情報処理学会論文誌 Vol.54 No.2 933–944 (Feb. 2013)

# OSS開発における不具合割当てパターンに着目した不具合修正時間の予測

正木 仁[1,a)] 大平 雅雄[2] 伊原 彰紀[1] 松本 健一[1]

受付日 2012年5月14日, 採録日 2012年11月2日

**概要**：大規模 OSS プロジェクトには日々大量の不具合が報告されており，不具合修正時間（OSS プロジェクトに報告された不具合が解決するまでに要する時間）をより正確に見積もることが重要になってきている．OSS プロジェクト管理者は，次期バージョンのリリースまでにどの不具合を修正すべきかを判断しなければならないためである．しかしながら，不具合の修正範囲の大きさや問題の複雑さの違い，ボランティアを主体とする開発者のスキルセットの違いなどの要因によって，個々の不具合の修正時間を見積もることは容易ではない．そのため近年，OSS 開発における不具合修正時間の予測に関する研究がさかんに行われている．本論文では，不具合割当てパターンを用いて不具合修正時間の予測モデルを構築する．不具合割当てパターンとは，不具合修正タスク割当て時の不具合報告者・管理者・修正担当者の 3 者の社会的関係を分類したものである．不具合割当てパターンの違いにより，修正作業にとりかかるまでの時間および修正作業自体に要する時間はそれぞれ大きく異なることが知られている．従来研究の多くは不具合情報（重要度や優先度など）に基づいて予測モデルを構築しているが，不具合管理パターンを考慮することでさらなる予測精度の向上を期待できる．本論文では，Eclipse Platform および JDT を対象として構築した予測モデルの評価を行った．実験の結果，不具合割当てパターンが不具合修正時間の予測精度向上に寄与するとともに，指定期間内（1 週間以内など）に不具合修正が完了するか否かの判断を支援できることが分かった．さらに，本予測モデルを利用することで，Platform で約 16%，JDT で約 10%多くの不具合を修正可能と判断できることが分かった．

**キーワード**：OSS 開発，不具合割当てパターン，不具合修正時間

# Predicting the Bug Fixing Time Based on the Bug Assignment Patterns in OSS Development

Hitoshi Masaki[1,a)] Masao Ohira[2] Akinori Ihara[1] Kenichi Matsumoto[1]

Received: May 14, 2012, Accepted: November 2, 2012

***Abstract:*** The number of reported bugs has been increasing especially in large-scale open source software (OSS) projects. Project managers in the projects have to decide which bugs should be resolved until they release the next version of their products. However it is not easy to estimate the time to resolve each bug due to the differences of the size of required modifications, the difficulty of each modification, skill set of each developer, and so forth. To address this issue, many studies have tried to predict the bug fixing time in OSS development. In this paper we constructs a prediction model for the bug fixing time, using the bug assignment patterns which have an impact on developer's performance of fixing bugs. The bug assignment patterns categorize the social relationships among bug reporters, managers, and developers in assigning bug fixing tasks. While most studies in the past only used the information extracted from bug reports itself, taking the bug assignment patterns into account would lead to prediction results with higher accuracy. Using data from the Eclipse Platform and JDT projects, we evaluates the prediction model. As a result, we found that the bug assignment patterns improved the prediction accuracy and help OSS managers make sure if a target bug will be resolved in a specified period (e.g., one week). We also found that our prediction model can contribute to resolve more bugs until the specified release (about 16% in Platform and about 10% in JDT).

***Keywords:*** OSS development, bug assignment patterns, bug fixing time

[1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology, Ikoma, Nara 630–0101, Japan
[2] 和歌山大学
Wakayama University, Wakayama 640–8510, Japan
[a)] hitoshi421@gmail.com

## 1. はじめに

Android OS を搭載したスマートフォンが広く普及するなど，我々の社会生活の様々な場面においてオープンソースソフトウェアが幅広く利用されるようになった結果，特に大規模 OSS プロジェクトでは，大量の不具合が日常的に報告されるようになっている．たとえば，Eclipse や Mozilla プロジェクトには 1 日に数百件の不具合が報告されることが知られている [1]．したがって，報告された個々の不具合がいつまでに解決されるかを見極めることは OSS プロジェクト管理者にとってきわめて重要な課題となっている．日々大量に不具合が報告される現状では，定期的に行うバージョンアップのリリース前にすべての不具合を修正することはできないため，修正を完了して次期リリースに含めることができるものとそうでないものの取捨選択を行わなければならない．また，次期リリースに含める予定の機能などに重大な不具合が存在する場合にも，リリースに間に合うよう優先的にコア開発者を割り当てるかどうか，あるいは，次期バージョンのリリースが遅延することをアナウンスするかどうかを判断するために，不具合の修正が完了する時期の検討が必要となる．

このような背景から，近年，OSS プロジェクトを対象とした不具合修正時間を予測するための研究がさかんに行われている．従来研究の多くは，不具合報告時に利用される不具合報告フォーマットそのものに含まれる基本的な情報（優先度や重要度など）に基づいて予測モデルを構築している [2], [3], [4], [5], [6]．しかし，比較的粒度の大きな予測期間（たとえば，6 カ月以内に修正されるかどうか，など）を用いて予測が行われており，OSS プロジェクト管理者のニーズを十分には満たせているとはいえない．

そこで本研究は，不具合修正プロセスにおける社会的関係，特に不具合修正作業の割当てパターン（不具合割当てパターン）に着目して，不具合修正時間の予測モデルの構築を試みる．不具合の再割当てが不具合修正時間の遅延の要因の 1 つであること [7]，不具合割当てには 4 つのパターンがありパターンごとに不具合修正時間が異なること [8], [9] が分かっており，予測モデルに不具合割当ての情報を加えることで粒度の小さな期間における予測精度の向上が期待できるためである．本研究の主な貢献は，予測精度の向上により，**OSS プロジェクト管理者が，指定期間（次期リリースなど）までに解決可能な不具合をより多く判断できるようにすること**，である．これにより，不具合修正に係る作業効率が向上すると期待できるため，より高品質な OSS の流通につながるものと考えられる．

本論文では，**不具合修正が完了するまでの日数と指定期間内に不具合修正が完了するか否か**をそれぞれ予測する不具合修正時間予測モデル構築し，Eclipse Platform および JDT プロジェクトを対象とした実験を行い予測モデルの精度評価を行う．本実験から，(1) 不具合修正時間を予測するのに最も適した数理モデルと (2) 予測精度の向上率を明らかにする．また，(3) 予測精度の向上により不具合がどの程度多く解決可能か判断できるようになるかを明らかにする．

本論文の構成は以下のとおりである．続く 2 章では不具合管理システムを利用した不具合修正プロセスの概要を説明し，本研究が着目する不具合割当てパターンについて説明する．また，関連研究を紹介し本研究との違いについて述べる．3 章では不具合修正時間予測モデルの構築に用いたメトリクスと予測モデルについて説明する．4 章では Eclipse Platform および JDT プロジェクトを対象に行った実験について述べ，5 章で実験結果について述べる．6 章では実験結果に対する考察を行い，最後に 7 章で本論文の結論を述べる．

