Vol.2015-SE-187 No.12
2015/3/12

適切な優先度設定のための修正作業の分析

吉行　勇人[1,a)]　大平　雅雄[1,b)]

概要：本研究の目的は，不具合修正における不具合の優先度を適切に設定するための手法を構築することである． OSS プロジェクトでは日々多くの不具合が報告されており，全ての不具合を次のリリースまでに修正することが困難であるため，各不具合に優先度を設定して，優先的に対処すべき不具合を決定している．不具合の優先度の決定には報告された不具合の情報を確認するコストがかかるため，不具合報告時の情報を用いて優先度を予測する手法が提案されている．しかし，先行研究で予測される優先度はこれまでに設定された優先度が正しいことを前提としている．OSS プロジェクトを対象とした予備調査の結果，高い優先度が設定された不具合のうち約 20%の不具合は次のリリース以降に修正完了しており，設定された優先度が必ずしも修正作業と一致していないことが分かった．そこで，優先度と修正作業が一致していない不具合の特徴を明らかにするための分析を行う．分析の結果，次のリリース以前に修正完了した不具合とリリース以後に修正完了した不具合の特徴の違いを明らかにした．

キーワード：OSS 開発，不具合修正，優先度予測

An Analysis for Appropriate Priority Assignments in a Bug Management Process

Abstract: The goal of our study is to construct a method to support an appropriate priority assignment for a bug-fixing task. A large number of bugs are reported to a large-scale open source software (OSS) project. Since fixing all the bugs until the next release is very difficult, a manager in the project needs to decide which bugs should be fixed preferentially. The existing study proposed a method to predict the priority of a bug, but it is based on the assumption that the priority put in the past bug-fixing was correct. In our preliminary investigation, about 20% of bugs with high priority are fixed after the next release. In other words, the priority does not necessarily coincide with the practice. In this paper, bug reports in two open source software projects (Apache Derby and Qpid) are analyzed and classified according to the priority and whether they were fixed by the next release or not. As a result of our analysis, we found that there are several differences between the bugs fixed before and after the next release.

Keywords: Open Source Software Development, Bug Fix, Priority Prediction

1. はじめに

近年，開発期間の短縮や品質の確保を目的として，オープンソースソフトウェア (OSS) を利用したソフトウェア開発が主流となっている．また，Mozilla Firefox や Android など，エンドユーザが OSS を直接利用する機会も増加している．OSS が広く利用されるようになった結果，大規模 OSS プロジェクトには日々多くの不具合が報告されている．例えば，Mozilla プロジェクトでは，1 日に数百件以上の不具合が報告される場合がある [6]．OSS プロジェクトに報告された不具合は，プロジェクトに参加している開発者によって確認され不具合であることが認められれば，適任の開発者に修正が依頼され，修正が行われる．

OSS プロジェクトでは，不具合管理システム (Bug Tracking System : BTS) を用いて，不具合に対処している．不具合管理システムには，報告された不具合の基本的な情報（不具合の重要度や不具合が発生したコンポーネント）や，不具合に対する議論，不具合を報告した人や修正を担当した人などの情報が記録されている．不具合の情報を詳細に

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

記録し，修正漏れを防止するのが目的である．

特に，日々多くの不具合が報告される現状では，全ての不具合を次のソフトウェアリリースまでに修正することは困難である．そのため，各不具合に優先度を設定し，優先的に取り組むべき不具合とそうでない不具合とを区別している．高い優先度が設定された不具合は，ユーザへの影響の大きい不具合など，次のリリースまでに修正が行われることが期待されている．反対に，低い優先度が設定された不具合は次のリリースまでに必ずしも修正する必要はないと判断されている．各不具合に優先度を設定するには，報告された不具合を確認し，優先順位を決定する必要がある．報告される膨大な量の不具合に対して人手ですべての優先度を決定することは困難なため，優先順位付け作業の負担を軽減するために．報告された不具合の優先度を予測する手法が提案されている [14].

