情報処理学会論文誌 Vol.59 No.3 834–844 (Mar. 2018)

開発状況メトリクスを用いた OSS不具合修正時間予測モデル

伊原 彰紀[1,a)] 若元 亮樹[1] 松本 健一[1]

受付日 2017年6月11日, 採録日 2017年12月8日

概要：本論文では，オープンソースソフトウェア（OSS）開発における不具合修正時間の予測に向けて，修正対象となる不具合の特徴量（不具合メトリクス）だけでなく，不具合修正作業に影響することが知られているプロジェクトの開発状況（開発者の作業量，プロダクトの変化量等）を表すメトリクス（開発状況メトリクス）を用いたモデルを提案する．具体的には，従来研究が使用していた不具合の特徴を表す不具合メトリクスに加え，本論文ではバージョン管理システム，開発者メーリングリスト，不具合管理システム，レビュー管理システムに記録された開発者の作業量，プロダクトの変化量を開発状況メトリクスとして用いる．代表的なOSSプロジェクトであるQtプロジェクト，および，OpenStackプロジェクトにおいて報告された不具合を対象としたモデル適用実験の結果，本論文が提案する不具合修正時間予測モデルは，2週間以内に修正される不具合の特定に有効であることが明らかとなった．

キーワード：オープンソースソフトウェア，不具合修正時期，開発状況メトリクス

A Predicting Method of Issue-fixing Time Using Development Metrics in OSS Projects

Akinori Ihara[1,a)] Ryoki Wakamoto[1] Ken-ichi Matsumoto[1]

Received: June 11, 2017, Accepted: December 8, 2017

Abstract: This study proposes a model to predict bug-fixing time using bug metrics and new metrics "development metrics" which describes developer's activities and product updates in OSS projects. Existing studies have proposed the model using only bug metrics which is measured from only bug tracking system. This study build a model using bug metrics and development metrics which is measured from version control system, mailing list, bug tracking system, and review management system. Using OpenStack project dataset and Qt project dataset as a case study, we built the bug fixing time prediction model. As the result, the prediction model achieved to improve the accuracy to identify bugs which the project could be fixed in two weeks after reporting.

Keywords: open source software, bug fixing time, development metrics

1. はじめに

今日，Android OSを搭載したスマートフォンやウェブブラウザFirefoxが広く普及し，我々が生活する多くの場面でオープンソースソフトウェア（OSS）が利用されるようになった．その結果，OSSを開発するプロジェクトに膨大な不具合情報が寄せられている．たとえば，Mozilla Firefoxプロジェクトでは，1日に数百件もの不具合報告が寄せられている．

OSSプロジェクトに報告された不具合は，管理者が修正順位の優先付けを行い，優先順位の高い不具合から順に開発者に修正を依頼する．管理者は，個々の不具合の修正時間を予測し，次期リリースに含める変更を決定する．

不具合の修正時間予測手法は，従来研究で多数提案されており [5], [12], [13], [25], [37]，多くの研究は不具合の特徴（不具合が混入した機能，優先度等）に基づいて予測モデルを構築している．しかし，OSS開発では，積極的に機能

[1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
[a)] akinori-i@is.naist.jp

を追加する時期，機能の追加を抑える保守的な時期が日々変化するため不具合の特徴のみが修正時間に影響するとは考え難い [26]．OSS プロジェクトの開発状況を考慮している研究の 1 つとして，Giger ら [11] は，不具合修正時間予測モデルを構築するための説明変数に報告年月の情報を使用している．Giger らの実験では，報告年月の情報が他の説明変数に比べてモデルに強く貢献していることを示している．その理由は，同じ時期に報告された不具合は，修正時間に差がなかったことが 1 つの原因と考えられる．つまり，不具合の修正時間は，不具合の特徴のみが修正時間に影響しているとはいえない．

本論文では，報告時期による不具合修正時間の違いを考慮するために，OSS プロジェクトに不具合が報告された直前の開発状況（開発者の作業量，プロダクトの変化等）を表すメトリクス（開発状況メトリクス）を説明変数として使用する不具合修正時間予測モデルを提案する．実験では，従来研究で使用している不具合の特徴のみを説明変数として用いた予測モデルの精度と比較する．

昨今のソフトウェア開発では，ソースコードをはじめ，不具合情報，開発者間の議論，コードレビュー記録等を様々な管理システムに蓄積している．本論文では，不具合報告日の直前の数日間に各管理システムに蓄積された開発状況を表す記録を計測し，不具合修正時間の予測に使用する．そして，不具合の報告日から指定期間内に開発者が修正を完了するか否かを予測するモデルを構築し，Qt プロジェクト，および，OpenStack プロジェクトを対象とした実験を行う．

続く 2 章では，OSS 開発における不具合修正プロセスと，関連研究を述べ本論文の立場を明らかにする．3 章では，開発状況メトリクス，および，不具合修正時間の予測モデルの構築方法について述べる．4 章では，Qt プロジェクト，および，OpenStack プロジェクトを対象とした実験方法について述べ，5 章で，予測結果を示す．6 章で，開発状況メトリクスを用いたモデルが出力する予測精度の妥当性の考察を行い，最後に 7 章で本論文のまとめを述べる．

2. 不具合修正プロセスと修正時間

2.1 不具合修正プロセス

OSS プロジェクトでは，不具合の情報を Bugzilla[*1]，JIRA[*2]をはじめとする不具合管理システム（BTS: Bug Tracking System）で管理し，修正を行う．**図 1** は，一般的な不具合修正プロセスを示す．従来研究では，この不具合修正プロセスにおいて，修正作業が遅延する際に直面する様々な課題に着目している．代表的な例を以下に示す．

1. 不具合報告の受付．不具合管理システムには，不具合を発見した開発者，利用者が不具合情報を登録する．一般的には不具合を発見した機能，利用している開発環境（コンピュータの種類，OS 等），優先度，機能の振舞い，再現方法等を報告するが，OSS の開発に関与していない参加者が報告すると，修正に必要な情報が欠けていることが多く，開発者が報告者に追加情報を問い合わせる手間が必要になる [4], [10], [21]．

2. 不具合の振り分け．プロジェクトに報告された不具合は，必ずしも修正されるとは限らない．すでに報告済みの不具合，一部の環境にのみ発生する優先度の低い不具合，等も含まれている．膨大に報告される不具合の中から迅速に修正すべき不具合を特定することは容易ではない [15], [27]．

3. 修正担当者の決定．OSS プロジェクトの管理者は，プロジェクトに参加する開発者の専門知識，開発経験を把握できていないため，不具合修正，機能追加を実装する適任の開発者を特定することは容易ではない [3]．また，修正を依頼したとしても長期間対応されず，別の開発者に再依頼することもある [17]．

