Vol.2016-SE-191 No.4
2016/3/14

High Impact Bugが不具合修正時間予測に与える影響の評価：OSSプロジェクトを対象としたケーススタディ

吉行　勇人[1,a)]　大平　雅雄[1,b)]

概要：本論文では，不具合修正時間予測手法の予測精度向上を目的として，High Impact Bug を考慮した不具合修正時間予測手法を提案する．近年，大規模な OSS プロジェクトでは，日々多くの不具合が報告されている．このような現状では，報告された全ての不具合を次のバージョンリリースまでに修正することは困難であるため，プロジェクトの管理者は，次のバージョンリリースまでにどの不具合を優先的に修正すべきか，という意思決定を行う必要がある．その意思決定の指標として，個々の不具合の修正にかかる時間がどのくらいかを見積もることが求められており，OSS 開発における不具合修正時間を予測する研究が盛んに行われている．しかし，先行研究の予測モデルは，全ての不具合がユーザや開発者に与える影響や修正難易度は同一であると仮定して構築されている．実際には，不具合の種類によって影響範囲や修正難易度は異なり，不具合修正時間にも違いが生じる可能性がある．そこで，本研究では不具合の分類方法のひとつとして注目されている High Impact Bug に着目し，個々の不具合が High Impact Bug であるかどうかを考慮した不具合修正時間予測手法を構築した．4 件の OSS プロジェクトを対象とした実験を行った結果，High Impact Bug が不具合修正時間予測モデルの予測精度向上に寄与することが確かめられた．

キーワード：OSS 開発，不具合修正時間予測，High Impact Bug

1. はじめに

オープンソースソフトウェア (OSS) は，多くのソフトウェア開発プロジェクトにおいて，品質確保や開発期間の短縮を目的として広く利用されている．OSS が日常的に利用され，その重要性が増す一方で，大規模な OSS プロジェクトには，日々多数の不具合が報告されている．例えば，Mozilla プロジェクトには，一日に数百件以上の不具合が報告される場合がある [9]．多数の不具合が報告されている状況では，開発者の不足やリリースまでの期間が限られていること等の理由から，全ての不具合を次のリリースまでに修正することは現実的でない．そのため，プロジェクトの管理者は，次のリリースまでにどの不具合を優先的に修正すべきか，という意思決定を行う必要がある．その意思決定の指標として，個々の不具合の修正にかかる時間がどのくらいかを見積もることが重要である．

このような背景から，近年 OSS 開発における不具合修正時間を予測する手法を構築する研究が盛んに行われている [8,13,14,19]．これらの先行研究の多くは，不具合報告に含まれる情報（不具合が発生したコンポーネント情報や，不具合に設定された重要度等）から，修正時間を予測する手法を提案している．また，不具合修正時間は人的要因に依る部分も大きいため，開発者個人の能力や開発者間の関係に着目した研究も多く行われている [14,19]．

一方で，先行研究のほとんどは，不具合の種類を十分に考慮しておらず，全ての不具合がユーザや開発者に与える影響や，修正難易度等は同一であると仮定して予測モデルを構築している．不具合には，タイプミスを修正するような軽微な不具合から，セキュリティ上の脅威となりうる重大な不具合まで多様に存在し，修正難易度および修正にかかる時間は異なると考えられる．そのため，より精度の高い修正時間予測を行うためには，個々の不具合の種類を考慮した分類が必要であると考えられる．

そこで，本研究では，不具合の分類方法として近年注目されている High Impact Bug [12] に着目する．High Impact Bug は先行研究において 6 種類提案されており，それぞれの High Impact Bug は修正時間に特徴があることが報告されている [5,11,16,18]．よって，High Impact Bug による分類を考慮して不具合修正時間予測を行うことで，予測精度の向上が期待できる．

本稿では，High Impact Bug が不具合修正時間予測の予

[1] 和歌山大学
Wakayama Uniersity
[a)] s151054@center.wakayama-u.ac.jp
[b)] masao@sys.wakayama-u.ac.jp

測精度に与える影響を評価することを目的として，4 件の OSS プロジェクトの不具合を対象としてケーススタディを行った．ケーススタディでは，個々の不具合が指定期間（例えば，1 日や 1 週間）内に修正完了するか否かを予測する予測モデルを構築し，予測精度の評価を行った．ケーススタディの結果，以下の知見が得られた．

- High Impact Bug を考慮することによって，不具合修正時間予測モデルの予測精度が 0.1%から 11.9%向上する
- 先行研究で提案されている 6 種類の High Impact Bug のうち，どの High Impact Bug が予測精度向上に寄与するかは，プロジェクトおよび指定期間によって異なる

本稿の構成は以下の通りである．続く第 2 章では関連研究として，本研究の着眼点である High Impact Bug と不具合修正時間予測について述べる．第 3 章ではケーススタディの設定について述べる．第 4 章では，ケーススタディの結果について述べる．第 5 章ではケーススタディの結果に対する議論とその妥当性について述べる．最後に，第 6 章で今後の課題について述べ，本稿をまとめる．

