NAIST-IS-MT1051098

修士論文

不具合割当パターンを用いた OSS の不具合修正時間の予測

正木 仁

2012 年 2 月 2 日

奈良先端科学技術大学院大学
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
修士 (工学) 授与の要件として提出した修士論文である。

正木 仁

審査委員：

松本 健一 教授	（主指導教員）
安本 慶一 教授	（副指導教員）
門田 暁人 准教授	（副指導教員）
大平 雅雄 助教	（副指導教員）

不具合割当パターンを用いた OSS の不具合修正時間の予測*

正木 仁

内容梗概

近年，大規模オープンソース (OSS) プロジェクトには，大量の不具合が日々報告されている．報告された不具合のすべてを修正することは現実的ではないため，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取捨選択する必要がある．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，オープンソース開発における不具合修正時間の予測に関する研究が盛んに行われている．

本論文では，不具合割当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．不具合割当パターンとは，不具合修正タスク割当時の不具合報告者・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．不具合割当パターンの違いにより，修正作業に取り掛かるまでの時間及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 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，不具合割当パターンが予測精度向上に寄与するとともに，指定期間内（1 週間など）に不具合修正が完了するか否かの管理者の判断を支援できることが分かった．

*奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 修士論文, NAIST-IS-MT1051098, 2012 年 2 月 2 日.

キーワード

不具合割当パターン，不具合管理システム，予測モデル，不具合修正時間，Eclipse Platform

Predicting the Bug Fixing Time based on the Bug Assignment Patterns in Open Source Software*

Hitoshi Masaki

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, since trying to resolve all the reported bugs until the next release is not realistic. 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.

This thesis 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 relationship 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 project, the thesis evaluates the prediction model. As a result, the thesis found that the bug

*Master's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-MT1051098, February 2, 2012.

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).

Keywords:

Bug Assignment Pattern, Bug Tracking System, Prediction Model, Bug Fixing Time, Eclisep Platform

目 次

図 目 次

表 目 次

1. はじめに

1.1 研究の背景と目的

Android OS を搭載したスマートフォンが広く普及するなど，オープンソースソフトウェア (OSS) は我々の社会生活の中でより身近な存在になってきている．従来，プロプラエタリなソフトウェア製品を主に開発してきた民間企業が OSS を自社製品に積極的に採用するようになった理由は，開発工数の大幅な削減が期待できるからである [1, 2]．OSS は原則無償で利用することができる [3] ため，自社製品の一部あるいは大部分に OSS を流用することによって，自社開発する場合に比べて開発工数の大幅な削減が可能となる．結果的に，開発期間の短縮化にもつながるため，携帯電話／スマートフォンのような競争の激しい市場においては，新製品をタイムリーに提供していくための手段となっている．

OSS を利用した製品開発は，上述のような生産性向上の側面のみならず品質確保の手段としても注目されている [4]．一般的な OSS は，ソースコードが無償で公開されており，定められたライセンス条件の下で誰しもが自由に改変や再配布を行うことができる．そのため，世界中の不特定多数のボランティア開発者の協力を得ることができれば，不具合の発見と修正を迅速に行うことが可能となる．Apache や Linux を代表とする大規模 OSS は，プロジェクト参加者を限定しないバザール方式 [5] で開発された OSS の代表例であり，プロプラエタリなソフトウェア製品に引けを取らない品質を有することで知られている．これら大規模 OSS が社会・経済活動において極めて重要な業務システム（あるいはその一部）として利用されていることも珍しくない．例えば，Linux カーネルをベースとして Redhat 社が開発・販売している Linux ディストリビューション Red Hat Enterprise Linux (RHEL) [1]は，ニューヨーク証券取引所の基幹システムとして採用され，世界の経済活動を支えている．

このように，OSS がソフトウェア開発における生産性および品質向上の手段として活用され，我々の社会生活の様々な場面においても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

[1] http://www.redhat.com/products/enterprise-linux/

れるようになった．例えば，Eclipse や Mozilla プロジェクトには一日に数百件の不具合が報告されていることが知られている [6]．バザール方式の OSS 開発では，OSS の品質向上に貢献する共同開発者としてエンドユーザからの積極的な不具合報告は本来歓迎されるべきものである．しかしながら，以下のような複数の要因により，多くの OSS プロジェクトにおいて不具合が解決されるまでの時間（不具合修正時間）が長期化している．

- OSS プロジェクト管理者は大量の不具合報告を一つ一つ読み，それぞれの不具合の解決に適任の開発者を決定し，不具合修正作業を依頼する必要がある [7, 8, 9, 10, 11]
- 不具合修正作業を依頼された開発者が必ずしも適任者であるとは限らず，最終的な作業担当者が決まるまで何度も不具合割当作業が必要になる [6, 12, 13, 14, 15]
- 過去の大量の不具合報告を調べる手間を嫌い，既に報告あるいは解決された不具合 (duplicate bug) が繰り返し報告される（OSS プロジェクト管理者が報告・解決済みの不具合報告かどうかを調べる必要がある）[16, 17, 18, 19, 20]
- エンドユーザからの不具合報告には，不具合修正のために開発者が必要としている情報が含まれていないことが多く，必要な情報を入手するために何度も議論する必要がある [21, 22, 23, 24]
- 不具合修正に積極的に貢献するコア開発者の絶対数が不足している（長期間プロジェクトに貢献し続ける開発者が少ない）ため，特定の少数の開発者のみで報告された不具合の多くを解決せざるを得ない [25, 26, 27, 28]

OSS を利用してソフトウェア製品を開発している企業（OSS 利用企業）にとって，不具合修正時間の長期化は極めて重要な問題である．OSS 利用企業が開発しているソフトウェア製品の品質に大きな影響を与える不具合であれば，OSS プロジェクト側の対応を待たずに自ら対応する必要があるためである．ただし，OSS 利用による工数削減効果が大きく損なわれるため，OSS 利用企業がすべての不具

合に対応するのは現実的ではない．OSS プロジェクト側での不具合修正が許容範囲内で完了するかどうかを不具合毎に見極める必要がある．

OSS プロジェクト管理者にとっても，個々の不具合の修正完了期間の見極めは重要な課題である．不具合が日々大量に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等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盛んに行われている．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 [22, 29, 30, 31, 32]．また，比較的粒度の大きな期間（例えば，6 か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 利用企業や OSS プロジェクト管理者のニーズを十分には満たせていない．

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パターン（不具合割当パターン）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．不具合の再割当が不具合修正時間の遅延の要因の一つであること [12]，不具合割当には 4 つのパターンがありパターン毎に不具合修正時間が異なること [33, 34] が分かっており，予測モデルに不具合割当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の予測モデルを 3 種類のメトリクスを用いて構築し，Eclipse Platform プロジェクトを対象として実験を行い精度評価を行った．実験の結果，不具合修正が完了するまでの日数の予測において，有用なモデルを構築することができなかった．しかし，指定期間内に不具合修正が完了するか否かの予測において，有

用なモデルを構築することができた．また，不具合割当パターンは，短期間で（1日以内に）不具合修正が完了するか否かの予測において，予測精度の向上に寄与することを明らかにした．

1.2 本論文の構成

本論文の構成は以下の通りである．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と問題点を説明し，不具合修正時間の予測の必要性について述べる．3章では不具合修正プロセスにおける社会的関係が不具合修正時間に与える影響について説明する．4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．5章では Eclipse Platform プロジェクトを対象に行った実験の方法について述べ，6章で実験結果について述べる．7章では実験結果に対する考察を行い，8章では関連研究について述べる．最後に，9章で本論文の結論を述べる．

2. OSS 開発における不具合修正プロセス

本章では，OSS 開発における不具合修正プロセスについて説明する．2.1 節では不具合修正プロセスの概要を説明する．2.2 節では不具合の管理方法について説明し，2.3 節では不具合修正プロセスの参加者とその役割を説明する．2.4 節では不具合修正プロセスにおける問題と不具合修正時間予測の必要性について述べる．

2.1 不具合修正プロセスの概要

OSS 開発では，開発者が世界中に点在しているため，不具合に関する情報を共有することが可能な不具合管理システムを利用して不具合の修正作業が行われている．まず，OSS に含まれる不具合が発見された場合，不具合を発見した人によって不具合管理システムに不具合に関する情報が報告される．その後，報告された不具合に対して修正を行う必要があるかないかが判断され，不具合の修正が必要な場合，不具合の修正が行われる．最後に，修正された不具合に問題がないかの確認が行われ，修正に問題がなければ，不具合の修正作業は完了となる．ただし，修正に問題があれば，不具合の修正が再度行われる．

不具合修正プロセスにおける修正作業の進捗は，Status として不具合管理システムに記録される．不具合修正プロセスにおける不具合の Status の遷移を図 1 に示し，各 Status の詳細を表 1 に示す．

2.2 不具合修正プロセスにおける不具合の管理方法

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を利用している．不具合管理システムには，Bugzilla[2]，Trac[3]，Redmine[4]等がある．いずれの不具合管理システムも共通して，以下のような特徴を持っている．

- OSS プロジェクトに関連する全ての不具合を一元管理できる．

[2]http://www.bugzilla.org/
[3]http://trac.edgewall.org/
[4]http://www.redmine.org/

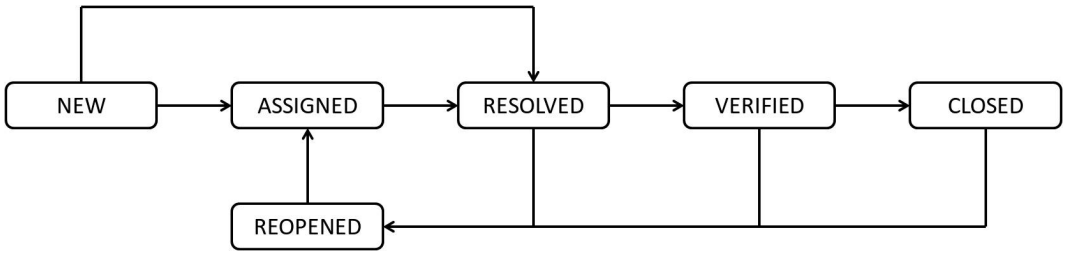

図 1　不具合票の Status の遷移

表 1　Bugzilla における Status の種類

Status 名	詳細
NEW	不具合の報告が行われた状態
ASSIGENED	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

- 不具合の修正に関連する作業の履歴が細かく記録される．
- OSS の開発者だけでなく，一般の利用者も不具合に関する情報を確認できる．

これにより，地理的に分散していても，開発者間で不具合に関する情報を随時共有することができるため，開発者は容易に不具合の修正を行うことができる．以降，本論文では多くの OSS プロジェクトで利用されている Bugzilla を対象に不具合管理システムの説明を行う．

