コードレビュープロセスに基づくパッチ再投稿の予測

平尾 俊貴[1,a)]　伊原 彰紀[1,b)]　松本 健一[1,c)]

概要：オープンソースソフトウェア (OSS) プロジェクトでは，不具合修正，機能追加などの要求に対して，開発者がソースコードを開発，更新し，変更したファイル（パッチ）を作成する．パッチは，プロダクトに統合される前に，レビューアによって変更内容の検証が行われる．レビューアがパッチを採択，または，不採択と判断した場合，それ以降の検証は行われないが，再度修正を要求する場合がある．本論文では，パッチのレビュープロセスに基づきパッチの再投稿が必要か否かを判断するパッチ再投稿予測モデルを構築する．予測モデルは，コードレビュープロセス中の変更対象プログラムの決定，変更，検証の 3 つの工程に関するメトリクスを用いて構築する．Qt プロジェクトのレビューデータセットを用いて実験を行った結果，予測精度は再現率が 0.686, 適合率が 0.891, F 値が 0.775 であり，レビューアの意見を示すメトリクスが再投稿に関係することが分かった．

キーワード：コードレビュープロセス，パッチ再投稿，オープンソースソフトウェア

Predicting Patch Re-submission Based on Code Review Process

Abstract: In order to fix a bug or to create a new function for Open Source Software (OSS), the developers update source codes. The piece of software designed to update source code is called patch. In OSS projects, some reviewers (not the patch creater) usually verified the patches before integrating into the products. If the reviewers do not decide to integrate or abandon a patch, the reviewers will request the patch creator to update again (Patch Re-submission). This study proposed a method to predict patch re-submission based on code review process. We built the prediction model using Qt project dataset. Our classifiers indicated that: we can accurately predict the patch re-submission with , a recall of 0.69, a precision of 0.89, and a F value of 0.78; and reviewers comment is the most important factor to understand the patch re-submission.

Keywords: Code Review Process, Patch Re-Submission, Open Source Software

1. はじめに

オープンソースソフトウェア (OSS) は，ソースコードが不特定多数の開発者の目に触れるため不具合が発見されやすい [1]．一方で，様々な専門性や開発スキルを持つ開発者によって変更される全てのファイル（パッチ）が高品質，高可読性であるとは限らない．そのため，すぐにプロダクトへ統合（採択）されず，パッチの修正が要求される場合がある．時には，すぐに破棄（不採択）されることもある．プロダクトに統合されない具体的なパッチの事例としては，欠陥が混入している，コーディングルールに従っていない，インラインコメントが不適切，過去に同様の機能を有するプログラムが投稿されている，などが挙げられる．このようなプロダクトに統合すべきでないパッチであるか否かを判定するために，多くの OSS 開発では，更新されたパッチの検証作業（レビュー）が行われる．

通常，更新されたパッチは，複数のレビューアが検証する．その後，プロジェクトを変更する権限を持つ開発者 (以降，コミッター) が認めたソースコードのみをプロダクトに統合する．従来のパッチの投稿・検証作業は，メーリングリストや不具合追跡システムなどを活用されていた．しかし，これらのツールはコードレビューのためのツールではないため，更新されたコードに対する議論，変更されたコードに対する評価方法などをテキストで議論されることも多く，プロジェクトや開発者間で管理できていな

[1] 奈良先端科学技術大学院大学，情報科学研究科
[a)] hirao.toshiki.ho7@is.naist.jp
[b)] akinori-i@is.naist.jp
[c)] matumoto@is.naist.jp

かった．昨今では，Gerrit，ReviewBoard をはじめとするレビュー管理システムが OSS プロジェクトや企業などでも導入されており，このようなツールベースのレビュー管理は Modern Code Review と呼ばれている [2]．レビュー管理システムからは，更新されたプログラムや，レビューを行った開発者を自動抽出可能であるため，コードレビューに関する研究が活発に進められている．

