

2015年度 修士論文

モジュールオーナーの適性を考慮した コミッター候補者予測

2016年2月

和歌山大学大学院システム工学研究科

学生番号: 60150120

山谷 陽亮

Predicting Candidates for Committers Considering Capability of Module Owners

by

Yosuke Yamatani

Master's Thesis

Graduate School of Systems Engineering

Wakayama University

February 2016

概要

近年、OSSの社会的普及により、OSSを利用するユーザーが年々増加している。ユーザーの多い大規模なOSSプロジェクトでは、日々大量の不具合が報告されており、報告された不具合は、ボランティアの開発者が中心となり、不具合修正のためのパッチが投稿され修正される。投稿されたパッチはバージョン管理システムに直接変更を加える権限を持つ開発者であるコミッターにより検証され、問題がないことが確認されればリポジトリに取り込まれることで不具合の修正が完了する。コミッターはソフトウェアに対して直接変更を加える権限を持つため、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。そのため、プロジェクトに参加す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、既存のコミッターからの推薦や承認によってコミット権限が付与される。このように、開発者にコミット権限を付与するかどうかの判断が慎重に行われているため、コミット権限を持たない開発者に比べコミッターは少人数で構成されている。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り、比較的少人数で構成されるコミッターのみではパッチの検証を効率的に行えなくなっている現状である。コミッターの不足を解消するために、ボランティアの開発者の中からコミッターに昇格可能な開発者を予測するための研究が盛んに行われている。先行研究では、既存のコミッターの昇格前の活動量メトリクス（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、ボランティアの開発者の中からコミッターに昇格し得る開発者を予測している。実際の大規模OSSプロジェクトでは、各コミッターが担当すべきモジュールが決まっているため、コミッターの不足を解消するためには、コミッターが不足しているモジュールを担当することのできる開発者を推薦する必要がある。しかし、特定のモジュールを担当するコミッターを推薦するための予測モデルが構築されていない。本研究では、特定のモジュールを担当するコミッターをモジュールオーナーと呼び、モジュールオーナーとなるべき開発者を特定するための予測モデルを構築する。予測モデル構築には、モジュールごとに計測した活動量メトリクスおよび、モジュールオーナーの適性を示すメトリクスであるパッチ貢献度を用いる。本研究では、モジュールオーナーとなる開発者を予測する際、本研究で新しく定義したメトリクスであるパッチ貢献度が予測精度向上に寄与するかどうかを確かめることを目的として5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、GCC プロジェクト、GIMP プロジェクト、WebKit プロジェクト）を対象に、評価実験を行った。評価実験の結果、5つのプロジェクトのすべてにおいて、パッチ貢献度を用いない予測モデルに比べ、パッチ貢献度を用いた予測モデルは、予測精度が向上することが明らかになった。

キーワード

モジュールオーナー、コミッター予測、大規模OSS開発、ソフトウェアメトリクス

Abstract

Open Source Software (OSS) have come into wide use recently, users of OSS are increasing. In the large scale of OSS projects, users report a lot of bugs every day. Reported bugs are fixed by patches which volunteer developers submit. Submitted patches are reviewed by committers who have permission to commit edited source code to the version control system. If committers accept submitted patches, they commit submitted patches to the version control system. Since committers have permission to commit edited source code to the version control system, they play an important role in OSS development. Project managers authorize developers to commit the version control system carefully, a developer is promoted to a new committer through the recommendation of current committers. There are less committers than developers who cannot commit to the version control system directly in OSS projects. In the large scale of OSS projects, a lot of patches are submitted every day, so minority of committers can not perform the verification of submitted patches effectively. To resolve lack of committers, researchers developed prediction model which identify future committers. In the prior study, the prediction model is based on developer activity metrics (eg. the number of submitted patches, the number of posted comment). In the large scale of OSS projects, modules which committers should be in charge have been determined. The prediction model which identify future committers suitable for the module is needed to resolve lack of committers. However, the prediction model which identify future committers suitable for the module has not been developed. Committers who are in charge of a module are called module owner in this thesis. The goal of this study is to build the prediction model which can identify the developer to be a module owner from among the volunteer developers. The prediction model in this study is based on developer activity metrics which were measured per module and metrics for capability of the module owner, patch contribution rate. This study conducted the experiment intended for five OSS projects (Eclipse Platform, Mozilla Firefox, GCC, GIMP, WebKit) to evaluate the prediction model. As a result of the experiment, the prediction model which contains patch contribution rate can predict future module owner more precisely.

Keywords

Module owner, Committer prediction, Large-scale open source software development, Software metrics

目次

第1章	はじめに	1
1.1	研究の背景と目的	1
1.2	本論文の構成	2
第2章	OSS 開発における不具合修正プロセスとコミッター	3
2.1	OSS 開発における不具合修正プロセス	3
2.2	OSS 開発におけるコミッターの役割と昇格プロセス	5
2.3	コミッター不足が引き起こす問題点	6
2.4	コミッター候補者を予測する必要性	7
第3章	先行研究	8
3.1	コミッター候補者予測	8
3.2	先行研究での問題点	8
3.3	本研究の着眼点	10
第4章	アプローチ	11
4.1	概要	11
4.2	モジュールオーナーの定義	11
4.3	予測に用いるメトリクス	11
4.4	予測モデル	12
第5章	実験	14
5.1	実験の目的	14
5.2	データセット	14
5.2.1	対象プロジェクト	14
5.2.2	データセット作成方法	16
5.2.3	基本統計量	20
5.3	評価指標	20
5.4	実験手順	21
第6章	実験結果	25
6.1	Mozilla Firefox プロジェクトでの実験結果	25
第7章	考察	28
7.1	予測精度に関する考察	28
7.2	予測に寄与するメトリクスに関する考察	28
7.3	予測できなかった開発者	28
7.3.1	False Positive となった開発者	28

7.3.2	False Negative となった開発者	28
7.3.3	追加分析	28
7.4	モジュール間でのモデルの差	28
7.5	妥当性の検証	30
7.5.1	パッチの行数	30
7.6	制約	31
7.6.1	モジュール分割方法	31
7.6.2	不具合情報とコミット情報のひも付け方法	31
第 8 章	関連研究	32
8.1	コミッター昇格に関する研究	32
8.2	開発者の適性に関する研究	33
8.3	モジュールオーナーに関する研究	34
第 9 章	おわりに	36
	外部発表	37

第1章 はじめに

本章では，本研究の背景および目的を述べる．また，本論文の構成についても述べる．

1.1 研究の背景と目的

近年，オープンソース（OSS）を用いたソフトウェア開発が主流になるなど，OSS が社会的に普及している．OSS が社会的に普及することにより，OSS を利用するユーザーが年々増加している．そのため，ユーザーの多い大規模な OSS プロジェクトでは，日々大量の不具合が報告される．報告された不具合は，ボランティアの開発者が中心となり，不具合修正のためのパッチが投稿され修正される．パッチとは，ソースコードの修正前と修正後の差分を示すファイルのことである．投稿されたパッチはバージョン管理システムに直接変更を加える権限を持つ開発者であるコミッターにより検証され，問題がないことが確認されればリポジトリに取り込まれることで不具合の修正が完了する．コミッターはソフトウェアに対して直接変更を加える権限を持つため，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．そのため，プロジェクトに参加す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，既存のコミッターからの推薦や承認によってコミット権限が付与される．このように，開発者にコミット権限を付与するかどうかの判断が慎重に行われているため，コミット権限を持たない開発者に比べコミッターは少人数で構成されている．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り，比較的少人数で構成されるコミッターのみではパッチの検証を効率的に行えなくなっている現状である．

コミッターの不足を解消するために，ボランティアの開発者の中からコミッターに昇格可能な開発者を予測するための研究が盛んに行われている．先行研究 [20, 33] では，既存のコミッターの昇格前の活動量メトリクス（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，ボランティアの開発者の中からコミッターに昇格し得る開発者を予測している．しかしながら，実際の大規模 OSS プロジェクトの不具合修正プロセスにおいては，各コミッターが担当すべきモジュールが決まっているため，有能なボランティアの開発者を全て昇格させれば不具合修正の長期化の問題が解決できるとは限らない．本研究では，特定のモジュールを担当するコミッターをモジュールオーナーと呼び，コミッター候補者予測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ボランティア開発者の中から特定するための予測モデルを構築する．予測モデル構築のために，先行研究 [20, 33] 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する．

本研究では、5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、GCC プロジェクト、GIMP プロジェクト、WebKit プロジェクト）を対象に、評価実験を行う。評価実験の目的は、

本研究で構築する予測モデルは、先行研究 [20, 33] で用いられている活動量メトリクスとパッチ貢献度を用いて、各開発者がモジュールオーナーになるか否かの予測を、ロジスティック回帰分析および、ランダムフォレスト法によって行う。メトリクスのすべての組み合わせについて予測モデルを構築し、適合率、再現率、F1 値で予測精度の評価を行った結果、5つのプロジェクトのすべてにおいて、パッチ貢献度を用いない予測モデルに比べ、パッチ貢献度を用いた予測モデルは、予測精度が向上することが明らかになった。

1.2 本論文の構成

本論文の構成は以下の通りです。続く 2 章では、OSS 開発における不具合修正プロセスおよび不具合修正プロセスにおいて重要な役割を担うコミッターについて説明し、大規模 OSS プロジェクトではコミッターが不足していることに触れる。3 章では、コミッターの不足を解消するために行われている研究について述べ、先行研究が抱える問題について述べる。4 章では、本研究のアプローチについて述べる。5 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、6 章で、評価実験の結果について述べる。7 章では、考察を行い、8 章で、関連研究と本研究の違いについて述べ、9 章でまとめる。

