OSS開発における不具合修正プロセスの改善に向けたモジュールオーナー候補者推薦

Module Owner Candidate Recommendationfor Improving Bug-Fixing Process in OSS Development

柏 祐太郎 * **山谷 陽亮** † **大平 雅雄** ‡

あらまし 大規模 OSS プロジェクトでは，モジュールオーナーと呼ばれるプロジェクトのコア開発者が，プロジェクトに日々大量に投稿されるパッチを検証している．モジュールオーナーは，ソフトウェアの品質管理の面で重要な役割を担っているため，プロジェクトに参加する一般の開発者にモジュールオーナーを任せるかの判断は，既存のモジュールオーナーやコアメンバーによって慎重に行われる．慎重な判断が行われる故，コミット権限を持たない開発者に比べモジュールオーナーは少人数で構成されている．パッチが大量に投稿される現状に対して，少数のモジュールオーナーで行うパッチの検証は効率的とは言えない．

本研究ではモジュールオーナーとして活動できる開発者をより早く見つけ出すために，特定のモジュールに対してモジュールオーナーとなる開発者を予測するモデルを構築する．実際にモジュールオーナーに昇格した開発者の昇格前の活動量メトリクスおよびコミット権限を持たない開発者の活動量メトリクスそれぞれをモジュールごとに計測し，ロジスティック回帰分析により学習することで，予測モデルの構築を行う．

1 はじめに

大規模 OSS（Open Source Software）プロジェクトでは，日々大量の不具合が報告されており，報告された不具合を修正するためにプロジェクトに参加する開発者によって不具合修正のソースコードの差分（パッチ）が投稿される．投稿されたパッチはコミット権限（ソースコードに変更を加える権限を指す）を持つ開発者である**モジュールオーナー**により検証され，パッチの内容に問題がないことが確認されればバージョン管理システムに変更内容をコミットし，不具合の修正が完了する．モジュールオーナーはソフトウェアの品質管理の面で重要な役割を担うため，モジュールオーナーに昇格させる（開発者にコミット権限を付与する）かどうかの判断は慎重に行われる．その結果，コミット権限を持たない開発者に比べ**モジュールオーナーは少人数で構成されており，大規模 OSS プロジェクトでは，大量の不具合を修正するために投稿されるパッチも膨大な数に上るため，パッチの検証が効率的に行われず，不具合の修正時間が長期化している**ことが問題となっている．パッチの検証を効率化するために，モジュールオーナーとして活動できる開発者をより早く見つけ出すことが，効果的と考えられる．しかしながら，開発者は数万人に及ぶこともあり，既存のモジュールオーナーやコアメンバーが**それぞれの開発者における過去の活動を把握し，コミット権限を付与すべき開発者を特定することは現実的に困難**である．また，コミット権限の付与の判断は，既存のモジュールオーナーやコアメンバーの経験的な判断によるもので，具体的な指標がないことも，コミット権限の付与が滞っている原因とも考えられる．

そのため，本研究ではモジュールオーナーとして活動できる開発者をより早く見つけ出すために，特定のモジュールに対してモジュールオーナーとなる開発者を予測するモデルを構築する．実際にモジュールオーナーに昇格した開発者の昇格前の活動量メトリクスおよびコミット権限を持たない開発者の活動量メトリクスそれぞ

*Yutaro Kashiwa, 和歌山大学および JSPS 特別研究員（DC）

†Yosuke Yamatani, 和歌山大学

‡Masao Ohira, 和歌山大学

れをモジュールごとに計測し，ロジスティック回帰分析により学習することで，予測モデルの構築を行う．

本論文の構成は次の通りである．続く 2 章では，OSS 開発における不具合修正プロセスにおいて重要な役割を担うモジュールオーナーについて説明し，モジュールオーナーの不足を解消するために行われている先行研究について述べ，先行研究の問題を指摘する．3 章では，本研究のアプローチである予測モデルの構築について述べ，4 章では，提案手法の有用性を確認するために行った評価実験について述べる．5 章と 6 章でそれぞれ，評価実験の結果とその考察を行い，7 章で本研究をまとめる．