不具合管理システム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と変更履歴ページが作成される．基本情報ページと変更履歴ページの例を図 2，図 3 に示す．

基本情報ページには，固有の ID 番号や不具合が発生したプロダクト，不具合の再現方法等の不具合に関する詳細な情報が記録されている．基本情報記録部に

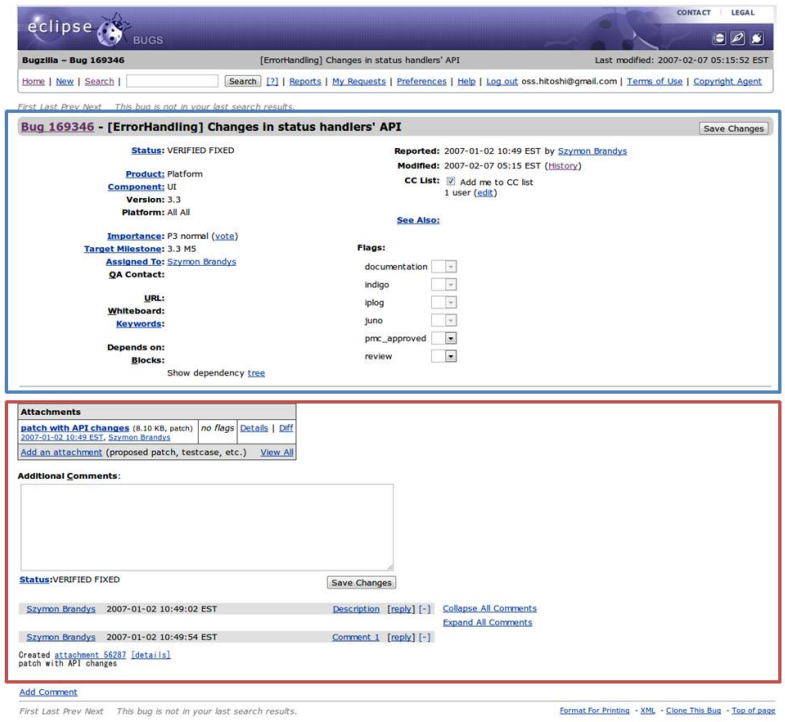

図 2　基本情報ページ

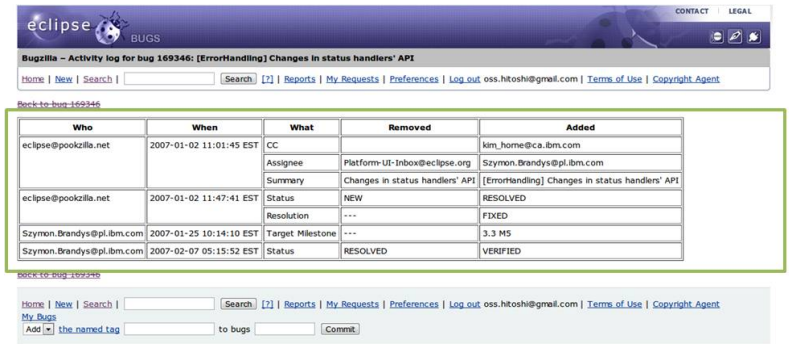

図 3　変更履歴ページ

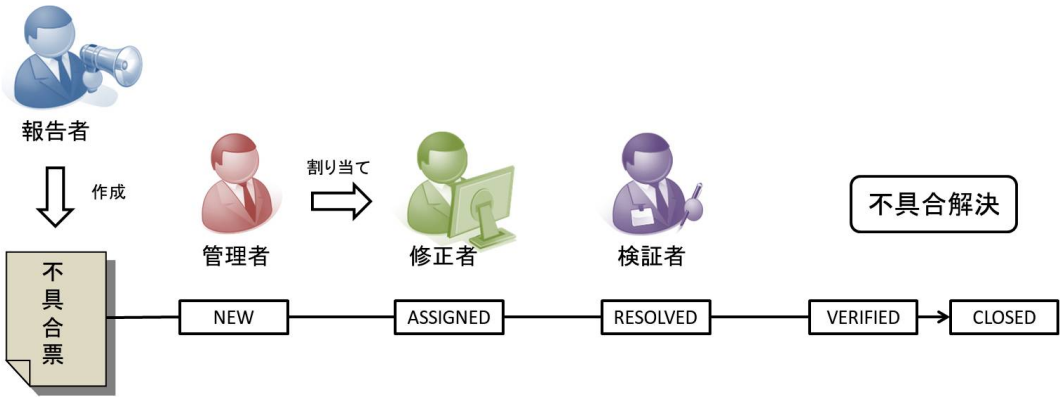

図 4 不具合修正プロセス

は，不具合の報告を行った人が登録した不具合に関する情報や報告した日時等が記録されている．議論情報記録部には，利用者又は開発者が投稿した不具合に対する意見や質疑等が記録されている．

変更履歴ページには，基本情報ページに登録されている不具合に関する情報の変更履歴が記録されている．変更情報記録部には，変更を行った人物，変更を行った日時，変更箇所，変更内容が一覧で記録されている．

本論文では，基本情報ページ及び変更履歴ページを併せて不具合票と定義する．

2.3 不具合修正プロセスの参加者

OSS 開発における不具合修正プロセスには，OSS 開発に携わっている開発者だけではなく，OSS 開発に携わっていない利用者も参加しており，それぞれが重要な役割を担っている．不具合票の各状態における参加者を図 4 に示し，それぞれの参加者の役割を以下に示す．

- 報告者

  報告者の役割は，発見した不具合に関する情報を不具合管理システムに登録することである．報告者は，OSS の利用者や開発者が担う役割であり，不具合修正プロセスの参加者の中で最も多くの人が関わっている．

- 管理者

  管理者の役割は，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ことと，不具合の修正を開発者に割り当てることである．管理者は，OSS の開発者の中でも中心的な開発者が担う役割である．

- 修正者

  修正者の役割は，依頼された不具合の修正を行うことである．修正者は，OSS の開発者が担う役割である．

- 検証者

  検証者の役割は，修正が完了した不具合に問題がないかを確認することである．検証者は，管理者と同じく OSS の中心的な開発者が担う役割である．

これらの役割は，全て同一の人物が担うこともあれば，それぞれ別な人物が担うこともある．例えば，開発者が不具合を報告し，自ら引き受けて不具合の修正，検証を行うことが考えられる．また，利用者が不具合の報告を行い，それぞれ異なる開発者が不具合の依頼，修正，検証を行うことも考えられる．ただし，当該不具合の修正に適切な開発者でなかった場合等に修正者の変更が行われることがあるため，1 つの不具合に対して複数人の修正者が参加することもある．

2.4 不具合修正プロセスの問題点と不具合修正時間予測の必要性

大規模 OSS プロジェクトでは，日々大量の不具合が報告されている．しかし，管理者の不足，不具合の再割当，不具合に関する情報の不足等の原因により，多くの OSS プロジェクトでは不具合の修正時間が長期化している．

このような現状では，定期的なバージョンアップのリリース前にすべての不具合の修正が完了することは困難である．そのため，管理者はそれぞれの不具合の修正時間を見積もり，どの不具合の修正を実施して次期リリースに含めるかの取捨選択を行う必要がある．そこで，本論文では個々の不具合の修正時間を見極める OSS プロジェクトの管理者を支援するために，不具合修正時間の予測を行う．

表 2 Bugzilla における Resolution の種類

Resolution 名	詳細
FIXED	不具合が修正された
INVALID	不具合ではないと判断された
WONTFIX	機能性の低下等の理由により，不具合の修正が見送られた
DUPLICATE	同一の不具合票が存在すると判断された
WORKFORME	不具合を再現できないため，不具合の修正を行っていない

本論文で扱う不具合修正時間は，不具合の修正者が決定してから不具合の修正が完了するまでの時間とする．具体的には，不具合票の Assignee に開発者が登録されてから，不具合票の Resolution に不具合の修正結果が登録されるまでの時間である．ただし，不具合の修正が完了した状態とは，不具合票の Resolution が表 2 に示すいずれかの状態にあることを指す．Resolution の状態には，FIXED，INVALID，WONTFIX，DUPLICATE，WORKSFORME があり，不具合が修正される場合だけでなく，不具合が修正されない場合も含まれる．

3. 社会的関係が不具合修正時間に与える影響

本章では，不具合修正プロセスにおける参加者の社会的関係が不具合修正時間に与える影響について述べる．3.1 節では不具合修正プロセスにおける不具合割当パターンの概要について述べ，3.2 節では不具合割当パターンと不具合修正時間の関係について述べる．

3.1 不具合割当パターンの概要

大澤 [33] は，不具合修正プロセスにおける参加者の社会的関係が不具合修正時間に与える影響を明らかにするために，報告者・管理者・修正者の関係を 4 種類のパターン（不具合割当パターン）として，以下のように分類した．各不具合割当パターンを図 5 に示す．

- 自己解決型

  自己解決型では，報告者，管理者，修正者の役割を全て同じ人物が担当する．自己解決型の例として，報告者は不具合の内容を熟知しており，不具合の修正を行うことができる場合が考えられる．

- 修正委託型

  修正委託型では，報告者と管理者の役割は同じ人物が担当するが，管理者の役割は異なる人物が担当する．修正委託型の例として，報告者は不具合の内容を熟知しており，かつ適切な開発者に修正を依頼できるが，不具合の修正を行うことができない場合が考えられる．

- 修正受託型

  修正受託型では，管理者と修正者の役割は同じ人物が担当するが，報告者の役割は異なる人物が担当する．修正受託型の例として，報告者が不具合の内容を熟知していない，又は適切な開発者に修正を依頼できないが，管理者が不具合の修正を行うことができる場合が考えられる．