## 2. 不具合修正プロセスにおける不具合割当てパターン

### 2.1 不具合情報の管理と不具合修正プロセス

現在，多くの OSS 開発では大量の不具合を効率的に管理するために不具合管理システム（BTS: Bug Tracking System）が利用されている．BTS には，Bugzilla [*1]，Trac [*2]，Redmine [*3] などがある．

BTS に不具合に関する情報が登録されると，1 つの不具合に対して**基本情報ページ**（**図 1**）と**変更履歴ページ**（**図 2**）が作成される．基本情報ページは，基本情報記録部と議論情報記録部からなり，優先度，重要度，不具合の再現方法，不具合に対しての意見や質疑など，不具合に関する詳細な情報を記録・管理する．変更履歴ページは，変更情報記録部からなり，個々の不具合の修正状況（変更者，変更日時，変更箇所，変更内容）を記録・管理する．本論文では，基本情報ページおよび変更履歴ページをあわせて不具合票と定義する．

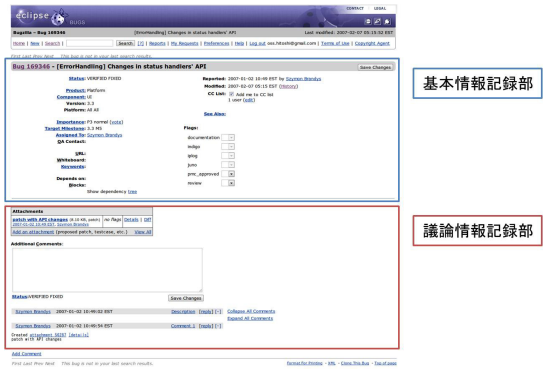


図 1 基本情報ページ
**Fig. 1** A page for basic information.

*1 http://www.bugzilla.org/
*2 http://trac.edgewall.org/
*3 http://www.redmine.org/

図 3 は，Bugzilla を用いた不具合修正プロセスの一例である．図 3 の各状態は，変更履歴ページで **Status** として記録されているものであり，不具合が報告されてから解決されるまでの修正状況の把握・管理に利用される．**表 1** は，各 Status の詳細である．

### 2.2 不具合割当てパターン

OSS 開発における不具合修正プロセスには，開発者だけではなくエンドユーザも多数参加しており重要な役割を担っている．**図 4** は，不具合修正プロセスにおける参加者の役割を示したものである．**表 2** は，役割の詳細である．

これらの役割は，すべて同一の人物が担うこともあれば，それぞれ別の人物が担うこともある．大澤 [8] は，大規模 OSS プロジェクト（Eclipse Platform および JDT，Mozilla Firefox および Thunderbird，Linux Kernel）を対象に分析を行い，不具合修正プロセスにおける参加者の関係が不具合修正時間に影響を与えることを明らかにした．また，報告者・管理者・修正者の関係を 4 種類の**不具合割当てパターン**（**図 5** および**表 3**）として定義し，自己解決型と修正受託型の場合，不具合修正時間は短く，修正委託型と分担解決型の場合，不具合修正時間が長くなることを示した．


図 2 変更履歴ページ
**Fig. 2** A page for status change history.

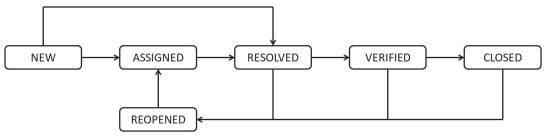


図 3 Bugzilla 利用時の不具合修正プロセスの一例
**Fig. 3** A bug fixing process in Bugzilla.

表 1 Bugzilla における Status の種類
**Table 1** Status in Bugzilla.

| Status 名 | 詳細 |
|---|---|
| NEW | 不具合の報告が行われた状態 |
| ASSIGNED | 不具合の修正を行う人が決定した状態 |
| RESOLVED | 不具合の修正が行われた状態 |
| REOPENED | 不具合の再修正が必要になった状態 |
| VERIFIED | 修正された不具合の検証が行われた状態 |
| CLOSED | 不具合が解決された状態 |

### 2.3 関連研究

#### 2.3.1 不具合修正プロセスにおける参加者の関係

本研究が不具合割当てパターンに着目する理由は，OSS 開発における不具合修正時間が，プロジェクト参加者の関係によって大きく影響を受けるためである．

- 報告者と管理者の関係：**管理者**は BTS の不具合票を基に不具合の場所と原因を特定し修正者に割り当てる．ただし，エンドユーザが**報告者**として作成した不具合票は，不具合の再現方法の記述がないなど管理者が不具合を特定できない場合も多い．結果的に不具合修正時間の長期化を招く [10] ため，不具合報告の品質を改善するための研究が行われている [2], [11], [12].
- 管理者と修正者の関係：**管理者**の主な役割は，報告された不具合の内容を理解し適任の**修正者**（**OSS 開発者**）に不具合修正タスクを割り当てることである．しかし，大量の不具合が日々報告されている現状では，すべての不具合に対して適切な修正者を割り当てることは困難である [13]．そこで，開発者の不具合割当て作業の負担を減らすために，不具合の修正に適任な開発者を推薦する研究が行われている [7], [14], [15].

上記既存研究は，不具合修正プロセスにおける 2 者間（報告者–管理者，または，管理者–修正者）の関係を考慮したものである．一方，本研究で用いる不具合割当てパターン [8] は，不具合修正時間への影響をより正確に取り扱うために，3 者間（報告者–管理者–修正者）の関係を考慮したものである．

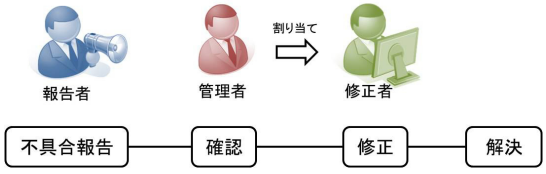


図 4 不具合修正プロセスにおける参加者の役割
**Fig. 4** Roles of participants in the bug fixing process.

表 2 参加者の役割
**Table 2** Roles of participants.

| 役割 | 詳細 |
|---|---|
| 報告者 | 発見した不具合に関する情報を BTS に登録する．OSS 利用者や開発者が担う役割であり，最も多くの参加者が関わっている． |
| 管理者 | 報告された不具合に対する修正が必要か否かを判断する．また，修正作業を開発者に割り当てる．OSS 開発者の中でも中心的な開発者が担う． |
| 修正者 | 依頼された不具合の修正を行う．OSS 開発者が担う． |

表 3 不具合割当てパターンと参加者の関係
Table 3 Bug assignment patterns and relationships among participants.

| パターン | 開発者の社会的関係 |
|---|---|
| 自己解決型 | すべての役割をすべて同じ人物が担当する．|
| 修正委託型 | 報告者と管理者は同一人物．修正者のみ異なる人物が担当する．|
| 修正受託型 | 管理者と修正者は同一人物．報告者のみ異なる人物が担当する．|
| 分担解決型 | 各役割をすべて異なる人物が担当する．|

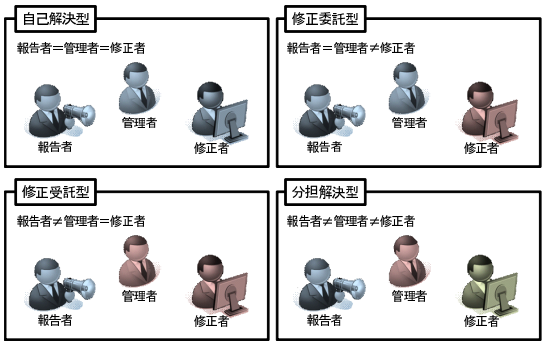


図 5 不具合割当てパターン
Fig. 5 Bug assignment patterns.