多くのソフトウェア開発プロジェクトで，コードレビューがレビュー管理システムで一元管理されるようになり，複数のレビューアの意見を容易に比較できるようになった．複数人によってパッチが検証されるようになると，レビューアの意見が採択，または，不採択として一致する場合もあれば，意見が割れる場合もある．もしパッチを修正する必要がある場合，修正を行って再度提出（再投稿）が要求される．例えば，Qt プロジェクトでは，過去に投稿されたパッチ 70,705 件のうち 28,276 件（39.9%）は再投稿されている．OSS 開発では，必ずしも投稿後に早速検証されるとは限らず，検証が完了するまでに数日かかることも多い．そのため，変更内容についてパッチ投稿者が記憶している早い段階で再投稿すべきか否かの判断を得ることが望まれる．

本論文では，パッチの更新や評価に関係する 3 つの観点（更新対象プログラム，更新，検証）のメトリクスを用いて，パッチの再投稿が必要か否かを判断するためのパッチ再投稿予測モデルの構築を行う．パッチ再投稿予測モデルは，レビュー管理システムに投稿されたパッチについて，再投稿が必要か否かを判断するモデルである．このモデルは，レビューアから検証結果（採択/不採択，または，再投稿）が報告される任意の時点で予測する．予測モデルに用いる 3 つの観点について，

更新対象プログラムは，更新対象となるプログラムファイル数など，変更規模に関するものを対象とする．更新は，プログラムの変更量，および，変更者の経験量など，変更に関するものを対象とする．検証は，レビューアからのコメント数や投票数など，検証作業に関するものを対象とする．本論文では，大規模かつ，不特定多数の開発者が貢献している Qt プロジェクトを対象に実験を行った．

続く 2 章では，コードレビュープロセスについて述べる．3 章では，実験方法を説明し，4 章では，パッチ再投稿予測の実験結果について述べる．5 章で本論文の考察を行い，最後に 6 章で本論文のまとめと今後の課題について述べる．

2. コードレビュープロセス

多くの OSS プロジェクトにおいて，コードレビューは，メーリングリスト [3][4][5]，不具合追跡システム [6][7][8] などを使って管理されている．しかし，各々のツールはコードレビューのためのツールではないため，変更されたコードに対する議論，変更されたコードに対する評価方法などをテキストで議論されることも多く，プロジェクトや開発者間で一元化できていなかった．

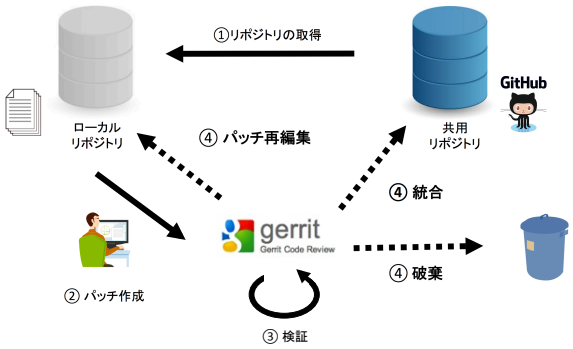

図 1 コードレビュープロセスの概念図

昨今では，Gerrit [*1]，Phabricator [*2]，ReviewBoard [*3] を始めとするコードレビュー管理システムが OSS プロジェクトや企業などで使用されている．コードレビュー管理システムは，容易な操作で利用することができ，変更箇所の閲覧や直接コメントを挿入するなど，ツール上でコードレビューに関する議論や評価を行うことができる．また，版管理システム Github とも連携することが可能であり，コードレビュー管理ツールから直接 Github にコミットすることも可能である．このようにコードレビュー管理をシステムで一元化した管理方式は Modern Code Review と呼ばれている [2]．

Modern Code Review におけるプロセスの概略図を図 1 に示す．開発者によってパッチが作成されてから，パッチの検証が終了するまでの流れは次の通りである．

(1) 開発者は開発対象のソースコードリポジトリをローカル環境にコピーする．
(2) 開発者はパッチを作成し，コードレビューシステム (Gerrit など) に提出する．
(3) パッチ投稿者が担当者 (レビューア) を決定し，レビューアによってパッチが検証 (レビュー) される．
(4) パッチがレビューアによる検証によって採択された場合，パッチは統合される．ただし，修正が必要であれば再投稿を要求され，不採択であれば破棄される．

