Vol.2022-SE-211 No.29
2022/7/30

成果物の作業状態を把握する手法の構築に向けたOSSの改訂履歴調査

塚本 良太[1]　石尾 隆[2]

概要：設計書やソースコード等の成果物は改訂を繰り返して作成される．そのため，作業中の状態を定量的に把握することは，プロジェクトの進捗管理や完了までの予測や見積りを行う上で非常に有用である．本稿では成果物の作業状態把握に向けて，まずはOSSの改訂履歴に対する調査を実施した．具体的には，リリースのタイミングを作業の完了とみなし，ソースコード全体の状態，コミットメッセージ，更新ファイルについて，リリースに向けた作業の状態とみなせるような周期的な作業の特徴がないか調査した結果について報告する．

キーワード：OSS，PostgreSQL，Firefox，作業状態，LDA，コミットメッセージ，更新ファイル，信頼度成長曲線

OSS revision history analysis aimed at building a method for grasping the working status of deliverables

RYOTA TSUKAMOTO[1]　TAKASHI ISHIO[2]

Abstract: Deliverables such as design documents and source code are created by repeating revisions. Therefore, quantitatively grasping the state during work is very useful for project progress management and prediction / estimation until completion. In this paper, we first report the results of investigating the revision history of OSS in order to understand the working status of deliverables. The release timing was regarded as the completion of the work, and the state of the entire source code, the commit message, and the update file were investigated for periodic features that could be regarded as the work state.

Keywords: OSS, PostgreSQL, Firefox, Working status, LDA, Commit message, Update file, Reliability growth curve

1. はじめに

ソフトウェアの開発は，最終的なプロダクトを生産するための設計書やソースコードといった中間成果物を作成する活動が大半を占める．これら成果物を作成する活動の生産性を改善・効率化するために，開発プロセスや各プロセス・タスクで使われるツールを改善する等，様々な施策が適用される．そのような施策自体は技術やツールの開発を通じて活発に導入されるが，その効果の計測はプロジェクト全体の工数や，開発プロセス単位の工数で計測されるのが一般的である．例えば，中村ら [1] はGitHubのプロジェクトをスマートシティに見立ててプロジェクトの状態を定量化し，ソフトウェア進化の計測を試みている．藤原 [2] はソフトウェア開発のテストプロセスに絞って障害の検出と修正活動の状態を定量的に計測することを試みている．

一方で，設計書やソースコード等の成果物は改訂を繰り返して作成される．改訂が進むごとに，その作業状態を把握することができれば，プロジェクトの進行度合いを迅速に把握したり，プロジェクト間で類似した変更の作業量を比較したりすることが可能になると考えられ，開発環境の

[1] 三菱電機株式会社
Mitsubishi Electric Corporation
[2] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

変更や些細な工夫による作業の改善効果の定量的な計測につながると期待できる．ソフトウェアのリリースに向けた作業状態を調べる手法の一種としては，信頼度成長曲線による残存バグ数の推定 [3] が類似しているが，バグの個数に限定しない作業状態の計測を試みている研究は見当たらない．

本研究では，ソフトウェアの改訂単位で，成果物の作業状態を把握する手法を構築することを目指して，作業状態を定量化するための特徴量を探す探索的な分析に取り組んだ．具体的には，題材として入手しやすい OSS のソースコードの改訂履歴を調査し，ソフトウェアの複数バージョンをリリースしていく過程で周期的に生じる成果物の状態変化が存在しないか，また，それを識別し作業状態を定量化するための特徴量の候補がないかを調査した．

本研究における「作業」，すなわち計測の対象とする活動は，ソフトウェア開発における成果物作成の活動である．特に，成果物に対するいくつもの改訂を取り込み，次のバージョンとしてリリースするようなライフサイクルを持つものを想定する．成果物としてはテキストを主体とした仕様書，設計書，ソースコード等を想定する．また，作業の具体的な内容は成果物に対するテキストの追加・変更・削除の操作とする．ソフトウェアにバージョン番号を付与してリリースするタイミングを，作業全体の開始・終了とする．Git 等に代表されるバージョン管理システムに当てはめると，作業全体の開始・終了のタイミングはタグ付けのタイミング．個々の作業（追加・変更・削除の操作）の単位はコミットの単位となる．

以降，本論文では，2 章で調査の方法を，3 章で調査の結果を述べる．また，4 章では，作業状況を推定するモデルを試作した結果を述べる．5 章で関連研究を述べ，6 章にて現在のまとめと今後の課題を述べる．