しかし，先行研究 [14] で予測される優先度は，過去に不具合に設定された優先度が正しいことを前提としている．高い優先度が設定された不具合は優先的に対処された結果，修正にかかる時間が短くなると考えられるが，不具合に設定された優先度は必ずしも不具合修正時間に影響を与える訳ではないことが報告されている [13]．したがって，OSS プロジェクトなどで行われている人手による優先度の設定は，必ずしも適切であるとは限らないと言える．さらに，本研究で OSS プロジェクトを対象とした予備調査を行った結果，高い優先度が設定された不具合のうち 20%から 30%の不具合は次のリリース後に修正完了していることが分かった．すなわち，設定された優先度どおりに修正作業が行われていない不具合が存在することが示唆された．優先度は次にリリースされるバージョンの品質を考慮して設定されているため，設定された優先度どおりに修正作業が行われていない場合，プロジェクトが当初想定した品質を満たせないままリリースされている可能性がある．

我々は，ソフトウェアの品質保証には，次のリリースまでに修正すべき不具合が全て修正されるための適切な優先度設定が必要であると考えている．本研究の最終目的は，適切な優先度設定のための支援手法を構築することである．本稿では，初期段階として，設定された優先度どおりに修正作業が行われていない不具合に注目して分析を行い，特徴を明らかにすることを目的とする．

本稿では，Apache Software Foundation の OSS プロジェクトである Apache Derby プロジェクトと Apache Qpid プロジェクトを対象として，ケーススタディを行う．ケーススタディの結果，以下の結論を得た．

- 優先度の高低に関わらず，約 20%〜30%の不具合はリリース後に修正完了している．
- リリース後に修正完了した不具合は修正規模が大きく，修正時間が長い
- 次のリリースに間に合わなかった不具合は長期間放置されやすい

本稿の構成は以下の通りである．続く第 2 章では関連研究と本研究のモチベーションについて述べる．第 4 章ではケーススタディとそ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

以下では，本研究に関連する先行研究を紹介し，本研究との関連性を説明する．

2.1 不具合予測

日々多数の不具合が報告される現状では，すべての不具合を目視で確認し，適切な処置を行うことが現実的でないため，機械学習や統計手法を用いて様々な要素を予測する研究が行われている [3,7,9–11,14].

例えば，不具合修正時間予測 [4,17,18] では，過去に報告された不具合の特徴と修正にかかった時間を機械学習し，新しく報告された不具合の修正にどれだけ時間がかかるかを予測している．不具合修正時間の予測結果は，OSS プロジェクトの開発者にとってどの不具合を次のリリースまでに修正できそうかを知る指標として活用することができる．その他には，過去に報告された（解決された）重複する不具合を検出するための研究 [3,9] や，一旦修正された不具合の再発 (Reopen) しやすさを予測する研究 [10,11] などがある．

また，本研究と類似する研究には，不具合の重要度 [7] や優先度 [14] を予測する研究がある．しかしながら，これらの研究では，OSS プロジェクトの過去の不具合報告を正解集合として利用している，すなわち，重要度や優先度が正しく設定されていたものとして不具合報告データを扱っている．したがって，[7] や [14] で報告されている予測精度は，実際にはより低い値をとるものと思われる．

2.2 不具合の特徴分析

これまで提案されている予測手法は，全ての不具合を等しく扱っているが，実際にはそれぞれの不具合が開発プロセスやプロダクトに与える影響は同一ではないため，不具合が与える影響を考慮した予測手法が求められている．そこで，先行研究では，プロセスやプロダクトに大きな影響を与える不具合の特徴を分析している [5,12,15,16]．[12] は，開発者のスケジュールに影響を与える不具合 (Surpeise Bug) と，ユーザの満足度に影響を与える不具合 (Breakage Bug) について分析を行っている．[15] は，他の不具合の修正を妨げる不具合 (Blocking Bug) の特徴を明らかにしている．[16] は，非機能要求に関する不具合として，セキュリティを脅かす不具合 (Security Bug) と，パフォーマンス

の低下を発生させる不具合 (Performance Bug) に着目し，特徴を明らかにしている．[5] は，あるバージョンに混入された不具合が，そのバージョンのリリースでは発見されず，次のバージョン以降で初めて発見される不具合 (Dormant Bug) に着目して分析を行っている．

これらの研究は，開発者やユーザへの影響を考慮した優先度設定においても活用すべき知見である．例えば，Security Bug は言うまでもなく，ユーザの満足度に影響を与える Breakage Bug は，ユーザが競合他社製品へ移行しないよう速やかに対処すべき不具合であると言える．本研究が最終的に目指す，優先度の自動設定においては不具合の特徴と影響についても考慮した手法にすべきである．