手順 (1) は，開発者がプロジェクトの変更するために行う．手順 (2), (3) は，手順 (4) の判定が再投稿であれば，繰り返す．手順 (4) において，パッチが統合もしくは破棄された時，パッチの検証が完了する．また，このプロセスにおいて，採択/再投稿/不採択の評価結果はレビューアが議論して決定する．

[*1] Gerrit: https://code.google.com/p/gerrit/
[*2] Phabricator: http://phabricator.org/
[*3] ReviewBoard: https://www.reviewboard.org/

3. 実験方法

3.1 メトリクスの選定

パッチ再投稿予測モデル構築のために，本論文では，コードレビュープロセスにおいて，パッチの開発，評価に影響する 3 つの観点（変更対象プログラム，変更，検証）に関するメトリクスを選定する．表 1 に，選定したメトリクスと選定した理由を示す．

パッチ再投稿予測モデルは，パッチをレビュー管理システムへ投稿後に再投稿が必要かどうかを判断するモデルであり，パッチが投稿されたのち，レビューから検証結果が報告（採択/不採択，または，再投稿）される任意の時点で予測することを想定している．従って，表 1 に示すメトリクスは，パッチ投稿後，レビューアから検証結果が報告されるまでに計測されるメトリクスを対象とする．

変更対象プログラムに関するメトリクスは，機能追加，不具合報告をプロジェクトが受け取った時点で，変更すべきファイルを特定することができれば，この時点で計測できる可能性がある．変更に関するメトリクスは，開発者がパッチをレビュー管理システムへ投稿した時点で計測できる．検証に関するメトリクスは，パッチ投稿からレビューアからの検証結果が報告されるまでの任意の時点までに計測するため，パッチ投稿から計測時点が経過するにつれて，メトリクスの値も更新される．

3.2 パッチ再投稿予測モデル構築方法と評価

本論文では，モデルを構築する前に，表 1 で示した説明変数間の多重共線性について分析を行った．多重共線性は，説明変数間に強い相関が発生していることであり，モデル構築に相関関係が強い変数が混入していると予測精度の低下につながる．本論文では，多重共線性を確認するために R パッケージの `redun` を用い，強い相関関係を持つ説明変数を削除する．

本論文では，モデル構築，及び，モデルを評価するためのデータ作成には，多くの研究で用いられている交差検証法を使用する [9][10]．交差検証法はデータセットを n 個のブロックに分割し，$n-1$ 個のブロックでモデルを構築し，それ以外のブロックを検証用に用いる．本論文では，多くの論文で使われるように n を 10 とする．3.1 節で示した変数を学習するためにランダムフォレストアルゴリズム [11] を用いてパッチ再投稿モデルを構築した．

モデルを評価するために交差検定を行うため，モデルは 10 回構築され，これを本実験では 100 回繰り返す．合計 1000 個のモデルが構築され，各モデルで得られた 4 つの評価結果（適合率，再現率，F 値，AUC (Area Under the Curve) ）の平均値を比較する．各評価指標は 0 以上 1 以下の値をとり，1 に近づくほど予測精度が高いことを示す．また，AUC は 0.5 以上超えると，ランダム抽出よりも有効であることを示す．

3.3 モデルに影響するメトリクスの重要度

本論文では，再投稿が必要であるか否かの判断に影響する要因を明らかにするために，ランダムフォレストアルゴリズムの分類精度に強く影響する度合いを示す値を計測する [11]．パッチ再投稿予測モデルは 1000 回構築されるため，モデルへの影響度合いを示す値に基づく説明変数のランキングも 1000 回出力される．本論文では，各説明変数がランキングのどの位置にランク付けされるかを分類するために `ScottKnott Test` [12] と呼ばれるクラスタ分析手法を用いる．`ScottKnott Test` は統計的に複数のスコアをグルーピングする手法であり，Fault-prone モジュール判別モデルに影響を及ぼす説明変数の特定などにも用いられている [13]．本論文では，`ScottKnott Test` を行うために R パッケージの `ScottKnott` を用いる．

