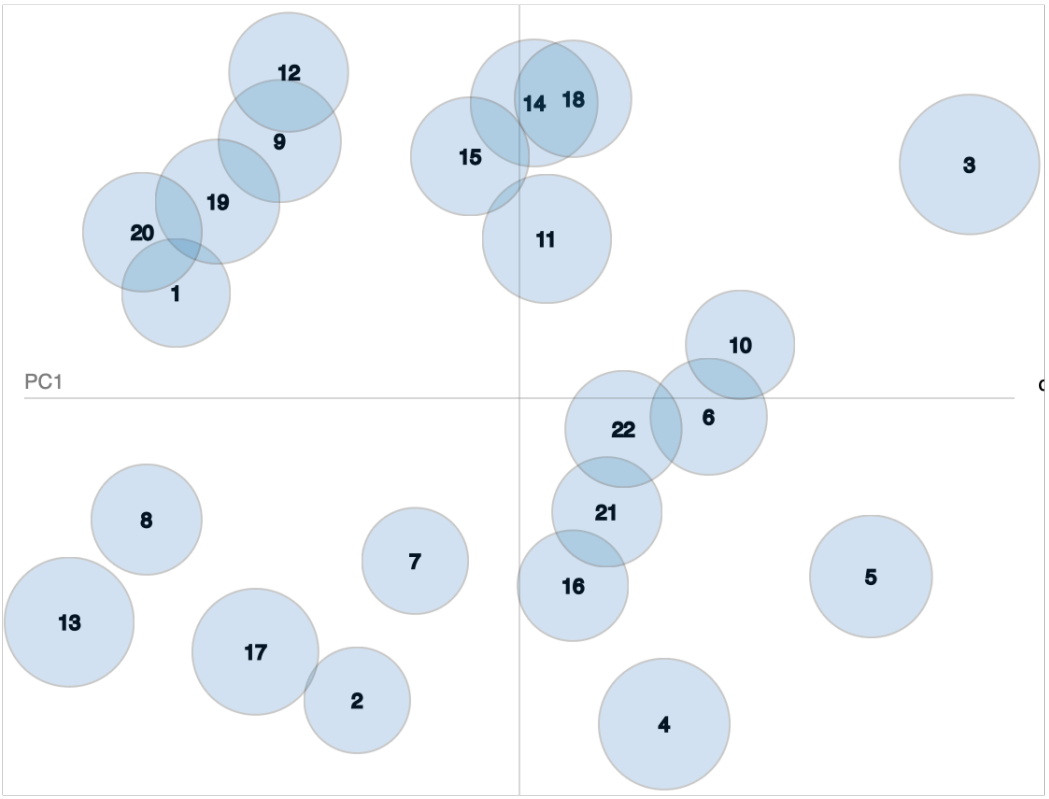


図 4.2　新たにコードクローンが生成された時のコミットメッセージをトピック分析した結果 (LDAvis)