#### 2.3.2 不具合修正時間の予測

これまで，様々な予測モデルと不具合票に関連するメトリクスを用いて，不具合修正時間を予測する研究が行われてきた [2], [3], [4], [5], [6]．たとえば，Weiss ら [3] は，不具合票に含まれる類似するテキスト情報を用いて，JBoss プロジェクトを対象に不具合修正時間を予測している．また，Hewett ら [5] は，不具合報告時に記録されるコンポーネントや優先度などの情報を用いた予測モデルを構築している．

近年では，新たに人（開発者）に関連するメトリクスを用いることで，予測精度の向上が試みられている [16], [17], [18], [19], [20]．たとえば，Bhattacharya ら [20] は，修正者の評判に基づく不具合修正時間の予測モデルを構築している．また，Marks ら [16] は，不具合が一定期間内（3 カ月，1 年，3 年）に修正されるか否かの予測を開発者の属性情報を用いて構築している．

従来研究の多くは，**不具合報告時から不具合修正完了時までの**不具合修正時間の予測，または，**予測実施日から不具合修正完了時までの**不具合修正時間の予測を行っている．一方，本研究では**修正者決定時から不具合修正完了時までの**不具合修正時間を予測しており，予測モデルを構築するために不具合修正時間へ影響が確認されている不具合割当てパターンを加えているため，予測精度の向上が期待できる．さらに本研究は，修正が行われた不具合だけでなく，多くの従来研究では対象としていなかった**修正が行われなかった不具合**も対象として予測モデルを構築している．従来の予測モデルは，現実には数多く存在する DUPLICATE（重複して報告された不具合）や WONTFIX（修正予定のない不具合）の予測には対応できないため，実用上大きな問題があったが，本研究により解決できる可能性がある．

## 3. 不具合修正時間予測モデルの構築

本章では，不具合修正時間予測モデルの構築方法について説明する．

### 3.1 予測期間

本研究では，不具合修正時間の予測に有用な手法を明らかにするために，不具合修正時間を連続値で予測する手法と不具合修正時間を離散値で予測する手法を用いる．目的変数を連続値とする手法では，不具合修正が完了するまでの日数の予測（以下，**日数予測**）を行う．目的変数を離散値とする手法では，指定期間内に不具合修正が完了するか否かの予測（以下，**期間指定予測**）を行う．特に，期間指定予測では，1 日以内，1 週間以内，1 カ月以内に不具合修正が完了するか否かの 3 種類の予測を行う．3 つの単位を用いる理由は，数時間以内に修正が可能か否かよりも，不具合修正に中長期的な時間が必要かどうかを知ることが，実際の不具合修正プロセスで管理者がタスク管理を行うために重要であると考えられるためである．

### 3.2 予測モデル構築のためのメトリクス

本実験では，不具合修正時間の予測モデルを構築するために，不具合票から抽出した 3 種類のメトリクスを用いる．

#### 3.2.1 不具合票に関連するメトリクス

不具合票に関連するメトリクスの概要を**表 4** に示す．不具合票に関連するメトリクスとは，不具合の報告者によって入力された不具合に関する情報や，開発者らの議論に関する情報などである．開発者が不具合を迅速に修正するために，不具合票に記録されている不具合情報は重要な変数である [21], [22]．

#### 3.2.2 時間に関連するメトリクス

時間に関連するメトリクスの概要を**表 5** に示す．時間に関連するメトリクスとは，不具合の報告から割当てまでの時間や不具合の修正者が決定した日時に関する情報である．管理者から不具合の修正を割り当てられた開発者は，限られた時間の中で不具合の修正を行うため時間は重要な変数である [23]．

表 4 不具合票に関連するメトリクス
Table 4 Metrics associated with a bug report.

| 変数 | 尺度 | 詳細 |
|---|---|---|
| Components | 名義 | 不具合が発生したコンポーネント名 |
| Milestone | 名義 | マイルストーン記載の有無 |
| Priority | 名義 | 優先度 |
| Serverity | 名義 | 重要度 |
| DependsOnCount | 比例 | 依存関係にある不具合の数 |
| BlocksCount | 比例 | 修正を妨げている不具合の数 |
| CCCount | 比例 | メーリングリストの登録者数 |
| DescriptionWords | 比例 | 不具合に関する概要の文字数 |
| CommentsCount | 比例 | コメントの数 |
| CommentsWords | 比例 | コメントの総文字数 |
| AttachmentsCount | 比例 | 添付ファイルの数 |

表 5 時間に関連するメトリクス
Table 5 Metrics associated with time.

| 変数 | 尺度 | 詳細 |
|---|---|---|
| AssignTime | 比例 | 修正者の決定までに要した時間 |
| AssignedMonth | 比例 | 修正者が決定した月 |
| AssignedDay | 比例 | 修正者が決定した日 |
| AssignedWeekEnd | 名義 | 修正者が決定した日が週末か否か |

表 6 参加者に関連するメトリクス
Table 6 Metrics associated with participants.

| 変数 | 尺度 | 詳細 |
|---|---|---|
| Reporter | 名義 | 報告者のメールアドレス |
| Assignor | 名義 | 管理者のメールアドレス |
| Fixer | 名義 | 修正者のメールアドレス |
| Patterns | 名義 | 不具合割当てパターン |

#### 3.2.3 参加者に関係するメトリクス

人に関連するメトリクスの概要を**表 6** に示す．人に関連するメトリクスとは，修正者決定プロセスの参加者に関する情報や不具合割当てパターンである．不具合修正プロセスにおいて，報告者，管理者，修正者は主要な役割を担っており，不具合修正時間の予測においても重要な変数となりうる [8]．

### 3.3 不具合修正時間予測のためのモデル

本研究で，日数予測は目的変数を連続値として予測可能な重回帰分析およびランダムフォレスト法を用い，期間指定予測は目的変数を離散値として予測可能な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．これらのモデルは，不具合修正時間予測に関する既存研究 [2], [5], [16], [17] においても利用されている．ただし，本論文では紙面の都合上，最も精度の高かったランダムフォレスト法による結果のみを示すため，本節ではランダムフォレスト法に関してのみ説明する．

#### 3.3.1 ランダムフォレスト法

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る [24]．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（復元抽出）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得る．従来の集団学習ではモデル構築時にすべての説明変数を用いるのに対して，ランダムフォレスト法では無作為に選択された説明変数を用いる．

表 7 Eclipse プロジェクトにおける不具合件数
Table 7 Number of bugs in Eclipse Plarform and JDT projects.

| | Platform | JDT |
|---|---|---|
| 全不具合票（件） | 89,087 | 45,015 |
| 対象不具合票（件） | 21,835 | 11,088 |

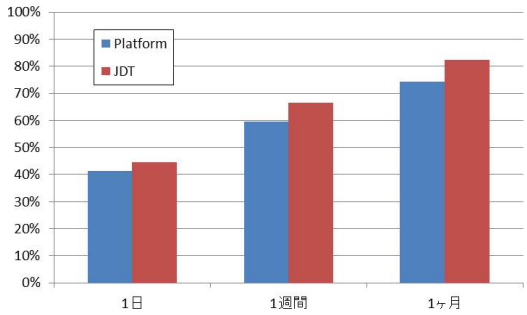


図 6 一定期間内に修正された不具合票の割合
Fig. 6 Ratio of bug reports fixed within a certain period of time.