2. 関連研究

2.1 High Impact Bug

近年，ユーザや開発者にとって重大な影響を与える不具合を High Impact Bug と呼び，多くの研究が行われている [5,6,11,15,16,18]．これまでに，6 種類の High Impact Bug が提案されており，以下に紹介する．

Surprise Bug [15] Surprise Bug は，予期せぬファイルに予期せぬタイミングで発見された不具合と定義されている．Surprise Bug が発生すると，修正計画の変更が必要となる等，開発者に重大な影響を与える．

Breakage Bug [15] Breakage Bug は，以前は利用可能だった機能が，不具合修正や新機能の追加のための変更によって利用不可能になる不具合と定義されている．Breakage Bug が発生すると，ユーザが日常業務で利用していた機能が利用不可能になり，ユーザ満足度等に影響を与える．

Dormant Bug [5] Dormant Bug は，あるバージョンの開発中に埋め込まれたが，そのバージョンリリース後にすぐには発見されず，さらに次のリリースになってから初めて発見された不具合と定義されている．リリース後に発見された不具合は，そのリリースの品質を測る指標として用いられているが，Dormant Bug は本来不具合が埋め込まれたリリースとは別のリリースの不具合として判断されるため，各リリースの品質評価を誤る可能性がある．

Security Bug [6] Security Bug は，不正アクセス等のセキュリティ上の脅威となりうる不具合と定義されている．Security Bug は，ユーザの個人情報流出等，重大な問題を引き起こす原因となりうるため，多くのプロジェクトで重要視されている．

Performance Bug [11] Performance Bug は，応答時間やメモリ使用量の増大等のパフォーマンス低下を引き起こす不具合と定義されている．ソフトウェアのパフォーマンスは非機能要求であり，ユーザによってどれだけパフォーマンスが低下したときに Performance Bug であると断定するかが曖昧であるため，判断が難しい．

Blocking Bug [16] Blocking Bug は，他の不具合の修正を阻害する不具合と定義されている．Blocking Bug が発生すると，修正を妨げられている他の不具合を早く修正しなければならないときであっても，先に Blocking Bug を修正する必要があり，修正時間の長期化の原因となる．

Ohira ら [12] は，4 件の OSS プロジェクト（Apache Ambari, Apache Camel, Apache Derby, Apache Wicket）に報告された不具合を対象に，6 種類の High Impact Bug に分類し，個々の不具合に各 High Impact Bug であるか否かをラベル付けしたデータセットを公開している．

先行研究で提案された 6 種類の High Impact Bug 以外にも，今後新たに定義される可能性があるが，本研究では既存の 6 種類の High Impact Bug のみを対象とする．

2.2 不具合修正時間予測

近年，不具合修正時間予測手法の研究が盛んに行われている [1,2,4,10,19]．先行研究では，個々の不具合が修正されるまでの時間を連続値で予測する手法 [1,4] や，指定期間内に修正完了するか否かを予測する手法 [2,10,19] が構築されている．修正時間を連続値で予測する手法 [1,4] では，回帰分析等によるモデルを構築しているが，いずれの先行研究もおおむね決定係数 0.4 程度で，有用なモデルを構築できているとは言えない．指定期間内に修正完了するか否かを予測する手法では，従来は 3 ヶ月や 1 年以内に修正完了するか否か，といった粒度での予測を行っていた [10]．しかし，意思決定の指標として利用するには，より細かい粒度での予測が求められており，近年では 1 日や 1 週間といった期間を対象に予測する手法が提案されている [19]．また，指定期間を全不具合の修正時間の中央値とすることで，不具合の修正時間が全体と比べて長いか短いかを予測する場合もある [2]．先行研究の多くは，不具合の報告時に登録される，不具合の発生したコンポーネントや不具合に設定された重要度，不具合の症状を記述する Description 等の不具合自体の特徴を表す情報をもとに予測モデルを構築している [17]．例えば，Weiss らは Summary と Description の文字列の類似度に基づいて過去の不具合報告から不具合修正時間の予測を行っている [17]．また，

表 1　各プロジェクトの情報

プロジェクト	Ambari	Camel	Derby	Wicket
プロジェクト開始時期	2011 年 9 月	2007 年 4 月	2004 年 9 月	2006 年 10 月
全報告数 (2015 年 2 月時点)	9,482	8,328	6,786	5,831

表 2　プロジェクトごとの不具合数および各 High Impact Bug の分布

プロジェクト	Ambari	Camel	Derby	Wicket
不具合数	821	867	852	840
はずれ値閾値（日）	3.45	12.19	327.72	34
Surprise	91(11.1%)	181(20.9%)	62(7.3%)	68(8.1%)
Breakage	8(1%)	41(4.7%)	174(20.4%)	59(7%)
Dormant	18(2.2%)	191(22%)	139(16.3%)	153(18.2%)
Security	26(3.2%)	17(2%)	59(6.9%)	7(0.8%)
Performance	34(4.1%)	60(6.9%)	63(7.4%)	50(6%)
Blocking	3(0.4%)	8(0.9%)	24(2.8%)	3(0.4%)

近年では不具合修正に関わる人的要因（例えば，誰が不具合を報告したのか，等）に着目して予測モデルを構築し，細粒度な予測を行う研究が報告されている [2,19].