2.3 不具合修正割当支援

OSS 開発における不具合を修正するプロセスにおける修正作業を支援するための研究が行われている．2.1 節でも述べたように，大量の不具合それぞれに対して適任の担当者を人手で決定することが困難なためである．不具合修正に最も適任の担当者を推薦する手法 [2,6] や，最も低い修正コストで修正を完了するための担当者推薦手法 [8]，修正担当者の負荷を考慮した担当者推薦手法 [19] などがある．本研究では，適切な優先度の自動設定を支援するための手法構築を最終目標としているが，最適な優先度の決定には，不具合をどの担当者が修正するべきかについても考慮するべきであり，これら先行研究を今後は参考にする予定である．

3. 予備調査

本章では，本研究で行った予備調査について説明する．

3.1 調査内容

本予備調査は，OSS プロジェクトにおける修正作業の実態を明らかにすることを目的とする．ある時点で報告された不具合が，次のバージョンリリースまでに修正完了できたかどうかを調査する．調査対象とする不具合は，第 4 章のケーススタディで用いるデータセットと同一である．

3.2 結果

表 1 に，予備調査の結果を示す．表 1 より，Derby では優先度が高い不具合のうち約 27% (25/(67+25)) の不具合がリリース後に修正完了している（不具合報告時点から次のリリースまでの間に修正が完了しなかった）ことが分かった．また，優先度の低い不具合のうち，約 36% (239/(419+239)) の不具合が次のリリース後に修正完了していることが分かった．

同様に表 1 より，Qpid では優先度が高い不具合のうち 20% (35/(140+35)) の不具合がリリース後に修正完了していることが分かった．また，優先度の低い不具合のうち，

表 1 予備調査結果

		優先度高	優先度低
Derby	リリース前	67	419
	リリース後	25	239
Qpid	リリース前	140	324
	リリース後	35	82

約 20% (82/(324+82)) の不具合が次のリリース後に修正完了していることが分かった．

予備調査の結果から，高い優先度が設定されている不具合のうち，約 20%から 30%程度の不具合は，高い優先度が設定されているにも関わらず直近のリリースまでに修正が間に合わなかったことが分かった．また，低い優先度が設定されている不具合でも，20%から 40%近い不具合が直近のリリース以降に修正完了していることが分かった．これらの結果から，プロジェクトによっては設定された優先度どおりに修正作業が行われていない場合が存在することが示唆された．次章では，設定された優先度どおりに修正作業が行われていない不具合に着目して行った分析する．

4. ケーススタディ

本章では，本研究で行った不具合の分析のケーススタディについて説明する．本ケーススタディの目的は，優先度と修正作業が一致していない不具合の特徴を明らかにすることである．

4.1 データセット

本ケーススタディでは，Apache Derby プロジェクトおよび Apache Qpid プロジェクトの不具合を分析対象とする．各プロジェクトの不具合管理システム JIRA に報告された不具合に対してフィルタリングを行い，分析対象とする不具合を決定する．なお，第 3 章で述べた予備調査で用いたデータセットは本ケーススタディで対象とした不具合と同一である．対象プロジェクトに関する情報を表 2 に示す．

以下に不具合に対して行ったフィルタリングの内容を示す．JIRA では，報告の種類を分類することができる．分類には，不具合 (Bug) の他に，既存機能の改善要求 (Improvement) や，新規機能の追加要求 (New Feature)，その他タスク (Task) 等がある．本ケーススタディでは不具合 (Bug) のみを対象とする．また，修正完了済みの不具合を対象とする．修正完了済み不具合は，Status が Fixed となっている不具合とする．修正規模の比較のために，修正にソースコードの変更を必要とした不具合を対象とする．ソースコードの変更を必要とした不具合を特定する方法は，4.3 節で説明する．

対象不具合は，Apache Derby は 2004 年 9 月から 2014 年 12 月に報告された 6,785 件の不具合，Apache Qpid は 2006 年 9 月から 2014 年 12 月に報告された 6,288 件の不