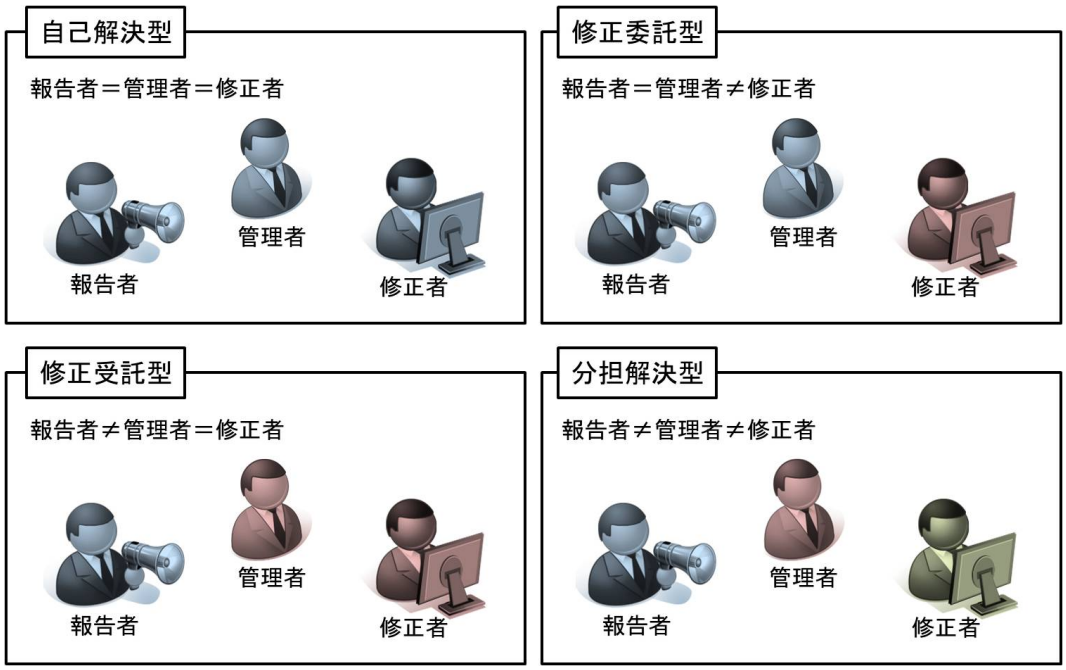

図 5 不具合割当パターン

- 分担解決型

  分担解決型では，報告者，管理者，修正者の役割を全て異なる人物が担当する．分担解決型の例として，報告者が不具合の内容を熟知していない，又は適切な開発者に修正を依頼できず，管理者も不具合の修正を行うことができない場合が考えられる．

大規模 OSS プロジェクト（Eclipse, Mozilla, Linux）を対象に分析を行った結果，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，不具合修正時間は長いことを明らかにした．ただし，OSS プロジェクトによっては修正委託型の不具合修正時間が短い場合もあり，修正開始前の議論が不具合修正時間に影響している可能性があると考察している．

3.2 不具合割当パターンと不具合修正時間の関係

先行研究では修正の行われた不具合票のみを分類の対象としていたが，本論文では修正の行われなかった不具合票も対象としているため，再度不具合票の分類を行い，不具合割当パターンと不具合修正時間の関係を調べた．Eclipse プロジェクトの 2 つのプロダクト（Platform，JDT）を対象に，不具合票を分類した結果を図 6 に示す．両プロダクトともに，修正受託型と分担解決型の比率が高く，自己解決型と修正委託型の比率が低いことが分かる．これは，自己解決型と修正委託型における報告者の役割を開発者だけが担っているのに対して，修正受託型と分担解決型における報告者の役割は利用者と開発者が担っていることに起因すると考えられる．

Eclipse プロジェクトにおける各不具合割当パターンの不具合修正時間に関する統計量を表 3，表 4 に示す．Platform において，修正委託型が最も不具合修正時間が短く，過半数の不具合が約 4 時間以内に修正されていた．一方，分担解決型の不具合修正時間が最も長く，過半数の不具合が約 1 週間以上修正に時間が掛かっていた．JDT において，自己解決型が最も修正時間が短かったが，Platform と同様に分担解決型が最も修正に時間が掛かっていた．

Eclipse プロジェクトの両プロダクトにおいて，各不具合割当パターンで不具合修正時間に違いが見られた．これより，不具合修正時間の予測に不具合割当パターンは有用であると考えられる．本論文では，不具合割当パターンを用いて不具合修正時間の予測を行うことで，不具合割当パターンが予測精度の向上に寄与するかどうかを明らかにする．

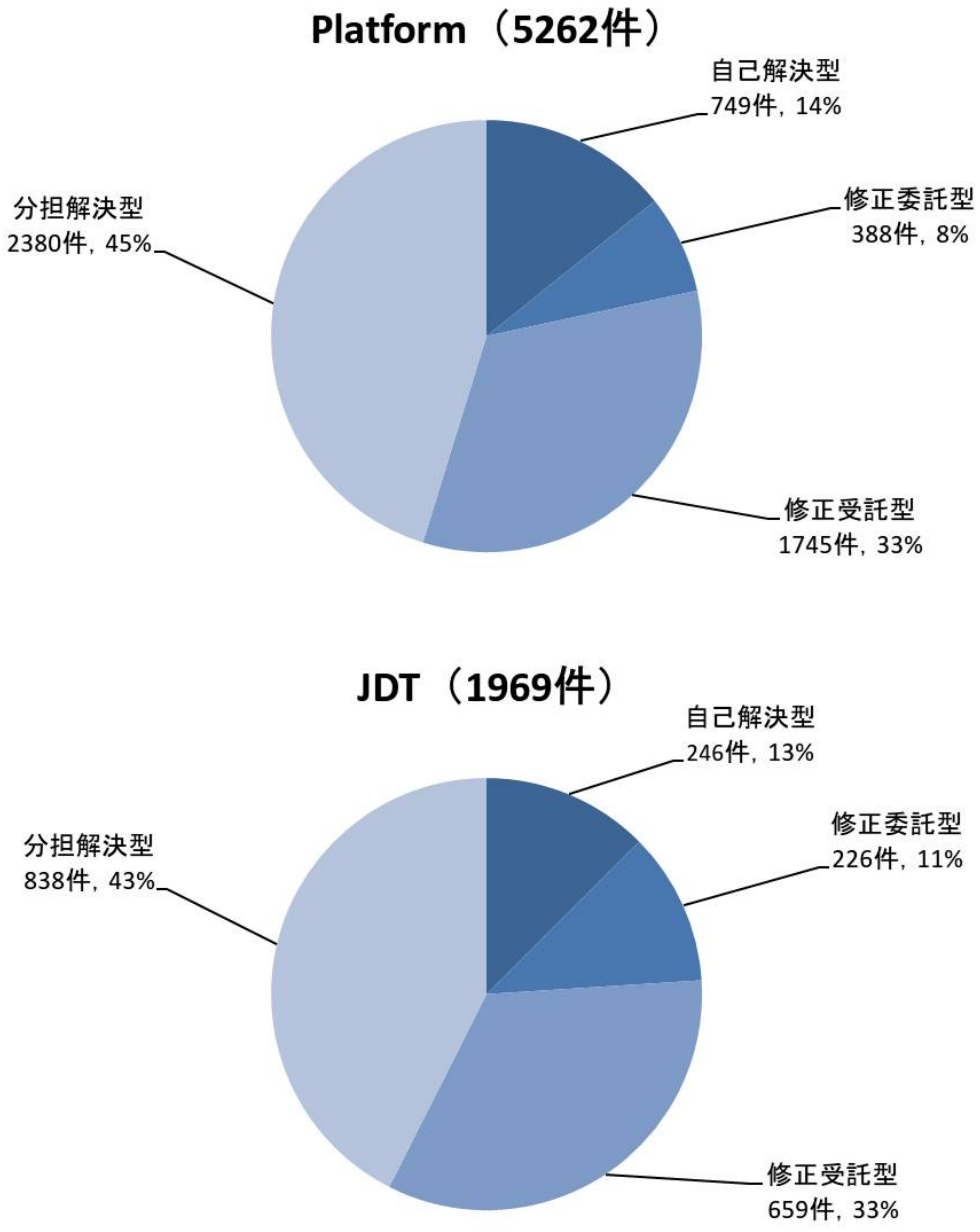

図 6 Eclipse プロジェクトにおける各不具合割当パターンの件数と比率

表 3 Platform における各不具合割当パターンの不具合修正時間

	不具合割当パターン			
	自己解決型	修正委託型	修正受託型	分担解決型
中央値（日）	1.03	0.16	2.02	6.95
平均値（日）	23.88	38.10	34.70	70.60
標準偏差	69.06	128.24	97.54	160.30
分散	4769.16	16444.36	9513.37	25696.33
最大値（日）	744.87	1119.90	1204.96	1392.53
最小値（日）	0.00	0.00	0.00	0.00

表 4 JDT における各不具合割当パターンの不具合修正時間

	不具合割当パターン			
	自己解決型	修正委託型	修正受託型	分担解決型
中央値（日）	0.71	0.84	0.85	3.06
平均値（日）	18.56	18.35	20.99	43.19
標準偏差	66.86	57.91	70.91	119.00
分散	4470.68	3353.10	5027.75	14161.91
最大値（日）	731.33	664.91	817.92	1153.87
最小値（日）	0.00	0.00	0.00	0.00

4. 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．4.1 節では不具合修正時間の予測手法について述べる．4.2 節では予測モデル構築に用いた 3 種類のメトリクスについて述べ，4.3 節では 3 つのモデルについて説明する．

4.1 概要

本論文では，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を行う．

一定期間内に修正された不具合票の割合を図 7 に示し，1 日以内，1 週間以内，1ヶ月以内に修正された不具合の件数と割合を表 5 に示す．図 7 より，解決済みの多くの不具合が 1 日以内で修正されていることが分かる．実際の不具合修正プロセスでは，開発者のタスク管理を行う上で，数時間以内に修正が可能か否かよりも，中長期的な時間が必要かどうかを知ることが重要である．そこで，本実験の目的変数を離散値とする手法では，1 日以内，1 週間以内，1ヶ月以内に不具合修正が完了するか否かの予測を行う．

4.2 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（不具合票に関連するメトリクス，時間に関連するメトリクス，人に関連するメトリクス）を用いる．

4.2.1 不具合票に関連するメトリクス

利用者又は開発者は不具合を発見した場合，不具合の発生環境や再現方法等の不具合情報を不具合管理システムに登録する．開発者は不具合票に記録されて

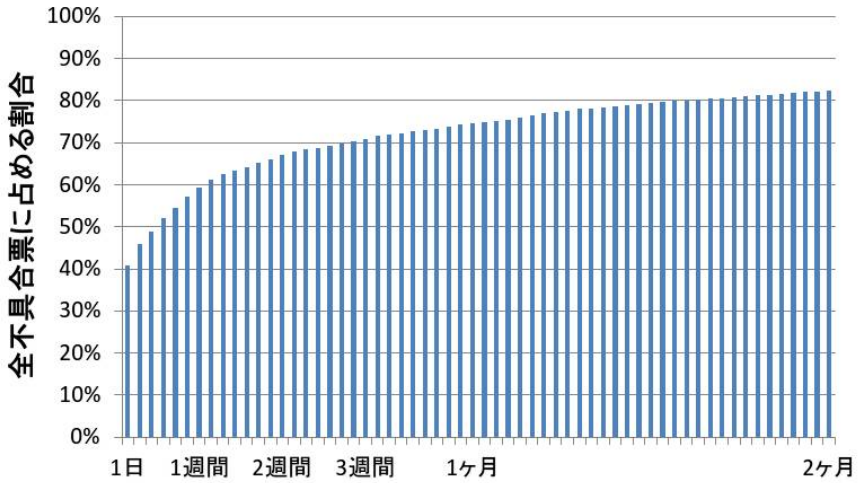

図 7 一定期間内に不具合修正が完了した不具合票の割合

表 5 指定期間内に不具合修正が完了した不具合票の件数と割合

	不具合票 (件)	比率 (%)
1 日以内	2151	40.88
1 週間以内	3128	59.45
1ヶ月以内	3942	74.91

いる不具合情報を基に，不具合を再現し，修正を行う．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である．例えば，Mockus ら [35] は基幹となるモジュールや影響力の高いモジュールの不具合修正時間は短いことを明らかにしている．また，Herraiz ら [36] は重要度の高い不具合ほど，不具合修正時間は短いことを明らかにしている．