## 4. 実験方法

本章では，前章で述べた不具合修正時間の予測モデルを評価するために行う実験方法について述べる．

### 4.1 目的

本実験の目的は，(1) 不具合修正時間を予測するのに最も適した数理モデルと (2) 予測精度の向上率を明らかにすることである．

### 4.2 データセット

本実験では，2001 年 10 月から 2012 年 4 月の間に Eclipse Platform プロジェクト[*4]および JDT プロジェクト[*5]の BTS に報告された不具合票をデータセットとして用いる．両プロジェクトは，長期間の開発が続けられており，実験を行うための十分なデータを取得可能なため本実験の対象とした．**表 7** に，データセットの内訳を示す．

本論文で用いた Platform，および JDT において，一定期間内（1 日，1 週間，1 カ月）に修正された不具合票の割合を**図 6** に示す．不具合の約 4 割は 1 日以内に修正され，1 週間以内に約 6 割，1 カ月以内に約 8 割の不具合が修正されていることが分かる．

[*4] http://www.eclipse.org/platform/
[*5] http://www.eclipse.org/jdt/

本実験では，以下の不具合票を除外したもの（対象不具合票）を用いる．

- 他の BTS から移行された不具合票：正確な報告日時を特定できないため，他の BTS からインポートされた不具合票は除外する．
- 機能拡張に関する不具合票：機能拡張に掛かる時間は本論文で定義する不具合修正時間と本質的に異なるため，機能拡張に関する不具合票は除外する．
- 記録情報が間違っている不具合票：実際の不具合修正時間を求めることは困難なため，不具合修正後に BTS に登録された不具合票は除外する．
- 修正者が変更されている不具合票：本実験では新規の不具合に対して修正者が初めて割り当てられた段階で管理者が不具合修正時間を予測することを想定しているため，修正者が複数回変更されている不具合票は除外する．
- Reopen された不具合票：不具合の再修正（REOPEN）は様々な理由で発生する [25] ため，本実験では将来再修正される可能性があるかどうかは考慮しない．そのため，Resolution が複数回変更された不具合票は除外する．
- 修正者の特定が不可能な不具合票：修正者を一意に特定する必要があるため，修正者としてメーリングリストが登録されている不具合票を除外する．

### 4.3 評価基準

日数予測における評価基準として，決定係数および自由度調整済み決定係数を用いる．決定係数 $R^2$ は，回帰方程式のあてはまりの良さ（予測精度）を評価するための指標であり，式 (1) のように定義される．

$$R^2 = 1 - \frac{\sum_{i=1}(y_i - \hat{y_i})^2}{\sum_{i=1}(y_i - \bar{y_i})^2} \tag{1}$$

決定係数は値域 $[0,1]$ をとり，0.5 を超えていることが，有用なモデルであることの目安とされる [26]．ただし，回帰方程式の説明変数が増えるに従い，決定係数は高くなる傾向にある．そこで，本論文では標本数と説明変数の数により補正された，自由度調整済み決定係数 $R^{*2}$ を用いる．標本数を $N$，説明変数の数を $k$ としたとき，自由度調整済み決定係数は式 (2) のように定義される．

$$R^{*2} = 1 - \frac{\sum_{i=1}(y_i - \hat{y_i})^2/(N-k-1)}{\sum_{i=1}(y_i - \bar{y_i})^2/(N-1)} \tag{2}$$

期間指定予測における評価基準として，適合率，再現率，F1 値 [27] を用いる．本論文における適合率（Precision）とは，一定期間よりも早く修正すると予測した不具合のうち，正しく予測できた不具合の割合を指す．また，再現率（Recall）とは，一定期間内に修正された不具合のうち，正しく予測できた不具合の割合である．本論文では適合率と再現率の調和平均である F1 値を評価基準として用いる．F1 値は式 (3) のように定義される．

$$F1\text{-}measure = \frac{2 \times Precition \times Recall}{Precition + Recall} \tag{3}$$

3 つの評価基準はいずれも値域 $[0,1]$ 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表す．

### 4.4 実験手順

3 章で説明したメトリクスとモデルを用いて，日数予測，および期間指定予測を交差検証により，それぞれ下記の手順で実施した．また，期間指定予測と比較を行うために，機械学習を用いない予測を行っているが，この予測方法については 5.2.1 項において詳述する．

#### 4.4.1 日数予測の実験手順

**Step1. 説明変数の準備** モデル構築に用いるメトリクスの中で名義尺度の変数をダミー変数に変換する．ただし，ダミー変数を増やしすぎると，多重共線性が生じやすいなどの問題があるため，本実験では各変数の中で出現頻度の高い上位 5 つをダミー変数に変換する．また，説明変数間で多重共線性が生じているかを確認するために VIF（Variance Inflation Factor）を求め，10 を超える場合は片方の説明変数を削除する．

**Step2. 目的変数の準備** 目的変数である不具合修正時間を対数変換する．

**Step3. データセットの分割** データセットを，10 個（$data_1, \cdots, data_{10}$）に分割する．そのうち，$data_1$ を 1 個のテストデータ $test$ とし，残りの $data_2, \cdots, data_{10}$ を 9 個のフィットデータ $fit_1, \cdots, fit_9$ とする．

**Step4. 不具合修正時間予測モデルの構築** フィットデータ $fit_1, \cdots, fit_9$ および 3.3 節で説明したモデルを用いて，不具合修正時間予測モデルを構築する．

**Step5. 不具合修正時間予測モデルの精度評価** テストデータ $test$ を用いて予測を行い，4.3 節で説明した方法を用いて予測精度を求める．

**Step6. 実験の繰返し** Step3 において，テストデータになっていないデータセットの中から，たとえば $data_2$ を 1 個のテストデータ $test$ とし，残りの $data_1, data_3, \cdots, data_{10}$ を 9 個のフィットデータ $fit_1, \cdots, fit_9$ とする．これを，テストデータに選択されていないデータセットがなくなるまで，Step4 から Step5 を繰り返し行い，予測精度の平均値を求める．

#### 4.4.2 期間指定予測の実験手順

**Step1. 説明変数の準備** 日数予測の場合と同様である．

**Step2. 目的変数の準備** 目的変数は，不具合修正が一定期間内（1 日，1 週間，1 カ月）に完了していれば 1 とし，完了していなければ 0 とした．

**Step3〜Step6** 日数予測の場合と同様である．

**Step7. 目的変数の変更** 期間指定予測では，すべての目的変数に対して Step2 から Step6 を繰り返し行う．

## 5. 実験結果

本章では，構築した 2 種類の予測モデルを評価するために行った実験の結果について述べる．ただし，本論文では紙面の都合上，最も精度の高かったランダムフォレスト法による結果のみを示す．