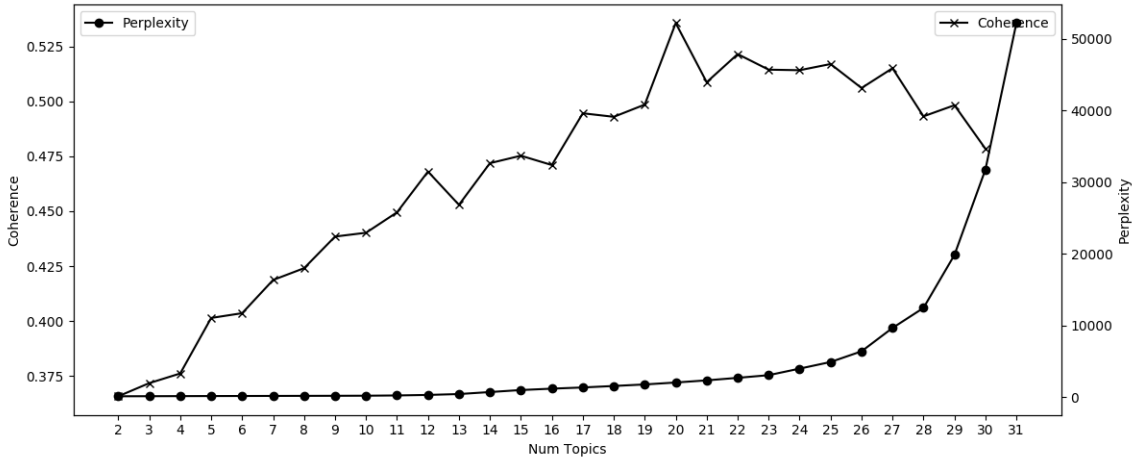


図 4.3　新たにコードクローンが生成された時以外のトピック数の検討

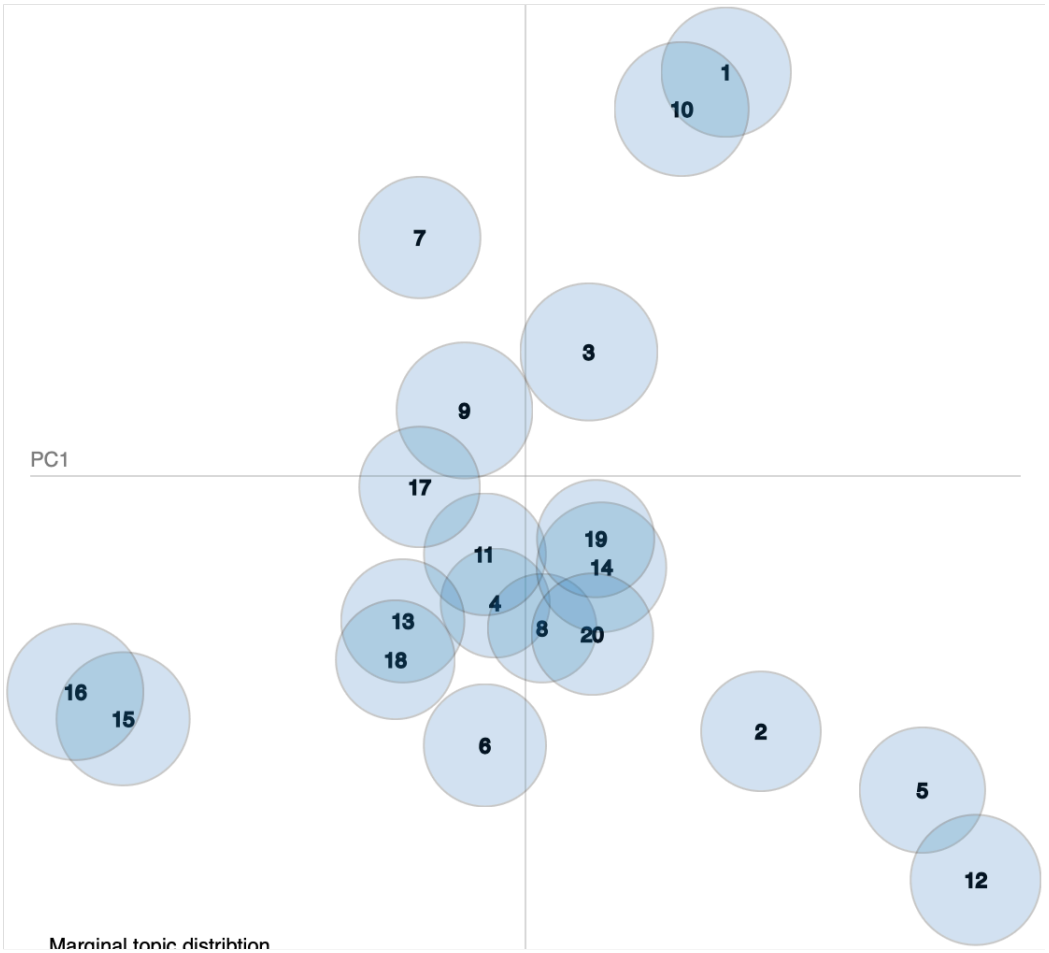


図 4.4　新たにコードクローンが生成された時以外のコミットメッセージをトピック分析した結果 (LDAvis)

表 4.5　グループ G_1^A に含まれるトピックとそのトピックに含まれる単語

トピック T_1^A	**import**, commit, **require**, get, **add**, attribute, task, **remove**, new, basedir, per, parent, part, set, configuration, testcase, **expand**, make, **unused**, **usage**, head, resource, without, **revert**, people, plugin, injection, outputfile, class, bug
トピック T_9^A	fix, move, **using**, plexus, enable, **api**, set, exception, better, correct, environment, **new**, **update**, test, **patch**, typo, consistent, xml, plugin, home, without, dependset, javacc, parent, special, file, adopt, checksum, mention, task
トピック T_{12}^A	**add**, **support**, **file**, ignore, usage, name, plugin, repository, shell, directoryscanner, post, broke, look, bugzilla, note, typo, message, ivy, case, checkstyle, code, care, use, **update**, complete, debug, example, task, attribute, allow
トピック T_{19}^A	**remove**, **documentation**, last
トピック T_{20}^A	pointer, make, java, stuff, back, different, task, compile, way, fix, class, **improve**, **convert**, parse, bug, two, post, easier, **add**, like, tree, todo, still, **option**, marmalade, description, better, bugzilla, hopedully, repository

表 4.6　グループ G_2^A に含まれるトピックとそのトピックに含まれる単語

トピック T_{11}^A	**fix**, allow, comment, **typo**, **version**, build, javadoc, vertifier, idea, context, text, branch, better, add, support, back, propagate, **debug**, message, example, directory, assembly, **release**, end, classloader, superpom, checkout, **error**, snapshot, use
トピック T_{14}^A	use, task, add, master, **file**, initial, **error**, remove, cache, **document**, type, **fix**, test, new, dependency, compare, include, ut, better, **merge**, plugin, basic, extract, embedder, update, definition, snapshot, time, flag, unit
トピック T_{15}^A	add, test, verbose, commit, **pr**, build, print, tar, run, pom, make, **integration**, **fix**, start, account, got, formatter, task, pointer, **option**, wljspc, current, bit, multiple, junit, method, another, artifact, fail, base
トピック T_{18}^A	**fix**, example, new, **error**, **improve**, npe, **remove**, **update**, **version**, move, support, optional, instead, two, use, magic, longer, **cleanup**, **release**, string, add, activation, allow, logo, seem, target, date, pass, bump, documentation

表 4.7　グループ G_3^A に含まれるトピックとそのトピックに含まれる単語

トピック T_3^A	add, fix, merge, longer, unused, pom, forgot, little, **update**, **website**, **provide**, **model**, pr, assume, branch, resource, **information**, unit, remove, structure, unit, remove, structure, directory, test, list, filter, bootstrap, need, check, license

表 4.8　グループ G_4^A に含まれるトピックとそのトピックに含まれる単語

トピック T_6^A	**change**, **import**, test, code, commit, add, javadoc, merge, branch, cleanup, update, **launcher**, model, help, checkstyle, selector, support, **patch**, jesse, similar, **resolution**, really, major, xalan, pom, **artifact**, collection, empty, generic, string
トピック T_{10}^A	merge, branch, **add**, minor, typo, snapshot, dependency, update, remove, **implementation**, cleanup, **previous**, **interface**, fix, possible, session, sun, **archive**, **manifest**, stuff, **package**, support, hierarchy, everything, due, progress, useless, revert, xsd, header
トピック T_{16}^A	**add**, whitespace, **build**, branch, merge, article, **new**, tim, vernum, **implementing**, upgrade, checkstyle, memory, typo, pointer, yet, metadata, force, ide, allow, move, couple, trailing, tag, **method**, even, wagon, need, read, make
トピック T_{21}^A	**version**, branch, merge, add, **property**, pr, **sync**, zip, tree, defult, fix, exist, another, target, code, **configure**, javadoc, line, beta, **source**, length, read, initial, prepare, generation, new, master, note, range requirement
トピック T_{22}^A	add, **style**, merge, unit, release, **javadoc**, well, family, root, try, **old**, **original**, **deprecate**, magic, whitespace, bugzilla, test, need, catch, pointer, use, mapper, deps, remove, text, module, **specification**, share, instead, small