2. 調査方法

本研究では，改訂の回数が豊富な OSS のソースコードを対象として調査を実施し，作業全体の開始から終了の期間で作業状態とみなせるような，周期性等の特徴がないか確かめる．ソースコードのみに限定した理由は，OSS では仕様書，設計書等のドキュメントが整備されるのは稀であるためである．具体的な調査の方法として，以下の 3 つの情報に対して時系列での変化を観察することで，周期性等の特徴がないか，分析を実施した．

- トピックモデル
  文章を特徴づける単語群（トピック）．トピックモデリングは自然言語処理の分野で用いられる統計的な潜在意味解析の一つである．文章が複数の潜在的なトピックから確率的に生成されると仮定し，文章内の各単語はあるトピックが持つ確率分布に従って出現すると仮定する．

表 1 PostgreSQL の調査対象範囲と規模等の情報
Table 1 Information such as the scope and scale of PostgreSQL research

項目	説明
リポジトリ	https://github.com/postgres/postgres 公式のミラー．2021/7/15 時点を clone.
ブランチモデル	メジャーバージョン単位で master からブランチする運用.
タグとバージョン	タグ REL_xx_yy でバージョン xx.yy がリリースされる.
調査対象範囲	REL_11_STABLE ブランチ．タグ REL_11_BETA1〜REL_11_12.
参考規模	REL_11_STABLE ブランチの規模．全 1,878 コミット．5,465 ファイル（REL_11_BETA1）.

- コミットメッセージ
  成果物に対するテキストの追加・変更・削除の操作のサマリを表す文章．Git 等のバージョン管理システムでは成果物に対する改訂内容の登録作業を，コミットと呼ぶ．このコミットに対する補足説明としてコミットメッセージも同時に登録できるのが一般的である．コミットメッセージには，この改訂で何をしたのか，なぜそれをしたのか，といった情報が記載されるのが一般的である．
- 更新ファイル
  コミットで改訂されたファイルのパス．Git 等のバージョン管理システムでは成果物に対する改訂の最小単位はコミットであり，そこには複数ファイルの更新ファイルが含まれる．各更新ファイルの情報としては，ファイルのパス，ファイルに対する改訂の内容（追加，変更，削除）が記録される．本調査ではこれらのうち，分析が容易なファイルのパスを対象とする．

ソフトウェアの開発が進み，バージョンのリリースが近づいてきたとき，特定のトピックに関する修正，あるいは特定のメッセージを持った変更，特定のファイルに対する修正などが毎回発生するのであれば，それらが開発の作業状態を表す指標となる可能性がある．

2.1 調査対象

OSS には作業全体の開始・終了のタイミング（前後のバージョンのリリースタイミング）が不規則なものと規則的なものがある．本調査ではリリースタイミングが不規則な OSS として PostgreSQL，規則的な OSS として Firefox を対象として選択した．なお，本調査の過程でトピックモデルの調査は PostgreSQL の結果で十分と判断したため，Firefox ではトピックモデルの調査は実施していない．

表 1，**表 2** に各 OSS の調査対象範囲と規模等の情報を

表 2　Firefox の調査対象範囲と規模等の情報
Table 2　Information such as the scope and scale of Firefox research

項目	説明
リポジトリ	https://hg.mozilla.org/mozilla-central 2021/10/3 時点を clone.
ブランチモデル	単一ブランチで運用.
タグとバージョン	タグ FIREFOX_BETA_xx_BASE でバージョン xx-1 がリリースされる.
調査対象範囲	タグ FIREFOX_BETA_75_BASE〜FIREFOX_BETA_93_BASE.
参考規模	調査対象範囲の規模．全 76,675 コミット．926,742 ファイル（FIREFOX_BETA_75_BASE）.

表 3　LDA の実行環境
Table 3　LDA execution environment

実装	説明
Python 3.9.5	実行基盤.
Antlr 4.9.2	構文解析器.
antlr-python-runtime 4.9.2	Antlr の Python ライブラリ.
nltk 3.6.2	Python における自然言語処理の前処理用ユーティリティ.
gensim 4.0.1	Python における自然言語処理ライブラリ.

示す．PostgreSQL はブランチが多数あるため，安定版として保守されているバージョン 11.x のリリース群を調査対象とした．Firefox では開発に用いられているブランチを調査対象とした．

2.2　調査手順

2.2.1　トピックモデル

トピックモデリングでは文章が，ある確率で選択されたトピックで生成され，各トピックは，ある確率で選択された単語で生成されると考える．トピック選択や単語選択の確率にはディリクレ分布が用いられるため LDA（Latent Dirichlet Allocation）とも呼ばれる [4].