4. 不具合の修正．開発者が，プロジェクトに報告された不具合情報，および，ソースコードの変更履歴から，修正すべきソースコードを特定することは容易ではない [35]．

5. 修正内容の検証と取り込み．不具合が改修されたか否かを判断するために，複数の開発者による検証作業が行われる．しかし，未熟な開発者の間違った評価により修正時間が遅延する [14], [28]．

不具合修正プロセスにおいて修正作業が遅延する要因には，不具合固有の要因（報告内容の不足，優先度の理解）だけでなく，プロジェクトの開発状況として開発者の作業量，プロダクトの変化を示すソースコードの変更量・頻度，開発者の貢献等も影響している．次節では，不具合修正時間の分析や予測のために着目された具体的なメトリクスを示す．

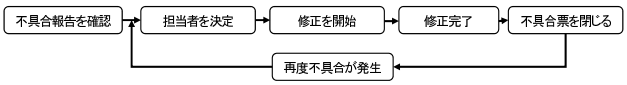

図 1 不具合票の状態遷移
Fig. 1 State transition diagram for bug-fixing.

表 1 不具合修正時間に影響するメトリクス
Table 1 Common metrics for analyzing bug-fixing time.

カテゴリ	メトリクス	関連研究
不具合の特徴	優先度，コンポーネント名，等	[7][11][12][13][16][25][32]
修正プロセス	コミット数，修正担当者の変更回数，等	[17][23][30][33][37]
コミュニティ	報告者，修正者の議論（コメント）数，等	[9][11][15][22][24]

2.2 不具合修正時間に着目した関連研究

従来研究では，不具合の特徴，修正プロセス，開発者コミュニティに着目した不具合修正時間の分析や予測が行われている．**表 1** は，具体的に使用されたメトリクスと関連研究を示す．

[*1] Bugzilla: https://www.bugzilla.org/
[*2] JIRA: https://www.atlassian.com/software/jira

不具合の特徴．多くの従来研究は，不具合票に記録された情報に基づいて修正時間（不具合報告から修正が完了するまでの時間）の分析，予測に取り組んでいる [7], [11], [12], [13], [16], [25], [32]．Herraiz ら [12] は，優先度が高い不具合は，その他の不具合に比べて修正時間が短いことを定量的に示し，優先度の違いによる修正時間の予測を行っている．Hewett ら [13] は不具合報告時の優先度に加え，コンポーネント（機能）名等をメトリクスとして使用し，不具合修正時間を予測するモデルを構築している．また，Giger ら [11] は，不具合の報告年月の情報を使った予測モデルを提案している．

不具合修正プロセス．不具合の修正を開始している不具合に対して，修正時間を予測する場合は，不具合の特徴だけでなく，修正プロセス中のメトリクスを使うことも可能である．たとえば，Jeong ら [17] は修正担当者の変更が繰り返されているプロセスに着目し，修正担当者の変更によって修正完了までの時間が長期化することを指摘している．また，正木ら [37] は，報告者，修正依頼者，修正担当者が同じ場合と異なる場合で修正時間が異なることを示し，報告者，修正依頼者，修正担当者を説明変数として用いて修正時間の予測を行っている．

開発者コミュニティ．OSS プロジェクトに参加する開発者の社会的関係や，人に着目して不具合修正活動を理解しようとする研究が行われている [6], [9], [15], [22], [24], [30]．Ortu ら [24] は，不具合の修正，および，修正にともなう議論に参加する開発者に着目し，個々の不具合の修正に参加する開発者が少ない場合に修正までの時間が短いことを示し，少ない開発者による迅速な意思決定が修正時間を短縮していると述べている．

2.3 不具合修正時間予測の課題

これまでに不具合修正時間の分析・予測に関する研究が行われてきたが，依然として不具合修正プロセスの課題が修正時間に与える影響を分析した研究が多い．その中でも，Bhattacharya ら [5]，Bosu ら [6] は，プロジェクトの状況，修正時期の違いで，不具合の修正時間に関係するメトリクスが異なることを示している．また，Giger ら [11] は，不具合の報告年月の情報を使った修正時間予測モデルを構築し，不具合の報告時期が修正時間の予測精度の向上に寄与することを確認している．

しかし，従来研究で使用する報告年月の情報では，報告時期が異なる不具合の修正時間を予測することは難しい．また，報告時期の開発状況を表すメトリクスではないため，修正時間が変動した理由を検証することはできない．本論文では，不具合報告直前にプロジェクトが管理するリポジトリへ登録された情報やその変動（開発状況メトリクス）を考慮するために，開発状況メトリクスを用いた不具合修正時間予測モデルを提案する．

3. 不具合修正時間予測モデルの構築

本章では，開発状況メトリクスを用いた予測モデルの構築に向けて，開発状況メトリクスの概要，メトリクスの計測方法，予測モデルに用いるアルゴリズムを説明する．

3.1 概要

本論文では，不具合報告から修正が完了がするまでの時間（不具合修正時間）を予測するモデルを構築する．OSS 開発者，または，管理者は，不具合が時期リリースまでに修正が完了するか否かを判断するために不具合修正時間を予測する．提案する予測モデルは，次期リリースまでに修正する不具合の特定を可能にする．昨今，ラピッドリリースを行う Firefox をはじめとして，多くの OSS プロジェクトではリリースサイクルが短期化しているため，本論文では短期間で修正される不具合を対象とし，不具合の報告日から 1 日以内，3 日以内，1 週間以内（7 日），2 週間以内（14 日），1 カ月以内（30 日）に修正されるか否かを予測するモデルを構築する．

3.2 予測モデル構築のためのメトリクス

本論文では，不具合修正時間の予測モデルを構築するために，従来研究で使用されている不具合メトリクスに加え，開発状況メトリクスを用いる．

3.2.1 不具合メトリクス

表 2 に本実験で用いる不具合メトリクスを示す．不具合票には不具合を発見した機能，利用している開発環境（コンピュータの種類，OS 等），優先度，機能の振舞い，再現方法等が記録されている．本論文では，報告時点で不具合票から取得可能な情報を不具合メトリクスとして用いる．特に優先度は，OSS プロジェクト固有の名前がつけられ，種類も多い．また，不具合に付される優先度の頻度が偏る（中程度の優先度が最も多く付される）．本論文では，各プロジェクトにおいて最も多く付されている優先度（OpenStack プロジェクトにおける “Medium”，Qt プロジェクトにおける “P2: Important”）を “Medium” とし，それより高い優先度を “High”，低い優先度を “Low”，優

表 2 不具合メトリクス
Table 2 Bug metrics.