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（不具合が発生したコンポーネント（Component），優先度（Priority），重要度（Serverity）等）や，利用者又は開発者らによる議論に関する情報（コメント数（CommentCount），コメントの平均文字数（CommentWords））等である．不具合票に関連するメトリクスの概要を表 6 に示す．

表 6 不具合票に関連するメトリクスの概要

変数	尺度	詳細
Component	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	間隔	依存関係にある不具合の数
BlocksCount	間隔	修正を妨げている不具合の数
CCCount	間隔	メーリングリストの登録者数
DiscriptionWords	間隔	不具合に関する概要の文字数
CommentsCount	間隔	コメントの数
CommentsWords	間隔	コメントの総文字数
AttachmentsCount	間隔	添付ファイルの数

4.2.2 時間に関連するメトリクス

管理者から不具合の修正を割り当てられた開発者は，限られた時間を使い不具合の修正を行う．休日や長期休暇のある月のように，不具合の修正を行うための時間を確保できるかにより，不具合の修正時間は異なると考えられる．例えば，Anbalagan ら [37] は不具合の修正時間は不具合の報告が行われた日の曜日と関係があることを明らかにしている．

時間に関連するメトリクスとは，不具合の報告から割当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．時間に関連するメトリクスの概要を表 7 に示す．

4.2.3 人に関係するメトリクス

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いる．いずれかの役割を担う利用者又は開発者の能力や経験により，不具合の修正時間は異なると考えられる．例えば，多くの不具合を報告している利用者や開

表 7 時間に関連するメトリクスの概要

変数	尺度	詳細
AssignTime	間隔	修正者の決定までに要した時間
AssignedMonth	間隔	修正者が決定した月
AssignedDay	間隔	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 8 人に関連するメトリクスの概要

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
Pattern	名義	不具合割当パターン

発者が報告者の場合，詳細な不具合の再現方法等の修正に必要な情報を不具合票に登録することにより，修正者は迅速に不具合の修正に取りかかることができるため，修正時間は短くなると考えられる．

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パターンである．人に関連するメトリクスの概要を表 7 に示す．

4.3 不具合修正時間予測のためのモデル

本実験では，不具合修正が完了するまでの日数の予測に目的変数を連続値として予測可能な重回帰分析及びランダムフォレスト法を用い，指定期間内に不具合修正が完了するか否かの予測に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及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の予測において，良く用いられているため [22, 31, 38, 39]，本研究においても採用する．

4.3.1 重回帰分析

重回帰分析は，目的変数 y に対して k 個の説明変数 $x_1, \cdots, x_k$ が与えられたとき，式 (1) のように定義される．

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k + \epsilon \tag{1}$$

重回帰分析では，誤差項は正規性，等分散性が仮定されている．等分散性とは，n をデータ数とするとき，誤差項の各誤差 $\epsilon_i \, (i = 1, \cdots, n)$ が同一の分散を持つことである．

4.3.2 ロジスティック回帰分析

ロジスティック回帰分析では，2 クラスの標本を判別する判別関数にロジスティック関数を用いて式 (2) のように表現する．

$$P\left(y|x_1, \cdots, x_p\right) = \frac{1}{1 + e^{-(\alpha_1 x_1 + \cdots + \alpha_p x_p + \beta)}} \tag{2}$$

ここで，$y \in \{0, 1\}$ は 2 値のクラスを表す目的変数，x_i は目的変数，α_i は回帰係数，$P\left(y|x_1, \cdots, x_p\right)$ は説明変数 x_i に対して y が 1 のクラスに属する確率である．本論文では，$P\left(y|x_1, \cdots, x_p\right)$ の値が 0.5 を超える場合に，目的変数 y が 1 のクラスに属すると判別する．

4.3.3 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り，2001 年に Breiman によって提案された [40]．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．そして，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全ての説明変数を用いていたのに対して，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ている．

5. 実験方法

本章では，不具合修正時間予測の実験方法について述べる．5.1 節では本実験を行う目的について述べる．5.2 節では不具合修正時間の予測モデル構築に用いたデータセットについて説明し，5.3 節で不具合修正時間の予測モデルの予測精度を求めるために用いた評価基準について説明する．5.4 節では交差検定により行った実験の手順について説明する．

5.1 目的

本実験では，不具合修正時間の予測に適した予測手法を明らかにすることを目的として，不具合修正が完了するまでの日数，及び指定期間内に不具合修正が完了するか否かの予測をそれぞれ 2 種類のモデルを用いて行う．さらに，不具合割当パターンが不具合修正時間の予測に与える効果を明らかにするために，それぞれの予測において，不具合割当パターンを説明変数に用いた場合と用いなかった場合で予測を行い，予測精度を比較する．

5.2 データセット

本実験では，Eclipse プロジェクトで利用されている不具合管理システムの不具合票をデータセットとして用いる．

5.2.1 Eclipse プロジェクト

Eclipse は IBM（International Business Machines Corporation）によって開発された統合開発環境（IDE: Integrated Development Environment）の開発プロジェクトである．1998 年に IBM カナダでプロジェクトが開始され，2001 年に OSS として公開された．その後，2004 年に現在の非営利法人 Eclipse Foundation が設立された．同年 6 月に Eclipse 3.0 がリリースされ，以降毎年 6 月に最新バージョンがリリースされている．

表 9 Eclipse プロジェクトにおける不具合件数

	Eclipse Platform
全不具合票 (件)	21322
解決済不具合票 (件)	15467
対象不具合票 (件)	5262

本実験では，Eclipse の中でも中心的なプロジェクトである Eclipse Platform プロジェクト[5]を対象に不具合修正時間の予測を行う．Eclipse Platform 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．

5.2.2 不具合票

本実験では，2007 年から 2009 年の間に不具合管理システムに報告された不具合票をデータセットとして用いる．期間中の全不具合票，解決済不具合票，対象不具合票の件数を表 9 に示す．解決済不具合票とは，Resolution が FIXED，INVALID，WONTFIX，DUPLICATE，WORKSFORME のいずれかの状態にある不具合票のことであり，修正された不具合票（FIXED）だけでなく，修正されなかった不具合票（INVALID，WONTFIX，DUPLICATE，WORKSFORME）も含む．

本実験では，下記の不具合票を除いた 5262 件の対象不具合票から，不具合修正時間予測モデルの構築に用いるメトリクスを抽出する．

修正者が変更されている不具合票

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の不具合修正時間を予測するため，1 人目の修正者によって修正された不具合票のみを対象とする．そのため，Assignee が複数回変更されている不具合票を除外する．

[5]http://www.eclipse.org/platform/

問題が再発した不具合票

不具合の修正が 1 度で完了することを想定しているため，再修正が行われていない不具合票を対象とする．そのため，Resolution が複数回変更されている不具合票を除外する．

修正者の特定が不可能な不具合票

不具合票に修正者として登録されているアカウントの中には，開発者のメーリングリスト (以下，ML) が登録されているものがある．本論文における不具合修正時間の予測では，修正者を一意に特定する必要があるため，修正者のアカウントに ML が登録されている不具合票を除外する．

機能拡張に関する不具合票

不具合票の中には不具合報告に関する不具合票だけでなく，機能拡張に関する不具合票も存在する．しかし，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を除外する．

記録情報が間違っている不具合票

修正者決定時刻と不具合修正完了時刻が同じ場合，開発者は修正者決定時に報告せずに，不具合の修正が完了してから同時に報告したと考えられる．修正者決定時刻よりも不具合修正完了時刻が前の場合，管理者が報告を怠った又は修正者が誤って報告を行ってしまったと考えられる．このような不具合票から実際の不具合修正時間を求めることは困難なため，該当する不具合票を除外する．

5.3 評価基準

不具合修正が完了するまでの日数の予測における評価基準として，決定係数及び自由度調整済み決定係数を用いる．決定係数とは，回帰方程式の当てはまりの

良さ（予測精度）を評価するための指標であり，式 (3) のように定義される．

$$R^2 = 1 - \frac{\sum_{i=1}(y_i - \hat{y}_i)^2}{\sum_{i=1}(y_i - \bar{y}_i)^2} \tag{3}$$

決定係数は値域 $[0, 1]$ を取り，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 [41]．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 (4) のように定義される．

$$R^{*2} = 1 - \frac{\sum_{i=1}(y_i - \hat{y}_i)^2/(N-k-1)}{\sum_{i=1}(y_i - \bar{y}_i)^2/(N-1)} \tag{4}$$

指定期間内に不具合修正が完了するか否かの予測における評価基準として，適合率，再現率，F1 値 [42] を用いる．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であり，表 10 に示す記号を用いて式 (5) のように定義される．

$$Precision = \frac{n_{11}}{n_{11} + n_{12}} \tag{5}$$

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り，表 10 に示す記号を用いて式 (6) のように定義される．

$$Recall = \frac{n_{11}}{n_{11} + n_{21}} \tag{6}$$

ただし，適合率と再現率は一方の割合が高くなると，もう一方の割合が低くなる関係にある．そこで，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 (7) のように定義される．

$$F1 - measure = \frac{2 \times Precition \times Recall}{Precition + Recall} \tag{7}$$

3 つの評価基準はいずれも値域 $[0, 1]$ を取り，値が高い程予測精度が高く，値が低い程予測精度が低いことを表す．

表 10 指定期間内に不具合修正が完了するか否かの予測における結果の分類

		実測値	
		完了する	完了しない
予測値	完了する	n_{11}	n_{12}
	完了しない	n_{21}	n_{22}

5.4 実験手順

4 章で説明したメトリクスとモデルを用いて，不具合修正が完了するまでの日数及び指定期間内に不具合修正が完了するか否かの予測を交差検定により，それぞれ下記の手順で実施した．

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過ぎると，多重共線性が生じやすい等の問題があるため，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 不具合修正が完了するまでの日数の予測の場合，目的変数の対数変換を行う．指定期間内に不具合修正が完了するか否かの予測の場合，目的変数の値は一定期間 (1 日以内，1 週間以内，1ヶ月以内) に修正が完了した不具合を 1 とし，修正が完了していない不具合を 0 とした．