表 4.9　グループ G_5^A に含まれるトピックとそのトピックに含まれる単語

トピック T_5^A	**format**, update, merge, jesse, branch, remove, tidy, date, forgot, need, commit, check, glick, execute, manager, jon, generation, avoid, whatsnew, **integration**, correct, **another**, **method**, code, mojo, csv, dependency, escape, header, main

表 4.10　グループ G_6^A に含まれるトピックとそのトピックに含まれる単語

トピック T_4^A	merge, branch, **try**, **project**, make, code, use, **mercury**, using, **rule**, rather, **html**, **save**, checkstyle, **jdk**, **new**, avoid, filter, enable, **repo**, **pr**, put, add, note, npe, **urlresource**, problem, **stream**, deoendency, mapper

表 4.11　グループ G_7^A に含まれるトピックとそのトピックに含まれる単語

トピック T_7^A	**documentation**, main, go, merge, **unit**, **metadata**, bad, line, **style**, **check**, test, empty, failure, allow, notifier, **simplify**, **proposal**, prevent, presence, make, management, interpolation, conor, macneill, gump, **junit**, buildfile, seem, java, log

表 4.12　グループ G_8^A に含まれるトピックとそのトピックに含まれる単語

トピック T_2^A	regenerate, jdk, back, make, merge, release, booystrap, access, latest, exist, **clean**, per, happen, empty, attribute, translate, builder, expand, branch, **refactor**, script, remove, copy, filter, cut, byte, notion, descriptor, document, repo
トピック T_{17}^A	**task**, say, **plugin**

表 4.13　グループ G_9^A に含まれるトピックとそのトピックに含まれる単語

トピック T_8^A	**work**, new, image, **problem**, **junit**, well
トピック T_{13}^A	make, order, **mutant**, **sure**

表 4.14　グループ G_1^B に含まれるトピックとそのトピックに含まれる単語

トピック T_1^B	**add**, **test**, stuff, jar, option, **fail**, quiet, split, result, bump, html, new, except, apt, eol, detect, year, faster, assert, bugzilla, run, path, correct, get, **file**, fixcrif, work, name, support, avoid
トピック T_{10}^B	**artifact**, default, **script**, allow, snapshot, **unit**, **pass**, **target**, yet, logic, **test**, new, throw, handler, make, know, attempt, long, key, scanner, use, true, add, meant, build, reason, stream, file, monitor, contributor

表 4.15　グループ G_2^B に含まれるトピックとそのトピックに含まれる単語

トピック T_7^B	**file**, case, **avoid**, note, **must**, build, turn, **return**, find, **never**, test, use, step, particular, **feedback**, catch, skip, thrown, recent, safe, **manual**, echo, **symlink**, new, speed, markup, need, dest, task, add

表 4.16　グループ G_3^B に含まれるトピックとそのトピックに含まれる単語

トピック T_3^B	new, **debug**, **pom**, report, **reactor**, comment, **line**, due, **command**, right, **wrong**, forgot, mail, **api**, prevent, easier, setup, final, **object**, load, print, use, input, get, favour, useless, argument, open, file, number

表 4.17　グループ G_4^B に含まれるトピックとそのトピックに含まれる単語

トピック T_9^B	**plugin**, **version**, need, repo, download, way, reflect, later, push, proper, use, happen, **refactor**, slient, far, see, new, central, depend, temp, job, appear, suppress, add, alias, overridden, sub, must, test

表 4.18　グループ G_5^B に含まれるトピックとそのトピックに含まれる単語

トピック T_{17}^B	build, set, core, tag, basedir, show, filter, mvn, manifest, look, even, **adjust**, **prefix**, **conflict**, var, use, zip, constant, tweak, project, task, antlib, timeout, add, kaunch, file, **fix**, name, problem, connot

表 4.19　グループ G_6^B に含まれるトピックとそのトピックに含まれる単語

トピック T_4^B	list, type, **problem**, **broken**, go, section, root, checkout, basic, transfer, index, bunch, **break**, extract, second, array, use, size, add, git, name, clearer, method, macrodef, play, task, rather, oh, word, avoid
トピック T_8^B	**typo**, commit, within, strip, via, **incorrect**, idea, given, normal, **broke**, filesset, various, parallel, trim, setter, append, use, macro, magic, adapt, new, **fix**, task, affect, gump, master, test, mix, sign, kind
トピック T_{11}^B	match, one, move, **null**, longer, **bug**, bit, whole, **junit**, account, think, convert, custom, constructor, contain, pattern, lot, global, use, **test**, allow, cast, unknown, processor, mention, disk, **fix**, instead, multi, mark
トピック T_{14}^B	next, make, **sure**, first, bootstrap, system, format, **upload**, **implement**, process, read, checksum, whether, timestamp, similar, **get**, use, tree, **save**, twice, work, fine, file, new, platform, still, understand, even, effect, linux, **problem**, **fix**
トピック T_{19}^B	**model**, **project**, place, found, **give**, minor, point, **call**, trunk, cannot, **static**, **new**, bound, state, trace, guard, web, **function**, branch, stack, dev, **code**, **error**, hide, one, hidden, wrt, great, use, frame
トピック T_{20}^B	**parent**, want, **class** , time, still, last, rest, **previous**, extra, full, real, xml, date, left, one, either, special, **clear**, space, flush, slash, say, made, regexp, **test**, commit, selector, author, hope, use