2 OSS 開発におけるモジュールオーナー

2.1 モジュールオーナーの昇格プロセス

ボランティアの開発者によって開発・保守が行われている OSS プロジェクトでは，開発者の役割が複数存在する．Jensen らは，OSS プロジェクトのコミュニティ内における開発者の役割昇格プロセス（周辺的役割から中心的役割へ役割が変化するプロセス）について分析を行った [1]．OSS 開発において，すべての開発者に対してコミット権限を与えるとソフトウェアの品質をコントロールすることが難しくなるため，ソースコードに対する専門知識や，開発者を取りまとめる能力を持っていると判断された開発者にコミット権限が与えられている [2]．また，コミット権限を持たない一般開発者がコミット権限を与えられるまでのプロセスが OSS プロジェクトの Web ページなどで明記されており [1]，これらの情報からも，既存のモジュールオーナーやコアメンバーがモジュールオーナーに昇格させる開発者を判断していることがわかる．既存のモジュールオーナーやコアメンバーは，コミット権限を持たない開発者にコミット権限を与えるかどうかの判断を開発者のプロジェクトに対するこれまでの活動（パッチ投稿や不具合に関するコメントなど）を基に行う．しかしながら，コミット権限を与えるかどうかの判断を行うための最低限必要となる活動内容や活動量は定義されておらず，既存のモジュールオーナーやコアメンバーがモジュールオーナーを経験的に判断している現状である．

また，コミット権限を持たない開発者に対してコミット権限を与えるか否かの判断に至るまでに，1 年以上のプロジェクトへの貢献が求められることが多い．しかしながら，コミット権限を持たない開発者の活動期間が 1 年を超えると，活動に対する意欲の喪失などの理由により，プロジェクトを離脱する開発者が増えることが知られている [3]．プロジェクトに対して長期貢献をしてもらうには，モジュールオーナーに昇格すべき開発者をできるだけ早い段階で特定する必要がある．

2.2 既存の支援手法と本研究の着眼点

開発者の中からモジュールオーナーに昇格して適切な開発者を特定するために，既存のモジュールオーナーの昇格前の活動と一般開発者（コミット権限を持たない開発者）の活動を学習し，将来モジュールオーナーに昇格する開発者を予測する研究が行われている [4] [5]．先行研究 [5] では，既存のモジュールオーナーの昇格前のプロジェクト全体での活動量（パッチ投稿数やコメント投稿数など）に基づいて予測モデルを構築し，一般開発者の中からモジュールオーナーに昇格し得る開発者を予測している．しかしながら，先行研究の予測モデルは，再現率（実際にモジュールオーナーである開発者のうち，モジュールオーナーであると予測された開発者の割合）が 0.70–0.83 と高いものの，適合率（モジュールオーナーであると予測した開発者のうち，実際にモジュールオーナーであった開発者の割合）が 0.02–0.07 と著しく低いモデルとなっている．つまり，モジュールオーナーであると予測した開

[1] Eclipse プロジェクトでのコミット権限付与プロセス:
https://eclipse.org/membership/become_a_member/committer.php

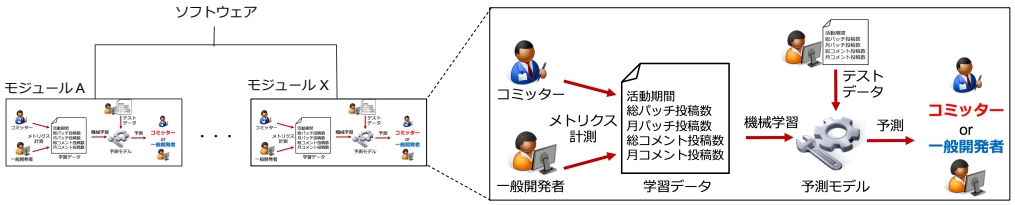

図 1　予測モデルのイメージ図

発者のほとんどは，モジュールオーナーでない結果となっている．先に述べたように，モジュールオーナーはソフトウェアの品質管理の面において重要な役割を担っている．問題のあるパッチが取り込まれてしまうと，ソフトウェアの品質を低下させる原因となってしまう．