Step3．データセットの分割 データセットを，9 個のフィットデータ $fit_1, \cdots, fit_9$ と 1 個のテストデータ $test$ 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 $fit_1, \cdots, fit_9$ 及び 4.3 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，5.3 節で説明した方法を用いて予測精度を求める．

Step6．実験の繰り返し Step3 から Step5 を 10 回繰り返し，予測精度の平均値を求める．

Step7．目的変数の変更 指定期間内に不具合修正が完了するか否かの予測では，全ての目的変数に対して Step2 から Step5 を繰り返し行う．

6. 実験結果

本章では，2 種類の予測方法を用いて不具合修正時間の予測を行った実験の結果について述べる．6.1 節では不具合修正が完了するまでの日数の予測結果について述べ，6.2 節では指定期間内に不具合修正が完了するか否かの予測結果について述べる．

6.1 不具合修正が完了するまでの日数の予測結果

目的変数の対数変換による残差の比較

重回帰分析を行うにあたり，(1) 残差の分散が等しいことと (2) 残差が正規分布に従うことが仮定されている．しかし，本実験を行った結果，実測値と予測値の残差は正規分布に従っていないことが分かった．そこで本実験では，目的変数となる不具合修正時間の対数変換を行い，不具合修正時間の予測を行った．

対数変換を行わなかった場合の残差分布を図 8 に示し，対数変換を行った場合の残差分布を図 9 に示す．対数変換を行わなかった場合に比べ，対数変換を行った場合，残差の分布はほぼ等しく，残差の等分散性が満たされていることが分かる．次に，対数変換を行った場合の正規 Q-Q プロットを図 10 に示し，対数変換を行った場合の正規 Q-Q プロットを図 11 に示す．対数変換を行わなかった場合に比べ，対数変換を行った場合，残差は直線状にプロットされており，残差の正規性も満たされていることが分かる．

予測精度の比較

重回帰分析及びランダムフォレスト法の決定係数及び自由度調節済決定係数を求めた結果を表 11 に示す．これより，不具合修正が完了するまでの日数を予測する場合，重回帰分析に比べ，ランダムフォレスト法の方が予測精度が高いことが分かる．具体的には，決定係数は 23 %高く，自由度調節済決定係数は 32 %高い値を示していた．しかし，重回帰分析及びランダムフォレスト法のどちらも，決

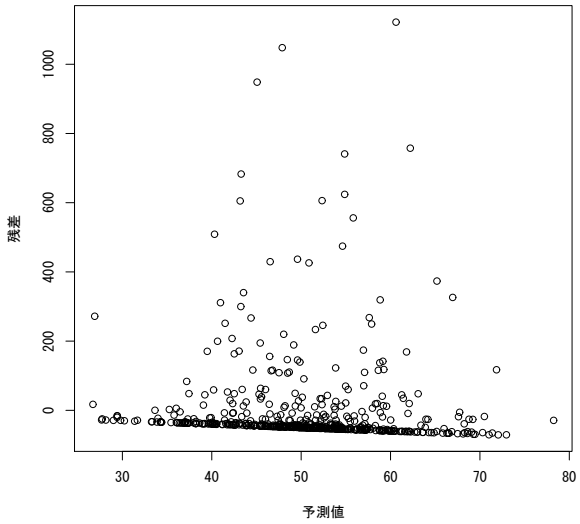

図 8 対数変換を行わなかった場合の残差分布

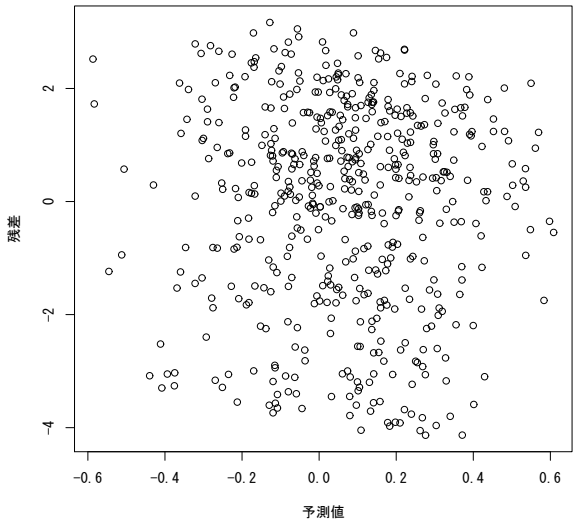

図 9 対数変換を行った場合の残差分布

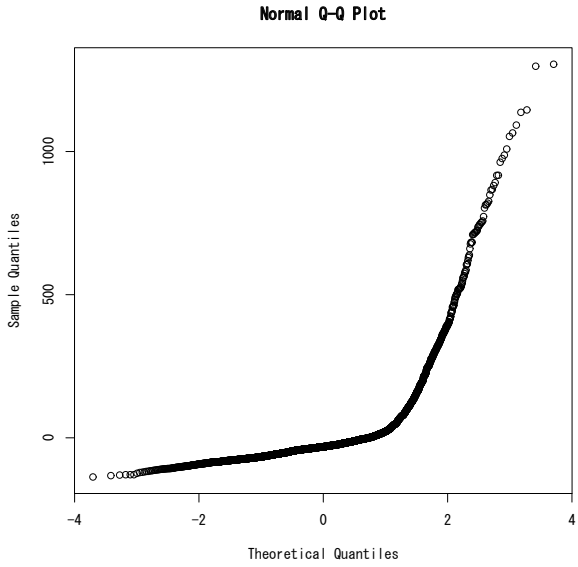

図 10　対数変換を行わなかった場合の残差の正規 Q-Q プロット

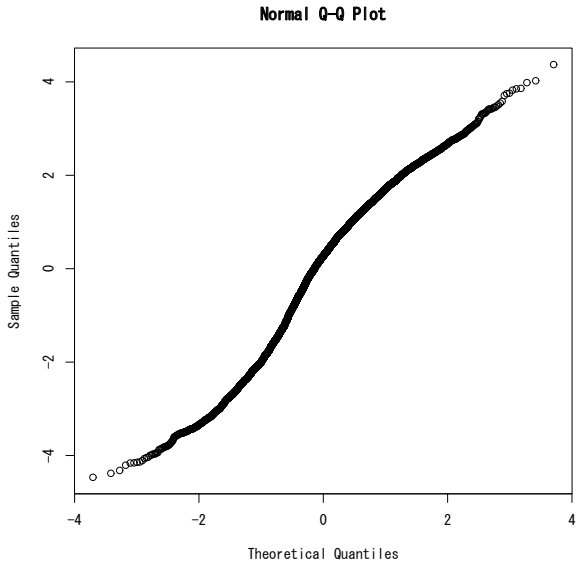

図 11　対数変換を行った場合の残差の正規 Q-Q プロット

表 11　重回帰分析及びランダムフォレスト法の予測精度

予測モデル名	決定係数　自由度調整済決定係数
重回帰分析	0.202　0.135
ランダムフォレスト法	0.264　0.201

定係数が 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することはできなかった．

各予測モデルにおける変数の重要度

各予測モデルにおいて，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，重回帰分析では各変数の標準化偏回帰係数を求め，ランダムフォレスト法では精度減少値を求めた．重回帰分析とランダムフォレスト法における各変数の標準化偏回帰係数及び精度減少値を表 12 に示す．標準化偏回帰係数とは，目的変数に対する各説明変数の影響度を表す偏回帰係数を標準化したものである．各変数の標準化偏回帰係数を比較することで，モデル構築に寄与している変数を求めることができる．精度減少値は，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程，モデル構築に寄与していることを表す．

重回帰分析では，不具合割当パターン（Pattern）が最もモデル構築に寄与していた．また，ランダムフォレスト法では，コンポーネント（Component）が最もモデル構築に寄与しており，次に不具合割当パターン（Pattern）がモデル構築に寄与していた．これより，不具合割当パターンは不具合修正が完了するまでの日数の予測に有用であることが分かった．

表 12 重回帰分析及びランダムフォレスト法における各変数の重要度

メトリクス	変数名	重回帰分析	ランダムフォレスト法
		標準化偏回帰係数	精度減少値
不具合票	Component	-0.07	47.98
	Priority	0.03	2.48
	Severity	0.02	3.46
	Milestone	-0.05	5.09
	DescriptionWords	0.02	5.47
	CommentsCount	-0.17	7.93
	CommentsWords	-0.17	7.93
	AttachmentsCount	0.05	5.02
	DependsOnCount	0.02	0.71
	BlocksCount	0.01	1.20
	CCCount	0.06	7.50
時間	AssignTime	0.04	11.13
	AssignedMonth	0.06	7.63
	AssignedDay	0.01	3.84
	AssignedWeekEnd	-0.01	0.02
人	Reporter	-0.08	16.32
	Assignor	0.02	22.36
	Resolver	-0.06	28.65
	Pattern	-0.43	34.12

6.2 指定期間内に不具合修正が完了するか否かの予測結果

予測精度の比較

ロジスティック回帰分析の予測精度（適合率，再現率，F1 値）を求めた結果，及びランダムによる予測に対する向上率を表 13 に示す．予測期間を 1 日とした予測では，適合率が大幅に向上していたのに対し，再現率は若干減少していた．予測期間を 1 週間及び 1ヶ月とした予測では，適合率，再現率ともに向上しており，特に再現率が大きく向上していた．また，F1 値は全ての予測期間で，ランダムによる予測に比べ，約 15 %以上向上していた．

ランダムフォレスト法の予測精度（適合率，再現率，F1 値）を求めた結果，及びランダムによる予測に対する向上率を表 14 に示す．予測期間を 1 日とした予測では，適合率が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 週間及び 1ヶ月とした予測では，ロジスティック回帰分析と同様に適合率，再現率ともに向上していた．また，F1 値は全ての予測期間で，ランダムによる予測に比べ，約 15 %以上向上していた．

各目的変数における F1 値の推移を図 12 に示す．予測期間を 1 日とした予測では，ロジスティック回帰分析に比べ，ランダムフォレスト法の方が予測精度は高かった．具体的な予測精度は，適合率は変わらず，再現率が 19 %高く，F1 値も 12 %高かった．予測期間を 1 週間及び 1 週間とした予測では，ロジスティック回帰分析とランダムフォレスト法の予測精度に違いはほとんど見られなかった．

実験の結果より，両予測モデルともにランダムによる予測に比べ，有効なモデルを構築できていることが分かった．特に，予測期間を 1 日とした予測では，ロジスティック回帰分析に比べ，ランダムフォレスト法の方が有効であることが分かった．

各予測モデルにおける変数の重要度

各予測モデルにおいて，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，ロジスティック回帰分析では各変数の残差逸脱度を求め，ランダムフォレスト法では精度減少値を求めた．逸脱度とは，モデ