本研究では，ソースコードのバージョンごとのトピック変化を分析した Thomas ら [5] の手法を用いて，ソースコードを文書として LDA を適用する．これにより，各ソースコードに含まれる，ソースコード全体の特徴を抽出し，開発において周期的に行われる変更を調査する．

分析に用いたソフトウェア環境を**表 3** に示す．LDA を文章に適用する場合，文章を単語に分解する必要があり，ソースコードの場合は，記述に用いられているプログラミング言語に応じた構文解析が必要である．本調査ではプログラミング言語向けの構文解析器として知られている ANTLR[*1] を用いて，C/C++言語の識別子群をソースファイルから抽出する．また，LDA の実行には Python の自然言語処理ライブラリである gensim[*2]を用いる．

LDA の前処理（ソースコードから LDA で分析する単語群を生成する処理）は，Thomas ら [5] と同等の以下の手順とした．また，LDA による分析時に指定したトピック数は 45 である．

(1) ANTLR でトークン化
　識別子属性のトークンのみ抽出.
(2) スネークケースを分解
　“_”（アンダースコア）で分離.
(3) キャメルケースを分解
　正規表現で分離.
```
.+?(?:(?<=[a-z])(?=[A-Z])|(?<=[A-Z])↩
↪(?=[A-Z][a-z])|$)
```
(4) 小文字化
(5) 数値を正規化
　固定値 “000” に置換.
(6) 変数の連番を正規化
　正規表現で数値部分を削除.
(7) 動詞の時制，名詞の複数形等を正規化
　nltk の averaged_perception_tagger で品詞を判別.
　nltk の wordnet の lemmatizer で正規化.
(8) ストップワードの除去
　nltk の stopwords を利用.
(9) 80%以上の文章（ファイル）に存在する単語を除去
　gensim の dictionary の filter_extremes を利用.
(10) 2%以下の文章（ファイル）に存在する単語を除去
　gensim の dictionary の filter_extremes を応用.
　全体から 2%以上の文章に存在する単語を除去した単語群を除去対象にする.

2.2.2　コミットメッセージ

本調査で対象としている PostgreSQL と Firefox の開発者はどちらも英語でコミットメッセージを記載している．英語のコミットメッセージでは，先頭単語に作業を表す動詞（add や fix 等）が現れることが多いことから，先頭単語がコミットメッセージ全体の特徴を表すと考え，その出現タイミングを分析するものとした．ただし，それぞれのプロジェクトに固有の記法が見られたため，以下のように分析ルールは個別に定義した．

2.2.2.1　PostgreSQL におけるコミットメッセージの先頭単語の抽出方法

表 4 に PostgreSQL のコミットメッセージの例を示す．PostgreSQL 場合は，書き出しから目的の単語が得られることが分かる．そこで，コミットメッセージを空白文字で分離して得られる最初の文字列に対し，自然言語処理で用いられる一般的な前処理を施す．また，複数バージョンに

[*1] https://www.antlr.org/
[*2] https://radimrehurek.com/gensim/

表 4　PostgreSQL のコミットメッセージ例
Table 4　PostgreSQL commit message example

コミットメッセージ
Stamp 11beta1.
Fix typo in comment.
Add missing files to src/backend/lib/README.
Widen COPY FROM's current-line-number counter ...
doc: PG 11 release note fixes: PGhost, typo
...
Translation updates
Prevent integer overflows in array subscriptin...
Fix mishandling of resjunk columns in ON CONFL...
Last-minute updates for release notes.
Stamp 11.12.

表 5　Firefox のコミットメッセージ例
Table 5　Firefox commit message example

コミットメッセージ
Bug 1620511 - Make ensure_mobile_android_packa...
No bug - Tagging mozilla-central 7ac664c145986...
No bug - Tagging mozilla-central 34377d8c0d2ad...
Update configs. IGNORE BROKEN CHANGESETS CLOSE...
Bug 1614294 - Increase the shutdown crash time...
...
Bug 1640779 - [X11][EGL] Implement xrandr-base...
Bug 1736684 - Part 1: Add test coverage for th...
Bug 1736431 - Allowlist URLs containing the st...
Bug 1737419 - Add note about mach npm test for...
Bug 1737424 - Markup inline code properly in n...