リポジトリ	変数名	尺度	概要
BTS	Priority	名義	不具合票に記載される優先度（High, Medium，Low，Unknown）
	TicketType	名義	報告内容の種類（Bug, Suggestion 等）
	NumWords	比例	不具合票に記載される詳細情報のワード数
	Year	間隔	不具合報告を行った西暦
	Month	名義	不具合報告を行った月
	Weekday	名義	不具合報告を行った日（平日，休日）
	BuggyComp	名義	不具合が発生したコンポーネント名
	Reporter	名義	不具合報告した開発者名

表 3 開発状況メトリクス
Table 3 Development metrics.

リポジトリ	変数名	尺度	概要
VCS	NumCommit	比例	コミット数
	NumFiles	比例	変更ファイルの延べ数
	NumSys	比例	変更サブシステムの延べ数
	NumAddLine	比例	コード追加行数
	NumDelLine	比例	コード削除行数
	NumActCom	比例	コミットした開発者（コミッタ）の延べ数
	NumCoreDev	比例	コア開発者の延べ数
ML	NumSend	比例	メール送信者の延べ数
	NumPostThread	比例	スレッド作成数
	NumReply	比例	返信メール数
BTS	NumTicket	比例	不具合票作成数
	NumWorkingTicket	比例	未修正の不具合数
	NumTicketFixed	比例	不具合修正数
	NumDevReported	比例	不具合報告している開発者の延べ数
	NumDevFixed	比例	不具合修正している開発者の延べ数
	NumDevRepFix	比例	不具合の報告・修正の両方に取り組む開発者の延べ数
	NumDevMess	比例	不具合票にコメントする開発者の延べ数
	NumFixer	比例	アサインされた開発者の延べ数
	NumHighTicket	比例	優先度 “High” の不具合票報告数
	NumMessageITS	比例	不具合票へのコメント数
	NumAssign	比例	アサインした回数
RMS	NumAccept	比例	アクセプトされたパッチ数
	NumReject	比例	リジェクトされたパッチ数
	NumPatchOwner	比例	パッチ作成者の延べ数
	NumReviewer	比例	パッチへコメントする開発者の延べ数
	NumReviewedDev	比例	レビューを行う開発者の延べ数
	NumPatch	比例	投稿パッチ数
	NumFirstPatch	比例	ファーストパッチの投稿数
	NumReviewMess	比例	パッチへのコメント総数

先度が不明なものを “Unknown” とする．

3.2.2 開発状況メトリクス

本論文で提案する開発状況メトリクスは，不具合が報告された直前の時期におけるプロジェクトの開発状況（開発者の作業量，プロダクトの変化量等）を理解するためのメトリクスである．具体的な開発状況メトリクスとして，バージョン管理システム（VCS: Version Control System），開発者メーリングリスト（ML: Mailing List），不具合管理システム（BTS），レビュー管理システム（RMS: Review Management System）の 4 つのリポジトリから**表 3** に示すメトリクスを計測する．これらのリポジトリは，OSS プロジェクトにおいて開発の進捗情報を随時記録するためのシステムとして使用され，不具合修正活動を理解するための研究で頻繁に利用されている．次に，各リポジトリの概要と，リポジトリから計測する開発状況メトリクスを説明する．

バージョン管理システム（VCS）． VCS は，ソフトウェアを構成するソースコードをはじめドキュメント等のファイルを管理し，それらのファイルの変更履歴を記録するシステムである．ソースコードの変更量（追加，削除）が多いほど，また，サブシステム（ソフトウェアを構成する最も上位のフォルダ，いい換えると，独立性の高いパッケージ）数の変更が多いほど，大規模な修正が必要となり欠陥が混入する原因となる [20], [29]．また，ソフトウェアの品質を維持するために一部の開発者（コミッタ）のみが VCS のファイルを変更する権限を持ち，コミッタがコードレビューの最終判断を行うため，コミッタ数もプロジェクトの状況を理解するために重要である [18]．しかし，コミット権限を持っていたとしても多くの開発者は活発に活動していない [36]．コミッタの中の約 2 割の開発者（コア開発者）が，プロダクトの変更の約 8 割を開発している [19]．したがって，コア開発者が少数の時期は，検証作業が遅れることが示唆される．本論文では，不具合が報告された直前の時期における OSS の変更規模を理解するために，コミッタ数と同様に，コア開発者（計測対象期間のコミット数の多い上位 2 割の開発者）を計測する．

開発者メーリングリスト（ML）． OSS プロジェクトに参加している開発者は ML を用いて，不特定多数の開発者がリリーススケジュールや開発の方針（機能追加，不具合修正等）に関する議論を行う．Abreu ら [1] は，開発者間のコミュニケーションの量が多い時期に，欠陥の混入数が多いことを実験的に示している．

不具合管理システム（BTS）． BTS は，OSS に発見された不具合の情報を 1 件ずつ不具合票に登録し，修正状況の把握や過去に報告された不具合を追跡するためのシステムである．2.1 節で述べたとおり，不具合修正プロセスにおいて修正作業が遅延する様々な要因は多岐にわたる．

レビュー管理システム（RMS）． RMS は，開発者が作成，または，変更したソースコードを登録し，当該ソースコードの振舞い，欠陥の有無を作成者以外の開発者が検証することを支援するシステムである．従来は BTS を使った検証作業が主流であり，BTS を対象とした分析で検証作業に関わる人数，議論（コメント数）が修正作業が遅延する要因となっている [2]．しかし，昨今 Gerrit，ReviewBoard をはじめとした RMS を利用する OSS プロジェクトが増加しており，検証作業における修正時間の遅延について多くの研究者が分析を進めている [28]．

3.3 予測モデルの構築方法

3.3.1 メトリクスの計測方法

不具合メトリクスは，不具合情報が BTS に登録された時点に取得できる情報を取得する．開発状況メトリクスは，不具合報告の直前の期間における活動量や開発者数を計測する．

図 2 は，本論文で提案する開発状況メトリクスの計測方法を示す．個々の不具合に対してメトリクスを計測する場合，不具合数とメトリクス数に比例してモデル構築の準備に時間を要するため，本論文では，Thomas ら [31] が不具合修正箇所を特定するために使用しているアプローチと同様に，分析対象期間を任意の期間（本論文では 10 日間）で

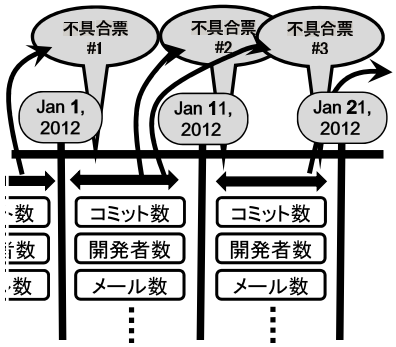

図 2 開発状況メトリクスの計測方法
Fig. 2 Extraction method for development metrics.