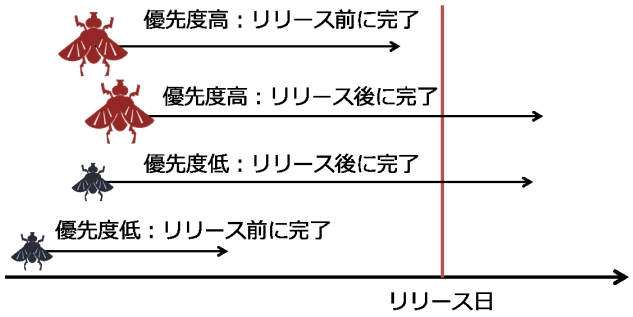

図 1　優先度と修正完了日による不具合の分類

表 2　対象プロジェクト

プロジェクト名	Derby	Qpid
期間	2004/9–2014/10	2006/9–2014/10
全不具合数	6,785	6,288
対象不具合数	750	581

具合とする．フィルタリングの結果，対象不具合の件数は Apache Derby は 750 件，Apache Qpid は 581 件となった．

4.2　不具合の分類

設定された優先度どおりに修正作業が行われていない不具合を分析するために，本ケーススタディでは不具合を「優先度の高低」と「次のリリースまでに修正完了したか否か」によって 4 種類の不具合に分類する．不具合の 4 種類の分類を図 1 に示す．

まず，「優先度の高低」による分類について説明する．本ケーススタディの対象として用いる不具合管理システム JIRA では，優先度が高い順に Blocker, Critical, Major, Minor, Trivial の 5 段階の優先度を設定することができる．不具合報告時の初期値は Major であり，対象プロジェクトでは記録されている不具合のうち約 60%の不具合は初期値のままとなっている．そのため，優先度が Major 以外である不具合は何らかの理由によりプロジェクトの管理者によって優先度を設定，変更された不具合であると示唆される．従って，Major を除いた 4 種類の優先度のうち，Blocker, Critical を「優先度高」，Minor, Trivial を「優先度低」として不具合を分類する．この分類によって，その不具合が報告された時点から次のリリースまでに修正すると想定されていたか否かを判断する．

次に，「次のリリースまでに修正完了したか否か」による分類について説明する．不具合報告には報告された日付と，既に修正が完了していれば修正完了日が記録されている．OSS プロジェクトに記録されている過去のリリースの日付を参照して，各リリースの間に報告された不具合の修正完了日が次のリリース日よりも前か後かによって不具合を分類する．この分類によって，

表 3　分析結果 (Derby)

		優先度高	優先度低
修正時間（日）	リリース前	10.19	6.42
	リリース後	139.81	286.50
コメント数	リリース前	10	4
	リリース後	14	8
修正ファイル数	リリース前	1	1
	リリース後	2	1
修正行数	リリース前	17	3
	リリース後	27	6
テストファイル数	リリース前	1	1
	リリース後	2	1
テスト行数	リリース前	11	5
	リリース後	36	8
変更ファイル数	リリース前	3	2
	リリース後	4	3
変更行数	リリース前	45	20
	リリース後	103	34

4.3　分析内容

分析内容として，前述の不具合の分類ごとに，不具合の件数，修正にかかった時間，コメント回数，修正時に変更したファイル数，修正時に変更したソースコードの行数をそれぞれメトリクスとして算出し，比較を行う．

修正にかかった時間は，不具合が報告されてから修正完了のタグ (Resolved) が付けられるまでの時間を日数に換算して計測する．不具合に対するコメント回数は，各不具合に対して記録されており，コメント回数が多いほど，その不具合に対する議論が活発だったといえる．修正時に変更したファイル数とソースコードの行数は，ソースコードを管理するためのバージョン管理システムから計測される．本ケーススタディで対象とするプロジェクトでは，バージョン管理システムとして Git を利用しており，公開された Git リポジトリから不具合修正のための変更を抽出する．Git ではソースコードの変更時にコミットメッセージと呼ばれる記述を残すことができ，変更の目的等を記録することができる．Apache Software Foundation のプロジェクトでは，不具合の修正のための変更にはプロジェクト名-JIRA の不具合番号（例えば，QPID-1234）と記録する方針をとっており，その記述のある変更を抽出することによって，どの不具合を修正するための変更であるかを特定することができる．また，修正時に変更したファイル数とソースコードの行数は，不具合修正のための変更と，テストケース修正のための変更に分割した結果も分析する．テストケース修正のための変更は，変更したファイルのファイルパス中に “test” の記述があるかどうかで特定する．例えば，“java/system/test/.....” といったファイルパスだった場合，このファイルはテストのためのファイルであると判断される．不具合修正のための変更は，テストケース修正のための変更以外の変更とする．