4. 実験

4.1 概要

本章では，3.1 節で示した変数を用いて，パッチ再投稿予測モデルを構築し，モデル精度に影響する変数の特徴を明らかにする．本論文では Qt プロジェクトのデータセットを用いて実験を行った．Qt は UI フレームワークであり，ソフトウェア開発企業 Digia の一部門によって開発されているが，Qt プロジェクトは OSS 開発のように不特定多数の開発者が貢献している．Qt プロジェクトではレビュー管理システム Gerrit を用いてレビューデータを管理している．本論文では，McIntosh ら [14] によって公開されているデータセットを利用する．そのデータセットには Qt プロジェクトのレビュー票 70,705 件が登録されている．レビュー票の中で，パッチ投稿後に評価結果が報告されるまでにレビューアからの評価がない場合は対象外とし，最終的にパッチが採択 (PS-M)，不採択 (PS-A)，再投稿 (Re-PS) と評価結果が報告された 51,671 件のレビュー票を対象とした．

レビューアからの評価は，レビューアが評価を報告した時に，レビュー票のコメント欄に自動的に生成されるメッセージのパターン（表 1）から計測する．自動生成されるコメントには点数（+2, +1, 0, -1, -2）が割り当てられており，その点数がパッチに対する評価となる．+2 が最も良い評価で，-2 が最も悪い評価となり，限られた開発者（コアレビューア）のみが+2 または-2 の評価をつけることができる．基本的には+2 と評価されたパッチのみがプロダクトに統合される．

表 1 パッチ再投稿予測モデルに使用するメトリクスとその選定理由

観点	メトリクス名	定義	選定理由
変更対象プログラム	#Files	パッチに存在するファイルの変更ファイル数	変更ファイル数が多いほど，修正漏れが発生する可能性が高くなる．
	#SubSystems	パッチに存在するファイルのサブシステム数	広範囲のモジュールの修正は，修正漏れが発生する可能性が高くなる．
変更	#AddLines	追加行数	追加行数が多いほど，間違った修正をする可能性が高くなる．
	#DeleteLines	削除行数	削除行数が多いほど，間違った修正をする可能性が高くなる．
	Core Reviewer	レビューアがコアコミッターであるか否か	開発の経験も豊富なコアレビューアが作成したパッチは，再投稿が発生する可能性が低いと考えられる．
	#Experiences	パッチ作成者が，過去にパッチが採択された経験の数	パッチの開発/採択の経験が多いほど，再投稿が起きにくいパッチを生成しやすい．
検証	Time	１回目の投稿から，判定 (採択，不採択，再投稿) までに要した時間	レビューアの合意形成が取れない，パッチの理解が困難などの理由から時間がかかっていることが考えられる．
	#Positive	肯定的な評価結果の報告数	肯定的な評価結果が多いほど，再投稿となる可能性が低いと考えられる．
	#Normal	肯定的でも否定的でもない評価結果の報告数	肯定でも否定でもない評価は，採択できるか否か判断することが難しく，再投稿の可能性があると考えられる．
	#Negative	否定的な評価結果の報告数	否定的な評価結果が多いほど，再投稿もしくは不採択になる可能性が高いと考えられる．
	#Comment	１回目の投稿から，最終評価結果 (採択，不採択，再投稿) までに投稿されたコメント数	議論が盛んに行われるほど，合意形成が取りにくいため再投稿が起きやすいと考えられる．
	#OwnComments	パッチ投稿者が投稿したコメントの総数	議論に対するパッチ投稿者の積極的な参加は，レビューアへの説明が必要であり，レビューアを納得させることができれば再投稿の必要はないと考える．

表 2 評価結果を表す Gerrit の自動投稿コメント

score	コミッターの種類	コメント内容
+2	core committer	”Looks good to me, approved”
		”Looks good to me”
+1	commiter	”Looks good to me, but someone else must approve”
		”Works for me”
		”Verified”
0	commiter	”No score”
-1	commiter	”I would prefer that you didn’t submit this”
-2	core commiter	”I would prefer that you didn’t merge this”
		”Do not submit”