第2章 OSS 開発における不具合修正プロセスとコミッター

本章では、OSS 開発における不具合を修正するプロセスについて述べる。また、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果たす開発者であるコミッターについて説明し、コミッターへの昇格方法、およびコミッターが不足することによって引き起こされる問題について述べる。

2.1 OSS 開発における不具合修正プロセス

OSS は個人の利用のみならず、OSS を用いたソフトウェア開発が主流になるなど、社会を支えるソフトウェアシステムに OSS が組み込まれるようになっている。OSS プロジェクトの多くは、ある特定の場所にプロジェクトメンバーが集まり開発が行われるといった開発形態ではなく、インターネット上で議論やソースコードの管理を行うことで、世界中に分散するボランティアの開発者による開発が可能となっている。そのような複数の拠点で同時に並行し進められる開発は分散開発（distributed development）と呼ばれている。世界中に分散する開発者が参加するプロジェクトでは、タイムゾーンや文化、主要言語などの違いが存在するため、対面同期型のコミュニケーションをとるソフトウェア開発と比べ、コミュニケーションが取りづらく、開発スピードが遅くなることが知られている [18]。

また、OSS の社会的普及により、OSS に対するニーズが多様化するなどの理由から OSS を開発するコミュニティが大規模になっている。Eclipse プロジェクトや Mozilla プロジェクトのようなコミュニティに参加している開発者が 1 万人を超える大規模なコミュニティを持つ OSS プロジェクトも少なくない。そのような大規模 OSS プロジェクトでは、1 日数百件の不具合が報告されているため [2]、日々報告される不具合を漏らさず修正し管理することが困難となっている。また、大規模 OSS 開発では、複数の開発者が同一のソフトウェアに対して変更を加え、開発が進められるため、最新のバージョンのソフトウェアを開発者全員でリアルタイムに共有することが困難となっている。

そのような問題を解決するために、多くの OSS プロジェクトでは、不具合管理システム¹や、バージョン管理システムなどのツールが用いられている。不具合管理システムとは、日々報告される不具合を漏らさず修正するために、発見された不具合の症状や、修正状況などを管理するソフトウェアであり、報告された不具合に関する議論なども行われ

¹不具合追跡システム、バグトラッキングシステム（BTS）と呼ばれることもある。

る。不具合管理システムには、Bugzilla²やJira³、Redmine⁴、Trac⁵などが存在する。バージョン管理システムとは、ソフトウェアのソースコードやドキュメントの変更履歴を記録し、特定の段階（バージョン）までさかのぼる機能を持つソフトウェアのことである。バージョン管理システムには、Subversion⁶やCVS⁷などの集中型のバージョン管理システムとGit⁸やMercurial⁹、Bazaar¹⁰などの分散型のバージョン管理システムが存在する。

次に、不具合管理システムおよびバージョン管理システムを用いたOSS開発における不具合修正プロセスについて説明する。ユーザーが不具合を発見した際、まず、発見した不具合の症状や不具合が発生した環境などを不具合管理システムに登録する。登録された不具合を修正するために、不具合に関する議論やパッチファイル（以下、パッチと呼ぶ。）の投稿が行われる。パッチとは、修正元と修正後のソースコードの差分を示すファイルを指す。投稿されたパッチは、プロジェクトのコア開発者であるコミッターによって検証が行われる。コミッターとは、バージョン管理システムに変更を直接加える権限を持つ開発者のことを指す。コミッターがパッチの検証を行った結果、パッチの修正内容が適切であると判断された場合、コミッターはバージョン管理システムにパッチの修正内容をコミットする。コミットとは、バージョン管理システムに対して変更を加えることである。このように、コミッターはソフトウェアに直接変更加える権限を持つため、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 [11]。

²Bugzilla : <https://www.bugzilla.org/>

³Jira : <https://ja.atlassian.com/software/jira/>

⁴Redmine : <http://www.redmine.org/>

⁵Trac : <http://trac.edgewall.org/>

⁶Subversion : <https://subversion.apache.org/>

⁷CVS : <http://www.nongnu.org/cvs/>

⁸Git : <https://git-scm.com/>

⁹Mercurial : <https://www.mercurial-scm.org/>

¹⁰Bazaar : <http://bazaar.canonical.com/>

2.2 OSS 開発におけるコミッターの役割と昇格プロセス

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、各開発者によって役割が異なる。Ye らは、OSS プロジェクトのコミュニティ構造はオニオンモデルによって表わされると述べている [30]。図 2.1 は OSS プロジェクトのコミュニティ構造をオニオンモデルで示している。Ye らは、OSS の開発者がプロジェクトに対する貢献を通じて、各開発者の役割がオニオンモデルの中心に向かい変化していくことを述べている。また、Jensen らは、OSS プロジェクトのコミュニティ内において、開発者の役割が変化するパターンについて分析を行い、4 種類のパターンが存在することを明らかにした [21]。

応募による変化・昇格

応募による変化・昇格パターンは、ボランティアの開発者が自ら特定の役割を志願し、より高いランクの開発者（オニオンモデルの内側の開発者）が志願した開発者に対して役割を与えるべきだと判断すれば、役割が与えられる。

選出による変化・昇格

選出による変化・昇格パターンでは、OSS プロジェクトのコアメンバーが行う選挙によって選出された開発者が役割を獲得する。

任命による変化・昇格

任命による変化・昇格パターンは、OSS プロジェクトのコアメンバーによって任命されることによって役割を獲得する。

ボランティアによる変化・昇格

ボランティアによる変化・昇格パターンでは、暗黙的・明示的にボランティア活動やタスクを実行することによって役割を獲得する。

Jensen らが明らかにした開発者役割の変化パターンは、コミッター昇格プロセスにも当てはまる。OSS 開発において、OSS に参加するすべての開発者に対してコミット権限を与えてしまう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 [12]。コミッターに昇格するためには、任命による昇格パターンを除き、既存のコミッターやプロジェクトのコアメンバーからの推薦や承認が必要となる。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^{11 12 13}、これらの情報からも、コミッター昇格には、既存のコミッターやプロジェクトのコアメンバーからの推薦や承認が必要であることがわかる。

¹¹Eclipse プロジェクト： https://eclipse.org/membership/become_a_member/committer.php

¹²Mozilla プロジェクト： <https://www.mozilla.org/en-US/about/governance/policies/commit/>

¹³WebKit プロジェクト： <https://webkit.org/commit-and-review-policy/>

既存のコミッターやプロジェクトのコアメンバーは、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、既存のコミッターやプロジェクトのコアメンバーが推薦や承認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、既存のコミッターやプロジェクトのコアメンバーが経験的にコミッターを選出している現状である。

2.3 コミッター不足が引き起こす問題点

コミッターはソフトウェアの品質管理において重要な役割を担う開発者であるため、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。そのため、コミット権限を持たない開発者に比べ、コミッターは少数である。Mozilla Firefox プロジェクトでは、約 10 年間でパッチの投稿や議論への参加などのプロジェクトに対する貢献を行った開発者は 76,000 人を超えたものの、そのうち、コミッターである開発者は約 550 人であった。投稿されたパッチの検証作業は、少数のコミッターが中心となって行われるため、コミッターに対して負担がかかっている。日々大量に不具合報告が行われている大規模 OSS プロジェクトにおいては、コミッターの過度な負担が原因となり、パッチ検証が効率的に行わず、その結果、不具合の修正時間が長期化していることが報告されている [6]。

コミッターの作業負担を軽減し、不具合の修正時間を短縮させるために、コミッターとして活動する開発者を増やすという手段が効果的であると考えられる。しかしながら、数

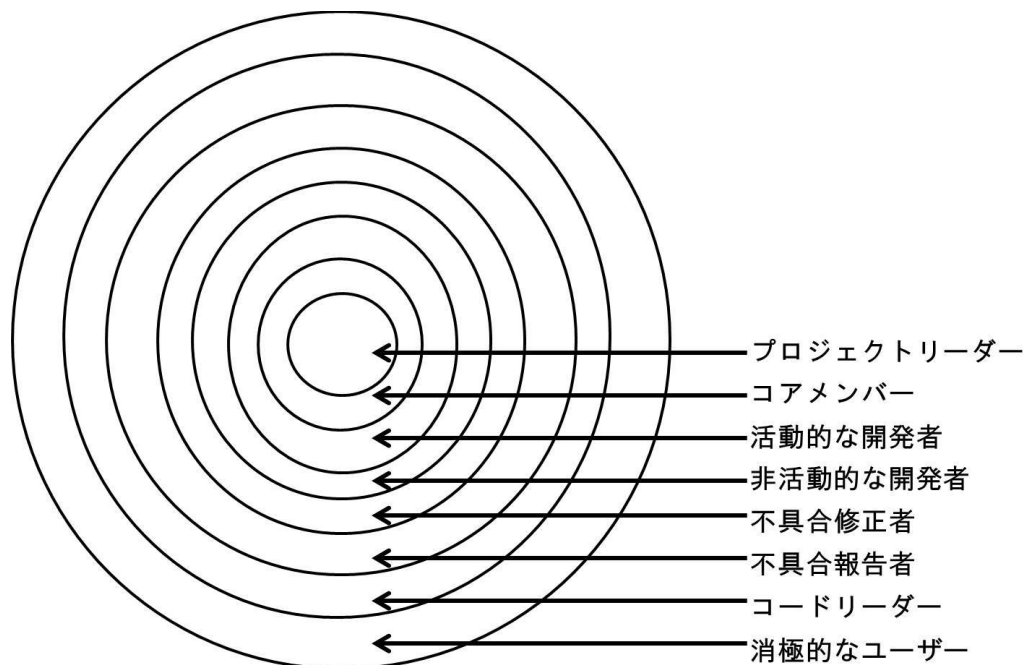


図 2.1: オニオンモデル ([30] をもとに改編)