表 4　分析結果 (Qpid)

		優先度高	優先度低
修正時間（日）	リリース前	5.02	1.28
	リリース後	202.00	171.48
コメント数	リリース前	3	2
	リリース後	3	2
修正ファイル数	リリース前	2	1
	リリース後	2	1
修正行数	リリース前	13	6
	リリース後	19	5
テストファイル数	リリース前	0	0
	リリース後	1	0
テスト行数	リリース前	0	0
	リリース後	2	0
変更ファイル数	リリース前	2	2
	リリース後	3	2
変更行数	リリース前	21.5	12
	リリース後	64	12

4.4　結果

本節では，ケーススタディの結果について述べる．

4.4.1　Derby

表 3 に Derby の分析結果を示す．表中の値は全て中央値である．

表 3 の修正時間をみると，リリース前に修正完了した不具合に関しては優先度が低い不具合の方が修正時間が短いことが分かった．一方で，リリース後に修正完了した不具合に関しては，優先度が高い不具合の方が修正時間が短いことが分かった．また，リリース後に修正完了した不具合はリリース前に修正完了した不具合よりも修正にかかる時間が 100 日から 200 日以上長くなる傾向にあることが分かった．

表 3 のコメント数をみると，優先度が高い不具合の方が優先度が低い不具合よりもコメント数が多く，リリース後に修正完了した不具合の方がリリース前に修正完了した不具合よりもコメント数が多いことが分かった．

表 3 の修正ファイル数，テストファイル数をみると，どちらのメトリクスも中央値では，優先度が高くリリース後に修正完了した不具合のみファイル数が多かった．修正ファイル数とテストファイル数の合計である変更ファイル数をみると，優先度が高い不具合の方が優先度が低い不具合よりも変更ファイル数が多く，リリース後に修正完了した不具合の方がリリース前に修正完了した不具合よりも変更ファイル数が多いことが分かった．

表 3 の修正行数，テスト行数，変更行数をみると，どのメトリクスも，優先度が高い不具合の方が優先度が低い不具合よりも行数が多く，リリース後に修正完了した不具合の方がリリース前に修正完了した不具合よりも行数が多いことが分かった．

4.4.2　Qpid

表 4 に Qpid の分析結果を示す．表中の値は全て中央値である．

表 4 の修正時間をみると，リリース前に修正完了した不具合とリリース後に修正完了した不具合のどちらも，優先度が低い不具合の方が修正時間が短かった．また，Derby の結果と同様に，リリース後に修正完了した不具合はリリース前に修正完了した不具合よりも修正にかかる時間が 200 日程度長くなる傾向にあることが分かった．

表 4 のコメント数をみると，優先度が高い不具合の方が優先度が低い不具合よりもコメント数が多いが，リリース前とリリース後では差がなかった．

表 4 の修正ファイル数をみると，優先度が高い不具合の方が優先度が低い不具合よりも修正ファイル数が多いが，リリース前とリリース後では差がなかった．テストファイル数をみると，優先度が高く，リリース後に修正完了した不具合以外は中央値が 0 であった．このことから，Qpid では過半数の不具合は不具合修正時にテストケースの変更を行っていないことが分かった．また，変更ファイル数をみると，優先度が高く，リリース後に修正完了した不具合は他の不具合よりもファイル数が多かった．

表 4 の修正行数をみると，優先度が高い不具合の方が優先度が低い不具合よりも修正行数が多いことが分かった．リリース前とリリース後では，優先度が高い不具合はリリース後の方が修正行数が多く，優先度が低い不具合はリリース前の方が修正行数が多かった．テスト行数をみると，優先度が高く，リリース後に修正完了した不具合以外は中央値が 0 であった．変更行数をみると，修正行数と同様に優先度が高い不具合の方が優先度が低い不具合よりも修正行数が多いことが分かった．優先度が高い不具合はリリース後の方が変更行数が多いが，優先度が低い不具合はどちらも差がなかった．