表 13　ロジスティック回帰分析の予測精度

	予測期間	適合率	再現率	F1 値
ロジスティック回帰分析	1 日	68.14	38.22	48.97
	1 週間	67.90	76.66	72.02
	1ヶ月	76.67	98.77	86.33
ランダム予測に対する向上率	1 日	66.68 %	-6.50 %	19.80 %
	1 週間	14.22 %	28.95 %	21.14 %
	1ヶ月	2.35 %	31.86 %	15.24 %

表 14　ランダムフォレスト法の予測精度

	予測期間	適合率	再現率	F1 値
ランダムフォレスト法	1 日	68.22	45.63	54.68
	1 週間	66.78	77.96	71.94
	1ヶ月	78.06	95.75	86.01
ランダム予測に対する向上率	1 日	66.87 %	11.62 %	33.76 %
	1 週間	12.33 %	31.14 %	21.01 %
	1ヶ月	4.21 %	27.82 %	14.81 %

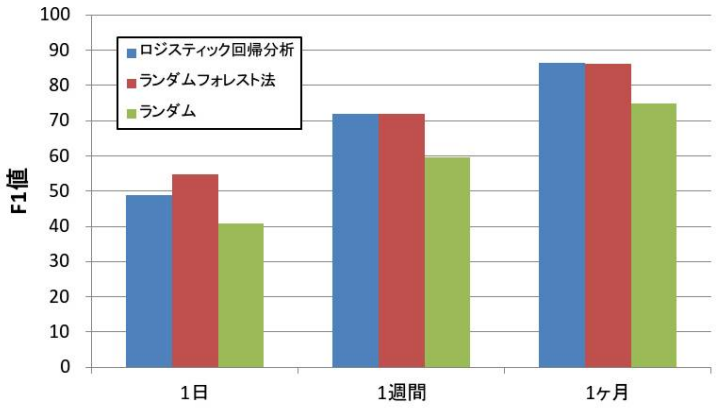

図 12　各予測モデル及びランダムによる予測の F1 値

ルの当てはまりの悪さ（予測精度）を評価するための指標である．残差逸脱度は，説明変数をモデルから取り除くことにより，どの程度当てはまりの悪いモデルになったかを示す．つまり，残差逸脱度が大きい程，モデル構築に寄与していることを表している．

ロジスティック回帰分析における各メトリクスの逸脱度を表 15 に示す．1 日以内及び 1 週間以内に修正されるか否かの予測では，不具合割当パターン（Pattern）が最もモデル構築に寄与していた．1ヶ月以内に修正されるか否かの予測では，コンポーネント（Component）が最もモデル構築に寄与しており，次に不具合割当パターン（Pattern）がモデル構築に寄与していた．ランダムフォレスト法における各メトリクスの精度減少値を表 16 に示す．1 日以内に修正されるか否かの予測では，コンポーネント（Component）が最もモデル構築に寄与しており，次に不具合割当パターン（Pattern）が寄与していた．1 週間以内及び 1ヶ月以内に修正されるか否かの予測では，コンポーネント（Component），修正者（Resolver），不具合割当パターン（Pattern）の順でモデル構築に寄与していた．これより，不具合割当パターンは指定期間内に不具合修正が完了するか否かの予測に有用であることが分かった．

表 15 ロジスティック回帰分析における各変数の重要度

メトリクス	変数名	残差逸脱度		
		1 日	1 週間	1ヶ月
不具合票	Component	263.06	177.64	132.95
	Priority	9.22	5.98	1.41
	Severity	1.38	3.70	3.88
	Milestone	6.31	5.04	4.96
	DescriptionWords	0.42	1.67	2.54
	CommentsCount	3.39	11.84	16.32
	CommentsWords	4.56	1.52	1.17
	AttachmentsCount	3.84	0.04	0.99
	DependsOnCount	6.26	2.72	0.67
	BlocksCount	0.70	0.61	1.40
	CCCount	12.08	8.33	5.04
時間	AssignTime	4.71	11.54	11.48
	AssignedMonth	9.87	9.08	22.23
	AssignedDay	0.61	1.90	0.05
	AssignedWeekEnd	0.10	0.09	1.75
人	Reporter	7.23	14.63	22.22
	Assignor	8.60	7.60	9.65
	Resolver	76.26	72.27	53.10
	Pattern	154.49	123.81	89.50

表 16 ランダムフォレスト法における各変数の重要度

メトリクス	変数名	精度減少値		
		1 日	1 週間	1ヶ月
不具合票	Component	129.97	83.67	62.15
	Priority	4.22	3.26	1.96
	Severity	6.11	8.57	7.87
	Milestone	10.70	11.03	13.45
	DescriptionWords	17.77	13.47	14.86
	CommentsCount	23.65	12.45	13.88
	CommentsWords	21.57	10.59	12.37
	AttachmentsCount	11.47	10.35	10.92
	DependsOnCount	1.93	3.34	-1.37
	BlocksCount	3.41	1.94	5.60
	CCCount	17.38	11.62	19.93
時間	AssignTime	27.24	24.95	30.08
	AssignedMonth	20.05	20.74	23.71
	AssignedDay	14.20	11.81	10.11
	AssignedWeekEnd	0.38	3.08	5.34
人	Reporter	45.79	23.36	22.28
	Assignor	59.48	41.74	43.72
	Resolver	81.94	67.89	59.29
	Pattern	83.51	63.72	58.29

7. 考察

本章では，6 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．7.1 節では不具合修正時間予測に不具合割当パターンが与える効果について考察を行い，7.2 節では不具合修正時間予測に用いた 2 種類の予測方法について考察を行う．7.3 節では，本研究の制約について述べる．

7.1 不具合割当パターンの効果

不具合修正が完了するまでの日数の予測において，不具合割当パターンを説明変数に用いたことによる各予測モデルの予測精度の向上率を表 17 に示す．表 17 より，不具合割当パターンを説明変数に用いることで各予測モデルの予測精度は向上することが分かった．これは，各不具合割当パターンにおける修正時間にそれぞれ有意な差があることから，修正時間を予測する上で不具合割当パターンが有用な説明変数となっているためである．

指定期間内に不具合修正が完了するか否かの予測において，不具合割当パターンを説明変数に用いたことによる各予測モデルの予測精度の向上率を表 18 に示す．1 日以内の予測において，不具合割当パターンを説明変数に用いることにより，各予測モデルの予測精度は向上することが分かった．特に再現率は 10 %から 23 %向上していた．これより，分担解決型を除く不具合割当パターンにおいて，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均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，不具合割当パターンが有用であったと考えられる．一方，1 週間及び 1ヶ月以内の予測において，不具合割当パターンを説明変数に用いても，各予測モデルの予測精度は向上しなかった．これより，不具合割当パターンの内，自己解決型，修正委託型，修正受託型では 1 週間経過した段階で大多数の不具合が修正されるため，1 週間及び 1ヶ月以内に修正されるか否かの判別基準として，不具合割当パターンが有用でなかったと考えられる．

先行研究において，不具合割当パターンは 4 種類（自己解決型，修正委託型，修正受託型，分担解決型）に分類されている．しかし，分担解決型には報告者，

表 17 重回帰分析及びランダムフォレスト法の予測精度の向上率

予測モデル名	決定係数	自由度調節済決定係数
重回帰分析	18.07 %	26.28 %
ランダムフォレスト法	12.96 %	14.51 %

管理者，修正者が全て異なる場合だけでなく，報告者と修正者が同じで管理者だけが異なる場合も含まれている．報告者と修正者が同一の人物である場合，自己解決型と同様に修正者は不具合の位置と原因を理解しているため，不具合修正時間は短くなると考えられる．そのため，分担解決型をさらに 2 種類に分類することで，短期間の予測における予測精度を向上させることができると考えられる．

分担解決型で修正された不具合の過半数は，修正に約 1 週間以上掛かっているが，1 日以内に修正されている不具合も多く存在する．先行研究では，不具合の修正時間が短い要因を明らかにするために，不具合管理システムのコメント数を分析している．分析の結果，コメント数が多い不具合ほど修正時間が短い傾向にあることが明らかにされた．しかし，議論の内容に関しては分析していないため，どのような議論が行われることで修正時間の短縮化に繋がっているかは明らかにされていない．そこで，本論文では議論の内容に関して実験的に分析を行った．分析の結果，修正時間の短縮化に繋がる議論と繋がらない議論が存在することが分かった．修正時間の短縮化に繋がる議論の一例として，不具合の再現方法に関する議論が挙げられる．また，修正時間の短縮化に繋がらない議論の一例として，不具合票の情報の変更に関する議論が挙げられる．今後，自然言語処理により議論の内容を分類し，予測モデル構築に用いることで予測精度の向上が期待される．

7.2 不具合修正時間の予測方法

重回帰分析及びランダムフォレスト法を用いて不具合修正が完了するまでの日数の予測を行ったが，不具合修正時間の予測に有効な予測モデルを構築することはできなかった．そこで，不具合修正時間と各説明変数の関係性を調べるために，不具合修正時間と各説明変数の相関係数を求めた．結果，相関係数は最大でも

表 18 ロジスティック回帰分析及びランダムフォレスト法の予測精度の向上率

予測モデル名	予測期間	適合率	再現率	F1 値
ロジスティック回帰分析	1 日	-1.81 %	23.42 %	14.35 %
	1 週間	2.79 %	-4.16 %	-0.47 %
	1ヶ月	0.37 %	-0.40 %	0.03 %
ランダムフォレスト法	1 日	5.47 %	10.59 %	8.54 %
	1 週間	0.62 %	0.44 %	0.54 %
	1ヶ月	0.40 %	-0.58 %	-0.04 %

-0.292 であり，大多数は無相関であった．つまり，不具合修正時間はコンポーネントやコメント数といった不具合票の情報や，修正者が決定した日時といった情報からは予測することは困難であると言える．以下，不具合修正が完了するまでの日数の予測モデルの実用性について述べる．

重回帰分析及びランダムフォレスト法で予測した修正時間と実際の修正時間の散布図を図 13，図 14 に示す．重回帰分析を行った場合，大多数の不具合は 6 時間から 4 日以内に修正されると予測した．約 1 日で修正されると予測した不具合が最も多かったが，修正時間の実測値はおよそ 1 分から 1000 日の間で幅広く分散していた．約 6 時間又は 4 日で修正されると予測した不具合も，同様である．ランダムフォレスト法で不具合修正時間を予測した場合も，重回帰分析による予測と同様に修正時間の予測値に対して，修正時間の実測値はおよそ 1 分から 1000 日の間で幅広く分散していた．これは，予測した修正時間の違いに関わらず，実際の修正時間は予測と大きく異なることを意味している．そのため，不具合修正が完了するまでの日数の予測モデルは，予測モデル間の予測精度に若干の違いがあるものの，予測値の誤差が極めて大きいため，実用性は低いと言える．