### 5.1 日数予測の実験結果

#### 5.1.1 不具合修正時間の予測精度

重回帰分析法よりも精度の高かったランダムフォレスト法による予測モデルの決定係数および自由度調節済み決定係数を求めた結果を**表 8** に示す．Platform プロジェクトよりも，JDT プロジェクトの方が高い精度で不具合修正時間を予測できていることが分かる．ただし，どちらも決定係数は 0.5 を超えておらず，不具合修正時間の予測に有効な予測モデルを構築できたとはいえない．

#### 5.1.2 予測モデル構築における変数の重要度

3 種類のメトリクスの各変数がモデル構築にどの程度寄与しているかを分析するために，各変数の精度減少値を求めた．ランダムフォレスト法を用いた予測モデルにおける各変数の精度減少値を**表 9** に示す．精度減少値とは，説明変数をモデルから取り除くことにより，どの程度予測精度が低下するかを示す．つまり，精度減少値が大きいほど，モデル構築に寄与していることを表す．表 9 より，両プロジェクトにおいて，コンポーネント（Components）が最もモデル構築に寄与しており，次に不具合割当てパターン（Patterns）がモデル構築に寄与していることが分かった．

**表 8** 予測モデルの精度
**Table 8** Accuracy of the prediction model using Random Forest.

| プロジェクト | 決定係数 | 自由度調整済決定係数 |
|---|---|---|
| Platform | 0.309 | 0.296 |
| JDT | 0.402 | 0.379 |

### 5.2 期間指定予測の実験結果

#### 5.2.1 予測精度の比較

Platform，および JDT プロジェクトにおいて，予測モデルの精度（適合率，再現率，F1 値），機械学習を用いない予測の精度（適合率，再現率，F1 値），機械学習を用いない予測に対する予測モデルの向上率を求めた結果を**表 10** に示す．

機械学習を用いない予測とは，過去に修正された不具合票の割合を基にして，一定期間内に不具合修正が完了するか否かを予測する方法である．たとえば，Platform プロジェクトでは不具合の約 41%が 1 日以内に修正されてい

**表 9** 各変数の重要度
**Table 9** Importance of each independent value.

| メトリクス | 変数名 | 精度減少値 | |
|---|---|---|---|
| | | Platform | JDT |
| 不具合票 | Components | 90.34 | 51.45 |
| | Priority | 4.86 | 1.34 |
| | Severity | 4.84 | 1.06 |
| | Milestone | 16.14 | 7.97 |
| | DescriptionWords | 8.30 | 9.05 |
| | CommentsCount | 10.58 | 12.08 |
| | CommentsWords | 14.30 | 13.45 |
| | AttachmentsCount | 8.43 | 6.92 |
| | DependsOnCount | 2.93 | 1.54 |
| | BlocksCount | 1.60 | 0.42 |
| | CCCount | 11.06 | 8.58 |
| 時間 | AssignTime | 19.84 | 13.74 |
| | AssignedMonth | 13.72 | 12.68 |
| | AssignedDay | 10.15 | 7.13 |
| | AssignedWeekEnd | 2.17 | 4.60 |
| 人 | Reporter | 49.02 | 15.57 |
| | Assignor | 29.44 | 34.70 |
| | Fixer | 32.38 | 37.66 |
| | Patterns | 76.63 | 51.34 |

**表 10** 予測精度の比較
**Table 10** Comparison of prediction accuracy.

| | 予測期間 | Platform 適合率 | Platform 再現率 | Platform F1 値 | JDT 適合率 | JDT 再現率 | JDT F1 値 |
|---|---|---|---|---|---|---|---|
| ランダムフォレスト法 | 1 日 | 65.82 | 45.24 | 53.62 | 68.24 | 54.51 | 60.61 |
| | 1 週間 | 68.81 | 75.71 | 72.10 | 73.23 | 86.80 | 79.44 |
| | 1 カ月 | 77.37 | 93.61 | 84.72 | 83.64 | 98.19 | 90.33 |
| 機械学習を用いない予測 | 1 日 | 41.22 | 41.22 | 41.22 | 44.66 | 44.66 | 44.66 |
| | 1 週間 | 59.45 | 59.45 | 59.45 | 66.40 | 66.40 | 66.40 |
| | 1 カ月 | 74.25 | 74.25 | 74.25 | 82.47 | 82.47 | 82.47 |
| 向上率 | 1 日 | **59.68**% | 9.73% | **30.08**% | **52.81**% | 22.07% | **35.72**% |
| | 1 週間 | 15.75% | **27.36**% | 21.27% | 10.29% | **30.73**% | 19.64% |
| | 1 カ月 | 4.21% | 26.07% | 14.11% | 1.42% | 19.07 | 9.54% |

太字は予測精度の向上率が最も高い適合率，再現率，F1 値の値．

表 11 各変数の重要度
Table 11 Importance of each variable.

| メトリクス | 変数名 | Platform 精度減少値 | | | JDT 精度減少値 | | |
|---|---|---|---|---|---|---|---|
| | | 1 日 | 1 週間 | 1 カ月 | 1 日 | 1 週間 | 1 カ月 |
| 不具合票 | Components | **68.85** | **57.61** | **52.43** | **43.64** | **37.38** | **24.21** |
| | Priority | 2.14 | 2.31 | 2.79 | 2.38 | 0.47 | 2.24 |
| | Severity | 1.97 | 2.40 | 4.69 | 1.32 | 0.52 | 1.01 |
| | Milestone | 15.10 | 13.55 | 12.68 | 8.64 | 5.43 | 3.39 |
| | DescriptionWords | 6.17 | 6.34 | 8.27 | 6.64 | 5.10 | 4.24 |
| | CommentsCount | 12.75 | 8.72 | 6.92 | 10.49 | 9.12 | 6.02 |
| | CommentsWords | 13.37 | 9.31 | 8.37 | 9.39 | 7.76 | 6.21 |
| | AttachmentsCount | 6.37 | 5.93 | 7.72 | 6.07 | 3.71 | 3.69 |
| | DependsOnCount | 3.45 | 0.30 | 0.18 | 0.89 | 1.14 | 0.96 |
| | BlocksCount | 1.80 | 1.89 | 0.95 | 0.33 | 0.33 | 0.28 |
| | CCCount | 5.67 | 7.35 | 9.92 | 6.38 | 5.20 | 3.59 |
| 時間 | AssignTime | 15.85 | 17.00 | **21.17** | 9.69 | 12.07 | **12.82** |
| | AssignedMonth | 10.85 | 14.03 | 18.62 | 7.03 | 15.94 | 11.68 |
| | AssignedDay | 9.73 | 9.17 | 9.99 | 7.10 | 6.82 | 5.59 |
| | AssignedWeekEnd | 2.68 | 1.77 | 1.92 | 7.75 | 2.21 | 3.93 |
| 人 | Reporter | **39.16** | **29.03** | 14.18 | **13.39** | **13.78** | 11.33 |
| | Assignor | **23.94** | **30.23** | **32.01** | **30.40** | **31.11** | **26.48** |
| | Fixer | **31.62** | **31.75** | **29.54** | **31.18** | **36.94** | **34.77** |
| | Patterns | **56.68** | **44.62** | **30.02** | **34.27** | **24.09** | **15.72** |

太字は各指定期間予測モデルの中で最も重要度の高い上位 5 件の変数の値.