分割し，対象とする不具合が報告日を含む期間の 1 つ前の期間で開発状況メトリクスを計測する．たとえば，不具合が任意の月の 13 日に報告された場合，同月の 1 日〜10 日の活動量を開発状況メトリクスとして計測する．

本論文が開発状況メトリクスを計測するために 10 日間の期間で分割した理由は，不具合ごとに開発状況メトリクスを計測した場合，実験対象の不具合の量から，メトリクスの計測だけで膨大な時間がかかるためである．また，分割期間を 10 日にした理由は，計測期間が長い場合は開発状況メトリクスに古い情報（開発者の離脱，大規模な修正等）がノイズとして混入し，短い場合は時期による開発状況メトリクスの違いをとらえることが難しいと考えたためである．分割期間の妥当性は 6 章で議論する．

3.3.2 予測アルゴリズム

本論文で提案する予測モデルではランダムフォレスト法を用いて，不具合の報告日から，ある期間（1 日，3 日，7 日，14 日，30 日）以内に修正完了するか否かを予測する．ランダムフォレスト法は，回帰木を用いて集団学習を行う手法であり，2001 年に Breiman によって提案された [8]．ランダムフォレスト法を用いた理由として，集団学習を行うことに適していることや，提案手法のように多くの説明変数を用いる場合にも利用できること，また各メトリクスの重要度を抽出できることがあげられる．モデル構築用のデータセットに対して繰り返しランダムサンプリングを行い，得られたサンプル群から多数の回帰木を構築する．各回帰木の出力の平均により最終的な予測結果を抽出する．

4. 実験方法

4.1 データセット

本実験では，予測モデルを評価するために，大規模 OSS（OpenStack, Qt）プロジェクトを対象に実験を行う．OpenStack は，Rackspace Hosting と NASA が OSS として立ち上げたクラウド環境構築システムである．また，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，OSS 開発のように不特定多数の開発者が貢献しているプロジェクトである．これらのプロジェクトでは，4 種類のリポジトリ（VCS，ML，BTS，RMS）をすべて利用している．本論文が各リポジトリで対象とするデータの概要を**表 4** に示す．

表 4 リポジトリの統計量
Table 4 Statics of target datasets.

	OpenStack	Qt
分析対象期間	Aug.2012-May.2014	Nov.2011-Oct.2013
コミット数	91,403	55,450
メール数	36,144	13,654
パッチ数	82,581	62,053
不具合票数	37,825	27,485

表 5 実験対象の不具合票件数
Table 5 Target bug reports.

	OpenStack	Qt
全不具合票数	37,825	27,485
解決された不具合票数	29,175	18,032
実験対象の不具合票数	17,117	7,913

表 6 特定の期間内に修正された不具合票数
Table 6 Bug reports identified by each prediction model.

プロジェクト名	OpenStack	Qt
対象不具合数	17,117	7,913
1 日以内	3,640 (21.3%)	826 (10.4%)
3 日以内	5,434 (31.7%)	1,307 (16.5%)
7 日以内	7,817 (45.7%)	2,155 (30.0%)
14 日以内	9,839 (57.5%)	2,463 (31.1%)
30 日以内	11,923 (69.7%)	3,938 (49.8%)

本実験では，2012 年 8 月から 2014 年 5 月までに OpenStack プロジェクトに報告された不具合票，および，2011 年 11 月から 2013 年 10 月までに Qt プロジェクトに報告された不具合票を対象に不具合修正時間を予測する実験を行う．**表 5** は，実験対象期間中に報告された全不具合票，そのうち解決された不具合票，そして本論文が実験対象とする不具合票の件数を示す．本論文で対象とする不具合は解決された不具合であるが，その中には，修正を行わないと決定した不具合，報告された内容がソフトウェアの仕様であった不具合を含む．本論文では，ソースコード等に修正が加えられた不具合のみを対象とする．また，不具合票に記載される修正の進捗状況を表すステータスが，実際の進捗に合わせて変更されず，多数の不具合票のステータスをまとめて切り替えていることがある [34]．本実験では，ステータスの変更を分析することで修正時間を計測しているため，このような場合は正しい修正時間を予測することができない．したがって，同日に多くの不具合が修正完了

を示すステータスに変更された不具合は実験対象から除外する．最終的に OpenStack プロジェクト 17,117 件，Qt プロジェクト 7,913 件を修正完了した不具合として実験対象とする．

4.2 評価基準

本論文では，不具合修正時間予測モデルの評価基準として，適合率，再現率，F1 値を用いる．適合率（Precision）は，指定期間以内に修正されると予測した不具合の中で，実際に修正された不具合数の割合を示す．再現率（Recall）は，指定期間内に修正された不具合の中で，修正する不具合と正しく予測できた不具合数の割合を示す．また，適合率と再現率はトレードオフ（一方の値が高くなると，他方が低くなる）の関係があるため，予測モデルを総合的に評価するために，適合率と再現率の調和平均である F1 値も評価基準として用いる．なお，適合率，再現率，F1 値はいずれも値域 [0, 1] をとり，値が高いほど予測精度が高く，値が低いほど予測精度が低いことを示す．

4.3 実験手順

本節では，3 章で説明した不具合メトリクス，開発状況メトリクスを用いて，指定期間内に不具合修正が完了するか否かを予測するための手順を説明する．

Step1. 説明変数の準備 名義尺度で扱っているメトリクスは，出現頻度が高い上位 5 つとその他 1 つの合計 6 種類のダミー変数に変換する．ただし，ダミー変数が多いと多重共線性の問題が生じるため，R パッケージ `redun` を用い，変数間の相関が 0.8 以上の場合，一方の変数を削除する．

Step2. 目的変数の準備 不具合が報告日から指定日数以内に修正完了するか否かを予測する．昨今の OSS プロジェクトではリリース間隔が短期化しているため，本論文で構築する予測モデルの指定日数を 1 日以内，3 日以内，7 日以内，14 日以内，30 日以内とする[*3]．

Step3. データセットの分割 予測モデルの評価として十分割交差検定を行うために，データセットを，9 個のフィットデータと 1 個のテストデータに分割する．

Step4. 不具合修正時間予測モデルの構築 フィットデータを学習するためにランダムフォレストを用いて予測モデルを構築する．本論文では説明変数の異なる 2 種類の予測モデル（不具合メトリクスのみで構築したモデル，不具合メトリクスと開発状況メトリクスで構築したモデル）を用意し，また，それぞれのモデルに対して目的変数の 5 種類の指定日数（1 日，3 日，7 日，14 日，30 日）を比較するため，合計 10 種類の予測モデルを構築する．