1 日以内に修正されるか否かの予測では，ロジスティック回帰分析に比べ，ランダムフォレスト法の方が予測精度（F1 値）は高かった．不具合の修正時間の多くは 1 日以内で修正されているが，中には修正に 2 年以上掛かっている不具合も存在する．ロジスティック回帰分析は，このようなケースに対しても過度に適合しようとしたため，予測精度が低下したと考えられる．以下，指定期間内に不具

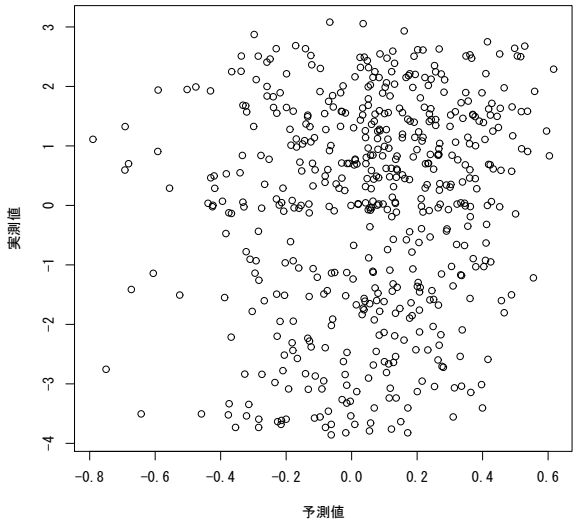

図 13 重回帰分析の予測結果の分布

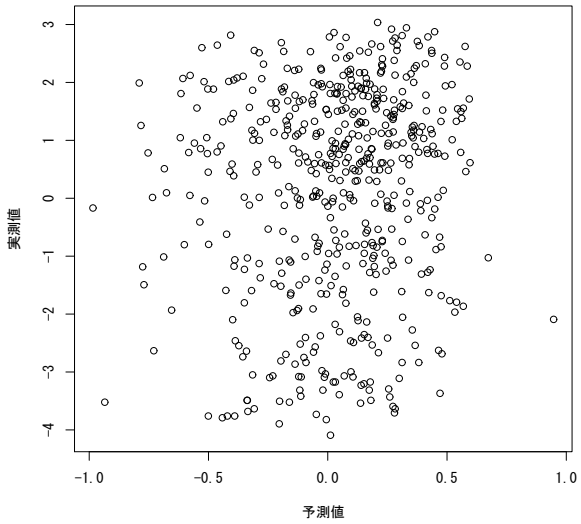

図 14 ランダムフォレスト法の予測結果の分布

合修正が完了するか否かの予測モデルの実用性について述べる．

1 日以内に修正されるか否かの予測における F1 値は，3 つの予測期間の中で最も低かった．1 日以内に修正される不具合のうち，1 分以内に修正されている不具合の割合は修正委託型で約 50 %，修正受託型で約 20 %も存在するが，修正時間があまりにも短いため，実際はあらかじめ修正作業を行っていたと考えられる．1 日以内に修正されると予測した不具合に占める修正時間が極端に短い不具合の割合が高いため，実際に現場において 1 日以内に修正されるか否かを予測する必要性は低いと考えられる．

1 週間以内に修正されるか否かの予測では，全不具合の約 70 %を 1 週間以内に修正されると予測しており，実際に 1 週間以内に修正された不具合の約 78 %を予測することができていた．実際の現場において，不具合修正の作業工程は週単位で計画すると考えられるため，1 週間以内に修正されるか否かの予測モデルの実用性は高いと考えられる．

1ヶ月以内に修正されるか否かの予測における F1 値は，3 つの予測期間の中で最も高かった．これは，1ヶ月以内に修正される不具合が全体の約 75 %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，実際は全不具合の約 96 %を 1ヶ月以内に修正されると予測しており，実際に 1ヶ月以内に修正される不具合の割合に比べ高過ぎるため，実際に現場における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，不具合修正が完了するまでの日数の予測方法よりも，指定期間内に不具合修正が完了するか否かの予測方法が不具合修正時間の予測において有用であると言える．特に，1 週間以内に修正されるか否かの予測モデルは実際の現場においても実用性が高いと考えられる．

7.3 制約

本論文では，大規模な OSS プロジェクトの 1 つである Eclipse において，不具合修正時間の予測に不具合割当パターンに基づく予測モデルが有用であることを示した．しかし，予測モデルの一般性については明らかにしていない．OSS プロ

ジェクトによって規模や開発形態は大きく異なるため，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．

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登録された時間を基に計算しているため，実際に開発者が不具合の修正を始めてから修正が完了するまでの時間とは異なる．

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．ただし，修正者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繋がる可能性がある．

8. 関連研究

本章では，関連研究について説明し，本論文との違いを述べる．

8.1 不具合修正プロセスにおける人的要因に関する研究

報告者と管理者の関係

不具合修正プロセスにおける不具合票の作成方法に関する研究が行われている [22, 24]．不具合票は報告者が不具合管理システムに不具合に関する情報を登録することにより作成され，管理者は不具合票を基に不具合の位置と原因を特定し，修正者に割り当てる．ここで，不具合票に登録されている情報が不足している場合，管理者は不具合を特定することが困難であるため，不具合修正時間の長期化を招くと言われている [43]．

このような問題を解決するために，開発者に対してアンケート調査を行い，開発者が必要とする情報を明らかにして，不具合票の記述方法を定義する研究が行われている [24]．さらに，Just ら [23] は上記のアンケート結果を基に，7 つの機能を現行の不具合管理システムに追加すべきであると提案している．

管理者と修正者の関係

不具合報告が行われた場合，管理者はまず不具合の内容を理解し，重複する不具合が報告されていないかを確認した後に，適任な開発者に不具合を割り当てる．しかし，大量の不具合が日々報告されている現状では，全ての不具合に対してこれらの作業を適切に実施していくことは困難である [44]．そのため，不具合を修正するために十分な知識とスキルを持たない開発者に割り当ててしまい，不具合再割当の問題を引き起こす一因となっている．

開発者の不具合割当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 [8, 12, 45]．例えば，Jeong ら [12] は，不具合修正プロセスに参加する開発者間の再割当関係をモデル化し，修正者の推薦モデルを構築している．

修正者間の関係

OSS 開発は世界中に点在する開発者によって共同で進められている．これらの開発者同士で情報交換を行う場合，主に電子メールや掲示板を用いて行われている [43, 46]．しかし，このような環境では時間のずれが生じるため，円滑に情報交換を行うことが困難であると言える．これにより，開発の遅延やソフトウェアの品質低下に繋がるとされている [47, 48]．

開発者間の情報交換の改善を目的として，時間のずれによる影響を調査する研究が行われている [49, 50, 51, 52]．例えば，亀井ら [51] は時間のずれが発生する実態を解明し，グローバル環境下における電子メールを用いた情報交換の所要時間短縮化のための指針を示している．

本研究との違い

先行研究の多くは，不具合修正プロセスにおける不具合修正までの参加者のうち，2 者間の関係が不具合修正プロセスに与える影響を調査したものである．一方，本研究で用いる不具合割当パターンは，全ての参加者間の関係が不具合修正プロセスに与える影響を調査したものである．

8.2 不具合修正時間の予測に関する研究

従来，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，不具合の修正時間を予測する研究が行われてきた [22, 29, 30, 31, 32]．例えば，Weiss ら [29] は不具合報告時に記録されるテキスト情報が類似する不具合の情報を用いて，JBoss プロジェクトを対象に不具合報告時から修正完了時までの平均時間を予測している．また，Hewett ら [31] は不具合報告時に記録されるコンポーネントや優先度等を用いて予測モデルを構築し，修正者決定時から検証完了時までの時間を予測している．

近年では，新たに人に関連するメトリクスを用いることで，不具合修正時間予測モデルの予測精度向上を図っている [38, 39, 53, 54, 55]．例えば，Bhattacharya ら [55] は 5 つの大規模な OSS プロジェクト（Eclipse，Mozilla Firefox Thunderbird

Seamonkey，Chrome）を対象に不具合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 [38] は 2 つの大規模 OSS プロジェクト（Mozilla，Eclipse）を対象に一定期間内（3ヵ月，1 年，3 年）に修正されるか否かの予測を行っている．実験の結果，不具合の情報と修正時間の関係は，プロジェクト間で異なり，同じプロジェクトでも時とともに変化することを明らかにした．

従来研究の多くは不具合修正時間の予測期間を不具合報告時から不具合修正完了時まで，又は予測実施日から不具合修正完了時までとしている一方で，本論文は修正者決定時から不具合修正完了時までを予測期間としている点が最も異なる．さらに，本論文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に予測モデルを構築している．

9. おわりに

本論文では，個々の不具合の修正完了期間を見極めるOSSプロジェクトの管理者を支援するために，不具合割当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．Eclipse Platformプロジェクトを対象に実験を行った結果，以下に示す知見が得られた．

- 不具合修正が完了するまでの日数の予測では，有効なモデルを構築することができないことが分かった．一方，指定期間内に不具合修正が完了するか否かの予測では，有効なモデルを構築することができることが分かった．特に1週間以内に不具合修正が完了するか否かの予測において，実用性の高い予測モデルを構築することが可能である．
- 不具合割当パターンは，1日以内に不具合修正が完了するか否かの予測において，予測精度の向上に寄与することが分かった．しかし，1週間及び1ヶ月以内に修正されるか否かの予測において，予測精度の向上に寄与しないことが分かった．不具合割当パターンの分担解決型を分類し，不具合修正プロセスにおける参加者間の議論内容毎に区別した予測モデルを構築することで，予測精度の向上に繋がると考えられる．

本論文で得られた知見を用いることで，管理者は事前に不具合の修正時間を知ることが可能になると考えられる．

今後，本研究の知見が効率的な不具合割当の実現へのきっかけとなることを期待する．

謝辞

本研究を進めるにあたり，多くの方々のご指導，ご支援を頂きました．この場を借りて感謝の意を表します．

奈良先端科学技術大学院大学 情報科学研究科 松本 健一 教授には，研究に対する直接的な指導に限らず，研究に取り組む上での心構えから研究者としての在り方まで，様々な面でご指導，ご助力を賜わりました．私が本研究を成し遂げることができたのは，先生の研究に対する深いご理解と暖かいご助言に励まされたおかげです．特に初めて助成金に応募した際には，的確なアドバイスを頂きました．心より感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 安本 慶一 教授には，本研究を進めるにあたり，広い視野からのご指導，ご助言を賜わりました．本研究の発表においても，的確かつ適切なご意見，ご指摘を頂きました．心より感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 門田 暁人 准教授には，本研究に対して多くの建設的なアドバイスを賜り，研究の進め方等的確かつ丁寧なご指導を頂きました．研究以外の様々な活動の場においてもお世話になりました．心より感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 大平 雅雄 助教には，本研究の全過程において，ありとあらゆる面で熱心なご指導，多大なご支援を賜わりました．先生のご協力がなければ私の研究は成しえませんでした．研究を進めるにあたり，ご心配，ご迷惑をお掛けしたにもかかわらず，最後まで我慢強く接して頂いたことに深く感謝致します．また，研究以外の場でも大変優しく親身になって接して頂いたこと，将来について何度もご相談にのって頂いたことに心より感謝申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 角田 雅照 特任 助教には，本研究を進めるにあたり，多くのご助言，貴重なご指摘を頂きました．特に，統計分野に関して丁寧かつ熱心なご指導を頂きました．日常生活においても心優しく接して頂きました．心より感謝申し上げます．