表 3 メトリクスの重要度ランキング

グループ	メトリクス
1	#Positive
2	#Negative
3	# Comments
4	# OwnComments
5	Time
6	# AddLines
7	# Files
8	# Experiences
9	# DeleteLines
10	# SubSystems
11	Core Reviewer
12	# Normal

4.2 実験結果

本論文で構築したパッチ再修正モデルの精度は，再現率が 0.686, 適合率が 0.891, F 値が 0.775, AUC が 0.912 であった．従って，本モデルはパッチが再投稿されるか否かをランダムに予測するよりも高い精度で予測できることが分かった．

次に，本実験で得られた説明変数の重要度ランキングを計測した結果を表 3 に示す．表 3 について，重要度の順位が統計的に似ているメトリクスをグルーピングした．上位グループのメトリクスであるほど，重要度が高いことを示す．実験結果より，#Positive が最も重要であり，次いで，#Negative と#Comments が高い重要度を示した. パッチの再修正が必要であるか否かを判断するために，#Positive，#Negative，#Comments が影響していることが分かった．一方で，パッチ投稿者の経験度 (#Experiences)，パッチの編集量 (#AddLines, #DeleteLines)，パッチの規模 (#SumSystem, #Files) を示すメトリクスはパッチの再投稿を予測するために強く影響していないことが分かった．従って，検証作業に関するメトリクスがパッチの再投稿を予測する上で重要であるため，本論文では，3 つのメトリクス（#Positive, #Negative, #Commetns）について，レビューアの最終的な評価結果（採択 (PS-M)，不採択 (PS-A)，再投稿 (Re-PS)）を分析する．

[**#Positive**] レビューアの最終的な評価結果が PS-M，PS-A，および，Re-PS のパッチを対象に，#Positive の分布を図 2 に示す．PS-M は 1 件以上の肯定的な評価を報告されることが多いことが分かった．また，半数

表 4 #Positive と#Negative の関係についての調査

	#Positive = 0		0 < #Positive < 2		2 ≤ #Positive	
#Negative = 0	**Pattern1**		**Pattern3**		**Pattern5**	
	PS-M	0 件	PS-M	20,188 件	PS-M	14,110 件
	PS-A	0 件	PS-A	0 件	PS-A	2 件
	Re-PS	0 件	Re-PS	3,922 件	Re-PS	870 件
#Negative > 0	**Pattern2**		**Pattern4**		**Pattern6**	
	PS-M	0 件	PS-M	218 件	PS-M	368 件
	PS-A	5,019 件	PS-A	1,035 件	PS-A	213 件
	Re-PS	4391 件	Re-PS	1,116 件	Re-PS	219 件

の Re-PS は 1 件以下の肯定的な評価が報告されていることが分かった．全ての組み合わせ（PS-M と PS-A，PS-A と Re-PS，Re-PS と PS-M）に対してマンホイットニーの U 検定により統計的有意差（有意水準 5%）を確認した．

[**#Negative**] レビューアの最終的な評価結果が PS-M，PS-A，および，Re-PS のパッチを対象に，#Negative の分布を図 3 に示す．PS-A は 1 件以上の否定的な評価を報告されることが多く，PS-M と Re-PS よりも多いことが分かった．また，一部の Re-PS は 1 件以上の否定的な評価が報告されていることが多く，図 2 と図 3 の結果より，Re-PS は肯定，否定の両方の意見が報告され合意形成に失敗している可能性がある．全ての組み合わせに対してマンホイットニーの U 検定により統計的有意差（有意水準 5%）を確認した．

[**# Comments**] レビューアの最終的な評価結果が PS-M，PS-A，および，Re-PS のパッチを対象に，#Comments の分布を図 4 に示す．PS-A は PS-M と Re-PS よりもコメント数が多く，図 3 の結果より，コメントの多くは否定的な内容の報告であると考えられる．

5. 考察

再修正されるパッチを予測するために#Positive，#Negative は強く影響するメトリクスであることが分かった．本章では，パッチを評価したレビューア数と最終評価結果との関係について考察し，その後，本論文の制約を述べる．