2.3　不具合修正タスク割当支援

不具合修正プロセスの支援手法として，不具合修正タスクの割当支援手法が提案されている [3]．先行研究では，不具合修正に適任の開発者を推薦する手法が提案されているが，これらの研究でも，不具合の種類を十分に考慮しているとは言えない．実際のプロジェクトでは，High Impact Bug に分類されるような不具合は，プロジェクトの中心的な開発者が担当する場合が多いと考えられる．そのため，本研究で着目した High Impact Bug による分類を考慮することで，より現実に即した支援手法が構築できる可能性がある．

3.　ケーススタディ

3.1　データセット

本ケーススタディでは，Ohira ら [12] が作成したデータセットを用いる．Ohira らは，4 件の OSS プロジェクト (Apache Ambari, Apache Camel, Apache Derby, Apache Wicket) に報告された不具合を High Impact Bug に分類し，ラベル付けしている．表 1 に，4 件の OSS プロジェクトの基本的な情報を示す．

[12] では，プロジェクトごとに，プロジェクト開始時期から 2015 年 2 月までの全報告から 1,000 件ずつ無作為に抽出し，個々の不具合に対してラベル付けを行った．6 種類の High Impact Bug のうち，Surprise Bug と Dormant Bug は機械的にラベル付けすることができるが，残りの 4 つの High Impact Bug については，不具合の説明や議論の内容を目視で確認して，ラベル付けを行っている．

ただし，Ohira らのデータセットは，不具合だけでなく改善要求も含めて 1,000 件の報告をラベル付けしている．本ケーススタディでは，不具合のみで 1,000 件になるように，Ohira らと同様の方法でデータセットを作成し直した．また，不具合の中には，修正が完了しているにも関わらず，完了報告がされておらず，長期間経ってから完了状態になるものが存在する．そういった不具合は，他の不具合と比較して修正時間が長く記録されることになり，予測モデル構築時に大きな影響を与えてしまう可能性がある．そこで，本ケーススタディでは，箱ひげ図を用いて，第三四分位値+IQR*1.5 以上離れている値（箱ひげ図における外れ値の定義の一つ）を外れ値としてデータセットから取り除いた．表 2 に，外れ値除去後のデータセットの件数および High Impact Bug の分布をプロジェクトごとに示す．表 2 より，プロジェクトによって，High Impact Bug の分布が異なることが分かる．

表 3　プロジェクトごとの指定期間内に修正完了した不具合の割合

プロジェクト	Ambari	Camel	Derby	Wicket
1 日以内	0.8624	0.6194	0.1303	0.4452
1 週間以内	1	0.9354	0.3709	0.7750
1 か月以内	1	1	0.6408	0.9845

表 4　プロジェクトごとの全不具合の修正時間（日）の中央値

プロジェクト	Ambari	Camel	Derby	Wicket
修正時間中央値	0.035	0.591	14.214	1.491

3.2　予測モデルの構築

本ケーススタディで構築する予測モデルは，指定期間内に修正完了するか否かを予測する．指定期間は 1 日，1 週間，1 ヶ月，全不具合の修正時間の中央値とする．表 3 に，プロジェクトごとに各指定期間内に修正完了した不具合の割合を示す．表 3 より，Ambari プロジェクトでは修正に 1 週間以上かかった不具合が存在しないため，指定期間 1 週間および 1 ヶ月では全ての不具合が修正完了している状態になる．同様に，Camel プロジェクトでは修正に 1 ヶ月以上かかった不具合が存在しない．そのため，Ambari プロジェクトの指定期間 1 週間と 1 ヶ月，Camel プロジェク

表 5　不具合票メトリクスの詳細

メトリクス	尺度	説明
Component	名義	不具合が発生したコンポーネント
Priority	順序	不具合の優先度
DescriptionWords	比率	Description の単語数
Reporter	名義	不具合報告者
Assignee	名義	修正担当者
Watches	比率	不具合に関する情報の変更の通知を受ける人数
CommentCount	比率	コメント数
CommentatorCount	比率	コメント投稿者数

トの指定期間 1 ヶ月については，予測モデルの構築は行わない．表 4 に，プロジェクトごとに全不具合の修正時間の中央値を示す．指定期間を全不具合の修正時間（日）の中央値とした場合，表 4 に示した日数よりも修正時間が短いか否かを予測するモデルを構築することになる．