上記の問題に対して，本研究では既存研究を改良し，適合度を改善を狙う．先行研究の予測モデルは，各モジュールオーナーがどのモジュールを担当するのにふさわしいモジュールオーナーであるかは考慮されておらず，一般開発者が将来コミット権限を持つか否かの予測を行っている． OSS プロジェクトでは各開発者が担当するモジュールが決められているため，各開発者のモジュールの適性を考慮する必要がある．そのため，本研究ではモジュールごとにモジュールオーナーとなるべき開発者を推薦するための予測モデルの構築を行う．

3　アプローチ

本章では，一般開発者の中から，モジュールオーナー候補者を予測する手法について説明する．

3.1　概要

本研究では，各モジュールに適したモジュールオーナーを推薦するための予測モデルの構築を行う．図 1 に構築する予測モデルのイメージ図を示す．まず，本手法では，モジュールオーナーに昇格した開発者と一般開発者の活動量メトリクスをそれぞれ計測し，両者の活動量メトリクスを分類器（ロジスティック回帰）に学習させて予測モデルを構築する．そして，一般開発者の活動量メトリクスを予測モデルに入力することで，将来モジュールオーナーになるか否かの予測を行う．

3.2　予測に用いるメトリクス

本研究では，先行研究 [5] で用いられているメトリクスをモジュールごとに計測する．この理由は，各モジュールオーナーが担当するモジュールが決められている OSS プロジェクトでは，あるモジュールにおいて新たなモジュールオーナーを選出する際に，モジュール内での活動を活発に行っている開発者を選出することが適切であると考えたためである．表 1 に，計測する各開発者の活動量メトリクスを示す．

なお，計測したメトリクスの分布はべき分布に従うため，計測したメトリクスに対して対数変換を行う（各メトリクスの値は 0 を含む可能性を持つため，各メトリクスの値に 1 を足した値の自然対数をとる）．これにより，メトリクスの値が大きい場合（1000 と 1001 など）に対して， 値が小さい場合（1 と 2 など） の差をより大きくすることができる．また，月コメント投稿数および月パッチ投稿数では，月初めに活動を開始（終了）した開発者と月末に活動を開始（終了）した開発者との月平均の活動量に大きな差が生じる可能性がある．そのため，計測開始時期を活動開始日の翌月とし，終了時期を活動終了日の前月までとしている．

表 1　活動量メトリクス

活動期間	開発者が初めて活動（不具合管理システム上でのパッチの投稿あるいはコメントの投稿）を行った日付から最後に活動を行った日付までの期間の日数．ただし，モジュールオーナーの場合，最後に活動を行った日付ではなく，モジュールオーナーに昇格した日付（バージョン管理システムで該当モジュールにコミットを初めて行った日付）とする．
総コメント投稿数	各開発者が特定のモジュールに対して不具合管理システム上でのコメントを投稿した回数．
月コメント投稿数	各開発者が 1ヵ月に特定のモジュールに対して不具合管理システム上でコメントを投稿した回数の中央値．
総パッチ投稿数	各開発者が特定のモジュールに対して不具合管理システム上でパッチを投稿した回数．
月パッチ投稿数	各開発者が 1ヵ月に特定のモジュールに対して不具合管理システム上でパッチを投稿した回数の中央値

3.3　予測モデル

本論文で構築する予測モデルは，コミット権限を持たない開発者が将来モジュールオーナーになるかどうかをロジスティック回帰分析を用いて予測する．

ロジスティック回帰分析は，2 値分類に優れている統計分析手法の 1 つであり，ソフトウェア工学の分野においてもよく用いられている [5] [6]．回帰分析とは，目的変数 y と説明変数 x の間に式を当てはめ，目的変数が説明変数によってどれほど説明できているのかを定量的に分析することである．ロジスティック回帰分析では，回帰式（目的変数を説明変数で計算する式）に非線形関数を用いる．ロジスティック回帰分析における回帰式を式 1 に示す．

$$P(y|x_1,\cdots,x_p) = \frac{1}{1+e^{-(\alpha_1 x_1+\cdots+\alpha_r x_r+\beta)}} \tag{1}$$