5.1 レビューアの評価数による最終評価結果の違い

本論文では，パッチが投稿されてからレビューアの最終評価結果が報告されるまでの間に，パッチの再投稿が必要であるか否かを判断することを想定し，パッチ再投稿予測モデルを構築した．本節では，レビューアが最終評価を決定した理由を，パッチ再投稿予測モデルに強く影響するレビューアの肯定的，否定的な意見数に基づき分類し，実際の事例を通して考察する．

本論文で対象としたレビュー票を，図 2 から肯定的なコメント数と，図 3 から否定的なコメント数から，6 つのパターンに分類した．肯定的な意見が報告された数（#Positive）は，図 2 より中央値が 2 である．そこで，中央値を基準に分類する．さらに，肯定的な意見が 0 件である場合は，1 件存在する場合と大きく異なるため，0 件の場合も分類する（#Positive = 0，0 < #Positive < 2，2 ≤ #Positive）．

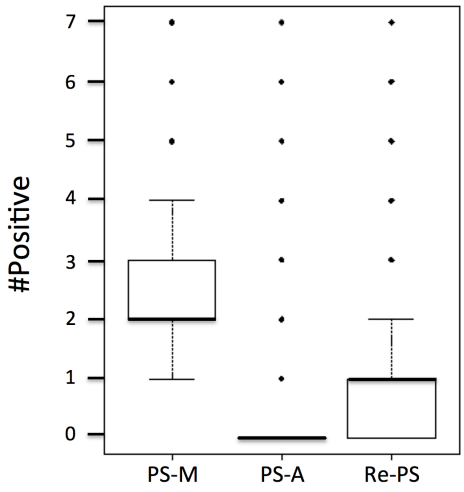

図 2 PS-M, PS-A, Re-PS の#Positive の分布

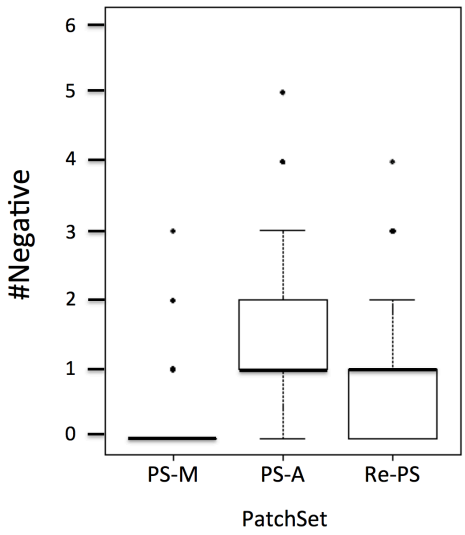

図 3 PS-M, PS-A, Re-PS の#Negative の分布

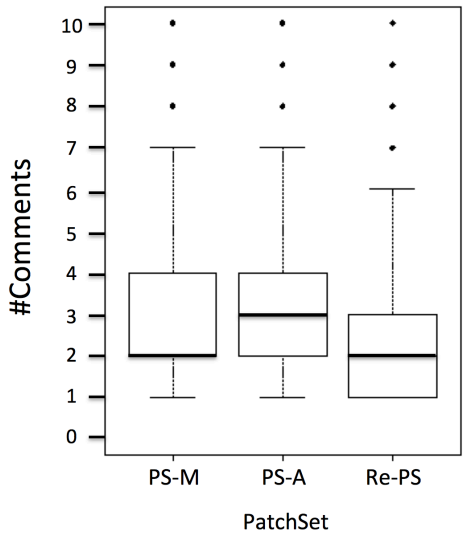

図 4 PS-M, PS-A, Re-PS の#Comments の分布

また，否定的な意見が報告された数（#Negative）は図 3 から，否定的な意見がほとんど確認されなかったため，0 件の場合と 1 件以上の場合に分類する (#Negative = 0, #Negative > 0)．