表 4.20　グループ G_7^B に含まれるトピックとそのトピックに含まれる単語

トピック T_{13}^B	**method**, like, put, without, rather, current, part, level, **page**, per, **string**, close, use, loop, generic, non, server, occur, respect, exec, task, stray, tar, add, rmic, miss, class, fix, bigger, extra
トピック T_{18}^B	local, old, access, sync, two, might, well, **document**, java, use, work, **text**, least, **content**, extend, less, **written**, perform, cleaner, unless, hold, ejbjar, length, get, one, make, still, byte, robust, **file**

表 4.21　グループ G_8^B に含まれるトピックとそのトピックに含まれる単語

トピック T_6^B	**path**, get, **name**, **javadoc**, patch, accept, take, keep, **expand**, host, **dir**, behaviour, behavior, **address**, follow, workaround, explain, use, fact, caught, error, leak, **destdir** , four, race, ago, fix, task, random, make

表 4.22　グループ G_9^B に含まれるトピックとそのトピックに含まれる単語)

トピック T_2^B	**clean**, work, site, **check**, **link**, **svn**, order, switch, end, pointer, **doc**, make, got, actual, older, warn, **fix**, jvm, port, commit, run, **error**, need, add, broken, **manual**, see, accident, **anchor**

表 4.23　グループ G_{10}^{B} に含まれるトピックとそのトピックに含まれる単語

トピック T_{5}^{B}	**fix**, run, **cleanup**, exist, **bad**, help, **syntax**, mode, id, public, **copyright**, revert, small, send, fork, align, tool, statement, certain, known, prompt, mapper, forget, ramdom, conform, make, equal, add, bit, **error**

表 4.24　グループ G_{11}^{B} に含まれるトピックとそのトピックに含まれる単語

トピック T_{12}^{B}	**log**, element, support, import, **branch**, **request**, **pull**, made, main, unix, today, header, present, number, **logger**, status, javac, publish, comparison, side, stylesheet, use, overwritten, lead, task, news, week, build, new, **git**

表 4.25　グループ G_{12}^{B} に含まれるトピックとそのトピックに含まれる単語

トピック T_{15}^{B}	**brack**, correct, **classpath**, explicit, instead, start, better, **url**, latest, scope, **use**, style, much, field, come, good, shell, flag, creation, layout, control, **code**, valid, exit, thing, like, upon approach, rule, three
トピック T_{16}^{B}	**use**, **error**, **code**, output, user, task, standard, deal, common, enough, built, write, **ibiblio**, gone, finish, scan, hard, **hint**, touch, lost, env, ssh, raw, strict, getter, grammar, join, spurious, regular, signjar

5. 考察

5.1 分析されたトピックに関する考察

ここでは，トピック分析の結果を元に，各トピックが示す特徴の考察を行う．

A) 新たにコードクローンが生成された時のトピック分析

新たにコードクローンが生成された時のトピック分析を行う．表 4.3 で記載しているグループ G_1^A〜G_9^A のそれぞれについて，各グループに含まれているトピックの単語から各グループが示す特徴の考察を行う．

グループ G_1^A

グループ G_1^A に含まれる単語は表 4.5 に示している．この表からわかるように，G_1^A にはトピック T_1^A，トピック T_9^A，トピック T_{12}^A，トピック T_{19}^A，トピック T_{20}^A が含まれる．G_1^A のトピックに共通する特徴は「add($T_1^A, T_{12}^A, T_{20}^a$)」「remove($T_1^A, T_{19}^A$)」「improve($T_{20}^A$)」「update($T_9^A, T_{12}^A$)」「convert($T_{20}^A$)」といった追加や削除, 改善に関する単語と,「api(T_9^A,)」「file(T_{12}^A)」「documentation(T_{19}^A)」「patch(T_9^A)」「option(T_{20}^A)」といったプログラムの機能に関する単語が考えられる．これらからプログラムの機能を新しいものに置き換えたりして改善をしたことを示すコミットメッセージが含まれると考えられる．

グループ G_2^A

グループ G_2^A に含まれる単語は表 4.6 に示している．この表からわかるように，G_2^A にはトピック T_{11}^A，トピック T_{14}^A，トピック T_{15}^A，トピック T_{18}^A が含まれる．G_2^A のトピックに共通する特徴は「typo(T_{11}^A)」「error($T_{11}^A, T_{14}^A, T_{18}^A$)」「fix($T_{11}^A, T_{14}^A, T_{15}^A, T_{18}^A$)」「debug($T_{11}^A$)」「cleanup($T_{18}^A$)」「improve($T_{18}^A$)」といった不具合とその改善に関する単語と,「pr(pull request)(T_{15}^A)」「merge(T_{14}^A)」「apdate(T_{18}^A)」「integration(T_{15}^A)」「version(T_{18}^A)」といったマージの要求に関する単語が考えられる．これらから不具合を改善したデータをマージするように要求に関するコミットメッセージが含まれると考えられる．

グループ G_3^A

グループ G_3^A に含まれる単語は表 4.7 に示している．この表からわかるよう

に，G_3^A にはトピック T_3^A が含まれる．T_3^A の特徴的な単語は「website」「model」「provide」「update」といった単語である．これらから website や model の提供やアップデートに関するコミットメッセージが含まれると考えられる．