同様に現れる傾向をつかむため，8 割以上のバージョンに現れる先頭単語を抽出するものとした．具体的な手順を以下に示す．実行環境は表 3 と同様である．

(1) スペースで分離して先頭の文字列を抽出
(2) 小文字化
(3) 末尾に ":" がある場合は除去
(4) 数値を正規化
　固定値 "000" に置換.
(5) 動詞の時制，名詞の複数形等を正規化
　nltk の averaged_perception_tagger で品詞を判別.
　nltk の wordnet の lemmatizer で正規化.
(6) 80%以上のバージョンに存在する単語を抽出

2.2.2.2　Firefox におけるコミットメッセージの先頭単語の抽出方法

表 5 に Firefox のコミットメッセージの例を示す．Firefox の場合，書き出しにバグトラッキングシステムの管理番号や，コンポーネント等の分類ラベルと考えられる記載が付与される場合があることが分かる．そこで，コミットメッセージからメッセージの主要部分を抽出した上で，空白文字で分離して得られる最初の文字列に対し，自然言語処理で用いられる一般的な前処理を施す．PostgreSQL の場合と同様に複数バージョンに同様に現れる傾向をつかむため，8 割以上のバージョンに現れる先頭単語を抽出する．具体的な手順を以下に示す．実行環境は表 3 と同様である．

(1) 正規表現を用いてメッセージの主要部分を抽出
　以下の順で正規表現を適用し，message 部分（赤字部分）を抽出する.
　1　Bug 1234567 - message
　`^(Bug|bug)␣[0-9]+␣*(-|:|,|\.|:|--|)␣+(.+)`
　2　Part 1 - message
　`^(part|Part|pt|PT|P)␣*[0-9]+(-[0-9]+|)␣↩`
　`↪ *(:|\)|-|\.|)␣+(.+)`
　3　No Bug - message
　`^(No␣Bug|No␣bug|no␣bug)␣*(-|,|:)␣+(.+)`
　4　[xxx] - message
　`^\[.+?\]␣*(-|)␣+(.+)`
　5　[xxx] message
　`^\[.+?\]␣+(.+)`
　6　(xxx) message
　`^\(.+?\)␣+(.+)`
　7　xxx: message
　`^\S+:␣+(.+)`
(2) スペースで分離して先頭の文字列を抽出
(3) 小文字化
(4) 数値を正規化
　固定値 "000" に置換.
(5) 動詞の時制，名詞の複数形等を正規化
　nltk の averaged_perception_tagger で品詞を判別.
　nltk の wordnet の lemmatizer で正規化.
(6) 80%以上のバージョンに存在する単語を抽出

2.2.3　更新ファイル

本調査では個別の更新ファイルの出現タイミングを分析する．1 つのコミットに含まれる複数の更新ファイルの組み合わせは考慮しない．80%以上のバージョンで更新が行われたファイルの名前を，各コミットから抽出する．

3.　調査結果

3.1　トピックモデル

図 1 に PostgreSQL のトピックモデルの変遷を示す．縦軸は Thomas ら [5] が定義した Weight（そのトピックに関するソースコードの量），横軸はコミット数，図中の縦線はバージョンリリースのタイミングである．**表 6** は図 1 中のトピック（variable）を構成する単語群の一部である．variable の括弧書きは Weight のトップ 3 であり，上位からデータオブジェクト取得関連，関数操作関連，メモリ管理関連のトピックであると考えられる．