万人に及ぶ開発者の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、既存のコミッターやプロジェクトのコアメンバーが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行うために、開発者は通常1年以上パッチの投稿や議論への参加などといったプロジェクトに対する貢献が求められる。しかし、OSSプロジェクトに参加する開発者は、活動に対する意欲の喪失などの理由から、活動期間が1年を超えるとプロジェクトから離脱してしまう確率が上がることが知られている [8]。

このような問題を解決するために、開発者の中からコミッタ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。

2.4 コミッター候補者を予測する必要性

第3章 先行研究

本章では，開発者の中から将来コミッターになると考えられる開発者を特定することを目的とした先行研究について説明し，先行研究が抱える問題点について述べる．そして，本研究で取り組む課題について述べる．

3.1 コミッター候補者予測

開発者の中からコミッターに昇格すべき開発者を特定するために，既存のコミッターとコミット権限を持たない開発者の活動を学習し，将来コミッターに昇格する開発者を予測する研究が行われている [20, 33]．伊原らは，コミッター昇格以前の開発者をコミッター候補者，コミット権限を持たない開発者を一般開発者と定義し，それぞれの開発者において，活動量メトリクスを計測している [20, 33]．活動量メトリクスとは，表 3.1 に示すように，総パッチ投稿数や総コメント投稿数などの開発者の活動を定量化したものである．月パッチ投稿数および月コメント投稿数は，各開発者が1ヵ月にパッチ投稿あるいはコメント投稿を行った回数の中央値を示しており，この値が高い開発者ほど，継続的に活動を行っている開発者であることが示される．

先行研究では，大規模 OSS プロジェクトである Eclipse Platform プロジェクトおよび，Mozilla Firefox プロジェクトを対象にコミッター候補者および一般開発者の活動量メトリクスを計測し，ロジスティック回帰分析およびランダムフォレスト法を用いて予測モデルを構築している [20]．実験の結果，両プロジェクトともに，再現率（実際にコミッター候補者であるもののうち，コミッター候補者であると予測されたものの割合）が高く，Eclipse Platform プロジェクトでは，総コメント投稿数，Mozilla Firefox プロジェクトでは活動期間が最も予測に寄与するメトリクスであることが明らかになった．

3.2 先行研究での問題点

先行研究では，既存のコミッターの昇格前の活動量（パッチ投稿数など）に基づいて予測モデルを構築し，ボランティアの開発者の中からコミッターに昇格し得る開発者を予測している．しかしながら，実際の大規模 OSS プロジェクトの不具合修正プロセスにおいて，各コミッターが担当すべきモジュールが決まっていることも少なくない．大規模 OSS 開発では，各コミッターがソフトウェア全体を詳しく理解することは困難であるため，各

表 3.1: 活動量メトリクス一覧

活動	メトリクス名	内容
パッチ投稿	総パッチ投稿数	各開発者がパッチを投稿した総回数
	月パッチ投稿数	各開発者が1ヶ月にパッチを投稿した回数の中央値
開発に伴う議論	総コメント投稿数	各開発者がコメントを投稿した総回数
	月コメント投稿数	各開発者が1ヶ月にコメントを投稿した回数の中央値
活動期間	活動期間	各開発者の活動（パッチ投稿，開発に伴う議論のいずれかの活動）が実施された月数

コミッターがパッチの検証やソフトウェアの品質管理をすることのできる範囲は限られている。実際に Mozilla プロジェクトでは、各モジュールを担当するコミッターが決められており、各モジュールを担当するコミッターはモジュールオーナーと呼ばれている¹。

大規模 OSS 開発では、各コミッターが担当すべきモジュールが決まっているため、先行研究のように有能なボランティアの開発者を全て昇格させれば不具合修正の長期化の問題が解決できるとは限らない。図を用いて説明する。図 3.1 に示すように、モジュール A、モジュール B、モジュール C、モジュール D から構成される OSS が存在し、各モジュールのオーナーが存在しているとする。今、モジュール C において、不具合修正のために投稿されたパッチが集中しており、パッチが効率的に検証されないことから不具合修正が長期化している状況である。このような状況において、投稿されたパッチを効率的に検証するためには、モジュール C のモジュールオーナーを増員することが求められる。しか

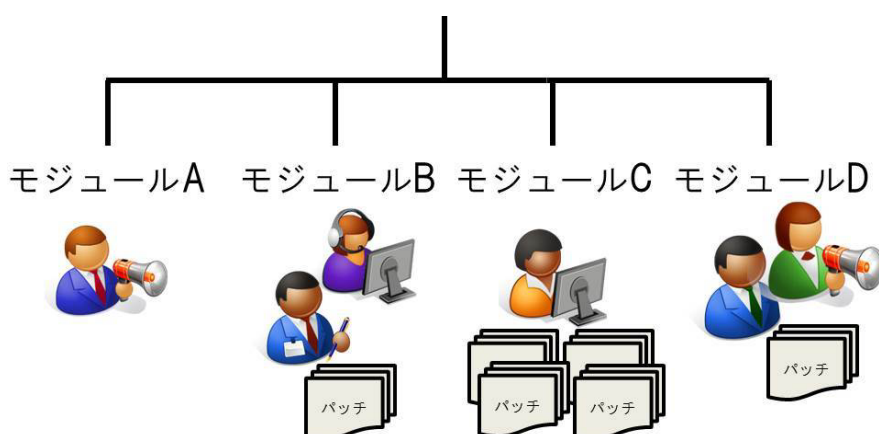


図 3.1: 特定のモジュールにコミッターが必要となる事例

¹<https://www.mozilla.org/en-US/about/governance/policies/module-ownership/>

しながら，先行研究の予測モデルでは，各開発者がどのモジュールを担当するのにふさわしい開発者なのかといったより細粒度な予測を行うことができないため，予測された開発者が本来求められるモジュールとは異なるモジュールの担当となる可能性が存在する．そのため，コミッターと予測された開発者が本来求められるモジュールとは異なるモジュールの担当となった場合，不具合修正の長期化の問題を解決することはできない．

3.3 本研究の着眼点

前節で述べた通り，大規模OSSプロジェクトにおいて，各コミッターが担当するモジュールが決定されているため，コミッターの不足による不具合修正の長期化を解決するためには，コミッターが不足しているモジュールに適した開発者を推薦する必要がある．しかしながら，先行研究のコミッター候補者予測モデルは，各開発者が将来コミッターになるか否かの予測を行うため，コミッターであると予測された開発者がどのモジュールを担当可能なかまでを知ることができない．また，開発者の中から特定のモジュールのオーナーを推薦するための予測モデルが構築する研究が行われていない．

そこで本研究では，コミッターの不足による不具合修正の長期化を解決するために，モジュールオーナーとなるべき開発者を特定するための予測モデルの構築を行う．モジュールオーナーとなる開発者を特定することにより，オーナーが不足しているモジュールに対して，モジュールオーナーを増員することができ，パッチの検証作業が効率的に行われることが期待される．

第4章 アプローチ

本章では、担当者が不足しているモジュールに適したコミッターを推薦するために、コミット権限を持たない開発者の中から、モジュールオーナーを予測する手法について説明する。まず、概要について述べ、次に提案するメトリクスについて述べる。さらに、予測モデル構築に用いる機械学習アルゴリズムについて述べる。

4.1 概要

本研究では、担当者が不足しているモジュールに適したコミッターを推薦するために、モジュールごとのコミッター候補者予測を行う。モジュールごとのコミッター候補者予測モデルを構築するために、先行研究 [20] 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する。また、モジュールオーナーの適性を表すメトリクスを定義する。モジュールオーナーの適性を表すメトリクスは、開発者が特定のモジュールに対して、どれだけパッチの投稿通じて貢献を行ったかの割合を示すものである。このメトリクスの値が大きいほど、特定のモジュール内での貢献度が大きいことを示しており、モジュールオーナーに昇格する可能性が高くなると想定している。また、予測モデル構築には、先行研究 [20] で用いられているロジスティック回帰分析および、ランダムフォレスト法を用いる。

4.2 モジュールオーナーの定義

4.3 予測に用いるメトリクス

モジュールごとのコミッター候補者予測モデルの構築を行うために用いるメトリクスについて説明する。本研究で用いるメトリクスは、モジュールごとの活動量メトリクスおよび、モジュールオーナーの適性を示すメトリクスである。以下にそれぞれのメトリクスについて説明する。

活動期間

活動期間は、各開発者が特定のモジュールに対してパッチ投稿あるいはコメント投稿を行った月数で表わされる。活動期間は、開発者がプロジェクトに参加している期間中に、パッチ投稿あるいはコメント投稿が行われていない月があればカウントされない。