グループ G_4^A

グループ G_4^A に含まれる単語は表 4.8 に示している．この表からわかるように，G_4^A にはトピック T_6^A，トピック T_{10}^A，トピック T_{16}^A，トピック T_{21}^A，トピック T_{22}^A が含まれる．G_4^A のトピックに共通する特徴は「change(T_6^A)」「import(T_6^A)」「add(T_{10}^A, T_{16}^A)」「new(T_{16}^A)」「version(T_{21}^A)」「sync(T_{21}^A)」「implemennatation(T_{10}^A)」「implementing(T_{16}^A)」といった変更や追加，実装に関する単語と,「launcher(T_6^A)」「artifact(T_6^A)」「interface(T_{10}^A)」「archive(T_{10}^A)」「manifest(T_{10}^A)」「package(T_{10}^A)」「method(T_{16}^A)」「source(T_{21}^A)」「style(T_{22}^A)」「specification(T_{22}^A)」「javadoc(T_{22}^A)」といったシステムの環境などに関する単語が考えられる．これらからシステム環境に対する変更や追加等に関するコミットメッセージが含まれると考えられる．

グループ G_5^A

グループ G_5^A に含まれる単語は表 4.9 に示している．この表からわかるように，G_5^A にはトピック T_5^A が含まれる．T_5^A の特徴的な単語は「format」「another」「method」「integration」といった単語である．これらからフォーマットやメソッドの統合に関するコミットメッセージが含まれると考えられる．

グループ G_6^A

グループ G_6^A に含まれる単語は表 4.10 に示している．この表からわかるように，G_6^A にはトピック T_4^A が含まれる．T_4^A の特徴的な単語は「project」「try」「save」「new」「rule」「html」「jdk」「urlresource」といった単語である．これらから web 系の新たな機能やルールの適用に関するコミットメッセージが含まれると考えられる．

グループ G_7^A

グループ G_7^A に含まれる単語は表 4.11 に示している．この表からわかるように，G_7^A にはトピック T_7^A が含まれる．T_7^A の特徴的な単語は「documentation」「unit」「metadata」「style」「junit」「check」「simplify」「proposal」といった単語である．これらからドキュメントやデータのチェックと簡素化の提案に関するコミット

メッセージが含まれると考えられる．

グループ G_8^A

グループ G_8^A に含まれる単語は表 4.12 に示している．この表からわかるように，G_8^A にはトピック T_2^A，トピック T_{17}^A が含まれる．G_8^A のトピックに共通する特徴は「plugin(T_{17}^A)」「task(T_{17}^A)」といったプラグインに関する単語と，「clean(T_2^A)」「refactor(T_2^A)」といったリファクタリングなどに関する単語が考えられる．これらからプラグインのリファクタリングに関するコミットメッセージが含まれると考えられる．

グループ G_9^A

グループ G_9^A に含まれる単語は表 4.13 に示している．この表からわかるように，G_9^A にはトピック T_8^A，トピック T_{13}^A が含まれる．G_9^A のトピックに共通する特徴は「junit(T_8^A)」「mutant(T_{13}^A)」といったテストに関する単語と，「problem(T_8^A)」「sure(T_{13}^A)」「work(T_8^A)」といったテストにより改善することを想起させる単語が考えられる．これらからテスト作業に関するコミットメッセージが含まれると考えられる．

B) 新たにコードクローンが生成された時以外のトピック分析

新たにコードクローンが生成された時以外のトピック分析を行う．表 4.4 で記載しているグループ G_1^B〜G_{12}^B のそれぞれについて，各グループに含まれているトピックの単語から各グループが示す特徴の考察を行う．

グループ G_1^B

グループ G_1^B に含まれる単語は表 4.14 に示している．この表からわかるように，G_1^B にはトピック T_1^B，トピック T_{10}^B が含まれる．G_1^B のトピックに共通する特徴は「test(T_1^B, T_{10}^B)」「unit(T_{10}^B)」「fail(T_1^B)」「pass(T_{10}^B)」といったテストに関する単語と，「file(T_1^B)」「target(T_{10}^B)」「artifact(T_{10}^B)」「script(T_{10}^B)」といったファイルやスクリプトに関する単語が考えられる．これらからテストに関連するファイルやスクリプトを追加等に関するコミットメッセージが含まれると考えられる．

グループ G_2^B

グループ G_2^B に含まれる単語は表 4.15 に示している．この表からわかるように，

G_2^B にはトピック T_7^B が含まれる．T_7^B の特徴的な単語は「rerturn」「feedback」「symlink」「manual」「avoid」「never」「must」といった単語である．このうち，「avoid」「never」「must」があることで何か強く避けたいものがあるように想起される．また,「rerturn」「feedback」「symlink」「manual」から何らかの手段や結果を返すことが想起される．これらから何らかの問題に対する解決策に関するコミットメッセージが含まれると考えられる．

グループ G_3^B

グループ G_3^B に含まれる単語は表 4.16 に示している．この表からわかるように，G_3^B にはトピック T_3^B が含まれる．T_3^B の特徴的な単語は「debug」「pom」「wrong」「api」「object」「reactor」「command」「line」といった単語である．このうち,「reactor」はプロジェクト maven の機能の一つであると「command」「line」といった関連単語が見受けられることから考えられる．また,「pom」もプロジェクト maven に関連する単語であることから，このトピックはプロジェクト maven に関するコミットメッセージが含まれていると考えられる．

グループ G_4^B

グループ G_4^B に含まれる単語は表 4.17 に示している．この表からわかるように，G_4^B にはトピック T_9^B が含まれる．T_9^B の特徴的な単語は「plugin」「version」「refactor」といった単語である．これらからプロジェクトで使用しているプラグインの修正に関するコミットメッセージが含まれると考えられる．

グループ G_5^B

グループ G_5^B に含まれる単語は表 4.18 に示している．この表からわかるように，G_5^B にはトピック T_{17}^B が含まれる．T_{17}^B の特徴的な単語は「conflict」「fix」「prefix」「adjust」といった単語である．これらからコンフリクトを引き起こしたこととそれに対する修正や調整に関するコミットメッセージが含まれると考えられる．