変遷を見ると，バージョンアップに従ってトピックの傾

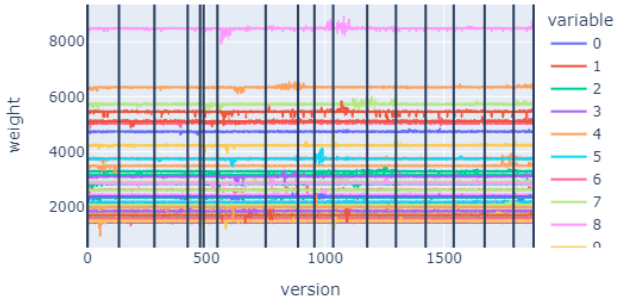

図 1 PostgreSQL のトピックモデルの変遷
Fig. 1 Transition of PostgreSQL topic model

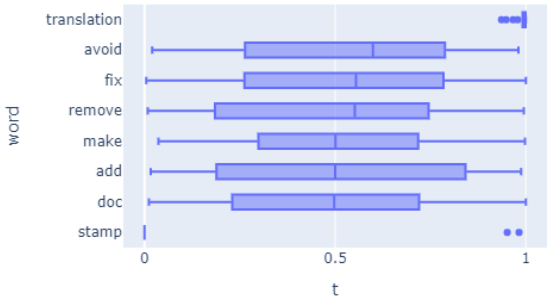

図 2 PostgreSQL のコミットメッセージ分析結果
Fig. 2 PostgreSQL commit message analysis results

表 6 PostgreSQL のトピック
Table 6 PostgreSQL topic

variable	トピックを構成する単語群
0	p, po, j, size, x, word, ptr, n, point, flag
1	field, node, stmt, copy, write, scalar, compare, alter, create, expr
2	log, x, rec, record, xlog, state, id, xlrec, ptr, lsn
...	...
7(3)	page, buffer, get, number, item, offset, block, tuple, opaque, data
...	...
24(2)	pg, datum, getarg, result, int, function, return, args, text, arg
...	...
38(1)	object, id, oid, get, pg, tuple, datum, heap, errcode, form
...	...
43	state, array, null, v, type, val, value, errcode, arr, start
44	null, guc, option, config, pgc, name, value, newval, extra, hook

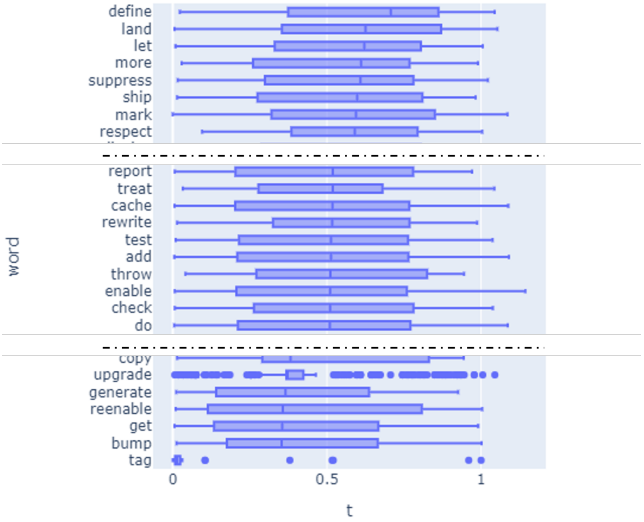

図 3 Firefox のコミットメッセージ分析結果
Fig. 3 Firefox commit message analysis results

向が大きく変わっていくことはなく，各トピックが占める割合は，ほぼ一定であることが分かる．これは，コードベースの規模に対して，ソースコードの変化量が相対的に非常に小さいためだと考えられる．バージョンリリース間においても，バージョンによらない周期的な特徴や傾きの変化などは見られなかった．

以上のことから，コードベースの規模が大きなプロジェクトの場合，トピックモデルに基づく作業状態の把握は難しいことが分かる．Firefox も PostgreSQL と同じく非常に規模が大きなコードベースであることから，Firefox のトピックモデルの変遷も PostgreSQL と同様と判断し，調査は未実施とした．

3.2 コミットメッセージ

図 2 が PostgreSQL のコミットメッセージの出現タイミングを，**図 3** が Firefox のコミットメッセージの出現タイミングを，それぞれ箱ひげ図によって図示したものである．横軸はバージョンごとの開発開始から終了（リリース）までの時刻を 0 から 1 に正規化した値であり，縦軸はコミットメッセージの先頭単語を中央値順に並べている．下のほうにある単語ほど，開発開始の初期に現れやすく，上のほうにある単語ほど，開発の後半になってから現れる単語である．Firefox の結果は全バージョンの 80%に登場する先頭単語の数が非常に多かったため，結果の一部のみを表示している．

出現タイミングの分布を見ると，特定のコミットメッセージは作業全体の開始直後もしくは終了直前に現れることが分かる．PostgreSQL であれば，“stamp”，“translation”，Firefox であれば，“tag” である．これらはタグ付けなどの作業に対応するコミットであり，全バージョンにまたがっ

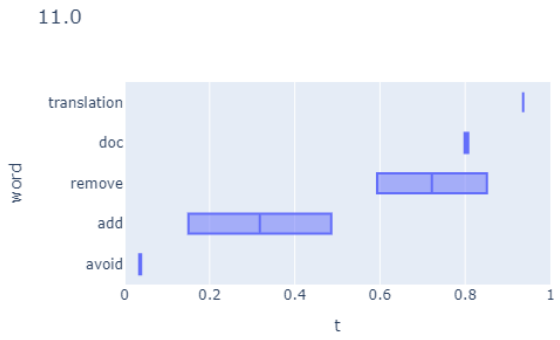

図 4 PostgreSQL 11.0 のコミットメッセージ分析結果
Fig. 4 PostgreSQL 11.0 commit message analysis results