Step5. 不具合修正時間予測モデルの評価 10 種類の予測モデルに対して，同じテストデータを用いて予測実験を行い，精度を評価する．

Step6. 実験の繰返し 十分割交差検定のテストデータを変えて検証を 10 回繰り返し，予測精度の中央値を求める．また，予測期間単位での精度評価に加え，提案する予測システム全体としての精度評価として Macro-average，および，Micro-average による予測精度を求める．

[*3] 実験対象とする Qt プロジェクトと OpenStack プロジェクトのリリース間隔は，それぞれ 29 日～617 日，71 日～364 日であった．

5. 実験結果

本章では，OpenStack プロジェクト，Qt プロジェクトに報告された不具合を対象に構築した修正時間予測モデルの評価実験の結果を述べる．

5.1 予測精度の比較

表 7 は，OpenStack プロジェクト，Qt プロジェクトで報告された不具合を説明変数の異なる 2 種類のモデル（提案手法：不具合メトリクスと開発状況メトリクスを用いて構築したモデル，従来手法：不具合メトリクスのみ用いて構築したモデル），また，それぞれのモデルに対して目的変数の 5 種類の指定日数（1 日，3 日，7 日，14 日，30 日）の予測精度（適合率，再現率，F1 値）を示す．太字は，予測期間別に提案手法と従来手法を比較し，予測精度の高い値を示す．

OpenStack プロジェクトでは，報告日から 1 日以内，3 日以内，7 日以内に修正される不具合の予測精度（F1 値）が従来手法よりも高く，Qt プロジェクトにおいては，30 日以内に修正される不具合の予測精度も従来研究よりも精度が高いことを確認した．特に，再現率が向上していることが分かった．いい換えると，従来手法に比べて予測期間以内に修正された不具合を正確に予測できたことを示す．また，本実験では表 7 に示す予測期間単位での精度評価に加え，提案する予測システム全体としての精度評価の結果を**表 8** に示す．実験の結果，OpenStack プロジェクトでは従来手法の方が高く，Qt プロジェクトでは提案手法の方が高くなった．OpenStack プロジェクトのデータセットから構築した予測モデルの提案手法と従来手法の F1 値の差は，macro-average が 0.17 から micro-average が 0.15 の差であり，それほど大きな差ではない．したがって，OpenStack プロジェクトにおいて 14 日以内，Qt プロジェクトにおいて 30 日以内に修正される不具合を従来手法に比べて高い精度で予測できる提案手法は有効性が期待できる．

5.2 予測モデルにおける変数の重要度

提案手法と従来手法における目的変数（予測期間）の異なる 5 種類の指定日数（1 日以内，3 日以内，7 日以内，14

表 7 不具合修正時間予測の結果
Table 7 Result of bug-fixing time prediction model.

プロジェクト	予測期間	提案手法（不具合 + 開発状況）			従来手法（不具合のみ）		
		適合率	再現率	F1 値	適合率	再現率	F1 値
OpenStack	1 日以内	46.73	**7.45**	**12.85**	**63.33**	0.78	1.55
	3 日以内	54.47	**22.13**	**31.47**	**58.07**	6.22	11.24
	7 日以内	**58.40**	**48.20**	**52.81**	57.84	48.16	52.56
	14 日以内	**64.34**	74.34	68.98	63.93	**77.74**	**70.16**
	30 日以内	**72.08**	91.52	80.64	70.04	**99.83**	**82.32**
Qt	1 日以内	**60.82**	**3.97**	**7.45**	50.00	1.13	2.21
	3 日以内	**40.49**	**5.06**	**8.99**	40.00	1.21	2.35
	7 日以内	46.06	**16.36**	**24.15**	**48.06**	7.09	12.36
	14 日以内	52.92	**35.03**	**42.15**	**61.40**	18.82	28.82
	30 日以内	60.18	**59.25**	**59.71**	**60.65**	56.37	58.43

表 8 Macro-average と Micro-average による不具合修正時間予測の結果
Table 8 Result of bug-fixing time prediction model by macro-average and micro-average.

プロジェクト	評価手法	提案手法（不具合 + 開発状況）			従来手法（不具合のみ）		
		適合率	再現率	F1 値	適合率	再現率	F1 値
OpenStack	macro-average	59.01	**48.85**	53.45	**63.37**	46.48	**53.62**
	micro-average	65.51	60.86	63.10	**65.54**	**61.12**	**63.25**
Qt	macro-average	52.09	**23.93**	**32.80**	**52.20**	17.13	25.79
	micro-average	55.90	**34.17**	**42.41**	**59.51**	26.62	36.79

日以内，30 日以内）で構築した予測モデルについて，モデル構築に使用した説明変数の重要度を分析するために，各説明変数を用いたときの精度減少値を求める [37]．精度減少値は，説明変数を予測モデルから取り除くことによる，予測精度の減少値を示す．精度減少値が高いほど，モデルの精度に寄与している説明変数であることを表す．

表 9 は，不具合修正時間予測モデルの構築で使用したメトリクスの精度減少値が高い 5 つの説明変数を示す．太字は，開発状況メトリクスを示す．提案手法で構築した予測モデルでは，不具合メトリクスだけでなく開発状況メトリクスも上位 5 つに表れ，開発状況メトリクスが予測モデルに貢献していることが分かる．特に従来研究に比べて最も予測精度が向上した 1 日以内に修正される不具合を特定する予測モデルでは，開発状況メトリクス（OpenStack では NumReply，NumFiles，NumDelLine，Qt プロジェクトでは NumSys，NumReject，NumFiles）貢献している．しかし，プロジェクトの違いで精度減少値が高いメトリクスは異なるため，プロジェクトによって不具合修正時間に影響するメトリクスに違いがあることが分かる．

6. 考察

6.1 開発状況メトリクスの推移

本実験結果として，開発状況メトリクスを用いた不具合修正時間予測モデルは，2 週間以内に修正される不具合を特定するために有効であることを確認した．その理由の 1 つとして，不具合報告日（モデル構築日）からの経過期間が長くなるほど，モデルに使用した開発状況メトリクスの影響が小さくなったことが考えられる．具体的には，Qt プロジェクトでは，1 日以内，または，3 日以内に修正される不具合を特定するためには，不採択されるパッチ件数（NumReject）等の開発状況メトリクスを使った予測モデルの精度が不具合メトリクスのみで構築したモデルよりも高くなる一方で，2 週間後以降に修正される不具合を特定するモデルでは開発状況メトリクスを使ったとしても予測精度は向上しない．2 週間後以降の予測精度は，不具合報告から時間経過にともない開発状況が変わり，報告時期の開発状況を表す開発状況メトリクスが予測モデルに与える貢献が小さくなったため，従来手法の予測精度との差がなくなったと考えられる．本節では，複数の予測モデルにおいて効果を示した開発状況メトリクス（OpenStack プロジェクト：NumReply，NumFiles，Qt プロジェクト：NumReject，NumSys）が，分析対象期間を 3 日間隔，7 日間隔，14 日間隔で区切った場合の変化量を分析する．

図 3 と**図 4** は，それぞれ OpenStack プロジェクト，Qt プロジェクトの開発状況メトリクスの変化量を箱ひげ図で示す．分析対象とした 4 つのメトリクスすべてにおいて，3 日間，7 日間で区切った場合の変化量の分散は，14 日間で区切った変化量の分散より小さい．また，各メトリクスにおいて，3 日間，7 日間で区切った場合と，14 日間で区

表 9 不具合修正時間予測モデルの構築に用いたメトリクスの中で精度減少値の高いメトリクス
Table 9 Top 5 important metrics in Bug-fixing time prediction model.