ここで，$y \in \{0,1\}$ は，2 値のクラスを表す目的変数，x_i は説明変数，α_i は回帰係数，$P(y|x_1,\cdots,x_p)$ は目的変数の条件付き確率である．

回帰式に線形関数を用いる重回帰分析では，目的変数が取り得る値の範囲に制約はないが，ロジスティック回帰分析では，ロジット変換を行っていることで目的変数の取り得る値の範囲が 0 から 1 となる．

本研究では，モジュールオーナーに昇格する場合 1 を，モジュールオーナーに昇格しない場合 0 を目的変数として学習し，予測モデルの出力値が 0.5 以上のときにモジュールオーナーに昇格すると判断することで，現在コミット権限を持たない開発者が将来モジュールオーナーに昇格するか否かの予測を行う．

4　実験

本章では，実験手順及び評価方法について述べる．

4.1　概要

本実験の目的は，モジュールのオーナーに適切な開発者を正しく推薦できるのかを確かめることである．3 つの大規模 OSS プロジェクトを対象に，先行研究 [38] で用いられている各開発者の活動量メトリクスをモジュールごとに計測し，ロジスティック回帰分析により予測モデルを構築する．構築した予測モデルを利用して，テストデータとなる開発者の活動量メトリクスを入力し，将来，モジュールオーナー

に昇格するか否かの予測を行う．予測モデルの評価には，適合率，再現率，F1 値を用いて交差検証によって行う．

4.2 対象プロジェクト

本論文では，大規模 OSS プロジェクトである Mozilla Firefox プロジェクト，Eclipse Platform プロジェクト，WebKit プロジェクトを対象にする．各プロジェクトは，長期に渡って開発・保守が行われているプロジェクトであり，開発に携わる開発者が多いため，本研究の対象プロジェクトとした．

4.3 定義

4.3.1 モジュール

本論文では，各機能を実現するためのソースファイルの集合をモジュールと定義する．プロジェクトの Wiki ページから機能を抽出し，各ソースコードがどの機能を実現しているかを特定した．Mozilla Firefox では，Web ブラウザが有する機能単位で分割を行った．ブラウザの共通部分である General，開発者向けツールである Developer Tools，ブックマーク機能を扱う Bookmark，ブラウザのツールバーやメニューバー部分に関する Bar，テーマの設定などを扱う Preferences，Web アプリに関する App に分割を行った．Eclipse Platform では，ユーザーインターフェース部分である UI，GUI 作成用ツールキットである SWT（Stantdard Widget Toolkit），ファイル等の各種リソースを管理するための Workspace に分割した．WebKit では，レンダリングライブラリである WebCore，JavaScript エンジンである JavaScript-Core，Web ページのソースの表示やデバッグのための機能を提供する Inspector に分割した．

4.3.2 モジュールオーナーと昇格日

本論文では，モジュールに対してソースファイルをコミットした開発者を，コミット先のモジュールオーナーとする．ひとりの開発者が複数のモジュールオーナーになることもありうる（確認要）．また，各モジュールオーナーが初めてモジュールに対してバージョン管理システム上で変更作業を行った日付をモジュールオーナーに昇格した日付とする．

4.4 評価指標

予測モデルの評価指標として適合率，再現率，F1 値を用いる．適合率とは，モジュールオーナーであると予測した結果のうち，実際にモジュールオーナーであった割合を指し，再現率とは，実際にモジュールオーナーである開発者のうち，モジュールオーナーと予測された割合を指している．また，適合率と再現率はトレードオフの関係にあるため，適合率と再現率の調和平均を取った値である F1 値を用いて予測精度の評価を行う．F1 値は式 (2) によって定義される．

$$\text{F1 値} = \frac{2 \times \text{適合率} \times \text{再現率}}{\text{適合率} + \text{再現率}} \tag{2}$$