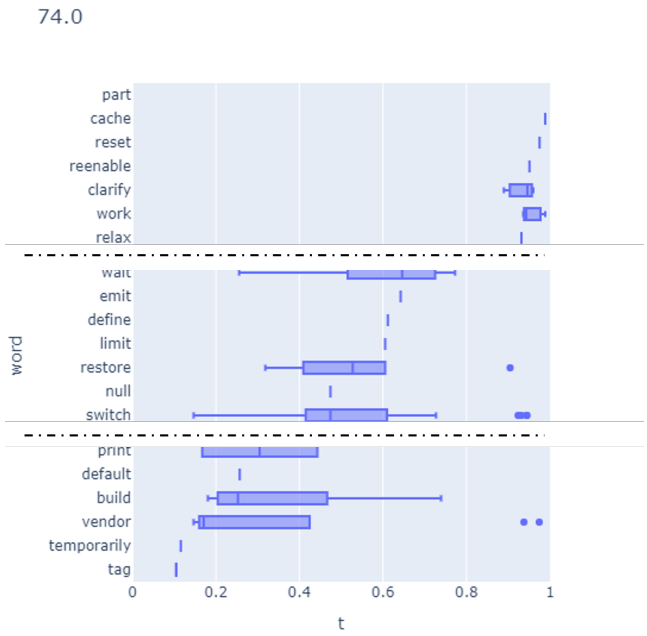

図 5 Firefox 74.0 のコミットメッセージ分析結果
Fig. 5 Firefox 74.0 commit message analysis results

たコミットメッセージの出現タイミングの偏りは，ほんの一部であることが分かった．

一方で，バージョンごとに分析すると，偏りが見られる単語も存在することが分かった．**図 4** は，調査対象の最初のバージョンにおける PostgreSQL のコミットメッセージの出現タイミングの分析結果である．同様に**図 5** は Firefox の調査対象の最初のバージョンにおけるコミットメッセージの出現タイミングの分析結果の一部である．

特定バージョンに絞って出現タイミングの分布を見ると，全バージョンでは偏りがなかった単語でも出現タイミングに偏りがあることが分かる．過去のバージョンにおけるコミットメッセージの出現タイミングを分析しておくことで，単語の出現が始まってから，そのコミットメッセージの出現が収束する時期をある程度見積もれることが期待できる．

3.3 更新ファイル

更新ファイルについてもコミットメッセージと同様の傾向が見られた．すなわち，全バージョンでは更新ファイルの出現タイミングの偏りは少ないが，特定バージョンに絞って出現タイミングの分布を見ると偏りがあることが分かった．従って，更新ファイルにおいても過去のバージョンにおける更新ファイルのタイミングを分析することで，その更新ファイルが収束する時期をある程度見積もれることが期待できる．

4. 作業状態の把握に向けた予備実験

3 章の調査結果から，過去の改訂履歴からコミットメッセージと更新ファイルの出現タイミングを事前に分析しておくと，そのコミットメッセージもしくは更新ファイルの出現が収束する時期を推定できる可能性があることが分かった．そこで，作業状態の把握に向けて，実際にコミットメッセージや更新ファイルの収束時期をどの程度の精度で推定できるか，予備実験を実施した．コミットメッセージと更新ファイルの特徴は同様であったことから，ここでは，コミットメッセージに絞って実験を実施した．

4.1 評価方法

バージョンごとにコミットメッセージの出現期間が偏るため，その期間がバージョンによらない時期となるように補正してから，信頼度成長曲線のモデルを当てはめる．

コミットメッセージの出現分布は同様でありその時期が異なると想定し，補正ではバージョンごとの出現分布を一様にずらして中央値を一致させる．複数バージョンの出現タイミングを重ねることで，モデルへの当てはめ（フィッティング）に利用できる点数を増やすことが狙いである．

信頼度成長曲線にはゴンペルツ曲線（ゴンペルツ分布）を用いる．フィッティングによって求められたゴンペルツ曲線を用いて，コミットメッセージの出現に対し，収束時期を推定する．フィッティングのためにコミットメッセージの出現分布を補正しているため，初回の出現タイミングも推定し，補正によってずらした分布を戻す必要がある．

本評価では初回の出現タイミングを 5 パーセンタイルの時期，収束時期はゴンペルツ分布の 95 パーセンタイルとした．また，ゴンペルツ曲線へのフィッティングには Python の科学技術ライブラリである SciPy[*3] を用いる．gensim の依存ライブラリでもあるため，実行環境は表 3 と同様である．

具体的な手順を以下に示す．

[*3] https://scipy.org/

表 7　パーセンタイル毎の推定精度
Table 7　Estimated accuracy for each percentile

パーセンタイル	PostgreSQL の平均誤差	Firefox の平均誤差
50	0.10	0.21
75	0.15	0.27
80	0.16	0.29
85	0.17	0.31
90	0.19	0.35
95	0.23	0.41
100	0.22	0.38