グループ G_6^B

グループ G_6^B に含まれる単語は表 4.19 に示している．この表からわかるように，G_6^B にはトピック T_4^B，トピック T_8^B，トピック T_{11}^B，トピック T_{14}^B，トピック T_{19}^B，トピック T_{20}^B が含まれる．G_6^B のトピックに共通する特徴は「broke(T_4^B, T_8^B)」「problem(T_4^B, T_{14}^B)」「break(T_4^B)」「typo(T_8^B)」「incorrect(T_8^B)」

「fix(T_8^B, T_{11}^B)」「bug(T_{11}^B)」「sure(T_{14}^B)」「error(T_{19}^B)」「clear(T_{20}^B)」といった不具合に関する単語が考えられる．これらからこのグループに共通する特徴は不具合やその修正に関するコミットメッセージが含まれると考えられる．

LDAvis によるトピックの可視化は近い距離にあるトピック同士は基本的に類似している部分があるためグループ化してトピックの内容を推定しているが，入力データが多いため，類似点が薄い場合がある．このグループは上記のような類似点は見受けられるが，各トピックに固有の特徴があると考えられる．次に各トピックの特徴を挙げる．

トピック T_4^B

特徴的な単語は「broken」「problem」「break」といった単語である．これらから不具合や問題が含まれていることを示唆しているコミットメッセージが含まれると考えられる．

トピック T_8^B

特徴的な単語は「typo」「incorrect」「broke」「fix」といった単語である．これらから不具合の修正に関するコミットメッセージが含まれると考えられる．

トピック T_{11}^B

特徴的な単語は「null」「bug」「junit」「fix」「test」といった単語である．これらからテストのバグ修正に関するコミットメッセージが含まれると考えられる．

トピック T_{14}^B

特徴的な単語は「save」「upload」「get」「implement」といった単語である．これらからコミットの保存や実装，またアップロードに関するコミットメッセージが含まれると考えられる．

トピック T_{19}^B

特徴的な単語は「model」「project」「static」「function」「code」「new」「give」「call」といった単語である．これらから新規のモデルやプロジェクト，機能等の提供に関するコミットメッセージが含まれると考えられる．

トピック T_{20}^B

特徴的な単語は「parent」「previous」「class」といった単語である．これらから親クラスの定義等に関するコミットメッセージが含まれると考えられる．

グループ G_7^B

グループ G_7^B に含まれる単語は表 4.20 に示している．この表からわかるように，G_7^B にはトピック T_{13}^B，トピック T_{18}^B が含まれる．G_7^B のトピックに共通する特徴的な単語は「method(T_{13}^B)」「document(T_{18}^B)」「text(T_{18}^B)」「page(T_{13}^B)」「string(T_{13}^B)」「written(T_{18}^B)」「file(T_{18}^B)」「content(T_{18}^B)」といった単語である．これらから文字列処理やテキストファイルの作業に関するコミットメッセージが含まれると考えられる．

グループ G_8^B

グループ G_8^B に含まれる単語は表 4.21 に示している．この表からわかるように，G_8^B にはトピック T_6^B が含まれる．T_6^B の特徴的な単語は「path」「name」「javadoc」「dir」「adress」「destdir(destination directory)」といった単語である．これらからディレクトリの作業に関するコミットメッセージが含まれると考えられる．

グループ G_9^B

グループ G_9^B に含まれる単語は表 4.22 に示している．この表からわかるように，G_9^B にはトピック T_2^B が含まれる．T_2^B の特徴としては「svn」「link」「check」「anchor」「doc(document)」といった SVN 関連の単語と,「fix」「clean」「error」といった不具合修正に関する単語である．これらから SVN のリンク等による不具合等，手動で確認や修正が必要なものに関するコミットメッセージが含まれると考えられる．

グループ G_{10}^B

グループ G_{10}^B に含まれる単語は表 4.23 に示している．この表からわかるように，G_{10}^B にはトピック T_5^B が含まれる．T_5^B の特徴的な単語は「fix」「cleanup」「bad」「syntax」「error」「copyright」といった単語である．これらからソースコードの著作権に対する修正に関するコミットメッセージが含まれると考えられる．

グループ G_{11}^B

グループ G_{11}^B に含まれる単語は表 4.24 に示している．この表からわかるように，G_{11}^B にはトピック T_{12}^B が含まれる．T_{12}^B の特徴的な単語は「git」「branch」「request」「pull」「logger」「log」といった単語である．これらから例えば git リポジトリの管理など，git 利用のために必要な作業に関するコミットメッセージが含まれると考えられる．

グループ G_{12}^B

グループ G_{12}^B に含まれる単語は表 4.25 に示している．この表からわかるように，G_{12}^B にはトピック T_{15}^B，トピック T_{16}^B が含まれる．T_{15}^B の特徴としては,「use」「code」「url」「classpath」といった単語で，classpath や URL に関するコードの修正に関するコミットメッセージが含まれると考えられる．また，T_{16}^B の特徴としては,「use」「code」「ibiblio」「error」「hint」「common」といった単語で，ibbiblio 等の一般的なヒントからソースコードのライブラリ名等の変更に関するコミットメッセージが含まれると考えられる．これらのトピックの共通点としては，何らかのソースコードの変更が行われたコミットメッセージが含まれていることだと考えられる．

5.2 研究設問への答え

3 章で説明した RQ1 と RQ2 への答えは次の通りである．

RQ1 コードクローンが新たに生成された時の作業にはどのような特徴があるのか．

5.1 節で述べたトピックの考察より，新たにコードクローンが生成された時の作業は次の通りである．

- 新たな機能等の追加や拡張 (表 4.3 のグループ $G_1^A, G_3^A, G_4^A, G_6^A$)
- 不具合の修正 (表 4.3 のグループ G_2^A)
- 既存のデータの確認と改善，拡張 (表 4.3 のグループ G_7^A)
- フォーマット等の統合 (表 4.3 のグループ G_5^A)
- リファクタリングやテストなどの保守作業 (グループ G_8^A, G_9^A)