る．これより，機械学習を用いない予測では報告された不具合の約 41%が 1 日以内に修正されると予測するが，実際に 1 日以内に修正される不具合はその中の約 41%であるとした（具体的な例として 100 個の不具合がある場合，41 個が 1 日以内に修正されると予測するが，実際に 1 日以内に修正される不具合は 17 個（$0.41 * 0.41 = 16.81$）となる）．他の予測期間においても，同様に過去に修正された不具合票の割合を求め，機械学習を用いない予測結果を求めた．

期間指定予測と機械学習を用いない予測において，予測期間を 1 日とした場合，両プロジェクトともに適合率は大幅に向上しており，再現率も向上していた．予測期間を 1 週間および 1 カ月とした場合，両プロジェクトともに適合率に比べ，再現率が大きく向上していた．また，Platform プロジェクトにおいて，機械学習を用いない予測よりも F1 値は約 14%から 30%向上しており，JDT プロジェクトにおいても F1 値は約 10%から約 36%向上していた．実験の結果より，機械学習を用いない予測に比べ，有効な予測を行えていることが分かった．特に，予測期間が短い場合の F1 値の向上率が高く，期間指定予測モデルを用いることで機械学習を用いない予測よりも，正確に不具合修正時間を予測することが可能になるといえる．

#### 5.2.2 各予測モデルにおける変数の重要度

3 種類のメトリクスの各変数がモデル構築にどの程度寄与しているかを分析するために，それぞれの指定期間における各変数の精度減少値を求めた．

Platform，および JDT プロジェクトにおける各変数の精度減少値を**表 11** に示す．指定期間を 1 日とした予測では，両プロジェクトともにコンポーネント（Components）が最もモデル構築に寄与しており，次に不具合割当てパターン（Patterns）が寄与していた．指定期間を 1 週間および 1 カ月とした予測においても，Platform プロジェクトでは上位 3 件，JDT プロジェクトでは上位 4 件以内に不具合割当てパターンが入っていた．これより，不具合割当てパターンは期間指定予測モデルの構築に有用であることが分かった．

## 6. 考察

本章では，5 章で行った不具合修正時間の予測実験の結果を基に考察を行う．

### 6.1 予測モデル構築に寄与するメトリクス

5.2.2 項より，不具合票メトリクスの Components，人メトリクスの Patterns，Reporter，Assignor，Fixer がモデル構築に寄与することが分かった．以下に，それぞれのメトリクスについて，考察する．

Platform，JDT における Components の不具合数と割合を**表 12** に示す．Components の寄与度が最も高かった理由として，不具合修正の優先度，Components の複雑度がそれぞれ異なることから不具合修正に必要な時間が Components によって明確に異なることが原因である

表 12 Components の統計量（上：Platform，下：JDT）
Table 12 Statistic of Components.

| | 不具合数 | 割合 |
|---|---|---|
| UI | 7,105 | 32.54% |
| SWT | 6,135 | 28.10% |
| Team | 1,477 | 6.76% |
| Debug | 1,136 | 5.20% |
| Releng | 917 | 4.20% |
| その他 | 5,065 | 23.20% |
| 合計 | 21,835 | 100.00% |

| | 不具合数 | 割合 |
|---|---|---|
| UI | 4,555 | 41.08% |
| Core | 4,479 | 40.40% |
| Debug | 1,176 | 10.61% |
| Text | 698 | 6.30% |
| APT | 127 | 1.15% |
| Doc | 53 | 0.48% |
| 合計 | 11,088 | 100.00% |

表 13 Patterns の統計量
Table 13 Statistic of Patterns.

| | Platform | | JDT | |
|---|---|---|---|---|
| | 不具合数 | 割合 | 不具合数 | 割合 |
| 自己解決型 | 1,896 | 8.68% | 792 | 7.14% |
| 修正委託型 | 2,320 | 10.63% | 1,029 | 9.28% |
| 修正受託型 | 5,558 | 25.45% | 2,708 | 24.42% |
| 分担解決型 | 12,061 | 55.24% | 6,559 | 59.15% |
| 合計 | 21,835 | 100.00% | 11,088 | 100.00% |

と考えられる．たとえば，Platform の Debug は，JDT の Debug からなる Eclipse Debug プロジェクトとして修正が行われており，他のプロジェクトにも影響することから，多くが 1 時間以内に修正が行われていた．一方で，Platform の SWT プロジェクトはツールキットの一種であることから，他のプロジェクトに対する影響力も低いため，多くは 1 週間以上かけて修正が行われていた．

Platform，JDT における各 Patterns の不具合数と割合を**表 13** に示す．Platform，JDT において，参加者間のミスコミュニケーションやコミュニケーションオーバヘッドが原因となり，分担解決型はその他の不具合割当てパターンに比べ，不具合修正時間が長くなることが明らかにされている [9]．両プロジェクトともに，分担解決型は不具合割当てパターンの過半数を占めており，不具合割当てパターンが分担解決型か否かを見極めることで，不具合修正に時間がかかるか否かを判別できるため，予測モデルの構築に Patterns が寄与したと考えられる．

Platform，JDT における報告者（Reporter），管理者（Assignor），修正者（Fixer）の一意な人数を**表 14** に示す．両プロジェクトともに，報告者は多数の人が行っているが，管理者および修正者を担っているのは限られた一部の人である．人によって，修正者に適任な開発者を選択する経験や知識，不具合の修正に必要な技術や知識に差があるため，どの人物がそれぞれの役割を担ったかによって，不具合修正時間は大きく異なるため，予測モデルの構築に Reporter，Assignor，Fixer が寄与したと考えられる．

表 14 報告者，管理者，修正者の一意な人数
Table 14 Unique users of Reporter, Assignor and Fixer.

| | Platform | JDT |
|---|---|---|
| 報告者数 | 3,495 | 1,862 |
| 管理者数 | 132 | 59 |
| 修正者数 | 213 | 87 |

表 15 Priority の統計量
Table 15 Statistic of Priority.

| | Platform | | JDT | |
|---|---|---|---|---|
| | 不具合数 | 割合 | 不具合数 | 割合 |
| P1 | 172 | 0.79% | 100 | 0.90% |
| P2 | 371 | 1.70% | 209 | 1.88% |
| P3 | 21,225 | 97.21% | 10,748 | 96.93% |
| P4 | 20 | 0.09% | 26 | 0.23% |
| P5 | 47 | 0.22% | 5 | 0.05% |
| 合計 | 21,835 | 100.00% | 11,088 | 100.00% |

表 16 Severity の統計量
Table 16 Statistic of Severity.