(1) 評価用のバージョンのデータを除く
(2) コミットメッセージ（先頭単語）の出現タイミングを補正する
　各バージョンの同じコミットメッセージに対し，出現タイミングの中央値が 0.5 となるように一様にずらす．
　コミットメッセージごとに各バージョンの出現点を重ねたデータを作成する．
(3) 重ねたデータからコミットメッセージごとにゴンペルツ分布のパラメータを推定する
　scipy.stats.gompertz の fit を用いる．
(4) 推定したゴンペルツ分布から X パーセンタイルとなる時期を求める
　scipy.stats.gompertz の interval を用いる．
　X は 5，50，75，80，85，90，95 とする．
(5) 評価用に除いたバージョンでの実際の最終コミット日時と比較し，誤差を求める
　5 パーセンタイルになると推定した時期と最初のコミット日時の差を推定したゴンペルツ分布の水平オフセットとし，推定した 50〜95 パーセンタイルの時期を水平オフセット分ずらす．
　95 パーセンタイルになると推定した時期と実際の最終コミット日時の差を求める．
　他のコミットメッセージ，他の評価用バージョンも含めた全ての誤差の平均を求める．また，実際のコミット日時から得られる 50〜95 パーセンタイルの日時についても同様に誤差の平均を求める．

4.2　評価結果

表 7 に評価結果を示す．100 パーセンタイルとしている行は実際の最終コミット日時と推定した 95 パーセンタイルの時期の誤差である．推定精度を見ると，パーセンタイルが大きくなるほど誤差が一様に大きくなっていることが分かる．また，常に誤差が一定以上あるため，一定のバイアスを与える，もしくは水平オフセットの求め方を変える等，推定方法を改善する余地があると考えられる．

以上のことから，過去の改訂履歴からコミットメッセージ（先頭単語）の出現タイミングを事前に分析しておくことでコミットメッセージが出現したときに同種の改訂がいつ収束できるかをある程度推定できると考えられる．

5.　関連研究

ソースコードを文書とみなして自然言語処理を適用することで，ソフトウェアに含まれた機能を認識しようという試みが行われている．Kawaguchi ら [6] は，ソフトウェアごとのソースコードに含まれる単語の集合に対して潜在意味解析 [7] を適用し，類似した機能を持つソフトウェアをグループ化する手法を提案した．Tian ら [8] は，LDA を用いて同じように分類を実施し，いくつかの機能カテゴリが認識できることを報告している．Thomas ら [5] は，バージョン管理システムに蓄積されたソースコードに対して LDA を適用し，ソフトウェアのバージョンごとに特定のトピックが占める割合の変遷を追うことで，ソフトウェアが実現している機能等の変化をバージョンごとに可視化した．本研究では，ソースコードに対するトピックモデルとして Thomas らの手法 [5] を元に，バージョンのリリースごとに周期性が存在するかどうかの分析を行った．

ソフトウェアのリリースに関しては，Openja ら [9] がリリースエンジニアリングのトピックを整理している．また，Hu ら [10] は Dynamic Topic Modeling を用いてコミットメッセージに登場するトピックの変化を分析し，時間経過に伴って開発トピックが実際に変わる様子を報告している．ソフトウェアのリリースに必要な周期性のある活動については今後も分析が必要であるが，前者のようなリリース準備等の作業内容や，時系列による変化が少ないトピックに手がかりがあるのではないかと考えている．

OSS を対象とした作業の進捗の予測モデルとして，バグ修正件数およびリリース日を予測するモデルが提案されている．Washizaki ら [3] は信頼度成長モデルによって，リリースまでに行われる Issue 修正件数の予測事例を報告している．また，Parveen ら [11] は，バグの修正件数と，ソースコードの変更対象ファイルのランダムさ（complexity of code changes）を使って，OSS のリリース時期を予測することを試みている．本研究では，リリース日の予測それ自体は目的ではないが，使用する特徴量がソフトウェアの開発作業状態を捉えられているのであれば，リリース日の予測精度が向上すると考え，評価指標として使用した．

リリースサイクルそのものは，本来，プロジェクトごとに設定されるものであり，他の活動に影響する制御変数であると考えられている．たとえば，da Costa ら [12] は，リリースサイクルを短縮すると，バグ修正がエンドユーザに届くまでにかかる時間がかえって延びてしまうことを報告している．また，Nourry ら [13] は，リリースサイクルを短縮するとリファクタリング活動が減少することを報告している．本研究は，リリースサイクルそのものの変更は考