4.4.3　まとめ

Derby と Qpid の結果を比較し，ケーススタディの結果をまとめる．

修正時間については，優先度の高低による影響はプロジェクトによって逆の結果であったが，リリース後に修正完了した不具合はどちらのプロジェクトでも 100 日から 200 日程度かかっており，修正時間が長くなる傾向にあることが分かった．

コメント数については，中央値では Qpid はコメント数が少なく顕著な差は出なかったが，Derby では優先度が高い不具合はコメント数が多く，またリリース後に修正完了した不具合の方がコメント数が多いことが分かった．

修正時に変更したファイル数とソースコードの行数は，不具合修正のための変更とテストケース修正のための変更に分けて分析を行ったが，Derby ではどちらも同じ特徴で，優先度が高い不具合は変更したファイル数やソースコード

の行数が多く，またリリース後に修正完了した不具合の方が変更したファイル数やソースコードの行数が多いことが分かった．Qpid では，不具合修正のための変更については Derby と同様の結果で，テストケース修正のための変更については，数が少なく大きな差が出なかった．

5. 考察

本章では，予備調査とケーススタディによって得られた結果と，本分析の妥当性について議論する．

5.1 予備調査とケーススタディの結果の考察

予備調査の結果から，優先度の高低に関わらず，20%から 30%の不具合はリリース後に修正完了していることが分かった．しかし，その理由は優先度の高低によって異なると考えられる．高い優先度が設定されている不具合でリリース後に修正完了した不具合は，多くの議論が必要であったり，修正規模が大きい等の理由で，リリースに間に合わなかった場合が多いと考えられる．低い優先度の場合は，修正するまでもないと判断されて放置され，リリース日を過ぎて余裕ができてから修正していると考えられる．低い優先度が設定されていてリリース前に修正完了した不具合は，タイプミス等のささいな問題ですぐに修正完了できると判断され，リリース前に終わらせてしまったと考えられる．限られた人的リソースの中で修正すべき不具合を全て修正するには，修正規模の見積もりや適切な修正担当者設定を行い，不具合に設定された優先順位の逆転が起こらないようにする必要がある．

ケーススタディの結果から，リリース後に修正完了した不具合は修正時間が長い傾向にあることが分かった．しかし，報告されてから次のリリース日を過ぎて修正完了した不具合は当然修正完了までの時間が長くなるため，報告からリリース日までの時間と，リリースを過ぎてからの時間を算出した．その結果を表 5 に示す．

表 5 をみると，Derby においては優先度の高低に関わらず，報告日からリリース日までの時間は 80 日程度であった．また，リリース日を過ぎてからの時間をみると，優先度が高い不具合で 50 日程度，優先度が低い不具合は 200 日程度かかっていることが分かった．Qpid においては，高い優先度が設定された不具合は報告日からリリース日までの時間が 80 日程度，低い優先度では 70 日程度であった．リリース日を過ぎてからの時間は，どちらも 100 日以上かかっていることが分かった．これらの結果から，リリース日以降は早く修正する必要が無くなったため，長期間放置され，総修正時間が長くなったと考えられる．また，プロジェクトによっては，優先度が高い不具合の方がリリースに間に合わなくても放置されにくい傾向にあると考えられる．

また，修正時に変更したファイル数やソースコードの行数の比較の結果から，リリース後に修正完了した不具合は変更行数等の修正規模が大きいことが分かった．このことから，リリース後に修正完了した不具合は修正が難しい不具合が多く，リリース前に修正完了させるためには適切な修正担当者の設定が必要であると考えられる．特に，変更ファイル数が多い場合はファイル間の依存関係を理解していなければ修正によって別の不具合を生じさせる可能性が高くなるため，コンポーネント毎に理解の深い開発者に修正を担当させるべきであると考えられる．

表 5　リリース日以降に完了した不具合の修正時間

		優先度高	優先度低
Derby	総修正時間	139.81	286.5
	報告日からリリース日まで	81.76	80.71
	リリース日以降	58.05	205.79
Qpid	総修正時間	202.00	171.48
	報告日からリリース日まで	82.72	66.48
	リリース日以降	119.28	104.99