総コメント投稿数

総コメント投稿数は、各開発者が特定のモジュールに対してコメントを投稿した回数で表わされる。

月コメント投稿数

月コメント投稿数は、各開発者が1ヵ月に特定のモジュールに対してコメントを投稿した回数の中央値で表される。

総パッチ投稿数

総パッチ投稿数は、各開発者が特定のモジュールに対してパッチを投稿した回数で表わされる。

月パッチ投稿数

月パッチ投稿数は、各開発者が1ヵ月に特定のモジュールに対してパッチを投稿した回数の中央値で表される。

4.4 予測モデル

本研究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予測する。そこで、先行研究 [33, 20] で用いられているロジスティック回帰分析およびランダムフォレスト法を用いて予測モデルの構築を行う。モジュールオーナーに昇格するか否かを目的関数（モジュールオーナーに昇格する場合 1，モジュールオーナーに昇格しない場合 0）とし、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。

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 つである。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 4.1 に示す。

$$P(y|x_1, \dots, x_p) = \frac{1}{1 + e^{-(\alpha_1 x_1 + \dots + \alpha_r x_r + \beta)}} \quad (4.1)$$

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、 $P(y|x_1, \dots, x_p)$ は目的変数の条件付き確率である。

本研究では、ロジスティック回帰分析を統計解析ソフトウェアである R¹ の glm 関数を用いて行う。

¹統計解析ソフトウェア R : <http://www.r-project.org/index.html>

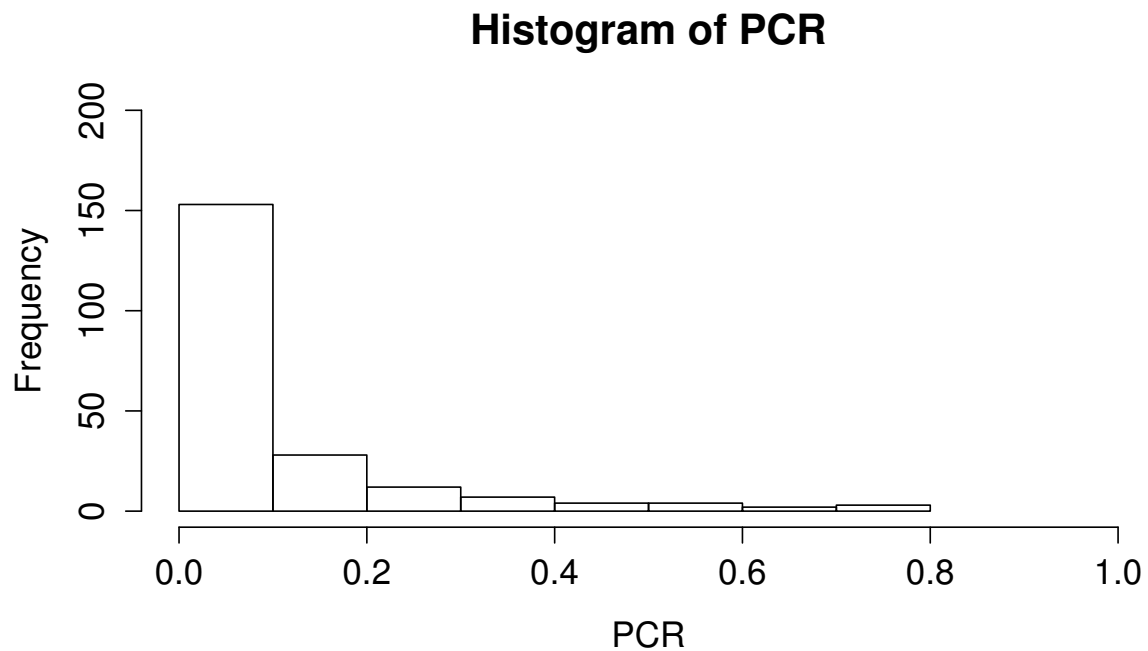


図 4.1: Eclipse Platform プロジェクトの SWT モジュールにおける PCR のヒストグラム

ランダムフォレスト法は、ブライマン [10] により提案された集団学習を用いた機械学習アルゴリズムである。集団学習とは、低性能なモデルを複数組み合わせることで、高性能な予測モデルを構築する方法であり、ランダムフォレスト法以外に、ブースティングやバギングが挙げられる。ランダムフォレスト法は、説明変数をランダムにサンプリングしたデータセットを用いているため、ブースティングやバギングに比べ、高次元データ解析に向いている点である。

ランダムフォレスト法では、まず、与えられたデータセットに対し、繰り返しランダムサンプリングを行い、多数のブートストラップサンプルを作成する。次に、各々のブートストラップサンプルデータを用いて未剪定の最大の決定・回帰木を作成する。そして、すべての結果を統合し、新しい予測・分類器を作成する。また、ランダムフォレスト法は、各説明変数の重要度を推定することができる。

本研究では、ランダムフォレスト法を統計解析ソフトウェアである R の `randomForest` 関数を用いて行う。

第5章 実験

本章では，評価実験を行う目的および実験に用いるデータについて述べる．また，実験を行う手順と評価方法についても述べる．

5.1 実験の目的

本実験の目的は，モジュールのオーナーにふさわしいコミッターを予測する際，本研究で新たに導入したパッチ貢献度メトリクス（PCR）が予測精度向上に寄与することを確認することである．5つの大規模OSSプロジェクトを対象に，先行研究[20]で用いられているメトリクスとパッチ貢献度メトリクスのすべての組合せについて予測モデルを構築する．予測精度の評価は，適合率，再現率，F1値を用いて交差検証によって行い，最も予測精度が高くなるメトリクスの組み合わせを明らかにする．

5.2 データセット

5.2.1 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Eclipse Platformプロジェクト¹，Mozilla Firefoxプロジェクト²，GCCプロジェクト³，GIMPプロジェクト⁴，WebKitプロジェクト⁵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．以下に各プロジェクトの概要を述べる．

Eclipse Platform プロジェクト

Eclipse Platform プロジェクトは，Eclipseを開発するためのプラットフォームである．Eclipseは，2001年11月にIBMがオープンソース化された統合開発環境（IDE: Integrated Development Environment）である．Eclipse PlatformはEclipseにおける共通フレームワークであると定義されている．EclipseにはEclipse Platform

¹<http://www.eclipse.org/eclipse/>

²<https://www.mozilla.org/ja/firefox/>

³<https://gcc.gnu.org/>

⁴<https://www.gimp.org/>

⁵<https://webkit.org/>

プロジェクト以外に、JDT(Java Development Tools) や、PDE(Plug-in Development Environment) などのサブプロジェクトが存在する。

Mozilla Firefox プロジェクト

Mozilla Firefox プロジェクトは、Mozilla Foundation が開発するオープンソース・クロスプラットフォームのウェブブラウザである。Mozilla Firefox は、Netscape プロジェクトの後継プロジェクトであり、Netscape プロジェクトがオープンソース化した 1998 年 2 月に開発組織 Mozilla Organization が立ち上げられ、プロジェクトが開始した。Mozilla プロジェクトには、Mozilla Firefox プロジェクトの他に、電子メールクライアントである Thunderbird やスマートフォンなどに搭載される OS である Firefox OS プロジェクトなどのサブプロジェクトが存在する。

GCC プロジェクト

GCC (GNU Compiler Collection) は、GNU が提供するコンパイラ群である。GCC は 1985 年、リチャード・ストールマンによって既存のコンパイラを拡張する形で開発が始められた。当初コンパイラは Pastel という言語によって書かれていたが、1987 年に C 言語に書き直され、GNU プロジェクトの一つとして 1987 年にオープンソース化された。GCC の標準パッケージには、C, C++, Fortran, Java, Ada のコンパイラ並びにライブラリが含まれている。

GIMP プロジェクト

GIMP (GNU Image Manipulation Program) は、GNU プロジェクトによって開発が行われている画像編集ソフトウェアである。1995 年に Spencer Kimball と Peter Mattis によって開発が始められ、1996 年 1 月に初めてリリースされた。GIMP は、線やテキストの描画、グラデーション加工などの画像処理とスクリプトを使った自動画像処理を行う機能を備えている。また、モザイク編集やアニメーション合成を行うためのフィルタ機能も数多く備えている。

WebKit プロジェクト

WebKit とは、HTML, CSS, Javascript, SVG, MathMLなどを解釈する HTML レンダリングエンジンのフレームワークの名称である。WebKit は、Web ページのレンダリングやアプリケーションとしてのインターフェースを形成するためのフレームワークとして用いられており、Mac OS 標準のブラウザである Safari やメールなどの様々なアプリケーションで採用されている。

表 5.1: 不具合管理システムの URL の一覧

プロジェクト名	データ取得先
Eclipse Platform	https://bugs.eclipse.org/bugs/
Mozilla Firefox	https://bugzilla.mozilla.org/
GCC	https://gcc.gnu.org/bugzilla/
GIMP	https://bugzilla.gnome.org/
WebKit	https://bugs.webkit.org/