本研究では，この 3 つの評価指標のうち，適合率を重視する．

4.5 実験手順

本研究で行う評価実験の手順を以下に示す．以下の手順は，各プロジェクトのそれぞれのモジュールに対して実施する．

手順 1：データセットの分割 モジュールオーナーと一般開発者のそれぞれのデータをランダムに 2 分割し，半分を学習データ，もう半分をテストデータとする．例えば，モジュールオーナーのデータが 100 件，一般開発者のデータが 10,000 件あったとする．このとき，ランダムに抽出したモジュールオーナーのデータ

表 2　各プロジェクトにおける予測精度

プロジェクト	モジュール名	適合率	再現率	F1 値
Firefox	App	0.571	0.563	0.552
	Bar	0.500	0.750	0.590
	Bookmark	0.480	0.560	0.511
	Developer Tools	0.800	0.786	0.800
	General	0.455	0.585	0.511
	Preferences	0.552	0.613	0.587
Platform	UI	0.455	0.667	0.545
	SWT	0.250	0.800	0.375
	Workspace	0.600	0.667	0.667
WebKit	Build	0.586	0.718	0.646
	Layout	0.452	0.559	0.500
	WebCore	0.626	0.591	0.608
	Inspector	0.533	0.762	0.627
	WebKit2	0.500	0.500	0.500
	JavaScriptCore	0.351	0.565	0.433

を 50 件，同じくランダムに抽出した一般開発者のデータを 5,000 件取得し，学習データとする．そして抽出されなかった残りのデータをテストデータとする．

手順 2：学習データのバランス調整　一般開発者に比べモジュールオーナーの割合は著しく低いため，正しい予測ができない恐れがある [7] [8]．そのため，学習データのバランスの調整を行うために，一般開発者の学習データをモジュールオーナーの学習データと同じ数に合わせる．例えば，モジュールオーナーの学習データが 50 件，一般開発者の学習データが 5,000 件あった場合，一般開発者の学習データから 50 件ランダムに抽出し，ランダムに抽出した 50 件のデータを学習データとする．

手順 3：学習　手順 2 で作成した学習データの学習を行う．本研究では，学習アルゴリズムとしてロジスティック回帰を用いる．モジュールオーナーである開発者を 1，一般開発者である開発者を 0 として学習する．

手順 4：予測　手順 3 で学習した結果に基づき，テストデータを用いて予測精度を算出する．予測精度の評価指標には，適合率，再現率，F1 値を用いる．予測の結果，目的変数が 0.5 以上であった場合，モジュールオーナーであると判断する．

手順 5：ホールドアウト検証　学習データとテストデータはランダム抽出により作成されるため，予測結果が毎回変化する．そのため，手順 1 から手順 4 を 1,000 回繰り返し，1,000 回行った結果の中央値を最終的な予測精度とする．

5　実験結果

各モジュールにおいて構築した予測モデルについての予測精度を表 5 に示す．各評価指標の値は，ホールドアウト検証により 1,000 回行われた結果の中央値を示している．

まず Firefox における，App モジュールでは，適合率 0.571，再現率 0.563 という予測精度の結果となった．これは，モジュールオーナーであると予測した開発者のうち，2 人に 1 人以上は実際にモジュールオーナーに昇格した開発者であり，実際にモジュールオーナーに昇格した開発者の約 56%を予測できたことを示している．また，DeveloperTools モジュールでは，適合率が 0.800 となっており，他のモジュールよりも高い精度でモジュールオーナーを予測できていることがわかった．表 5 に示す結果から，DeveloperTools モジュール以外のモジュールでは，適合率が 0.5 前

後となっており，モジュールオーナーであると予測した開発者のうち，2 人に 1 人は実際にモジュールオーナーに昇格した開発者であった．

次に Eclipse Platform プロジェクトでは，Workspace モジュールが最も適合率が高くなり，その値は，0.600 であった．また，最も適合率が低くなったモジュールは SWT モジュールで適合率が 0.250 となった．SWT モジュール以外のモジュールでは，適合率が 0.5 前後となっており，モジュールオーナーであると予測した開発者のうち，2 人に 1 人は実際にモジュールオーナーに昇格した開発者であった．