奈良先端科学技術大学院大学 ソフトウェア工学研究室 乾 純子 氏には，国際

会議の出張手続き等多くの事務処理に対し，多大なご助力を頂きました．心より感謝申し上げます．

奈良先端科学技術大学院大学 ソフトウェア工学研究室 伊原 彰紀 氏には，研究の進め方から，論文執筆，プレゼンテーション作法まで様々な面からご指導を頂きました．氏のご協力なしには研究を完遂することはできませんでした．また，最後まで諦めることなく研究を進めることができたのも，氏のおかげです．心より感謝致します．

研究室の卒業生である株式会社リクルートの高井 雄治 氏には，日頃より研究や日常生活に関してご相談に乗って頂きました．心より感謝致します．

奈良先端科学技術大学院大学 ソフトウェア工学研究室ならびにソフトウェア設計学研究室の皆様には，日頃より多大なご協力とご助言を頂きました．お陰さまで非常に楽しく充実した二年間を送ることができました．心より感謝致します．

最後に，大学院において研究を進めるにあたり，経済的にも精神的にも支えてくれた家族に深く感謝致します．

参考文献

[1] C. Dibona, D. Cooper, and M. Stone, Open sources 2.0, O'Reilly Media, Gravenstein Highway North, SE, Oct. 2005.

[2] S. Weber, The Success of Open Source, Harvard University Press, Cambridge, MA, Apr. 2004.

[3] Open Source Initiative, "http://www.opensource.org/docs/osd".

[4] J. Sandred, Managing open source projects: A wiley tech brief, John Wiley & Sons, Hoboken, NJ, 2001.

[5] E.S. Raymond, The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary, O'Reilly Media, Gravenstein Highway North, SE, Oct. 1999.

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

[7] A. Zeller, Why programs fail: A guide to systematic debugging, Morgan Kaufmann, Waltham, MA, Oct. 2005.

[8] J. Anvik, L. Hiew, and G.C. Murphy, "Who should fix this bug?," Proceedings of the 28th International Conference on Software Engineering (ICSE '06), pp.361–370, 2006.

[9] G. Canfora and L. Cerulo, "Supporting change request assignment in open source development," Proceedings of the 21st ACM Symposium on Applied Computing (SAC'06), pp.1767–1772, 2006.

[10] Z. Lin, F. Shu, Y. Yang, C. Hu, and Q. Wang, "An empirical study on bug assignment automation using chinese bug data," Proceedings of the 3rd

International Symposium on Empirical Software Engineering and Measurement (ESEM'09), pp.451–455, 2009.

[11] S. Rastkar, G.C. Murphy, and G. Murray, "Summarizing software artifacts: A case study of bug reports," Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10), pp.505–514, 2010.

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

[13] E. Shihab, A. Ihara, Y. Kamei, W.M. Ibrahim, M. Ohira, B. Adams, A.E. Hassan, and K.-i. Matsumoto, "Predicting re-opened bugs: A case study on the eclipse project," Proceedings of the 17th Working Conference on Reverse Engineering (WCRE'10), pp.249–258, 2010.

[14] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," Proceedings of the 2010 IEEE International Conference on Software Maintenance (ICSM'10), pp.1–10, 2010.

[15] P.J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, ""not my bug" and other reasons for software bug report reassignments," Proceedings of the 2011 ACM Conference on Computer Supported Cooperative Work (CSCW'11), pp.395–404, 2011.

[16] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," Proceedings of the 30th International Conference on Software Engineering (ICSE'08), pp.461–470, 2008.

[17] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful ... really?," Proceedings of the 24th IEEE In-

ternational Conference on Software Maintenance (ICSM '08), pp.337–345, 2008.

[18] A. Schröter, N. Bettenburg, and R. Premraj, "Do stacktraces help developers fix bugs?," Proceedings of the 8th International Working Conference on Mining Software Repositories (MSR'10), pp.118–122, 2010.

[19] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10), pp.45–54, 2010.

[20] A.J. Ko and P.K. Chilana, "How power users help and hinder open bug reporting," Proceedings of the 28th International Conference on Human Factors in Computing Systems (CHI'10), pp.1665–1674, 2010.

[21] A.J. Ko, B.A. Myers, and D.H. Chau, "A linguistic analysis of how people describe software problems," Proceedings of the Visual Languages and Human-Centric Computing (VL/HCC'06), pp.127–134, 2006.

[22] P. Hooimeijer and W. Weime, "Modeling bug report quality," Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07), pp.34–43, 2007.

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

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

[25] D.M. German, "Mining cvs repositories, the softchange experience," Proceedings of the 1st International Workshop on Mining Software Repositories (MSR'04), pp.17–21, 2004.

[26] B. Shibuya and T. Tamai, "Understanding the process of participating in open source communities," Proceedings of the 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development (FLOSS'09), pp.1–6, 2009.

[27] E. Shihab, N. Bettenburg, B. Adams, and A.E. Hassan, "On the central role of mailing lists in open source projects: An exploratory study," Proceedings of of the 3rd International Workshop on Knowledge Collaboration in Software Development (KCSD'09), pp.34–48, 2009.

[28] R. Tang, A.E. Hassan, and Y. Zou, "A case study on the impact of global participation on mailing lists communications of open source projects," Proceedings of the 3rd International Workshop on Knowledge Collaboration in Software Development (KCSD'09), pp.63–76, 2009.

[29] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?," Proceedings of the 4th International Workshop on Mining Software Repositories (MSR'07), pp.1–8, 2007.

[30] L.D. Panjer, "Predicting eclipse bug lifetimes," Proceedings of the 4th International Workshop on Mining Software Repositories (MSR'07), p.29, 2007.

[31] R. Hewett and P. Kijsanayothin, "On modeling software defect repair time," Empirical Software Engineering (ESE'09), vol.14, no.2, pp.165–186, 2009.

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

[33] 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,” 奈良先端科学技術大学院大学，修士論文，NAIST-IS-MT0951021，2011.

[34] 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.

[35] A. Mockus, R.T. Fielding, and J.D. Herbsleb, “Two case studies of open source software development: Apache and moxilla,” ACM Transactions on Software Engineering and Methodology (TOSEM’02), vol.11, no.3, pp.309–346, 2002.

[36] I. Herraiz, D.M. German, J.M. Gonzales-Barahona, and G. Robles, “Towards a simplification of the bug report form in eclipse,” Proceedings of the 5th International Working Conference on Mining Software Repositories (MSR’08), pp.145–148, 2008.

[37] P. Anbalagan and M. Vouk, ““days of the week” effect in predicting the time taken to fix defects,” Proceedings of the 2nd International Workshop on Defects in Large Software Systems (DEFECTS’09), pp.29–30, 2009.

[38] L. Marks, Y. Zou, and A.E. Hassan, “Studying the fix-time for bugs in large open source projects,” Proceedings of the 7th International Conference on Predictive Models in Software Engineering (PROMISE’11), no.11, 2011.

[39] N. Duc Anh, D.S. Cruzes, R. Conradi, and C. Ayal, “Empirical validation of human factors in predicting issue lead time in open source projects,” Proceedings of the 7th International Conference on Predictive Models in Software Engineering (PROMISE’11), no.13, 2011.

[40] L. Breiman, “Random forests,” Machine Learning, vol.45, pp.5–32, 2001.

[41] 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，Jan. 2001.

[42] J.L. Herlocker, J.A. Konstan, L.G. Terveen, and J.T. Riedl, “Evaluating collaborative filtering recommender systems,” ACM Transactions on Information Systems (TOIS’04), vol.22, no.1, pp.5–53, 2004.

[43] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, “Information needs in bug reports: Improving cooperation between developers and users,” Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work (CSCW’10), pp.301–310, 2010.

[44] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler, “An empirical study of operating systems errors,” Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP’09), pp.73–88, 2001.

[45] D. Cubranic and G.C. Murphy, “Automatic bug triage using text categorization,” Proceedings of the 16th International Conference on Software Engineering and Knowledge Engineering (SEKE’04), pp.92–97, 2004.

[46] K. Fogel, Producing open source software: How to run a successful free software project, O’Reilly Media, Gravenstein Highway North, SE, Oct. 2005.

[47] J.D. Herbsleb, T.A. Mockus, Audris Finholt, and R.E. Grinter, “An empirical study of global software development: Distance and speed,” Proceedings of the 23rd International Conference on Software Engineering (ICSE’01), pp.81–90, 2001.

[48] M. Cataldo and S. Nambiar, “On the relationship between process maturity and geographic distribution: An empirical analysis of their impact on software quality,” Proceedings of the the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE’09), pp.101–110, 2009.

[49] E. Carmel, Global Software Teams: Colloborating Across Borders and Time Zones, Prentice Hall, Upper Saddle River, NJ, Jan. 1999.

[50] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, "Does distributed development affect software quality? an empirical case study of windows vista," Proceedings of the 31st International Conference on Software Engineering (ICSE'09), pp.518–528, 2009.

[51] 亀井靖高，大平雅雄，伊原彰紀，小山貴和子，柗本真佑，松本健一，鵜林尚靖，"グローバル環境下におけるOSS開発者の情報交換に対する時差の影響，" 情報社会学会誌，vol.6，no.2，pp.13–30，2011.

[52] J. Anvik and G.C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," ACM Transactions on Software Engineering and Methodology (TOSEM'11), vol.20, no.3, p.Article 10 (35 pages), 2011.

[53] P. Anbatagan and M. Vouk, "On predicting the time taken to correct bug reports in open source projects," Proceedings of the 2nd international Workshop on Recommendation Systems for Software Engineering (RSSE'10), pp.523–526, 2009.

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

[55] P. Bhattacharya and I. Neamtiu, "Bug-fix time prediction models: Can we do better?," Proceedings of the 8th International Workshop on Mining Software Repositories (MSR'11), pp.207–210, 2011.