OpenStack				Qt			
1 日以内				1 日以内			
不具合		不具合+開発状況		不具合		不具合+開発状況	
変数	値	変数	値	変数	値	変数	値
Priority_Medium	4.33	**NumReply**	4.11	BuggyComp	3.43	**NumFiles**	3.89
Priority_Low	3.84	**NumFiles**	3.51	Year_2011	2.05	**NumReject**	3.28
NumWords	3.64	**NumDelLine**	3.47	Reporter	1.44	**NumSys**	3.22
Month_Mar	1.84	Priority_High	2.90	TicketType_Bug	1.41	**NumActCom**	3.01
3 日以内				3 日以内			
不具合		不具合+開発状況		不具合		不具合+開発状況	
変数	値	変数	値	変数	値	変数	値
NumWords	5.34	**NumDelLine**	4.69	BuggyComp	3.15	TicketType_Bug	2.46
Priority_Medium	5.15	Priority_Medium	4.54	NumWords	1.98	**NumSys**	2.30
Year_2012	4.09	**NumReply**	4.20	TicketType_Bug	1.86	**NumCoreDev**	2.19
Priority_High	3.74	Priority_High	4.09	Priority_High	1.41	**NumCommit**	2.10
7 日以内				7 日以内			
不具合		不具合+開発状況		不具合		不具合+開発状況	
変数	値	変数	値	変数	値	変数	値
Priority_Medium	10.18	Priority_Medium	5.25	BuggyComp	4.28	BuggyComp	3.19
Priority_Low	8.33	Priority_Low	5.13	TicketType_Bug	2.56	Priority_Low	2.46
NumWords	4.77	**NumReply**	4.93	Reporter	2.16	**NumSys**	2.32
Priority_High	4.27	**NumDelLine**	3.68	Priority_High	2.08	Priority_High	2.02
14 日以内				14 日以内			
不具合		不具合+開発状況		不具合		不具合+開発状況	
変数	値	変数	値	変数	値	変数	値
Priority_Medium	11.17	Priority_Medium	6.60	Priority_High	5.98	Priority_High	5.27
Priority_Low	8.09	Priority_Low	5.44	BuggyComp	3.81	TicketType_Bug	2.99
Priority_High	3.90	BuggyComp	4.23	TicketType_Bug	2.80	Priority_Low	2.09
NumWords	3.73	Priority_High	3.99	Priority_Low	2.18	BuggyComp	1.92
30 日以内				30 日以内			
不具合		不具合+開発状況		不具合		不具合+開発状況	
変数	値	変数	値	変数	値	変数	値
Priority_Low	4.70	Priority_Medium	5.31	Priority_High	8.91	Priority_High	6.26
Priority_Medium	4.16	**NumFiles**	4.80	Priority_Low	4.17	Priority_Low	4.80
BuggyComp	2.77	**NumReply**	4.46	TicketType_Bug	3.58	TicketType_Bug	3.58
NumWords	2.54	**NumReject**	4.28	BuggyComp	2.92	**NumActCom**	2.14
Priority_High	2.29	**NumCommit**	4.26	Year_2013	2.20	**NumSys**	1.97

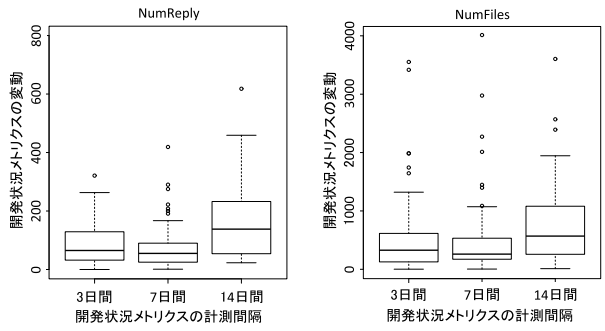

図 3 OpenStack プロジェクトにおける開発状況メトリクスの変動
Fig. 3 Change score of development metrics for each period in OpenStack project.

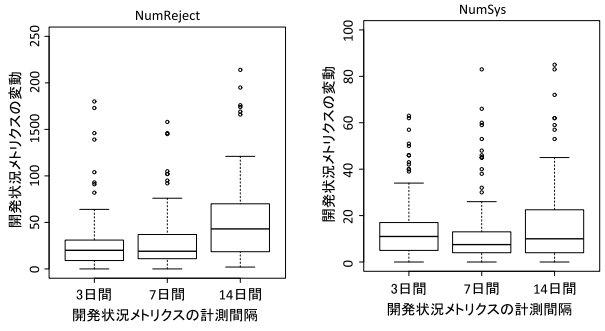

図 4 Qt プロジェクトにおける開発状況メトリクスの変動
Fig. 4 Change score of development metrics for each period in Qt project.

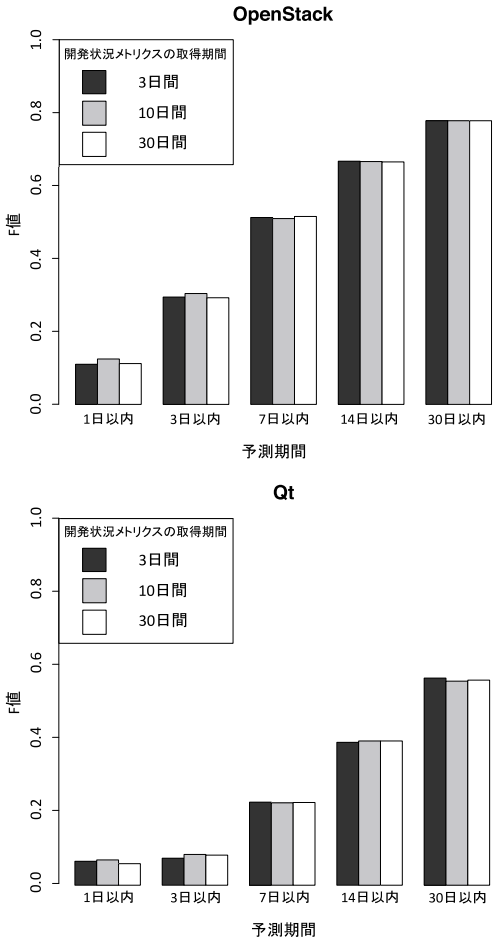

図 5 開発状況メトリクス計測期間による F1 値
Fig. 5 F1-value in each extracted period for development metrics.