最後に WebKit プロジェクトでは，最も適合率が高くなったモジュールは WebCore で，適合率が 0.626 となった．一方，最も適合率が低くなったモジュールは JavaScriptCore で 0.351 となった．JavaScriptCore モジュール以外のモジュールでは，適合率が 0.5 前後となっており，モジュールオーナーであると予測した開発者のうち，2 人に 1 人は実際にモジュールオーナーに昇格した開発者であった．

6　考察

6.1　予測モデルの有用性

実験の結果，プロジェクトやモジュールによってばらつきはあるものの，予測モデルの適合率が 0.5 前後となった．0.5 である適合率は，モジュールオーナーであると予測した開発者のうち，半数が実際のモジュールオーナーであることを意味している．

Eclipse Platform プロジェクトの UI モジュールでは，一般開発者の数が 194 人に対してモジュールオーナーの数が 25 人であり，約 9 人の開発者の中に 1 人のモジュールオーナーしか存在しない．しかし，本研究で構築した予測モデルはモジュールオーナーであると予測した開発者のうち，2 人に 1 人は実際のモジュールオーナーが存在するため，本研究で構築した予測モデルによりモジュールオーナー候補者を 2 人推薦することで，既存のモジュールオーナーやコアメンバーは，この推薦された開発者の中から，コミット権限を与える開発者を決定すればよい．そのため，本研究で構築した予測モデルにより，既存のモジュールオーナーやコアメンバーがコミット権限を付与する意思決定を支援できると考えられる．

6.2　閾値が予測精度に与える影響

本手法では，ロジスティック回帰分析の出力値が閾値 0.5 以上であった場合に，入力された活動量メトリクスをもつ開発者がモジュールオーナーであると予測する．本実験における閾値は，ロジスティック回帰の出力値の範囲（0 から 1）の中間の値を利用した．ロジスティック回帰分析の出力値はモジュールオーナーである確率を意味しており，閾値を変更することで，適合率や再現率を制御することができる．例えば，閾値を 0.9 に設定することは，モジュールオーナーである確率が 0.9 以上である場合にモジュールオーナーであると判断することを指す．そのため適合度が向上することが見込まれる．そこで本節では，閾値を変化させた場合での予測精度の影響について考察する．

Mozilla Firefox プロジェクトの Bookmark モジュールで，閾値を 0.1 から 0.9 まで 0.1 ずつ変化させて再実験し，得られた各評価指標の遷移を図 2 に示す．図から，閾値を高く設定することにより，適合率が高くなることがわかる．閾値を 0.9 に設定すると，適合率が 0.8 を超える結果となった．また，閾値を高く設定することにより，再現率が低下し，モジュールオーナーであると予測できる開発者が減少することがわかる．再現率の低下に伴い，F1 値も低くなっていることがわかる．この結果は他のモジュールでも同様な結果となった．これらの結果から，閾値を高く設定することにより，適合率が高くなるのものの，再現率が低くなることがわかった．OSS プロジェクトは日々進化しており，時期よってプロジェクトの方針が変化する [9]．そのため，本予測モデルを利用する場合，プロジェクトの時期や方針によって

表 3　開発者 A の各モジュールでの活動量メトリクス

モジュール名	活動期間	総パッチ	月パッチ	総コメント	月コメント
App	342	2	0	17	0
Bookmark	364	55	1	182	3
General	379	46	0	117	2
Preferences	216	0	0	1	0

表 4　開発者 B の各モジュールでの活動量メトリクス

モジュール名	活動期間	総パッチ	月パッチ	総コメント	月コメント
Bar	357	3	0	13	0
App	172	1	0	2	0
Bookmark	169	24	1	52	6
General	240	0	0	4	0