表 5.2: バージョン管理システムのリポジトリ URL の一覧

プロジェクト名	リポジトリ URL
Eclipse Platform	https://git.eclipse.org/r/platform/eclipse.platform.common.git
	https://git.eclipse.org/r/platform/eclipse.platform.debug.git
	https://git.eclipse.org/r/platform/eclipse.platform.git
	https://git.eclipse.org/r/platform/eclipse.platform.releng.git
	https://git.eclipse.org/r/platform/eclipse.platform.resources.git
	https://git.eclipse.org/r/platform/eclipse.platform.runtime.git
	https://git.eclipse.org/r/platform/eclipse.platform.swt.git
	https://git.eclipse.org/r/platform/eclipse.platform.team.git
	https://git.eclipse.org/r/platform/eclipse.platform.text.git
	https://git.eclipse.org/r/platform/eclipse.platform.ua.git
	https://git.eclipse.org/r/platform/eclipse.platform.ui.git
Mozilla Firefox	http://hg.mozilla.org/mozilla-central
GCC	git://gcc.gnu.org/git/gcc.git
GIMP	git://git.gnome.org/gimp
WebKit	git://github.com/WebKit/webkit.git

5.2.2 データセット作成方法

データ収集

本研究では、先行研究 [20] と同様に、開発者の活動期間やコメント投稿数、パッチ投稿数などのメトリクスを不具合管理システムから計測する。そのために、各プロジェクトが利用している不具合管理システムからデータを収集する。表 5.1 に各プロジェクトのデータの取得先を示す。

また、各開発者がコミッターに昇格したかどうかを特定するために、バージョン管理システムからコミット情報を取得する。表 5.1 に各プロジェクトが利用しているバージョン管理システムのリポジトリ URL を示す。Eclipse Platform プロジェクトでは、Git リポジトリが複数に分けられ管理されているため、複数の Git リポジトリからデータを収集する。また、Mozilla Firefox プロジェクトは、Mozilla プロジェクト全体で一つのバージョン管理システムを利用しているため、表 5.2 に示すリポジトリの /browser 以下のコミット

表 5.3: データセットの基本情報一覧

プロジェクト名	対象期間	不具合数	コミット回数	コミッター	一般開発者数
Eclipse Platform	2001/10-2012/10	96,387	94,990	195	13,297
Mozilla Firefox	1999/07-2013/01	120,468	13,179	530	76,931
GCC	1999/07-2014/03	59,983	106,636	141	10,282
GIMP	2001/02-2013/09	12,030	28,163	87	4,068
WebKit	2000/12-2014/04	121,685	150,392	363	8,017

表 5.4: Eclipse Platform プロジェクトでのモジュール分割

モジュール名	不具合管理システム上でのコンポーネント名
UI	UI, Text, Runtime, Search
SWT	SWT
Workspace	Team, CVS, Resources, Compare
Debug	Debug
Releng	Releng

ログを取得する。表 5.3 に収集したデータの基本情報を示す。

モジュールの特定

モジュールごとにパッチ投稿数やコメント投稿数を計測するために、どのモジュールに対して投稿されたパッチあるいはコメントなのかの特定を行う。不具合管理システムにおいて、報告された不具合には、ソフトウェアのどのモジュール（コンポーネント）⁶の不具合なのかといった情報が記載されている。基本的には、この情報を基に分割を行う。しかしながら、不具合情報に記載されているモジュールの情報では、各モジュールの粒度に大きく差が出てしまう。そのため、ソースコードの構造および、不具合管理システムのコンポーネント情報、プロジェクトの Wiki ページを用いて総合的にモジュールの特定を行う。本研究で行った、各プロジェクトのモジュールの特定方法および結果を以下に示す。

Eclipse Platform プロジェクト

Eclipse Platform プロジェクトに関する情報が Wiki ページ⁷に記載されており、Wiki ページによると、Eclipse Platform プロジェクトは UI, SWT (Standard Widget Toolkit), Workspace, Debug, Releng に分割することができる。表 5.4 に分割したモジュール名および不具合管理システムにおけるコンポーネント名を示す。

⁶本研究では、モジュールとコンポーネントを同じ意味として捉える。

⁷<http://wiki.eclipse.org/Platform>

表 5.5: Mozilla Firefox プロジェクトでのモジュール分割

モジュール名	不具合管理システム上でのコンポーネント名
General	General
DeveloperTools	Developer Tools, Developer Tools: Console, Developer Tools: Inspector, Developer Tools: Debugger, Developer Tools: Graphic Commandline and Toolbar, Developer Tools: Style Editor, Developer Tools: Scratchpad, Developer Tools: Source Editor, Developer Tools: Framework, Developer Tools: 3D View, Developer Tools: Responsive Mode, Developer Tools: Profiler
Bookmark	Bookmarks & History, RSS Discovery and Preview
Tab	Tabbed Browser, Panorama, Session Restore
Bar	Toolbars, Downloads Panel, Location Bar, Search, Menus
Preferences	Preferences, Theme, Shell Integration
App	File Handling, Installer, Security, Web Apps

表 5.6: GCC プロジェクトでのモジュール分割

モジュール名	不具合管理システム上でのコンポーネント名
C	c++, c, libstdc++, preprocessor
Java	java, libgcj
Ada	ada
Fortran	fortran, libfortran
MiddleEnd	tree-optimization, middle-end, rtl-optimization
BackEnd	target
Build	bootstrap

Mozilla Firefox プロジェクト

Mozilla Firefox プロジェクトは、Web ブラウザであるため、Web ブラウザが保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 DeveloperTools、ブックマーク機能を扱う Bookmark、Web ページのタブに関する Tab、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。表 5.5 に分割したモジュール名および不具合管理システムにおけるコンポーネント名を示す。

GCC プロジェクト

GCC プロジェクトは、コンパイラであり、標準パッケージでは、C, Fortran, Java, Ada の言語に対応している。また、GCC は、字句解析や構文解析を行うフロントエンド部、最適化を行うミドルエンド部、コード生成を行うバックエンド部から構成

表 5.7: GIMP プロジェクトでのモジュール分割

モジュール名	不具合管理システム上でのコンポーネント名
General	General
UserInterface	User Interface
Plugin	Plugins
Tools	Tools

表 5.8: WebKit プロジェクトでのモジュール分割

モジュール名	不具合管理システム上でのコンポーネント名
WebCore	WebCore Misc., WebCore JavaScript, Platform
JavaScriptCore	JavaScriptCore
Build	Tools / Tests
Layout	Layout and Rendering, Text, SVG, CSS, Animations, WebGL
API	WebKit API, Web Audio, WebKit BREWMP, WebKit EFL, WebKit Gtk, WebKit Qt, WebKit wx, WebKit Misc.
Inspector	Web Inspector, Web Inspector (Deprecated)
WebKit2	WebKit2

されている⁸。フロントエンド部は各言語によって異なるが、ミドルエンド部およびバックエンド部は共通であるため、表 5.6 に示すように分割を行った。

GIMP プロジェクト

GIMP プロジェクトは、不具合管理システムのコンポーネント情報を基に、GIMP の共通部分、ユーザーインターフェース部分、プラグイン、画像の編集を行うための機能を持つツールに分割した。表 5.7 に分割を行った結果を示す。

WebKit プロジェクト

WebKit は、プロジェクトの Wiki ページ⁹やソースコードの構造を基に分割した。分割した結果を表 5.8 に示す。WebCore とは、レンダリングライブラリを指し、JavaScriptCore とは、WebKit 実装のための JavaScript エンジンである。また、Inspector とは、Web ページのソースの表示やデバッグのための機能である。

モジュールオーナーの判別

各コミッターがどのモジュールのモジュールオーナーであるかの判別方法について述べる。本研究では、モジュールオーナーの判別方法として、特定のモジュールに対して投稿

⁸<http://www.redhat.com/magazine/002dec04/features/gcc/>

⁹<https://trac.webkit.org/wiki/>

表 5.9: コミットの説明文に含まれる不具合 ID の抽出方法一覧

プロジェクト名	抽出パターン例	説明
Eclipse Platform	Bug 25453 fixed.	Bug の直後の数字を不具合 ID として抽出
Mozilla Firefox	Bug 1229076 -	Bug の直後の数字を不具合 ID として抽出
GCC	PR target/32000, PR1324	PR の直後の数字を不具合 ID として抽出
GIMP	(bug#155325)	#の直後の数字を不具合 ID として抽出
WebKit	bug.webkit.org/show_bug.cgi?id=1479	id=の後の数字を不具合 ID として抽出

されたパッチをバージョン管理システムに反映しているかどうかを調べる。以下に判別手順を示す。

1. 特定のコミッターがコミットを行ったコミットログを取得する。
2. コミットログに含まれるコミットの説明文から不具合 ID を取得する。
3. 不具合 ID から不具合管理システムのコンポーネント情報を参照し、モジュールを特定する。
4. 1 から 3 の処理をすべてのコミッターに対して行う。

不具合情報とコミット情報のひも付けを行う方法として、コミットの説明文に含まれる不具合 ID を取得する方法が多くの研究で用いられている [28, 19, 16, 26]。コミットの説明文に含まれる不具合 ID の抽出方法はプロジェクトによって異なるため、表 5.9 に各プロジェクトでの不具合 ID 抽出パターンを示す。

以上の方法で判別した各プロジェクトのモジュールオーナーの数を表 5.10 に示す。5.10 から、GCC プロジェクトでの Ada, Build, Java モジュールは、モジュールオーナーの数が 1 人であるため、予測することができない。そのため、本研究で対象とするモジュールから除外する。

5.2.3 基本統計量

5.3 評価指標

予測精度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 5.1 によって定義される。

$$F1 \text{ 値} = \frac{2 \times \text{適合率} \times \text{再現率}}{\text{適合率} + \text{再現率}} \quad (5.1)$$