えていないが，リリースのために必要と考えられる作業が順調に進められているか，あるいは工程の工夫によって作業が削減されたかを自動的に判断する手法の開発を目的としている．

6. おわりに

本研究では，ソフトウェアの作業状況を推定する手段として，周期性のある特徴量を探すための分析を行った．コミットメッセージに出現する先頭単語，更新ファイルはそれぞれ登場時期が異なり，個別に信頼度成長曲線に似た収束のパターンを見せたことから，同種の改訂がいつ収束するか，作業の種別ごとに推定できる可能性があると考えている．

今後の課題として，作業終了までの作業量の推定，設計書等のドキュメント類に対する同様の分析が挙げられる．本研究では，特定のコミットメッセージの出現や特定のファイル更新の収束時期の推定について調査を行ったが，たとえばすべてのコミットメッセージ，更新ファイルが収束する時期等から現在の作業状況を推定できる可能性があると考えている．また，ドキュメント類に対する同様の分析については，企業における製品開発のドキュメントを題材に同様の検討を進める予定である．

参考文献

[1] 中村 匡秀, 戸田 航史, 玉田 春昭, 松本 健一, 自発的ソフトウェア進化を促すプロジェクト状態の推定, 設計工学・システム部門講演会講演論文集, 2019.29 巻, p.2413, (2019)
[2] 藤原 隆次, ソフトウェア信頼度成長モデルとその効果的実務適用事例　(特集「ソフトウェア信頼性工学の新展開」), 日本信頼性学会誌　信頼性, 27 巻, 7 号, p.461-470, (2005)
[3] H. Washizaki, K. Honda, and Y. Fukazawa, “Predicting Release Time for Open Source Software Based on the Generalized Software Reliability Model,” in 2015 Agile Conference, Aug. 2015, pp. 76–81, doi: 10.1109/Agile.2015.19.
[4] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” J. Mach. Learn. Res., vol. 3, pp. 993–1022, Mar. 2003.
[5] Stephen W. Thomas, Bram Adams, Ahmed E. Hassan, Dorothea Blostein, Studying software evolution using topic models, Science of Computer Programming, Volume 80, Part B, 1 February 2014, Pages 457-479, (2014)
[6] S. Kawaguchi, P. K. Garg, M. Matsushita, and K. Inoue, “MUDABlue: An automatic categorization system for Open Source repositories,” Journal of Systems and Software, vol. 79, no. 7, pp. 939–953, Jul. 2006, doi: 10.1016/j.jss.2005.06.044.
[7] T. K. Landauer, P. W. Foltz, and D. Laham, “An introduction to latent semantic analysis,” Discourse Processes, vol. 25, no. 2–3, pp. 259–284, 1998, doi: 10.1080/01638539809545028.
[8] K. Tian, M. Revelle, and D. Poshyvanyk, “Using Latent Dirichlet Allocation for automatic categorization of software,” in 2009 6th IEEE International Working Conference on Mining Software Repositories, May 2009, pp. 163–166, doi: 10.1109/MSR.2009.5069496.
[9] M. Openja, B. Adams, and F. Khomh, “Analysis of Modern Release Engineering Topics : – A Large-Scale Study using StackOverflow –,” in 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), Adelaide, Australia, Sep. 2020, pp. 104–114, doi: 10.1109/ICSME46990.2020.00020.
[10] J. Hu, X. Sun, D. Lo, and B. Li, “Modeling the evolution of development topics using Dynamic Topic Models,” in 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Mar. 2015, pp. 3–12, doi: 10.1109/SANER.2015.7081810.
[11] T. Parveen and H. D. Arora, “Applying Information Measure for Predicting Release Time of Open Source Software,” Walailak Journal of Science and Technology (WJST), vol. 15, no. 1, Art. no. 1, 2018.
[12] D. A. da Costa, S. McIntosh, C. Treude, U. Kulesza, and A. E. Hassan, “The impact of rapid release cycles on the integration delay of fixed issues,” Empir Software Eng, vol. 23, no. 2, pp. 835–904, Apr. 2018, doi: 10.1007/s10664-017-9548-7.
[13] O. Nourry, Y. Kashiwa, Y. Kamei, and N. Ubayashi, “Does shortening the release cycle affect refactoring activities: A case study of the JDT Core, Platform SWT, and UI projects,” Information and Software Technology, vol. 139, p. 106623, Nov. 2021, doi: 10.1016/j.infsof.2021.106623.