Pattern1: 本論文では，レビューアからの意見が 1 つ以上報告されたレビュー票に制約しているため Pattern1 は 0 件であった．

Pattern2: 肯定的な意見が報告されず，否定的な意見のみ報告される場合，初めて投稿したパッチがプロダクトに統合されることはなく，1 回目で破棄される，または，再投稿が要求される．特に，不完全な不具合修正，または，新たな不具合の混入が指摘されていることが多かった [*4] [*5]．

Pattern3: 否定的な意見が報告されず，肯定的な意見が 1 件報告された場合，全体の PS-M の約 57%が pattern3 に存在した．しかし，否定的な意見が報告されていないにもかかわらず再投稿が求められるパッチが存在する．それは，パッチ投稿者がレビューアからの軽微な修正に対応している，または，レビューアから指摘はないがパッチ投稿者が自主的に修正しているパッチを確認した [*6] [*7]．否定的な意見がなく，肯定的な意見の場合，再投稿を要する場合はあるが，統合される可能性が高いパッチであると考えられる．

Pattern4: 肯定的，否定的な意見を報告するレビューアを共に確認した．Pattern3 に比べて否定的な意見を報告するレビューアも存在するため不採択と判断されるパッチが多かったと示唆される．しかし，レビューアが否定的な意見を投稿した後すぐに肯定的な意見を投稿するケースが存在したため Pattern4 に分類される事例もあった [*8]．

Pattern5: 肯定的な意見を報告するレビューアが多く，Pattern5 では最終評価結果が PS-M のうち約 40%存在した．しかし，再投稿が要求される場合もあり，パッチ投稿者が自身がパッチ提出後，すぐに統合し，その後に自動テストシステムやレビューアによるレビューによって，不具合が見つかる場合が存在した.[*9]．パッチ投稿者はパッチに自信がある，もしくは，コアレビューアであるため自信の判断で統合する傾向がある可能性がある．

Pattern6: 肯定的，否定的な意見を報告するレビューアを共に確認した．Pattern5 に比べて否定的な意見を報告するレビューアも存在するため不採択と判断されるパッチが多かったと示唆される．しかし，Pettern4 と同様に、レビューアが否定的な意見を投稿した後すぐに肯定的な意見を投稿するケースが存在したため Pattern6 に分類される事例もあった [*10]．また，Pattern6 の場合，PS-M，PS-A，Re-PS のレビュー票数の差が小さく，再投稿が発生するか否か判断しにくいケースである．

5.2 制約

構成概念妥当性: 本論文で扱った変数（メトリクス）は，変更対象プログラム，変更，検証を完全に評価できたとは限らない．例えば，LOC(Lines of Code) やサイクロマティック複雑度などによる評価も必要であると考えられる．

内的妥当性: 今回対象としたレビューアのコメントの内容は，評価結果に限定した．しかし，実際のレビュー票には，評価結果以外のコメントも存在する．それらの影響を今後考慮する必要がある．

外的妥当性: 本実験は，Qt プロジェクトだけを対象とした．しかし，それ以外のプロジェクトでも同様の結果が得られるかまだ検証できていない．他のプロジェクトに対しても，本実験と同様の傾向が得られるか調査することが今後の課題である．

信頼性: 本実験では，予測精度の信頼性を確保するために，1000 回の交差検定を行った．また，重要度ランキングについても，1000 回の交差検定を行った結果に基づいて求めた．ただし，本論文が対象とした Qt プロジェクトのレビュー表が更新されたデータセットを対象とする場合，同様の結果が得れるとは限らない．

6. おわりに

本研究では，再投稿が必要となるパッチの予測モデルを構築した．Re-PS 予測精度は，再現率が 0.686, 適合率が 0.891, F 値が 0.775 であり，予測に影響するメトリクスは，パッチに対するレビューアのコメント (#Positive, #Negative, # Comments) であった．肯定的な意見のみが存在する場合は，1 回目で採択される可能性が高く（Pattern 3, 5），一方で，否定的な意見のみが存在するときは，不採択もしくは再投稿が発生する可能性が高いことが分かった（Pattern 2）．ただし，肯定的な意見と否定的な意見が共に投稿される場合は，再投稿を要する場合があるが，レビューアの誤判断で否定的な意見が投稿される場合があることを確認した（Pattern 4, 6）．