5.4 実験手順

本研究で行う評価実験の手順を以下に示す。

手順 1：データセットの分割 モジュールオーナーと非モジュールオーナー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、非モジュールオーナーのデータが 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を 50 件、同じくランダムに抽出した非モジュールオーナーのデータを 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 非モジュールオーナー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 [17, 25]。そのため、学習データのバランスの調整を行うために、非モジュールオーナー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、非モジュールオーナーの学習データが 5,000 件あった場合、非モジュールオーナーの学習データから 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および、ランダムフォレスト法を用いる。モジュールオーナーである開発者を 1、非モジュールオーナーである開発者を 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、目的関数が 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：交差検証 学習データとテストデータはランダムに作成されるため、予測結果が毎回変化する。そのため、手順 1 から手順 4 を 100 回繰り返し、100 回行った結果の中央値を最終的な予測精度とする。

手順 6：全モデルでの実行 手順 1 から手順 5 を??で述べたパターンについて行う。

手順 7 : 全モジュールに対しての実行 手順 1 から手順 6 は，特定のモジュールについて行われる．そのため，手順 1 から手順 6 を全モジュールに対して行う．

表 5.10: 各プロジェクトでのモジュールオーナーの数

プロジェクト名	モジュール名	モジュールオーナーの数
Eclipse Platform	Debug	15
	Releng	4
	SWT	12
	UI	52
	Workspace	16
Mozilla Firefox	App	18
	Bar	66
	Bookmark	54
	DeveloperTools	45
	General	73
	Preferences	66
	Tab	64
GCC	Ada	1
	BackEnd	13
	Build	1
	C	9
	Fortran	3
	Java	1
	MiddleEnd	16
GIMP	General	9
	Plugin	13
	Tools	7
	UserInterface	11
WebKit	API	151
	Build	155
	Inspector	40
	JavaScriptCore	42
	Layout	140
	WebCore	180
	WebKit2	21

表 5.11: 構築する予測モデル一覧

モデル名	モデル構築に用いるメトリクス
PCR	パッチ貢献度のみ
活動期間	活動期間のみ
コメント	総コメント投稿数, 月コメント投稿数
パッチ	総パッチ投稿数, 月パッチ投稿数
PCR+活動期間	パッチ貢献度, 活動期間
PCR+コメント	パッチ貢献度, 総コメント投稿数, 月コメント投稿数
PCR+パッチ	パッチ貢献度, 総パッチ投稿数, 月パッチ投稿数
活動期間+コメント	活動期間, 総コメント投稿数, 月コメント投稿数
活動期間+パッチ	活動期間, 総パッチ投稿数, 月パッチ投稿数
コメント+パッチ	総コメント投稿数, 月コメント投稿数, 総パッチ投稿数, 月パッチ投稿数
PCR+活動期間+コメント	パッチ貢献度, 活動期間, 総コメント投稿数, 月コメント投稿数
PCR+活動期間+パッチ	パッチ貢献度, 活動期間, 総パッチ投稿数, 月パッチ投稿数
PCR+コメント+パッチ	パッチ貢献度, 総コメント投稿数, 月コメント投稿数, 総パッチ投稿数, 月パッチ投稿数
先行研究のモデル	活動期間, 総コメント投稿数, 月コメント投稿数, 総パッチ投稿数, 月パッチ投稿数
全メトリクス	パッチ貢献度, 活動期間, 総コメント投稿数, 月コメント投稿数, 総パッチ投稿数, 月パッチ投稿数

第6章 実験結果

本章では，各プロジェクトでの実験結果について述べる．??に示した予測モデルについての予測精度を示し，モジュールオーナーを特定するのに有用なメトリクスを明らかにする．

6.1 Mozilla Firefox プロジェクトでの実験結果

モジュール名	適合率	再現率	F1 値
App	0.571	0.563	0.552
Bar	0.500	0.750	0.590
Bookmark	0.480	0.560	0.511
DeveloperTools	0.800	0.786	0.800
General	0.455	0.585	0.511
Preferences	0.552	0.613	0.587
Tab	0.286	0.571	0.364

表 6.1: Mozilla Firefox プロジェクトでの結果

モジュール名	活動期間	総パッチ投稿数	月パッチ投稿数	総コメント投稿数	月パッチ投稿数	(切片)
App	0.420	1.780	NA	-0.459	27.477	-3.846
Bar	0.267	1.548	NA	0.834	-1.271	-7.077
Bookmark	0.037	1.224	0.041	0.419	-0.528	-3.0480
DeveloperTools	11.461	7.975	-4.218	-8.126	8.114	-54.385
General	-0.063	0.741	-3.643	0.435	0.657	-2.212
Preferences	0.708	1.660	NA	-0.453	1.904	-5.839
Tab	0.764	1.357	NA	-0.371	2.002	-5.876

表 6.2: 回帰式 : Mozilla Firefox プロジェクト

第7章 考察

7.1 予測精度に関する考察

7.2 予測に寄与するメトリクスに関する考察

7.3 予測できなかった開発者

7.3.1 False Positive となった開発者

7.3.2 False Negative となった開発者

7.3.3 追加分析

分析方法

分析結果

7.4 モジュール間でのモデルの差

本研究で行った評価実験によって、モジュールオーナーになる開発者を予測する際、パッチ貢献度を用いることにより、予測精度が向上することが明らかとなった。本節では、予測精度が向上することによりどのような効果がもたらされるのかについて考察を行う。具体的には、パッチ貢献度を用いることによって予測することのできたモジュールオーナーに注目し、分析を行う。

表 7.1 に Mozilla Firefox プロジェクトの General モジュールにおける、パッチ貢献度を用いて最も予測精度が高くなったモデルであるパッチ貢献度のみを用いた予測モデルと、パッチ貢献度を用いずに最も予測精度が高くなったモデルである先行研究の予測モデルの結果を示す。表 7.1 における 1, 0 とは、モジュールオーナーであるかどうかを示しており、1 はモジュールオーナーであることを、0 はモジュールオーナーでないことを示している。表 7.1 に示すように、パッチ貢献度のみを用いた予測モデルによりモジュールオーナーであると判断された開発者は、817 (36+781) 人存在し、そのうち実際にモジュールオーナーであった開発者は 36 人であった。表 7.1 に示すように、パッチ貢献度のみを用いた予測モデルでは、モジュールオーナーとなる開発者を漏れなく予測できているものの、先行研究のモデルでは、実際にモジュールオーナーになった開発者を、モジュールオー

ナーでないと予測した開発者が3人いた。このパッチ貢献度を用いることによって初めてモジュールオーナーであると予測することのできた開発者に注目する。

パッチ貢献度を用いることによって初めてモジュールオーナーであると予測することのできた開発者がパッチの検証作業をどの程度行っていたかを分析する。表 7.2 に、パッチ貢献度を用いることによって初めてモジュールオーナーであると予測することのできた各開発者の Mozilla Firefox プロジェクトの General モジュールに対して投稿されたパッチを検証し、バージョン管理システムにコミットした回数を示す。表 7.2 に示すように、パッチ貢献度を用いることによって初めてモジュールオーナーであると予測することのできた開発者であるモジュールオーナー A は、パッチ検証を 34 回行っており、モジュールオーナー B は 3 回、モジュールオーナー C は 2 回であった。今、General モジュールに投稿されたパッチの総数は 2,015 であったため、モジュールオーナー A は、投稿されたパッチの約 1.7%(34/2015) のパッチに対して検証作業を行っていることがわかる。これは、General モジュールの 73 人のモジュールオーナーのうち、6 番目に多いパッチ検証数であった。

表 7.1: Mozilla Firefox プロジェクトの General モジュールの結果

パッチ貢献度のみ		実際の結果	
		1	0
予測結果	1	36	781
	0	0	37,881
先行研究のモデル		実際の結果	
		1	0
予測結果	1	33	1,794
	0	3	36,868

表 7.2: 先行研究のモデルで予測できなかった開発者のパッチ検証回数

	パッチ検証回数
モジュールオーナー A	34
モジュールオーナー B	3
モジュールオーナー C	2

これらの結果から、パッチ貢献度を用いずに予測を行い、モジュールオーナーを決定していた場合、モジュールオーナー A のようなモジュールに対して貢献の多い開発者を予測することができない恐れがあることがわかった。

表 7.3: Mozilla Firefox プロジェクトの General モジュールでの予測結果

パッチ貢献度のみ		実際の結果	
		1	0
予測結果	1	36	177
	0	0	38,485

7.5 妥当性の検証

本研究で行った評価実験の結果では、Mozilla Firefox プロジェクトおよび、GCC プロジェクト、GIMP プロジェクトにおいて、再現率が 1.000 となる予測モデルが存在した。本来、予測精度が 100% である予測モデルは、実験や評価の方法が間違っている可能性が高いため、本研究で行った実験の妥当性を検証するために、再現率が 1.000 となったモデルについて詳しく調査する。

再現率が 1.000 となった結果である Mozilla Firefox プロジェクトの General モジュールにおける、パッチ貢献度のみを用いた予測モデルでの予測結果を表 7.3 に示す。表 7.3 における 1, 0 とは、モジュールオーナーであるかどうかを示しており、1 はモジュールオーナーであることを、0 はモジュールオーナーでないことを示している。表 7.3 に示すように、予測によりモジュールオーナーであると判断された開発者は、213 (36+177) 人存在し、そのうち実際にモジュールオーナーであった開発者は 36 人であった。実際にモジュールオーナーであった開発者が 36 人存在し、そのうち、予測によりモジュールオーナーでないと判断された開発者は存在しなかった。そのため、再現率が 1.000 となった。