予測モデルの説明変数には，不具合票メトリクスと High Impact Bug メトリクスの 2 種類を用いる．不具合票メトリクスは，不具合票に記録された情報から得られるメトリクスで，先行研究で，予測に有用であると報告されたメトリクスである．表 5 に，不具合票メトリクスの詳細を示す．不具合票メトリクスのうち，名義尺度である Component, Reporter, Assignee は，ダミー変数化する．本ケーススタディにおいては，モデルの単純化のために，登場頻度上位 5 つをダミー変数化した．順序尺度である Priority も，ダミー変数化する必要がある．本ケーススタディで用いた OSS プロジェクトで不具合管理システムとして用いている Jira では，優先度は 5 段階（Blocker, Critical, Major, Minor, Trivial）設定されている．多くの OSS プロジェクトでは，優先度はデフォルト値 (Jira の場合 Major) のまま設定されている不具合の割合が多い [7]．そのため，本研究では優先度のデフォルト値を除いて，デフォルト値よりも優先度が高い 不具合 (Jira の場合，Blocker と Critical) と優先度が低い不具合 (Jira の場合 Minor と Trivial) に分類して 2 つのダミー変数に変換した．

High Impact Bug メトリクスは，本研究で新たに取り入れるメトリクスである．個々の不具合が各 High Impact Bug であるか否かをメトリクスとする．

予測モデルの構築には，ランダムフォレスト法，ロジスティック回帰分析，サポートベクターマシン (SVM) を用いる．ただし，紙面の都合上，各プロジェクトおよび指定期間ごとに，最も予測精度の良かった機械学習法の結果のみを示す．

予測精度の評価は，適合率，再現率，F1 値によって行う．適合率は，予測モデルによって指定期間内に修正完了すると予測された不具合のうち，実際に指定期間内に修正完了した不具合の割合を表す．再現率は，実際に指定期間内に修正完了した不具合のうち，予測モデルによって指定期間内に修正完了すると予測できた不具合の割合を表す．F1 値は，適合率と再現率の調和平均によって算出される．適合率と再現率はトレードオフの関係にあるため，F1 値は適合率と再現率のバランスの良さを表す．各評価指標は，0 から 1 の値を取り，値が大きいほど予測精度が高いことを表す．

予測精度は，10 分割交差検証によって算出する．データセットをランダムに 10 分割し，そのうち 9 つをトレーニングデータとして予測モデルを構築し，残りの 1 つをテストデータとして予測精度を算出する．テストデータを入れ替えながら 10 回繰り返し，10 回分の予測精度の平均を最終的な予測精度とする．また，データセットの分割はランダムであり，実行するたびに結果が異なるため，10 分割交差検証を 100 回行い，予測精度の平均を最終的な予測精度とする．

3.3　リサーチクエスチョン

本稿では，High Impact Bug が予測精度に与える影響を明らかにするために，2 つのリサーチクエスチョンを設定した．

RQ1:どの High Impact Bug が予測精度向上に寄与するか？ Ohira らは，作成したデータセットにおける各 High Impact Bug の修正時間の特徴に差があることを明らかにしている．また，プロジェクトによって各 High Impact Bug の修正時間の特徴は異なる．そのため，各 High Impact Bug が予測精度に与える影響は，一様ではなく，またプロジェクトによって影響が異なることが考えられる．そこで，6 種類の High Impact Bug メトリクスのうち，どの High Impact Bug メトリクスが予測精度向上に寄与するかを確かめる．不具合票メトリクスに 6 種類の High Impact Bug メトリクスを 1 つずつ加えた予測モデルを構築し，予測精度を評価する．不具合票メトリクスのみを用いたモデル（既存モデル）と，不具合票メトリクスに 6 種類の High Impact Bug メトリクスのうち 1 つを用いたモデルを比較して，予測精度が向上すれば，その High Impact Bug メトリクスは予測精度向上に寄与したと判断する．

RQ2:High Impact Bug を考慮したことによる予測精度の変化はどの程度か？ RQ1 で明らかになった，予測精度向上に寄与する High Impact Bug メトリクスを考慮することによって，予測精度がどの程度向上したかを評価する．RQ1 で予測精度向上に寄与した High Impact Bug メトリクスが複数あれば，組み合わせて用いる．不具合票メトリクスのみを用いた既存モデルと，不具合票メトリクスに加えて RQ1 で明らかになった予測精度向上に寄与する High Impact Bug メトリクスを用いた提案モデルの予測精度を比較する．また，構築した予測モデルの有用性を評価するために，非機械学習モデルとの比較を行う．非機械学習モデルとは，指定期間内に修正完了した不具合の割合から予測を行うモデルである．例えば，1 日以内に修正完了した

表 6　プロジェクトおよび指定期間ごとに予測精度の最も高かった機械学習法

プロジェクト	Ambari	Camel	Derby	Wicket
1 日	SVM	SVM	ランダムフォレスト法	ロジスティック回帰分析
1 週間		SVM	SVM	ロジスティック回帰分析
1 か月			SVM	SVM
中央値	ランダムフォレスト法	ロジスティック回帰分析	SVM	ロジスティック回帰分析

表 7　プロジェクトおよび指定期間ごとに予測精度向上に寄与した High Impact Bug メトリクス

プロジェクト	Ambari	Camel	Derby	Wicket
1 日	Per	Dor, Per, Blo	Bre, Dor, Sec, Per	Sur, Sec
1 週間		なし	Bre	なし
1 か月			Sur, Sec	なし
中央値	Sur, Bre, Dor, Blo	Sur, Bre	Sur	Bre, Dor, Sec, Per