閾値を変化させることが求められる．

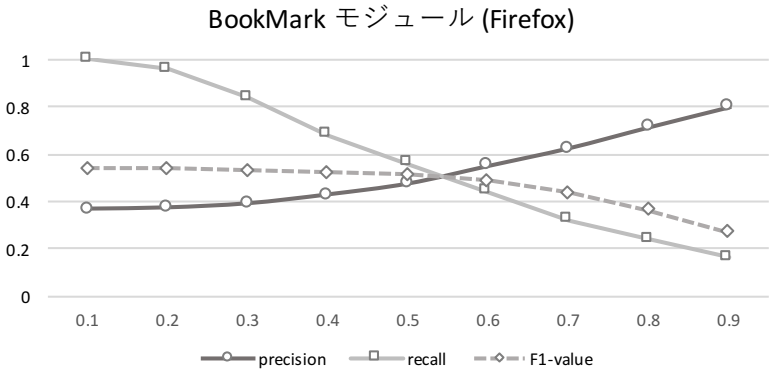

図 2　閾値を変化させた場合の各評価指標の遷移

6.3　予測精度向上に向けて

本節では，予測精度を向上させるため，正しく予測することのできなかった開発者に着目し，正しく予測できなかった理由について考察する．正しく予測することのできなかった開発者は 2 通り存在する．それは False Negative となった開発者と False Positive になった開発者である．False Negative になった開発者とは，モジュールオーナーであるのにも関わらず，一般開発者と予測された開発者を指し，False Positive となった開発者とは，一般開発者であるのにも関わらず，モジュールオーナーであると予測された開発者である．以下に，False Negative となった開発者と False Positive となった開発者それぞれについて述べる．

6.3.1　False Negative となった開発者

False Negative となったモジュールオーナーは，正しく予測することができていたモジュールオーナーに比べ，モジュール内での活動量が低い傾向にあった．そのため，モジュール内での活動量が低い開発者がなぜモジュールオーナーに昇格したのかを理解するために，False Negative となった開発者のモジュールオーナー昇格

以前の活動を分析した．以下ではいくつかの具体例を挙げて説明する．

Mozilla Firefox プロジェクトの App モジュールにおいて，False Negative となった開発者の 1 人（開発者 A とする）の App モジュール昇格以前の各モジュールでの活動量メトリクスを表 3 に示す．3 に示すように，開発者 A は App モジュールでの活動はあまり行っていないものの，General モジュールおよび Bookmark モジュールで活発に活動していることがわかる．

また，Mozilla Firefox プロジェクトの Bar モジュールにおいて，False Negative となった別の開発者（開発者 B とする）の Bar モジュール昇格以前の各モジュールでの活動量メトリクスを表 4 に示す．表 4 に示すように，開発者 B は Bar モジュールでの活動はあまり行っていないものの，Bookmark モジュールで活発に活動していることがわかる．

これらの結果は，他のモジュールでも同様に確認することができた．ここから，False Negative となった開発者は，対象モジュールでの活動はあまり行っていないものの，他のモジュールにおいて活発に活動している開発者を含むことがわかった．他のモジュールで与えられたコミット権限により開発者 A および B は App モジュールおよび Bar モジュールのモジュールオーナーとしても活動できるが，実質的なモジュールオーナーでなかった可能性がある．また，ソフトウェアは複数のモジュールによって構成されており，あるモジュールを変更すると，変更の影響が波及する依存関係にあるモジュールを変更する必要がある．そのため，ある特定のモジュールのモジュールオーナーである開発者が，依存関係のあるモジュールを変更することも珍しくない [10]．今後，依存関係にあるモジュールにおける活動量メトリクスを考慮した予測モデルを構築することにより，適合率の向上を見込むことができる．

6.3.2 False Positive となった開発者

False Positive となった開発者は，正しく予測することができていた開発者に比べ，モジュール内での活動量が大きい傾向にあった．そのため，モジュール内において，活発に活動している開発者がなぜモジュールオーナーでないのかを理解するために，False Positive となった開発者を分析する．