この結果から，新たな機能等を追加や拡張に関する作業の割合が大きいことが分かる．つまり，新しい機能の作成や拡張の際には，ソースコードを複製して

開発されていることが分かる．

RQ2 コードクローンが新たに生成された時の作業の特徴とその他の作業の特徴に違いはあるのか．

5.1 節で述べたトピックの考察より，新たにコードクローンが生成された時以外の作業は次の通りである．

- 不具合等の修正に関する作業 (表 4.4 のグループ $G_2^B, G_3^B, G_4^B, G_5^B, G_6^B, G_9^B, G_{10}^B, G_{12}^B$)
- フォルダの処理に関する作業 (表 4.4 のグループ G_7^B, G_8^B)
- テストに関する作業 (表 4.4 のグループ G_1^B)
- git の管理に関する作業 (表 4.4 のグループ G_5^B, G_{11}^B)
- リファクタリングやテストなどの保守作業 (表 4.4 のグループ G_3^B)

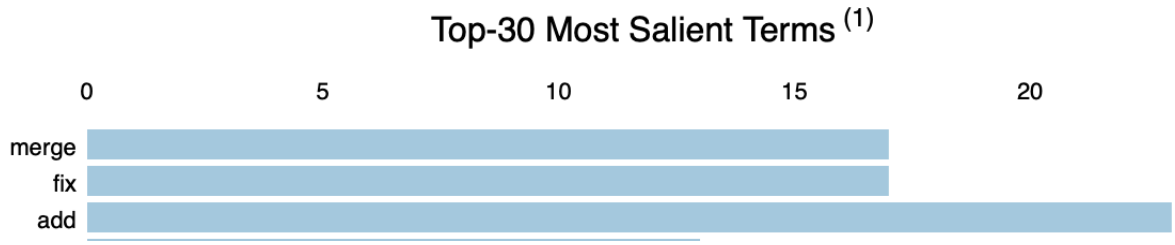
この結果から，不具合の修正に関する作業の割合が大きいことが分かる．

よって，新たにコードクローンが生成された時は，新たな機能等の追加する作業が多く，その他の作業では，不具合の修正に関する作業が多いと考えられる．図 5.1 にトピック分析の結果，それぞれの作業のコミットメッセージに含まれる特徴的な単語の上位 3 つを示す．この図から，その他の作業の時は「add」と「fix」の割合がほぼ同じであるのに対し，新たにコードクローンが生成された時の作業では明らかに「add」の方が多いことが分かる．つまり，新たにコードクローンが生成される際には機能等の追加を行う作業が多いことが分かる．

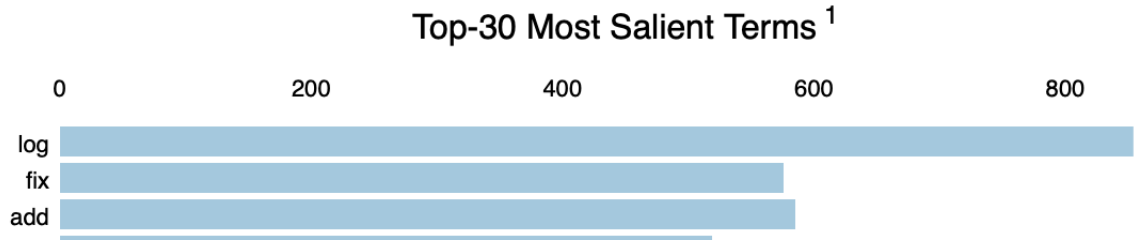
5.3 妥当性の脅威

妥当性の脅威として，次の 5 点が挙げられる．

1. トピック分析でのトピック数が多い．
   トピック数を一般的なトピックモデルの評価指標である Perplexity(トピックモデルの予測性能) と Coherence(各トピックの品質) を利用して検討しているが，トピック数が 22 と 20 と大きな値だったため，各トピックがそれぞれ示す話題についての推定がしづらくなっていた．



(a) 新たにコードクローンが生成された時



(b) 新たにコードクローンが生成された時以外

図 5.1　コミットメッセージに含まれる特徴のある単語上位 3 位

2. 新たにコードクローンが生成された時のデータ数が少ない．

   新たにコードクローンが生成された時のデータ数が少ないため，トピック分析を行った結果，各トピックの間の差異を見分け，話題の推定がしづらくなっていた．特に LDAvis による可視化 (図 4.2) で，近い距離にあるトピック同士は同じ意味を持つトピックであるように感じた．

3. 新たにコードクローンが生成された時以外のデータ数が多い．

   新たにコードクローンが生成された時以外のデータ数が多いため，トピック分析を行い，LDAvis による可視化 (図 4.4) の結果を参照した際，近い距離にあるトピック同士の共通点が見当たらない場合があった．これは，一般的な単語 (例えば「use」や「make」等) が共通しているために近い距離にあるが，コミットメッセージとして考察を行った際は異なる意味であると捉えられるからだと考えられる．

4. 新たにコードクローンが生成された時とそれ以外の時のデータ数に大きな差がある．

   表 4.1 からわかるように，新たにコードクローンが生成された時とそれ以外のデータ数に大きな差があるため，トピック分析を行い，LDAvis による可視化の結果を見た時に，直感的に同じ距離感にあるトピック同士だとしても，これらのトピック同士の共通点の範囲が大きく異なっていた．