不具合が全体の 55%だった場合，全体の 55%が 1 日以内に修正完了すると予測すると，そのうちの 55%が実際に 1 日以内に不具合修正が完了していると考えられる．この場合，適合率，再現率および F1 値は，定義に従って計算すると全てもともとの割合である 0.55 (55%) になる．

予測精度の比較は，予測精度同士の差分をパーセントで解釈して行う．例えば，モデル A による予測精度が 0.70, モデル B による予測精度が 0.75 だった場合，モデル B はモデル A に対して 0.05 の向上であるため，予測精度が 5%向上したと評価する．

4. 結果

本章では，ケーススタディの結果を示す．本稿では，紙面の都合上 3 つの機械学習法のうち，プロジェクトおよび指定期間ごとに最も予測精度の高かったモデルの結果のみを示す．表 6 に，各プロジェクトおよび指定期間でどの機械学習法を用いたかを示す．

4.1 RQ1

表 7 に，プロジェクトおよび指定期間ごとに，予測精度向上に寄与した High Impact Bug メトリクスを示す．表 7 より，Ambari プロジェクトの指定期間 1 日では Performance Bug, 指定期間中央値では Surprise Bug, Breakage Bug, Dormant Bug, Blocking Bug が予測精度向上に寄与した．Camel プロジェクトの指定期間 1 日では Dormant Bug, Performance Bug, Blocking Bug, 指定期間中央値では，Surprise Bug, Breakage Bug が予測精度向上に寄与した．Camel プロジェクトの指定期間 1 週間では，いずれの High Impact Bug メトリクスも予測精度向上に寄与しなかった．Derby プロジェクトの指定期間 1 日では Breakage Bug, Dormant Bug, Security Bug, Performance Bug, 指定期間 1 週間では Breakage Bug, 指定期間 1 ヶ月では Surprise Bug, Security Bug, 指定期間中央値では Surprise Bug が予測精度向上に寄与した．Wicket プロジェクトの指定期間 1 日では Surprise Bug, Security Bug, 指定期間中央値では Breakage Bug, Dormant Bug, Security Bug, Performance Bug が予測精度向上に寄与した．Wicket プロジェクトの指定期間 1 週間および 1 ヶ月では，いずれの High Impact Bug メトリクスも予測精度向上に寄与しなかった．

このことから，プロジェクトおよび指定期間によってどの High Impact Bug メトリクスが予測精度向上に寄与するかが異なることが分かった．いずれの High Impact Bug メトリクスも予測精度向上に寄与しなかったプロジェクトおよび指定期間のうち，Camel プロジェクトの指定期間 1 週間および Wicket プロジェクトの指定期間 1 ヶ月については，有効な予測モデルを構築できなかったこと（詳細は RQ2 の結果から述べる）が，いずれの High Impact Bug メトリクスも予測精度向上に寄与しなかった理由であると考えられる．

> プロジェクトおよび指定期間によって，どの High Impact Bug メトリクスが予測精度向上に寄与するかが異なる

4.2 RQ2

表 8 に，Ambari プロジェクトにおける各予測モデルの予測精度および精度向上度を示す．精度向上度は，提案モデルが各モデルに対してどれだけ予測精度が向上したかを表す．表 8 より，指定期間 1 日において，提案モデルおよび既存モデルは，適合率が非機械学習モデルと一致し，再現率が 1 に近い値となった．この場合，全ての不具合に対して指定期間内に修正完了すると予測しており，有効な予測モデルが構築できなかったことを表す．指定期間中央値において，提案モデルは非機械学習モデルと比較して，適合率は 17.4%向上，再現率は 17.7%，F1 値は 17.5%向上した．既存モデルと比較して，適合率は 2%向上，再現率は 1.8%向上，F1 値は 1.9%向上した．このことから，Ambari プロジェクトでは，提案モデルは指定期間中央値において

表 8 Ambari プロジェクトにおける各モデルの予測精度および精度向上度

		予測精度			精度向上度		
	指定期間	適合率	再現率	F1 値	適合率	再現率	F1 値
提案モデル	1 日	0.862	0.997	0.925			
	1 週間	なし	なし	なし			
	1 か月	なし	なし	なし			
	中央値	0.674	0.677	0.675			
既存モデル	1 日	0.862	0.996	0.924	0%	0.1%	0.1%
	1 週間	なし	なし	なし	なし	なし	なし
	1 か月	なし	なし	なし	なし	なし	なし
	中央値	0.654	0.659	0.656	2%	1.8%	1.9%
非機械学習モデル	1 日	0.862	0.862	0.862	0%	13.5%	6.3%
	1 週間	なし	なし	なし	なし	なし	なし
	1 か月	なし	なし	なし	なし	なし	なし
	中央値	0.500	0.500	0.500	17.4%	17.7%	17.5%