False Positive となった開発者の活動の一部を確認したところ，一部の開発者において，不具合管理システムとコミットログで異なるメールアドレスを用いているが確認できた（Firefox プロジェクトの App モジュールでは 6 人中 2 人）．本論文の実験では，不具合管理システムとバージョン管理システムで異なるメールアドレスを用いている開発者が存在した場合，正しいモジュールオーナーの特定を行うことができない．そのため，複数のメールアドレスの中から同一の開発者を特定する手法 [11] などの利用する必要がある．

7 おわりに

本研究では，人手の不足しているモジュールに適したモジュールオーナーを推薦するために，先行研究よりも細粒度なモジュールオーナーとなるべき開発者をモジュールごとに特定するための予測モデルを構築した．先行研究 [5] で用いられている活動量メトリクスをモジュールごとに計測し，ロジスティック回帰分析により予測モデルの構築を行った．

3 つの大規模 OSS プロジェクト（Eclipse Platform プロジェクトおよび Mozilla Firefox プロジェクト WebKit プロジェクト）を対象に，評価実験を行い，実験の結果，モジュールによってばらつきはあるものの，適合率が 0.5 前後で予測することができていることがわかった．また，モジュールオーナーであると判別するロジスティック回帰分析の目的変数である閾値を変化させることにより，プロジェクトの時期や方針に従った予測モデルを構築することができることがわかった．今後の課題は，パッチの投稿数やコメント投稿数などの，開発者の活動量だけでなく，活動の質や内容を定量化することである．

謝辞 本研究の一部は，JSPS 特別研究員奨励費（JP17J03330）および JSPS 科研費（基盤 (A): JP17H00731，基盤 (C): JP15K00101）による助成を受けた．

参考文献

[1] Chris Jensen and Walt Scacchi. Role migration and advancement processes in OSSD projects: A comparative case study. In *Proceedings of the 29th International Conference on Software Engineering (ICSE'07)*, pp. 364–374, 2007.

[2] Karl Fogel. *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly Media, 2005.

[3] Christian Bird, Alex Gourley, Prem Devanbu, Anand Swaminathan, and Greta Hsu. Open borders? Immigration in open source projects. In *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR'07)*, p. No.6, 2007.

[4] Akinori Ihara, Yasutaka Kamei, Masao Ohira, Ahmed E. Hassan, Naoyasu Ubayashi, and Kenichi Matsumoto. Early identification of future committers in open source software projects. In *Proceedings of the 14th International Conference on Quality Software (QSIC'14)*, pp. 47–56, 2014.

[5] 伊原彰紀, 亀井靖高, 大平雅雄, 松本健一, 鵜林尚靖. OSS プロジェクトにおける開発者の活動量を用いたコミッター候補者予測. 電子情報通信学会論文誌, Vol. 95, No. 2, pp. 237–249, 2012.

[6] Mohammad Gharehyazie, Daryl Posnett, Bogdan Vasilescu, and Vladimir Filkov. Developer initiation and social interactions in OSS: A case study of the apache software foundation. *Empirical Software Engineering*, Vol. 20, No. 5, pp. 1318–1353, 2015.

[7] Haibo He and E.A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 21, No. 9, pp. 1263–1284, 2009.

[8] Art B. Owen and Yi Lin. Infinitely imbalanced logistic regression. *Machine Learning*, Vol. 8, pp. 761–773, 2007.

[9] Yunwen Ye, Kumiyo Nakakoji, Yasuhiro Yamamoto, and Kouichi Kishida. The co-evolution of systems and communities in free and open source software development. In Stefan Koch, editor, *Free/Open Source Software Development*, chapter 3, pp. 59–82. Idea Group Publishing, Hershey, PA., 2004.

[10] Christian Bird, Nachiappan Nagappan, Brendan Murphy, Harald Gall, and Prem Devanbu. Don't touch my code! Examining the effects of ownership on software quality. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (FSE'11)*, pp. 4–14, 2011.

[11] Christian Bird, David Pattison, Raissa D'Souza, Vladimir Filkov, and Premkumar Devanbu. Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'08)*, pp. 24–35, 2008.