5. トピックの推定は主観的なものである．

   トピック分析を行い，検出されたトピックが持つ意味を推定するのは，主観によるものが大きい．本研究では，トピックが持つ単語群からコミットメッセージとして特徴的な単語を抜き出し，トピックの推定を行ったが，トピックの推定を行う人が異なれば，また違う結果が得られると考えられる．

6. 結言

本研究では，新たにコードクローンの生成が行われる作業の傾向について調査を行った．コミットの作業内容をコミットメッセージが要約していることに着目し，これを用いてトピック分析を行うことで作業内容の傾向の考察を行った．オープンソースプロジェクトである「Apache Ant」プロジェクトと「Apache Maven」プロジェクトを用いて調査を行った．調査対象のプロジェクトで，新たにコードクローンが生成されたコミットを CCEvovis を用いて取得し，新たにコードクローンが生成された時のコミットメッセージとそれ以外の時のコミットメッセージを用いて，トピック分析を行った．トピック分析により，新たにコードクローンが生成された場合の作業内容とそれ以外の場合の作業内容を推定し，比較を行った結果を以下に示す.

- ソースコードが変更される作業全体の傾向としては，不具合の修正などの保守作業や新たな機能の追加等が考えられる．
- 新たにコードクローンが生成される際に行われる作業の傾向としては，新たな機能等の追加に関する作業が多い．
- 新たにコードクローンが生成されない場合の作業の傾向としては，不具合の修正に関する作業が多い

謝辞

本研究を行うにあたり，研究課題の設定や研究に対する姿勢，本論文の作成に至るまで，全ての面で丁寧なご指導を頂きました，本学情報工学部門水野修教授，崔恩瀞助教に厚く御礼申し上げます．本論文執筆にあたり貴重な助言を多数頂きました，本研究室の　西浦生成先輩，近藤将成先輩，洪浚通先輩，植村佳治君，北村紗也加さん，國領正真君，塩津拓真君，脇上幸洋君，渡辺大輝君，若林奎人君，舟山優君，Khine Yin Mon さん，上北裕也君，里形洋道君，杉浦智基君，徳舛大樹君，山本凱君，学生生活を通じて筆者の支えとなった家族や友人に深く感謝致します.

参考文献

[1] 肥後芳樹，楠本真二，井上克郎，“コードクローン検出とその関連技術,” 電子情報通信学会論文誌 D，vol.91，no.6，pp.1465–1481，2008.

[2] C.J. Kapser and M.W. Godfrey, “‘ cloning considered harmful ”considered harmful: Patterns of cloning in software,” Empirical Softw. Engg.，vol.13，no.6，pp.645–692，Dec. 2008.

[3] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, “Comparison and evaluation of clone detection tools,” IEEE Transactions on software engineering, vol.33, no.9, pp.577–591, 2007.

[4] 本田紘貴，徳井翔梧，横井一輝，崔　恩瀞，吉田則裕，井上克郎，“コードクローン保守支援を目的とした変更履歴可視化システム,” report 471，電子情報通信学会技術研究報告，March 2019.

[5] H. Honda, S. Tokui, K. Yokoi, E. Choi, N. Yoshida, and K. Inoue, “Ccevovis: A clone evolution visualization system for software maintenance,” Proceedings of the 27th International Conference on Program Comprehension (ICPC), pp.122–125, May 2019.

[6] 高橋仁，中川博之，土屋達弘他，“文書中の単語出現頻度を利用したトピックモデル洗練化,” 研究報告ソフトウェア工学 (SE)，vol.2017，no.22，pp.1–8，2017.

[7] D.M. Blei, A.Y. Ng, and M.I. Jordan, “Latent dirichlet allocation,” Journal of machine Learning research, vol.3, no.Jan, pp.993–1022, 2003.

[8] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, “Comparison and evaluation of clone detection tools,” IEEE Transactions on Software Engineering, vol.33, no.9, pp.577–591, Sep. 2007.

[9] J.R. Cordy and C.K. Roy, “The nicad clone detector,” Proceedings of the 19th International Conference on Program Comprehension, pp.219–220, June 2011.

[10] H. Sajnani, V. Saini, J. Svajlenko, C.K. Roy, and C.V. Lopes, “Sourcerercc: Scaling code clone detection to big-code,” Proceedings of the 38th International Conference

on Software Engineering, pp.1157–1168, 2016.

[11] T. Kamiya, S. Kusumoto, and K. Inoue, “Ccfinder: a multilinguistic token-based code clone detection system for large scale source code,” IEEE Transactions on Software Engineering, vol.28, no.7, pp.654–670, 2002.

[12] 横井一輝，崔恩瀞，吉田則裕，井上克郎，“情報検索技術に基づく細粒度ブロッククローン検出,” コンピュータ ソフトウェア，vol.35，no.4，pp.16–36，2018.

[13] 山中裕樹，崔恩瀞，吉田則裕，井上克郎，佐野建樹，“コードクローン変更管理システムの開発と実プロジェクトへの適用,” 情報処理学会論文誌，vol.54，no.2，pp.883–893，2013.

[14] Y. Yamanaka, E. Choi, N. Yoshida, K. Inoue, and T. Sano, “Applying clone change notification system into an industrial development process,” Proceedings of the 21st International Conference on Program Comprehension (ICPC), pp.199–206, May 2013.

[15] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman, “Indexing by latent semantic analysis,” Journal of the American society for information science, vol.41, no.6, pp.391–407, 1990.

[16] T. Hofmann, “Probabilistic latent semantic indexing,” Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp.50–57, 1999.

[17] D. Newman, J.H. Lau, K. Grieser, and T. Baldwin, “Automatic evaluation of topic coherence,” Human Language Technologies: Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics, pp.100–108, 2010.

[18] M. Röder, A. Both, and A. Hinneburg, “Exploring the space of topic coherence measures,” Proceedings of the Eighth ACM International Conference on Web Search and Data Mining, pp.399–408, 2015.