表 9 Camel プロジェクトにおける各モデルの予測精度および精度向上度

		予測精度			精度向上度		
	指定期間	適合率	再現率	F1 値	適合率	再現率	F1 値
提案モデル	1 日	0.716	0.849	0.777			
	1 週間	0.935	1	0.967			
	1 か月	なし	なし	なし			
	中央値	0.650	0.630	0.640			
既存モデル	1 日	0.701	0.851	0.769	1.5%	-0.2%	0.8%
	1 週間	0.935	1	0.967	0%	0%	0%
	1 か月	なし	なし	なし	なし	なし	なし
	中央値	0.653	0.627	0.640	-0.3%	0.3%	0%
非機械学習モデル	1 日	0.619	0.619	0.619	9.7%	23%	15.8%
	1 週間	0.935	0.935	0.935	0%	6.5%	3.2%
	1 か月	なし	なし	なし	なし	なし	なし
	中央値	0.500	0.500	0.500	15%	13%	14%

表 10 Derby プロジェクトにおける各モデルの予測精度および精度向上度

		予測精度			精度向上度		
	指定期間	適合率	再現率	F1 値	適合率	再現率	F1 値
提案モデル	1 日	0.610	0.314	0.414			
	1 週間	0.656	0.611	0.633			
	1 か月	0.727	0.886	0.798			
	中央値	0.677	0.765	0.718			
既存モデル	1 日	0.491	0.273	0.351	11.9%	4.1%	6.3%
	1 週間	0.650	0.611	0.630	0.6%	0%	0.3%
	1 か月	0.724	0.885	0.797	0.3%	0.1%	0.1%
	中央値	0.676	0.755	0.713	0.1%	1%	0.5%
非機械学習モデル	1 日	0.130	0.130	0.130	48%	18.4%	28.4%
	1 週間	0.371	0.371	0.371	28.5%	24%	26.2%
	1 か月	0.641	0.641	0.641	8.6%	24.5%	15.7%
	中央値	0.500	0.500	0.500	17.7%	26.5%	21.8%

表 11 Wicket プロジェクトにおける各モデルの予測精度および精度向上度

		予測精度			精度向上度		
	指定期間	適合率	再現率	F1 値	適合率	再現率	F1 値
提案モデル	1 日	0.640	0.554	0.594			
	1 週間	0.794	0.972	0.874			
	1 か月	0.985	1	0.992			
	中央値	0.645	0.651	0.648			
既存モデル	1 日	0.636	0.549	0.589	0.4%	0.5%	0.5%
	1 週間	0.794	0.972	0.874	0%	0%	0%
	1 か月	0.985	1	0.992	0%	0%	0%
	中央値	0.648	0.644	0.646	-0.3%	0.7%	0.2%
非機械学習モデル	1 日	0.445	0.445	0.445	19.5%	10.9%	14.9%
	1 週間	0.775	0.775	0.775	1.9%	19.7%	9.9%
	1 か月	0.985	0.985	0.985	0%	1.5%	0.7%
	中央値	0.500	0.500	0.500	14.5%	15.1%	14.8%

有用であることが分かった．

表 9 に，Camel プロジェクトにおける各予測モデルの予測精度および精度向上度を示す．表 9 より，指定期間 1 日において，提案モデルは非機械学習モデルと比較して，適合率は 9.7%向上，再現率は 23%向上，F1 値は 15.8%向上した．既存モデルと比較して，適合率は 1.5%向上，再現率は 0.2%低下，F1 値は 0.8%向上した．指定期間 1 週間では，Ambari プロジェクトの指定期間 1 日と同様に，有効な予測モデルが構築できなかった．指定期間中央値において，提案モデルは非機械学習モデルと比較して，適合率は 15%向上，再現率は 13%向上，F1 値は 14%向上した．既存モデルと比較して，適合率は 0.3%低下，再現率は 0.3%向上，F1 値は変化しなかった．このことから，Camel プロジェクトでは，提案モデルによって指定期間 1 日では適合率が向上し再現率が低下することが分かった．指定期間中央値では，向上度は小さいが，適合率が低下し，再現率が向上することが分かった．

表 10 に，Derby プロジェクトにおける各予測モデルの予測精度および精度向上度を示す．表 10 より，指定期間 1 日において，提案モデルは非機械学習モデルと比較して，適合率は 48%向上，再現率は 18.4%向上，F1 値は 28.4%向上した．既存モデルと比較して，適合率は 11.9%向上，再現率は 4.1%向上，F1 値は 6.3%向上した．指定期間 1 週間において，提案モデルは非機械学習モデルと比較して，適合率は 28.5%向上，再現率は 24%向上，F1 値は 26.2%向上した．既存モデルと比較して，適合率は 0.6%向上，再現率は変化せず，F1 値は 0.3%向上した．指定期間 1 ヶ月において，提案モデルは非機械学習モデルと比較して，適合率は 8.6%向上，再現率は 24.5%向上，F1 値は 15.7%向上した．既存モデルと比較して，適合率は 0.3%向上，再現率は 0.1%向上，F1 値は 0.1%向上した．指定期間中央値において，提案モデルは非機械学習モデルと比較して，適合率は 17.7%向上，再現率は 26.5%向上，F1 値は 21.8%向上した．既存モデルと比較して，適合率は 0.1%向上，再現率は 1%向上，F1 値は 0.5%向上した．このことから，Derby プロジェクトでは，提案モデルによって，いずれの指定期間