| | Platform | | JDT | |
|---|---|---|---|---|
| | 不具合数 | 割合 | 不具合数 | 割合 |
| blocker | 80 | 0.37% | 16 | 0.14% |
| critical | 224 | 1.03% | 74 | 0.67% |
| major | 352 | 1.61% | 131 | 1.18% |
| normal | 21,043 | 96.37% | 10,776 | 97.19% |
| minor | 85 | 0.39% | 73 | 0.66% |
| trivial | 51 | 0.23% | 18 | 0.16% |
| 合計 | 21,835 | 100.00% | 11,088 | 100.00% |

一方で，不具合修正時間に大きく影響すると考えられた Priority と Severity の寄与度は低かった．これは，**表 15**，**表 16** から分かるように，不具合の大多数が初期値状態（P3，normal）であったことが原因であると考えられる．

### 6.2 予測モデル構築における不具合割当てパターンの効果

説明変数として不具合修正パターンを用いると（用いない場合に比べて），不具合修正時間予測モデルの精度がどの程度向上するかについて議論する．

日数予測において不具合割当てパターンを説明変数に用いたことによる予測モデルの精度向上率を**表 17** に示す．Platform プロジェクトで自由度調節済み決定係数は約 12%向上しており，JDT プロジェクトにおいても自由度調節済み決定係数は約 4%向上していた．これは，各不具合割当てパターンにおける不具合修正時間にそれぞれ有意な差があることから，不具合修正時間を予測するうえで不具

表 17 日数予測における予測モデルの精度向上率
**Table 17** Improvement rate of the model for predicting required days to resolve a bug.

| プロジェクト | 決定係数 | 自由度調節済み決定係数 |
|---|---|---|
| Platform | 11.40% | 12.24% |
| JDT | 4.01% | 3.81% |

表 18 期間指定予測における予測モデルの精度向上率
**Table 18** Improvement rate of the model for predicting whether a bug is resolved in a specified period.

| プロジェクト | 予測期間 | 適合率 | 再現率 | F1 値 |
|---|---|---|---|---|
| Platform | 1 日 | 1.79% | 4.83% | 3.59% |
| | 1 週間 | 0.87% | −1.45% | −0.24% |
| | 1 カ月 | −0.03% | −0.40% | −0.20% |
| JDT | 1 日 | 2.78% | 6.40% | 4.79% |
| | 1 週間 | 1.23% | −0.38% | 0.49% |
| | 1 カ月 | −0.27% | −0.10% | −0.19% |

合割当てパターンが有用な説明変数となっているためであると考えられる．

期間指定予測において不具合割当てパターンを説明変数に用いたことによる予測モデルの精度向上率を**表 18** に示す．予測期間を 1 日とした予測では，両プロジェクトともに不具合割当てパターンを説明変数に用いることで，予測モデルの F1 値は向上していた．これは，分担解決型を除く不具合割当てパターンにおいて，1 日以内に修正される不具合と修正されない不具合の比率がほぼ同一であったことから，1 日以内に修正されるか否かの判別基準として，不具合割当てパターンが有用であったためであると考えられる．一方，予測期間を 1 週間および 1 カ月とした予測では，両プロジェクトともに不具合割当てパターンを説明変数に用いても，予測モデルの F1 値向上につながらなかった．これは，不具合割当てパターンのうち，自己解決型，修正委託型，修正受託型では 1 週間を経過した段階で過半数の不具合が修正されており，指定期間内に不具合が修正されるか否かの判別基準として，不具合割当てパターンが有用でなかったためであると考えられる．これらの結果より，不具合割当てパターンは指定期間を短期間（1 週間未満）とした予測では有用であるが，指定期間を長期間（1 週間以上）とした予測では有用でないといえる．

### 6.3 不具合修正時間予測モデルの実用性

それぞれの予測期間の観点から期間指定予測モデルの実用性について議論する．予測期間を 1 日とした予測では，両プロジェクトともに機械学習を用いない予測に比べ，正しく予測した不具合の数は増加し，誤って予測した不具合の数は減少しており，予測精度が大幅に向上していた．しかし，1 日以内に修正されると予測した不具合の割合は，Platform プロジェクトで約 28%，JDT プロジェクトで約 36%と元々低い．修正に何日かかるか不明な不具合の数が多すぎるという理由から，予測期間を 1 日とした予測モデルだけで OSS プロジェクト管理者が不具合の修正計画を立てることは困難であると考える．

予測期間を 1 週間とした予測で，1 週間以内に修正されると予測した不具合の割合は，Platform プロジェクトが約 65%，JDT プロジェクトが約 79%であった．適合率も機械学習を用いない予測の約 59%（Platform）と約 66%（JDT）に比べ，本予測モデルでは約 69%（Platform）と約 73%（JDT）と高いため，不具合修正の作業工程を週単位で計画する際に，予測期間を 1 週間とした予測モデルは有効であると考えられる．

予測期間を 1 カ月とした予測モデルの精度（F1 値）は，3 つの予測期間の中で最も高かった．これは，実際に 1 カ月以内に修正される不具合が Platform プロジェクトで全体の約 74%，JDT プロジェクトで全体の約 82%を占めることから，一定の適合率を維持しながら高い再現率を実現することが可能なためである．しかし，1 カ月以内に修正されると予測した不具合の割合は，Platform プロジェクトで約 90%，JDT プロジェクトで約 97%ときわめて高く，大多数の不具合は 1 カ月以内に修正されると予測してしまうため，予測期間を 1 カ月とした予測モデルの実用性は低いと考えられる．

これらの結果より，予測期間を 1 週間とした予測モデルは実際の適用現場において最も実用性が高いと考えられる．具体的には，予測期間を 1 週間とした予測モデルを用いることで，同期間内に修正可能と見積もれる不具合の数は，Platform プロジェクトで約 16%（機械学習を用いないで予測する場合，1 週間以内に 8,491 件修正される．一方で，予測モデルを用いると 1 週間に 9,828 件修正される．以上より，$15.74 = \frac{9828}{8491} - 100$），JDT プロジェクトで約 10%（機械学習を用いないで予測する場合，1 週間以内に 5,794 件修正される．一方で，予測モデルを用いると 1 週間に 6390 件修正される．以上より，$10.28 = \frac{6390}{5794} - 100$）増加することになる．また，予測期間を 1 日とした予測モデルでは Platform プロジェクトで約 60%，JDT プロジェクトで約 53%多くの不具合を修正可能と判断可能になるため，1 日および 1 週間を予測期間とする予測モデルを併用することで，よりきめ細かな不具合修正計画の立案が可能になる．

### 6.4 制約

本論文では，大規模な OSS プロジェクトである Eclipse Platform および JDT プロジェクトについて，不具合修正パターンを考慮したランダムフォレスト法の利用による不具合修正時間予測モデルの有用性を示した．ただし，予測モデルの一般性については明らかになっておらず，今後 OSS プロジェクトの種類を増やすなどしてさらに検証する

必要がある．特に，OSS プロジェクトによっては，開発規模や開発形態が大きく異なるため，それぞれのプロジェクトに適した予測モデルを構築する必要があると考えられる．

なお，本論文で用いた名義尺度のメトリクスの中には，報告者のメールアドレスのように多数の項目を持つ変数が存在するため，出現頻度の高い上位 5 つの項目をダミー変数に変換している．コンポーネントのようにそれぞれのダミー変数がモデル構築に強く寄与している変数は，ダミー変数に変換する項目の数を増やすことで予測精度の向上につながる可能性があり，今後取り組む必要がある．