7.5.1 パッチの行数

本研究では、モジュールオーナーの適性を測るメトリクスとして、パッチ貢献度を定義した。パッチ貢献度として、モジュールに投稿されたパッチの回数に占める開発者が投稿したパッチの回数を算出しているが、特定のモジュール内における開発者のソースコードに対する貢献を定量化する際、パッチの回数ではなく、パッチの行数で判断する方法も考えられる。

本研究でパッチの行数ではなく、パッチの回数を採用した理由は、OSS 開発において、投稿されるパッチのサイズは小さいものが多いためである。Weißgerber らは、OSS プロジェクトに投稿されたパッチを分析し、投稿された 80% 以上のパッチが 10 行未満であることを明らかにした [29]。投稿されるパッチの大部分がサイズの小さいパッチであれば、パッチの行数およびパッチの回数の間に高い相関関係があることになる。そのため、本研究では、パッチの行数ではなく、パッチの回数を採用した。

7.6 制約

7.6.1 モジュール分割方法

本研究で行った評価実験では、各プロジェクトをモジュール単位で分割する際、ソースコードの構造や不具合管理システムのコンポーネント情報、プロジェクトの Wiki ページを参照し、これらの情報を基に、著者が分割を行った。そのため、モジュール分割の際の明確な基準は存在しないため、本研究のモジュール分割方法では、著者の主観による分割となっている。そのため、分割を行う人によって、モジュール分割の結果および、モジュールの粒度が異なる可能性がある。モジュール分割の結果が異なることにより、予測結果が異なる可能性があることに留意されたい。

7.6.2 不具合情報とコミット情報のひも付け方法

本研究では、モジュールオーナーを特定のモジュールに投稿されたパッチをバージョン管理システムにコミットしている開発者とした。そのため、不具合の情報と不具合修正のために行ったコミットの情報とのひも付けを行う必要があった。不具合情報とコミット情報のひも付け方法として、コミットの説明文に含まれるテキストから、不具合 ID を抽出するアプローチをとった。しがしながら、不具合修正のためのコミットであるのにも関わらず、コミットの説明文に不具合 ID を記載しないコミットが存在する。そのようなコミットが存在することにより、データセットに偏りが発生してしまい、本来の結果とは異なる結果になってしまう可能性があることが報告されている [4, 7, 5]。

本研究においても、この不具合情報とコミット情報のひも付けミスによるデータセットの偏りが生じる可能性があるものの、今回行った実験では、総コミット回数が 15 万回以上あるプロジェクトを対象にしていることから、各コミットが不具合修正のためのコミットなのかどうかを目視で判断することは現実的に不可能である。そのため、今回では、コミットの説明文に含まれるテキストから、不具合 ID を抽出するアプローチをとり、機械的に不具合情報とコミット情報のひも付けを行った。

第8章 関連研究

本章では，本研究の位置づけを明らかにするために，コミッター昇格に関する研究および，開発者の適性に関する研究，モジュールオーナーに関する研究について紹介する．

8.1 コミッター昇格に関する研究

本研究は，OSS 開発において重要な役割を担うコミッターに注目した．コミッターに関する研究が盛んに行われており，本節では，OSS 開発における，開発者のコミッター昇格プロセスに関する研究について述べる．

Jensen らは，OSS プロジェクト内での開発者の役割が変化するプロセスを明らかにするために，開発者にインタビューを行うことや，公開されているドキュメントなどの分析を行っている．分析の結果，プロジェクトのコア開発者からの推薦や承認を経て，コミッターへの昇格が行われていることを明らかにしている [21]．

Zhou らは，活動期間の長い開発者を Long-Term Contributor (LTC) とし，開発者がプロジェクトに参加した初期の時点で将来開発者が LTC となるかどうかの予測を行っている [31, 32]．ケーススタディの結果，開発者が LTC となるかどうかは，プロジェクトを取り巻く環境および，開発者の意欲，開発者の能力によってモデル化できることを明らかにした．Zhou らは，コミッターに限定した予測ではなく，コミット権限を持たなくとも，活動期間の長い開発者を予測している．

Shinha らは，コミッターに昇格するなどといったプロジェクトのコア開発者になる要因を，ソースコードの変更履歴や，報告された不具合の修正履歴などの情報を用いて分析している [27]．Shinha らは，コア開発者になる開発者は，OSS にとってより重要な部分のコードに対しての貢献があることや，プロジェクトのコアメンバーである開発者と協調作業を行っていることを明らかにした．

伊原らは，コミッターと呼ばれるリポジトリへ直接変更を反映する権限を持つ開発者を選出することを支援するために，開発者の中からコミッターに推薦すべき有能な開発者を見つけ出す有用な活動量を Eclipse Platform プロジェクトおよび，Mozilla Firefox プロジェクトを対象に分析している [33, 20]．[33] では，活動量としてパッチ投稿数，パッチ検証数，コメント投稿適合率数，活動期間などのメトリクスを用いてコミッターに昇格する開発者を予測しており，適合率は両プロジェクトともに 10%以下となったが，再現率が 70%を超える結果となっており，正確にコミッターを予測できてはいないが，コミッター

に昇格した開発者のうち、70%を超える開発者は提案モデルで予測することができていると結論付けている。

Gharehyazie らは、コア開発者の昇進にはパッチ投稿などの技術的な貢献だけでなく、議論に参加するなどの社会的な貢献も考慮され则认为、開発者間のコミュニケーションによるソーシャルネットワークメトリクスを用いて、将来の開発者（コミッターなど）を予測している [14]。

Bird らは、プロジェクトのコア開発者に昇進するプロセスを明らかにするために、開発者の活動内容および活動期間に関する分析を生存分析手法を用いて行っている [8]。Apache プロジェクトおよび、Python プロジェクト、Postgres プロジェクトにおいて分析を行った結果、コア開発者に昇進している開発者は活動開始から約 1 年前後で昇進しており、1 年を過ぎるとコア開発者に昇進する確率が下降していることを明らかにした。また [8] では、開発者の活動期間が 1 年よりも短い場合、コア開発者にふさわしいかどうかを判断することは難しく、1 年よりも長い場合、開発者はモチベーションを失ってしまう可能性がある」と述べられている。

これらの研究では、OSS に参加している開発者がプロジェクトのコア開発者になるまでのプロセスのモデル化を行っているが、本研究では、各開発者のモジュールごとの適性を考慮している点で異なっている。

8.2 開発者の適性に関する研究

本研究では、モジュールのオーナーにふさわしい開発者を推薦するために、モジュールオーナーの適性を定義した。本研究と同様に、各開発者の適性を把握するための研究が行われている。以下に、各開発者の適性を把握するための研究について述べる。

McDonald らは、Expertise Recommender と呼ばれるシステムを提案した [22]。Expertise Recommender は、Web ブラウザで動作するシステムであり、ユーザーがエラーメッセージなどの検索したいキーワードを入力すると、Expertise Recommender は、そのキーワードに連想される開発者のリストを返す。キーワードに連想される開発者は、バージョン管理システムの変更履歴からキーワードが含まれるファイルなどを変更している開発者を抽出することにより、判定される。

Mockus らは、開発者の適性を可視化するために、Expertise Browser というツールを開発した [23]。Expertise Browser は、開発者のソースコードの変更履歴を収集することにより、各開発者がどういった言語を得意としているか、といったことや、特定のファイルはどの開発者によって主に変更されているかなどといった情報をユーザーに提示している。Expertise Browser を用いることにより、特定の専門分野に詳しい開発者を発見することが容易になり、開発に関する議論をより効率的に行うことが期待できる。

Alonso らは、各コミッターがどの技術分野に特に詳しいのかを明らかにするために、各コミッターが得意とする技術分野を可視化するツールを開発した [1]。Alonso らが開発し

たツールでは、ソースコードの構造および、コミット時の説明文から、各コミッターがよく変更しているファイルに含まれるキーワードや、コミット時の説明文によく記述しているキーワードを抽出する。そして、それらのキーワードをタグクラウドによって表示している。Alonso らが開発したツールは、コミッターではない開発者がコミッターに質問をする際に利用されることを期待している。

Anvik らは、報告された不具合を担当する適任の開発者を推薦するために、バージョン管理システムのデータを用いた場合と、不具合管理システムのデータを用いた場合での予測結果の比較を行っている [3]。バージョン管理システムのデータを用いる場合では、報告された不具合に関連するモジュールを編集している開発者を推薦し、不具合管理システムのデータを用いる場合では、報告された不具合と関連する不具合の CC List に入っている開発者や、関連する不具合にコメントを投稿している開発者などを推薦している。実験の結果、どちらのモデルにおいても、高い精度で予測することができたことがわかった。

このように、各開発者の適性を把握するための研究が盛んに行われており、本研究でも、各開発者のモジュールごとの適性を考慮していることから、これらの研究と共通する部分も少なくない。しかしながら、本研究は、モジュールオーナーとなる開発者を予測している点でこれらの研究とは異なっている。

8.3 モジュールオーナーに関する研究

本研究では、特定のモジュールを担当するコミッターをモジュールオーナーとし、モジュールオーナーを開発者の中から特定するための指標を定義した。本研究と同様に、モジュールオーナーに着目した研究が行われている。以下に、モジュールオーナーに関する研究について述べる。