でも予測精度が向上することが分かった．特に，指定期間1日において，適合率が約12%向上し，予測精度向上に大きく寄与していることが分かった．

表11に，Wicketプロジェクトにおける各予測モデルの予測精度および精度向上度を示す．表11より，指定期間1日において，提案モデルは非機械学習モデルと比較して，適合率は19.5%向上，再現率は10.9%向上，F1値は14.9%向上した．既存モデルと比較して，適合率は0.4%向上，再現率は0.5%向上，F1値は0.5%向上した．指定期間1週間において，提案モデルは非機械学習モデルと比較して，適合率は1.9%向上，再現率は19.7%向上，F1値は9.9%向上した．既存モデルと比較すると，いずれのHigh Impact Bugメトリクスを用いても予測精度は向上せず，提案モデルと既存モデルは一致する結果となった．指定期間1ヶ月において，Ambariプロジェクトの指定期間1日と同様に，有効な予測モデルを構築できなかった．指定期間中央値において，提案モデルは非機械学習モデルと比較して，適合率は14.5%向上，再現率は15.1%向上，F1値は14.8%向上した．既存モデルと比較して，適合率は0.3%低下，再現率は0.7%向上，F1値は0.2%向上した．このことから，Wicketプロジェクトでは，提案モデルによって，指定期間1日において予測精度が向上することが分かった．指定期間中央値では，適合率が低下し，再現率が向上した．指定期間1週間では，いずれのHigh Impact Bugメトリクスを用いても予測精度は向上せず，提案モデルは有用でなかった．

提案モデルによって，予測精度が0.1%から11.9%向上した

提案モデルによって，適合率が向上し再現率が低下，または適合率が低下し再現率が向上する場合があった

5. 考察

5.1 予測精度向上に寄与したHigh Impact Bug

RQ1の結果から，プロジェクトおよび指定期間によって予測精度向上に寄与するHigh Impact Bugが異なることが分かった．そこで，ケーススタディで用いたデータセットにおける，各High Impact Bugの修正時間の特徴を分析する．表12に，プロジェクトごとに，全不具合の修正時間の中央値および，各High Impact Bugの修正時間の中央値を示す．表12より，いずれのHigh Impact Bugも全体の修正時間に比べて差があることが分かる．このことから，いずれのHigh Impact Bugも予測精度向上に寄与する可能性があるが，実際には寄与しないHigh Impact Bugもあった．この原因として，不具合報告メトリクスのみで十分予測可能な不具合においては，High Impact Bugであるかどうかを考慮する効果が小さくなることが考えられる．

表12 プロジェクトごとのHigh Impact Bugの修正時間（日）の中央値

プロジェクト	ambari	camel	derby	wicket
全体	0.035	0.591	14.214	1.491
Surprise	0.026	0.545	21.223	2.348
Breakage	0.482	0.546	10.491	2.737
Dormant	0.065	0.783	19.027	1.941
Security	0.056	1.245	16.801	0.405
Performance	0.053	0.799	22.154	1.140
Blocking	0.007	1.625	11.135	6.946

5.2 High Impact Bugを考慮した効果

RQ2の結果から，High Impact Bugメトリクスを用いることで，0.1%から11.9%の精度向上が見られた．最も精度向上が見られたDerbyプロジェクトの指定期間1日におけるランダムフォレスト法を用いた実験結果について，提案モデル（プロジェクト特化モデル）と既存モデルの予測結果の違いを詳しく分析する．提案モデルと既存モデルで予測結果が異なった不具合が，全体（840件）のうち，27件存在した．27件のうち，正しく予測できるようになったのは17件で，そのうち12件がいずれかのHigh Impact Bugに分類される不具合であった．12件のうち2件がSecurity bug, 5件がbreakage bug, 6件がSurprise bug, 1件がDormant bugだった（ただし，重複あり）．正しく予測できるようになった2件のSecurity bugは，既存モデルでは1日以内に修正完了しないと予測していたが，実際には1日以内に修正完了しており，提案モデルでは正しく予測できていた．Security bugはユーザに与える損害が大きく，早く修正しなければならないと判断され，実際に早く修正が行われたと考えられる．このことから，既存モデルで予測を行うと，早く修正すべき不具合を修正できないと判断して，後回しにしてしまう場合が生まれる可能性があるが，提案モデルではそれを回避できると効果があった．

5.3 構築した予測モデルの有用性

本ケーススタディで構築した予測モデルは，ほとんどのプロジェクトおよび指定期間において，非機械学習モデルよりも予測精度が高く，有用な予測モデルが構築できた．ただし，Ambariプロジェクトの指定期間1日，Camelプロジェクトの指定期間1週間，Wicketプロジェクトの指定期間1ヶ月では，有効な予測モデルが構築できなかった．この3つプロジェクトおよび指定期間では，指定期間内に修正完了した不具合の割合が85%以上である．このことから，指定期間内に修正完了した不具合の割合が高すぎると，有効な予測モデルが構築しにくいと考えられる．