本論文は，レビューアの評価結果がパッチ再投稿の予測に寄与することが分かった．しかしながら，本モデルの構築に使用した変数 (メトリクス) は，パッチを投稿してからレビューアの評価結果 (採択/再投稿/不採択) を待つ間に

[*4] https://codereview.qt-project.org/#/c/13312/
[*5] https://codereview.qt-project.org/#/c/58132/
[*6] https://codereview.qt-project.org/#/c/48/
[*7] https://codereview.qt-project.org/#/c/80/
[*8] https://codereview.qt-project.org/#/c/6295/
[*9] https://codereview.qt-project.org/#/c/31132/
[*10] https://codereview.qt-project.org/#/c/19299/

発生する情報を基に計測している．従って，パッチを投稿する前に，将来的に再投稿が必要となるパッチか否かを予測することができない．今後は，パッチを投稿する前に，再投稿が起きやすいパッチを予測する手法を提案することで，効率的な検証作業を支援できると考える．

謝辞 本研究の一部は，頭脳循環を加速する戦略的国際研究ネットワーク推進プログラムによる助成を受けた．

参考文献

[1] Raymond, E. S.: *The cathedral and the bazzar: musings on linux and open source by an accidental revolutionary*, O'Reilly and Associates, Sebastopol, CA (1999).

[2] Bacchelli, A. and Bird, C.: Expectations, Outcomes, and Challenges of Modern Code Review, *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*, pp. 712–721 (2013).

[3] Bird, C., Gourley, A. and Devanbu, P.: Detecting Patch Submission and Acceptance in OSS Projects, *Proceedings of the Fourth International Workshop on Mining Software Repositories (MSR'07)*, pp. 26–29 (2007).

[4] Jiang, Y., Adams, B. and German, D. M.: Will My Patch Make It? And How Fast?: Case Study on the Linux Kernel, *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR'13)*, pp. 101–110 (2013).

[5] Rigby, P. C. and Storey, M.-A.: Understanding Broadcast Based Peer Review on Open Source Software Projects, *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*, pp. 541–550 (2011).

[6] Weißgerber, P., Neu, D. and Diehl, S.: Small patches get in!, *Proceedings of the 5th International Working Conference on Mining Software Repositories (MSR'08)*, pp. 67–76 (2008).

[7] Tao, Y., Han, D. and Kim, S.: Writing Acceptable Patches: An Empirical Study of Open Source Project Patches, *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME'14)*, pp. 271–280 (2014).

[8] Nurolahzade, M., Nasehi, S. M., Khandkar, S. H. and Rawal, S.: The role of patch review in software evolution: an analysis of the mozilla firefox, *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution and Software Evolution Workshops (IWPSE-Evol'09)*, pp. 9–18 (2009).

[9] Efron, B.: Estimating the error rate of a precision rule: improvements on cross-validation, *Journal of the American Statical Association*, Vol. 78, No. 382, pp. 316–331 (1983).

[10] Lee, T., Nam, J., Han, D., Kim, S. and In, H. P.: Micro interaction metrics for defect prediction, *Proceedings of the 8th Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering (ESEC/FSE'11)*, pp. 311–321 (2011).

[11] Breiman, L.: Random forests, *Machine learning*, Vol. 45, No. 1 (2001).

[12] Scott, A. J. and Knott, M.: A Cluster Analysis Method for Grouping Means in the Analysis of Variance, *Biometrics*, Vol. 30, No. 3, pp. 507–512 (1974).

[13] Tantithamthavorn, C., McIntosh, S., Hassan, A. E., Ihara, A. and Matsumoto, K.: The Impact of Mislabelling on the Performance and Interpretation of Defect Prediction Models, *Proceedings of the 37th International Conference on Software Engineering (ICSE'15)*, pp. 812–823 (2015).

[14] McIntosh, S., Kamei, Y., Adams, B. and Hassan, A. E.: The Impact of Code Review Coverage and Code Review Participation on Software Quality: A Case Study of the Qt, VTK, and ITK Projects, *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR'14)*, pp. 192–201 (2014).