切った場合の分布には統計的な有意差を確認した（ウィルコクソンの順位和検定 P 値 < 0.05）．したがって，報告日直前の期間から計測した開発状況メトリクスを用いたとしても，2 週間後以降では開発状況が変化しているため，不具合を特定することは困難であると考えられる．

6.2 開発状況メトリクスの計測期間

本論文の予測実験では，開発状況メトリクスの計測時間として，分析対象期間を 10 日間で分割し，各期間に計測したメトリクスを開発状況メトリクスとして用いた．本節では，開発状況メトリクスを計測する期間の妥当性を確認する．**図 5** は，開発状況メトリクスの計測期間を 3 日間，10 日間，30 日間とした場合の予測精度（F1 値）の違いを示す．分析の結果，開発状況メトリクスの計測期間の違いによる F1 値の差はごくわずかであったため，計測期間による精度への影響はない．

6.3 制約

外的妥当性：本論文では，大規模な OSS プロジェクトである OpenStack，Qt の 2 つのプロジェクトのみを対象に実験を実施した．両プロジェクトでは，ソースコード，不具合，コミュニケーション，コードレビューをそれぞれ一元管理するリポジトリを保持し，これらのリポジトリは，多数の既存研究が開発状況を理解するための研究対象としている．現在，これらすべてのリポジトリを保持するプロジェクトは少なく，本論文では 2 プロジェクトのみを対象とした実験となった．しかし，本実験では，両プロジェクトで開発状況メトリクスの貢献が大きいことを実験で確認することができたため，他のプロジェクトでも同様の結果が期待できる．ただし，本論文でもプロジェクトの違いにより不具合修正時間予測モデルに貢献するメトリクスは異なっていたように，異なるプロジェクトを対象とした場合，予測モデルに貢献するメトリクスは異なる可能性がある．

内的妥当性：本論文で定義した不具合修正時間は，不具合管理システムに記録している時間を基に修正時間を計測している．本論文で計測した修正時間は，開発者が実際に不具合修正に取り組んだ時間とは異なる場合がある．開発者が実際に不具合修正に取り組んだ時間を正確に計測することは，OSS プロジェクト，および，開発者が活動時間をリポジトリに登録しない限り，現時点では難しい．

また，本論文では，開発状況メトリクスの計測期間を 10 日区切りで計測し，各不具合の開発状況メトリクスを近似的に不具合報告時期直前の期間から計測した．各期間の末に不具合が報告された場合，当該不具合の開発状況メトリクスは約 10 日前の期間から計測されたことになる．6.2 節でも検討したように，開発状況メトリクスの計測期間を変更したとしても予測精度に違いがないため，開発状況メトリクスの計測方法が本提案手法に与える影響は小さいと考える．

7. おわりに

本論文では，OSS プロジェクトの日々変化する開発状況下においても，開発者が不具合の修正時期を正確に予測することを目的として，不具合に関するメトリクスと不具合報告時期の開発状況を考慮したメトリクスを用いた不具合修正時間予測モデルを構築した．OpenStack プロジェクト，Qt プロジェクトに報告された不具合を対象に実験を行った結果，短中期間（2 週間）以内に修正される不具合について，開発状況メトリクスが修正時間を予測するために貢献することが分かった．今後，開発状況メトリクスを用いてプロジェクトや修正時期による修正時間に影響する原因の特定が可能になることを期待する．

謝辞 本論文の一部は，文部科学省科学研究補助費（若手 B：課題番号 16K16037，基盤 A：課題番号 17H00731）による助成を受けた．

参考文献

[1] Abreu, R. and Premraj, R.: How developer communication frequency relates to bug introducing changes, *Proc. Joint International and Annual ERCIM Workshops on Principles of Software Evolution and Software Evolution Workshops* (*IWPSE-Evol'09*), pp.153–158 (2009).

[2] Anbalagan, P. and Vouk, M.: On predicting the time taken to correct bug reports in open source projects, *Proc. International Conference on Software Maintenance* (*ICSM'09*), pp.20–26 (2009).

[3] Anvik, J., Hiew, L. and Murphy, G.C.: Who should fix this bug?, *Proc. 28th International Conference on Software Engineering* (*ICSE'06*), pp.361–370 (2006).

[4] Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R. and Zimmermann, T.: What makes a good bug report?, *Proc. 16th International Symposium on Foundations of Software Engineering* (*FSE'08*), pp.308–318 (2008).

[5] Bhattacharya, P. and Neamtiu, I.: Bug-fix time prediction models: Can we do better?, *Proc. 8th Working Conference on Mining Software Repositories* (*MSR'11*), pp.207–210 (2011).

[6] Bosu, A. and Carver, J.C.: Impact of developer reputation on code review outcomes in oss projects: An empirical investigation, *Proc. International Symposium on Empirical Software Engineering and Measurement* (*ESEM'14*), pp.1–10 (2014).

[7] Bougie, G., Treude, C., German, D.M. and Storey, M.-A.: A comparative exploration of freebsd bug lifetimes, *Proc. 4th International Workshop on Mining Software Repositories* (*MSR'10*), pp.106–109 (2010).

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

[9] Breu, S., Premraj, R., Sillito, J. and Zimmermann, T.: Information needs in bug reports: Improving cooperation between developers and users, *Proc. Conference on Computer Supported Cooperative Work* (*CSCW'10*), pp.301–310 (2010).

[10] Davies, S. and Roper, M.: What's in a bug report?, *Proc. 8th International Symposium on Empirical Software Engineering and Measurement* (*ESEM'14*), pp.1–10 (2014).

[11] Giger, E., Pinzger, M. and Gall, H.: Predicting the fix time of bugs, *Proc. 2nd International Workshop on Recommendation Systems for Software Engineering* (*RSSE'10*), pp.52–56 (2010).

[12] Herraiz, I., German, D.M., Gonzalez-Barahona, J.M. and Robles, G.: Towards a simplification of the bug report form in eclipse, *Proc. 2008 International Working Conference on Mining Software Repositories* (*MSR'08*), pp.145–148 (2008).

[13] Hewett, R. and Kijsanayothin, P.: On modeling software defect repair time, *Empirical Software Engineering*, Vol.14, No.22, pp.165–186 (2009).