5.4 制約

本研究で用いたデータセットは，4件のOSSプロジェクトの不具合を用いているが，いずれのOSSプロジェクト

も Apache Software Foundation (ASF) のサブプロジェクトである．ASF はサブプロジェクトごとに独立した開発体系を採っており，プロジェクトによって High Impact Bug の影響は異なると考えられるが，今後は一般性のために，その他のプロジェクトを対象としたケーススタディを行う必要がある．

本研究で用いたデータセットは，各プロジェクト 1,000 件ずつ無作為に抽出したものである．そのため，各プロジェクトに報告された全ての不具合を対象とする場合と比べて偏りが生じる可能性がある．また，各 High Impact Bug の分類は，目視によって行われている．そのため，目視を行った人の解釈によって High Impact Bug であるか否かの分類に差が生じる可能性がある．

6. まとめと今後の課題

本稿では，High Impact Bug が不具合修正時間予測の予測精度に与える影響を明らかにすることを目的として，4 件の OSS プロジェクトを対象にケーススタディを行った．ケーススタディの結果，以下の知見が得られた．

- High Impact Bug を考慮することによって，不具合修正時間予測モデルの予測精度が 0.1%から 11.9%向上する
- 先行研究で提案されている 6 種類の High Impact Bug のうち，どの High Impact Bug が予測精度向上に寄与するかは，プロジェクトおよび指定期間によって異なる

今後は，多くのプロジェクトを対象にケーススタディを行い，一般性を確かめる必要がある．

謝辞 本研究の一部は，文部科学省科学研究補助金（基盤 (C): 15K00101）による助成を受けた．

参考文献

[1] Anbalagan, P. and Vouk, M.: "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).

[2] Anh, N. D., Cruzes, D., Conradi, R. and Ayala, C. P.: 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)*, pp. 13:1–13:10 (2011).

[3] Anvik, J., Hiew, L. and Murphy, G. C.: Who should fix this bug?, *Proceedings of the 28th international conference on Software engineering (ICSE '06)*, pp. 361–370 (2006).

[4] Bhattacharya, P. and Neamtiu, I.: Bug-fix Time Prediction Models: Can We Do Better?, *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11)*, pp. 207–210 (2011).

[5] Chen, T.-H., Nagappan, M., Shihab, E. and Hassan, A. E.: An Empirical Study of Dormant Bugs, *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR '14)*, pp. 82–91 (2014).

[6] Gegick, M., Rotella, P. and Xie, T.: Identifying security bug reports via text mining: An industrial case study, *Proceedings of the 7th Working Conference on Mining Software Repositories (MSR '10)*, pp. 11–20 (2010).

[7] Herraiz, I., German, D. M., Gonzales-Barahona, J. M. and Robles, G.: 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).

[8] Hewett, R. and Kijsanayothin, P.: On Modeling Software Defect Repair Time, *Empirical Software Engineering*, Vol. 14, No. 2, pp. 165–186 (2009).

[9] Jeong, G., Kim, S. and Zimmermann, T.: Improving bug triage with bug tossing graphs, *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. 111–120 (2009).

[10] Marks, L., Zou, Y. and Hassan, A. E.: 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)*, pp. 11:1–11:8 (2011).

[11] Nistor, A., Jiang, T. and Tan, L.: Discovering, Reporting, and Fixing Performance Bugs, *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*, pp. 237–246 (2013).

[12] Ohira, M., Kashiwa, Y., Yamatani, Y., Yoshiyuki, H., Maeda, Y., Limsettho, N., Fujino, K., Hata, H., Ihara, A. and Matsumoto, K.: A Dataset of High Impact Bugs: Manually-Classified Issue Reports, *Proceedings of 12th Working Conference on Mining Software Repositories (MSR 15)*, pp. 518–521 (2015).

[13] Panjer, L. D.: Predicting Eclipse Bug Lifetimes, *Proceedings of the Fourth International Workshop on Mining Software Repositories (MSR '07)*, p. 29 (2007).

[14] Robbes, R. and Röthlisberger, D.: Using Developer Interaction Data to Compare Expertise Metrics, *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*, pp. 297–300 (2013).

[15] Shihab, E., Mockus, A., Kamei, Y., Adams, B. and Hassan, A. E.: High-impact Defects: A Study of Breakage and Surprise Defects, *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE '11)*, pp. 300–310 (2011).

[16] Valdivia Garcia, H. and Shihab, E.: Characterizing and Predicting Blocking Bugs in Open Source Projects, *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR '14)*, pp. 72–81 (2014).

[17] Weiss, C., Premraj, R., Zimmermann, T. and Zeller, A.: How Long Will It Take to Fix This Bug?, *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR '07)*, p. 1 (2007).

[18] Zaman, S., Adams, B. and Hassan, A. E.: Security Versus Performance Bugs: A Case Study on Firefox, *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR '11)*, pp. 93–102 (2011).

[19] 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).