## 7. おわりに

本論文では，個々の不具合の修正完了期間を見積もる必要のある OSS プロジェクトの管理者を対象として，不具合割当てパターンを用いた不具合修正時間の予測モデルを構築した．2 つの Eclipse プロジェクト（Platform，JDT）を対象に実験を行った結果，以下に示す知見が得られた．

- 不具合修正時間の予測において，日数を予測する方法では有効な予測モデルを構築することはできなかったが，期間を指定する方法ではランダムフォレスト法を用いて有効な予測モデルを構築することができた．特に，予測期間を 1 日および 1 週間とした予測において，実用性の高い予測モデルを構築することが可能である．
- 不具合割当てパターンを用いることで，予測期間が短期間（1 週間未満）の場合，予測モデルの精度向上に寄与することが分かった．しかし，予測期間が長期間（1 週間以上）の場合，予測モデルの精度向上に寄与しないことが分かった．
- 予測期間を 1 週間とした予測モデルを用いることで，同期間内に修正可能と見積もれる不具合の数は，Platform プロジェクトで約 16%，JDT プロジェクトで約 10%増加することが分かった．また，予測期間 1 日の予測モデルと併用することで，よりきめ細かな不具合修正計画の立案が可能になると期待できる．

今後は，OSS プロジェクトの数を増やした実験を行い予測モデルのより高い一般性について検証するとともに，予測モデルのさらなる精度向上を目指して不具合票に含まれる議論情報の分析（どのような議論が素早い不具合修正につながるか，など）を行う予定である．

**謝辞** 本研究の一部は，文部科学省科学研究補助金（基盤（B）：23300009）および（基盤（C）：24500041）による助成を受けた．

### 参考文献

[1] Guo, P.J., Zimmermann, T., Nagappan, N. and Murphy, B.: Characterizing and predicting which bugs get fixed: An empirical study of microsoft windows, *Proc. 32nd International Conference on Software Engineering* (*ICSE'10*), pp.495–504 (2010).

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

[3] 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).

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

[5] Hewett, R. and Kijsanayothin, P.: On modeling software defect repair time, *Empirical Software Engineering* (*ESE'09*), Vol.14, No.2, pp.165–186 (2009).

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

[7] Jeong, G., Kin, S. and Zimmermann, T.: Improving bug triage with bug tossing graphs, *Proc. 7th Joint Meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering* (*ESEC/FSE'09*), pp.111–120 (2009).

[8] 大澤直哉：OSS 開発における不具合修正遅延と修正者決定プロセスとの関係分析，修士論文，奈良先端科学技術大学院大学，NAIST-IS-MT0951021 (2011).

[9] 大平雅雄，大澤直哉，アハマドハッサン，松本健一：不具合管理パターンが不具合修正に与える影響の分析，ソフトウェア工学の基礎 XVIII，日本ソフトウェア科学会 FOSE2011，pp.237–242 (2011).

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

[11] 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).

[12] Just, S., Premraj, R. and Zimmermann, T.: Towards the next generation of bug tracking systems, *Proc. 2008 Symposium on Visual Languages and Human-Centric Computing* (*VL/HCC'08*), pp.82–85 (2008).

[13] Chou, A., Yang, J., Chelf, B., Hallem, S. and Engler, D.: An empirical study of operating systems errors, *Proc. 18th Symposium on Operating Systems Principles* (*SOSP'09*), pp.73–88 (2001).

[14] 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).

[15] Cubranic, D. and Murphy, G.C.: Automatic bug triage using text categorization, *Proc. 16th International Conference on Software Engineering and Knowledge Engineering* (*SEKE'04*), pp.92–97 (2004).

[16] Marks, L., Zou, Y. and Hassan, A.E.: Studying the fix-time for bugs in large open source projects, *Proc. 7th International Conference on Predictive Models in Software Engineering* (*PROMISE'11*), pp.11:1–11:8 (2011).

[17] Duc Anh, N., Cruzes, D.S., Conradi, R. and Ayal, C.: Empirical validation of human factors in predicting issue lead time in open source projects, *Proc. 7th International Conference on Predictive Models in Software*

*Engineering* (*PROMISE'11*), pp.13:1–13:10 (2011).
[18] Anbatagan, P. and Vouk, M.: On predicting the time taken to correct bug reports in open source projects, *Proc. 2nd International Workshop on Recommendation Systems for Software Engineering* (*RSSE'10*), pp.523–526 (2009).
[19] 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).
[20] Bhattacharya, P. and Neamtiu, I.: Bug-fix time prediction models: Can we do better?, *Proc. 8th International Workshop on Mining Software Repositories* (*MSR'11*), pp.207–210 (2011).
[21] Mockus, A., Fielding, R.T. and Herbsleb, J.D.: Two case studies of open source software development: Apache and moxilla, *Transactions on Software Engineering and Methodology* (*TOSEM'02*), Vol.11, No.3, pp.309–346 (2002).
[22] Herraiz, I., German, D.M., Gonzales-Barahona, J.M. and Robles, G.: Towards a simplification of the bug report form in eclipse, *Proc. 5th International Working Conference on Mining Software Repositories* (*MSR'08*), pp.145–148 (2008).
[23] Anbalagan, P. and Vouk, M.: "Days of the week" effect in predicting the time taken to fix defects, *Proc. 2nd International Workshop on Defects in Large Software Systems* (*DEFECTS'09*), pp.29–30 (2009).
[24] Breiman, L.: Random Forests, *Machine Learning*, Vol.45, pp.5–32 (2001).
[25] Guo, P.J., Zimmermann, T., Nagappan, N. and Murphy, B.: "Not My Bug" and other reasons for software bug report reassignments, *Proc. 2011 Conference on Computer Supported Cooperative Work* (*CSCW'11*), pp.395–404 (2011).
[26] 涌井良幸，涌井貞美：図解でわかる多変量変換，日本実業出版社，東京 (2001).
[27] Herlocker, J.L., Konstan, J.A., Terveen, L.G. and Riedl, J.T.: Evaluating collaborative filtering recommender systems, *Transactions on Information Systems* (*TOIS'04*), Vol.22, No.1, pp.5–53 (2004).

**正木 仁**

2010 年熊本工業高等専門学校電子情報システム工学専攻修了．2012 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．同年富士通株式会社入社．修士（工学）．オープンソースソフトウェア工学，ヒューマン・コンピュータ・インタラクションに興味を持つ．

**大平 雅雄** （正会員）

1998 年京都工芸繊維大学工芸学部電子情報工学科卒業．2000 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．2003 年同大学博士課程修了．同大学情報科学研究科助教を経て，2012 年和歌山大学システム工学部講師．博士（工学）．ソフトウェア工学，特にリポジトリマイニング，ソフトウェア開発における知的協調作業支援の研究に従事．電子情報通信学会，ヒューマンインタフェース学会，ACM 各会員．

**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，IEEE 各会員．

**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．電子情報通信学会，日本ソフトウェア科学会，ACM 各会員，IEEE Senior Member．