Microsoft 社社員である Bird らは、Windows Vista と Windows 7 の 2 つのリリースに対して、モジュールのオーナーシップと不具合数との関係を分析した [9]。オーナーシップを測るメトリクスとして、モジュールに変更を加えた開発者の合計や、最もモジュールを変更した開発者の変更回数が全変更回数に占める割合などとしている。ここで、Bird らの研究では、モジュールの単位をファイル単位として分析を行っている。分析の結果、オーナーシップメトリクスが低ければ、不具合数が多い結果となったことから、オーナーシップが明確に決められていないモジュールは、品質が低下することを明らかにした。この結果は、Nagappan らによる研究 [24] での結果を支持している。

Greiler らは、[9] の研究のリプリケーションスタディとして、オーナーシップを定量化する新たなメトリクスを定義し、モジュールのオーナーシップと不具合数との関係を分析した [15]。分析の結果、新たに定義したオーナーシップメトリクスを用いることにより、[9] に比べ高い精度で不具合予測を行うことができた。Greiler らの研究では、モジュールの単位をファイル単位および、ディレクトリ単位として分析を行っている。

Foucault らは、OSS プロジェクトでの開発者の流出入に着目し、開発者の流出入とソフ

ソフトウェアの品質の関係について分析を行った [13]。プロジェクトレベルで流入した開発者を External Newcommer, モジュールレベルで流入した開発者を Internal Newcommer とし, それぞれの開発者が変更を加えた行数を算出している。分析の結果, External Newcommer に比べ, Internal Newcommer は, 不具合を誘発しないことがわかった。つまり, プロジェクト内でモジュールオーナーが移動する場合, ソフトウェアの品質が低下しない傾向にあることが明らかになった。

これらの研究では, モジュールオーナーとソフトウェア品質の関係について分析が行われているものの, モジュールオーナーを特定するための指標については言及されていないため, 本研究と目的が異なる。また, これらの研究の結果から, 各コミッターが担当するモジュールを明確に決定することがソフトウェアの品質を向上させる要因になり得ることがわかる。そのため, 本研究で行ったように, 各モジュールのオーナーとなり得る開発者を特定することは, ソフトウェアの品質維持活動により影響を与えられられる。

第9章 おわりに

本研究では，特定のモジュールを担当するコミッターをモジュールオーナーと呼び，モジュールオーナーとなるべき開発者をボランティアの開発者の中から特定するための予測モデルを構築した．予測モデル構築のために，先行研究 [20, 33] で用いられている活動量メトリクス（活動期間，パッチ投稿数，コメント投稿数など）をモジュールごとに計測した．また，モジュールオーナーの適性を表現するメトリクスであるパッチ貢献度を新たに定義した．そして，モジュールオーナーとなる開発者を予測する際，パッチ貢献度が予測精度向上に寄与するかどうかの評価実験を行った．評価実験は，5つの大規模プロジェクト（Eclipse Platform プロジェクト，Mozilla Firefox プロジェクト，GCC プロジェクト，GIMP プロジェクト，WebKit プロジェクト）を対象に行った．評価実験の結果，5つのOSSプロジェクトのすべてにおいて，パッチ貢献度を用いない予測モデルに比べ，パッチ貢献度を用いた予測モデルは，予測精度が向上することが明らかになった．

今回行った評価実験では，5つのすべてのOSSプロジェクトにおいて，モジュールオーナーを高い再現率で予測することができた．実際にモジュールオーナーとなった開発者を漏れなく予測することができることが明らかとなったため，実際のOSSプロジェクトにおいて，本手法が適用されることを期待する．

外部発表

国内会議

- 山谷 陽亮, 大平 雅雄, “モジュールごとの活動量を考慮したモジュールオーナー候補者予測：大規模 OSS プロジェクトへの適用” 情報処理学会研究報告 第 191 回ソフトウェア工学研究発表会, 大阪府吹田市, 2016 年 3 月（発表予定）.

参考文献

- [1] Omar Alonso, Premkumar T. Devanbu, and Michael Gertz. Expertise identification and visualization from cvs. In *Proceedings of the 5th International Workshop on Mining Software Repositories (MSR'08)*, pp. 125–128, 2008.
- [2] John Anvik, Lyndon Hiew, and Gail C. Murphy. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)*, pp. 361–370, 2006.
- [3] John Anvik and Gail C. Murphy. Determining implementation expertise from bug reports. In *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR'07)*, pp. 9–16, 2007.
- [4] Adrian Bachmann and Abraham Bernstein. Software process data quality and characteristics – a historical view on open and closed source projects. In *Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops (IWPSE-EVOL'09)*, pp. 119–128, 2009.
- [5] Adrian Bachmann, Christian Bird, Foyzur Rahman, Premkumar Devanbu, and Abraham Bernstein. The missing links: Bugs and bug-fix commits. In *Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'10)*, pp. 97–106, 2010.
- [6] Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and Sunghun Kim. Duplicate bug reports considered harmful . . . really? In *Proceedings of the 24th International Conference on Software Maintenance (ICSM'08)*, pp. 337–345, 2008.
- [7] Christian Bird, Adrian Bachmann, Eirik Aune, John Duffy, Abraham Bernstein, Vladimir Filkov, and Premkumar Devanbu. Fair and balanced? bias in bug-fix datasets. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (FSE'09)*, pp. 121–130, 2009.
- [8] 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.
- [9] 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.
- [10] Leo Breiman. Random forests. *Machine Learning*, Vol. 45, pp. 5–32, 2001.
- [11] Laura Dabbish, Colleen Stuart, Jason Tsay, and James D. Herbsleb. Leveraging transparency. *IEEE Software*, Vol. 30, No. 1, pp. 37–43, 2013.
- [12] Karl Fogel. *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly Media, 2005.
- [13] Matthieu Foucault, Marc Palyart, Xavier Blanc, Gail C. Murphy, and Jean-Rmy Falleri. Impact of developer turnover on quality in open-source software. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (FSE'15)*, pp. 829–841, 2015.
- [14] Mohanmmad Gharehyazie, Daryl Posnett, and Vladimir Filkov. Social activities rival patch submission for prediction of developer initiation in oss projects. In *Proceedings of the 29th International Conference on Software Maintenance (ICSM'13)*, pp. 340–349, 2013.
- [15] Michaela Greiler, Kim Herzig, and Jacek Czerwotka. Code ownership and software quality: A replication study. In *Proceedings of the 12th Working Conference on Mining Software Repositories (MSR'15)*, pp. 2–12, 2015.
- [16] Monika Gupta, Ashish Sureka, and Srinivas Padmanabhuni. Process mining multiple repositories for software defect resolution from control and organizational perspective. In *Proceedings of the 11th International Workshop on Mining Software Repositories (MSR'14)*, pp. 122–131, 2014.
- [17] 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.
- [18] James D. Herbsleb, Audris Mockus, Thomas A. Finholt, and Rebecca E. Grinter. An empirical study of global software development: Distance and speed. In *Proceedings of the 23rd International Conference on Software Engineering (ICSE'01)*, pp. 81–90, 2001.
- [19] Kim Herzig, Sascha Just, and Andreas Zeller. It's not a bug, it's a feature: how misclassification impacts bug prediction. In *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*, pp. 16–24, 2013.

- [20] 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.
- [21] 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.
- [22] David W. McDonald and Mark S. Ackerman. Expertise recommender: A flexible recommendation system and architecture. In *Proceedings of the 3th ACM Conference on Computer Supported Cooperative Work (CSCW'00)*, pp. 231–240, 2000.
- [23] Audris Mockus and James D. Herbsleb. Expertise browser: A quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering (ICSE'02)*, pp. 503–512, 2002.
- [24] Nachiappan Nagappan, Brendan Murphy, and Victor R. Basili. The influence of organizational structure on software quality: An empirical case study. In *Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, pp. 521–530, 2008.
- [25] Art B. Owen and Yi Lin. Infinitely imbalanced logistic regression. *Machine Learning*, Vol. 8, pp. 761–773, 2007.
- [26] Foyzur Rahman and Premkumar Devanbu. How, and why, process metrics are better. In *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*, pp. 432–441, 2013.
- [27] Vibha Singhal Sinha, Senthil Mani, and Saurabh Sinha. Entering the circle of trust: Developer initiation as committers in open-source project. In *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR'11)*, pp. 133–142, 2011.
- [28] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In *Proceedings of the 2th International Workshop on Mining Software Repositories (MSR'2005)*, pp. 1–5, 2005.
- [29] Peter Weißgerber, Daniel Neu, and Stephan Diehl. Small patches get in! In *Proceedings of the 5th International Working Conference on Mining Software Repositories (MSR'08)*, pp. 67–76, 2008.

- [30] Yunwen Ye and Kouichi Kishida. Toward an understanding of the motivation open source software developers. In *Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*, pp. 419–429, 2003.
- [31] Minghui Zhou and Audris Mockus. Developer fluency: achieving true mastery in software projects. In *Proceedings of the 18th ACM SIGSOFT international symposium on Foundations of software engineering (FSE'10)*, pp. 137–146, 2010.
- [32] Minghui Zhou and Audris Mockus. What make long term contributors: Willingness and opportunity in oss community. In *Proceedings of the 34th International Conference on Software Engineering (ICSE'12)*, pp. 518–528, 2012.
- [33] 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.