[14] Hirao, T., Ihara, A. and Matsumoto, K.: The impact of a low level of agreement among reviewers in a code review process, *Proc. International Conference on Open Source Systems* (*OSS'16*), pp.97–110 (2016).

[15] Hooimeijer, P. and Weimer, W.: Modeling bug report quality, *Proc. 22nd International Conference on Automated Software Engineering* (*ASE'07*), pp.34–43 (2007).

[16] Ihara, A., Yasutaka, K., Monden, A., Ohira, M., Keung, J.W., Ubayashi, N. and Matsumoto, K.-I.: An investigation on software bug fix prediction for open source software projects–a case study on the eclipse project, *Proc. International Workshop on Software Analysis, Testing and Applications* (*SATA'12*), pp.135–144 (2012).

[17] Jeong, G., Kim, S. and Zimmermann, T.: Improving bug triage with bug tossing graphs, *Proc. ESEC/FSE'09*, pp.111–120 (2009).

[18] Jongyindee, A., Ohira, M., Ihara, A. and Matsumoto, K.-I.: Good or bad committers? – A case study of committer's activities on the eclipse's bug fixing process, *IEICE Trans. Information and Systems*, Vol.E95-D, No.9, pp.2202–2210 (2012).

[19] Koch, S. and Schneider, G.: Effort, cooperation and coordination in an open source software project: Gnome, *Information Systems Journal*, Vol.12, No.1, pp.27–42 (2002).

[20] Mockus, A. and Weiss, D.M.W.: Predicting risk of software changes, *Bell Labs Technical Journal*, Vol.5, No.2, pp.169–180 (2000).

[21] Moran, K.: Enhancing android application bug reporting, *Proc. Joint Meeting on Foundations of Software Engineering* (*ESEC/FSE'15*), pp.1045–1047 (2015).

[22] Nguyen, A., Cruzes, D., Conradi, R. and Ayala, C.P.: Empirical validation of human factors in oredicting issue lead time in open source projects, *Proc. 7th International Conference on Predictive Models in Software Engineering* (*PROMISE'11*) (2010).

[23] Ohira, M., Hassan, A.E., Naoya, O. and Matsumoto, K.: The impact of bug management patterns on bug fixing: A case study of eclipse projects, *Proc. 28th IEEE International Conference on Software Maintenance* (*ICSM'12*), pp.264–273 (2012).

[24] Ortu, M., Adams, B., Destefanis, G., Tourani, P., Marchesi, M. and Tonelli, R.: Are bullies more productive?: Empirical study of affectiveness vs. issue fixing time, *Proc. 12th Working Conference on Mining Software Repositories, MSR'15*, pp.303–313, IEEE Press Piscataway, NJ, USA (2015).

[25] Panjer, L.D.: Predicting eclipse bug lifetimes, *Proc. 4th International Workshop on Mining Software Repositories* (*MSR'07*), pp.29–33 (2007).

[26] Phannachitta, P., Ihara, A., Jirapaiwong, P., Ohira, M. and Matsumoto, K.: An algorithm for gradual patch acceptance detection in open source software repository mining, *IEICE Trans. Information and Systems*, Vol.E95-A, No.9, pp.1478–1489 (2012).

[27] Rastkar, S., Murphy, G.C. and Murray, G.: Summarizing software artifacts: A case study of bug reports, *Proc. 32nd International Conference on Software Engineering* (*ICSE'10*), pp.505–514 (2010).

[28] Rigby, P.C. and Bird, C.: Convergent contemporary software peer review practices, *Proc. 9th Joint Meeting on Foundations of Software Engineering* (*ESEC/FSE'13*), pp.202–212 (2013).

[29] Shihab, E., Hassan, A.E., Adams, B. and Jiang, Z.M.: An industrial study on the risk of software changes, *Proc. ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering* (*FSE'12*), pp.1–11 (2012).

[30] Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W.M., Ohira, O., Masao, Adams, B., Hassan, A.E. and Matsumoto, K.-I.: Studying re-opened bugs in open source software, *Empirical Software Engineering*, pp.1–38 (2013).

[31] Thomas, S.W., Nagappan, M., Blostein, D. and Hassan, A.E.: The impact of classifier configuration and classifier combination on bug localization, *IEEE Trans. Software*

Engineering, Vol.39, No.10, pp.1427–1443 (2013).
[32] Weiss, C., Premraj, R., Zimmermann, T. and Zeller, A.: How long will it take to fix this bug?, *Proc. 4th International Workshop on Mining Software Repositories* (*MSR'07*), pp.1–8 (2007).
[33] Zhang, F., Khomh, F., Zou, Y. and Hassan, A.E.: An empirical study on factors impacting bug fixing time, *Proc. 19th Working Conference on Reverse Engineering* (*WCRE'12*), pp.225–234 (2012).
[34] Zheng, Q., Mockus, A. and Zhou, M.: A method to identify and correct problematic software activity data: Exploiting capacity constraints and data redundancies, *Proc. 2015 10th Joint Meeting on Foundations of Software Engineering* (*ESEC/FSE'15*), pp.637–648 (2015).
[35] Zhou, J., Zhang, H. and Lo, D.: Where should the bugs be fixed? – More accurate information retrieval-based bug localization based on bug reports, *Proc. International Conference on Software Engineering* (*ICSE'12*), pp.14–24 (2012).
[36] 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.J95-D, No.2, pp.237–249 (2012).
[37] 正木 仁，大平雅雄，伊原彰紀，松本健一：OSS 開発における不具合割当パターンに着目した不具合修正時間の予測，情報処理学会論文誌，Vol.54, No.2, pp.933–944 (2013).

伊原 彰紀 （正会員）

2007 年龍谷大学理工学部卒業．2009 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．2012 年同大学博士課程修了．同年同大学情報科学研究科助教．博士（工学）．ソフトウェア工学，特にオープンソースソフトウェア開発・利用支援の研究に従事．電子情報通信学会，日本ソフトウェア科学会，IEEE 各会員．

若元 亮樹

2014 年岡山県立大学情報工学部スポーツシステム工学科卒業．2016 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．修士（工学）．ソフトウェア工学の研究に従事．

松本 健一 （正会員）

1985 年大阪大学基礎工学部情報工学科卒業．1989 年同大学大学院博士課程中退．同年同大学基礎工学部情報工学研究科助手．1993 年奈良先端科学技術大学院大学助教授．2001 年同大学院教授．工学博士．エンピリカルソフトウェア工学，特に，プロジェクトデータ収集/利用支援の研究に従事．日本ソフトウェア科学会，プロジェクトマネジメント学会．電子情報通信学会フェロー，IEEE Senior Member．本会フェロー．