最後に，コメント数の比較の結果から，高い優先度が設定された不具合はコメント数が多く，リリース後に修正完了した不具合もコメント数が多いことが分かった．このことから，高い優先度が設定されている不具合は開発者の目にとまりやすく，多くの開発者が興味を持ったことで活発な議論がされやすいと考えられる．また，リリース後に修正完了した不具合は修正方針について意見が分かれたり，修正方法の特定が難しい等の理由で議論が長引き，修正に時間がかかっていると考えられる．

5.2 妥当性の検証

本稿でケーススタディに用いた OSS プロジェクトは，どちらも Apache Software Foundation に属するプロジェクトである．どちらも独立した開発を行っているプロジェクトではあるが，上位組織の影響を受けていないとは言い切れないため，今後は Apache Software Foundation 以外のプロジェクトを対象とした分析も行う必要がある．

本稿でケーススタディに用いた OSS プロジェクトでは，不具合管理システムに JIRA を用いている．一方で，Bugzilla という不具合管理システムでは，優先度の他に，重要度というパラメータが存在する．Bugzilla を利用している OSS プロジェクトである Eclipse プロジェクトでは，優先度は開発者によって設定され，どの不具合の修正を優先して行うかを表し，重要度はユーザによって設定され，その不具合が与える影響度を表している，と定義されている [1]．JIRA では，優先度のパラメータしかないため，Bugzilla における重要度を加味したパラメータが優先度として設定されている可能性がある．そのため，今後は Bugzilla における優先度の影響についても，分析を行う必要がある．

本稿のケーススタディでは，5 段階の優先度のうち，高

い方から 2 項目を高い優先度，低い方から 2 項目を低い優先度として不具合を分類している．そのため，高い優先度の中での優先度の高低（例えば，$Blocker > Critical$）を考慮していない．今後は，より細粒度での優先度の分析を行う必要がある．

また，修正作業の分類のために，次のリリース前に修正完了したか次のリリース後に修正完了したかによって不具合を分類している．しかし，実際にはリリース前には一切の修正作業を行わず，リリース後に初めて修正作業が始まる場合があり，次のリリースを目標にしているのかどうかを判断して，分類を行う必要があると考えられる．

本稿ではリリース日以前に修正完了した不具合はそのリリースに間に合ったと判断しているが，不具合の検証期間を考慮しておらず，記録された修正完了日がリリース日以前でも実際にそのリリースに不具合修正が含まれたかどうかを判断していない．今後はプロジェクトのリリースノート等を参照して，日付以外の情報を用いてリリースに間に合ったか否かを判断する必要がある．

6. まとめと今後の課題

本稿では，不具合修正プロセスにおける不具合の優先度と修正作業の関係を理解するための分析を行った．分析の結果，リリース前に修正完了した不具合とリリース後に修正完了した不具合の特徴の違いが明らかとなった．本稿で得られた知見として，以下のことが分かった．

- 設定された優先度どおりに修正作業が行われていない不具合が存在する
- 優先度の高低に関わらず，約 20%〜30%の不具合はリリース後に修正完了している．
- リリース後に修正完了した不具合は修正規模が大きく，修正時間が長い
- 次のリリースに間に合わなかった不具合は長期間放置されやすい

今後の課題は，他のプロジェクトや他の不具合管理システムを利用しているプロジェクトを対象とした分析を行うことである．また，不具合に関わる人に関する分析や不具合報告に含まれる情報を用いた分類の細分化を行い，より詳細な分析を行う．また，本稿では，主に修正完了時点で初めて分かる結果を中心に分析を行っている．適切な優先度設定の手法構築のためには，不具合報告時点での情報について分析を行い，リリースに間に合う不具合かそうでないかを分類する特徴を明らかにする必要がある．今後の分析の結果から，適切な優先度を設定するための手法を構築する．

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

参考文献

[1] : What is the difference between Severity and Priority?, http://wiki.eclipse.org/Bug_Reporting_FAQ.
[2] 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).
[3] Bettenburg, N., Premraj, R., Zimmermann, T. and Kim, S.: Duplicate bug reports considered harmful ... really?, *Proceedings of the 24th IEEE International Conference on Software Maintenance (ICSM '08)*, pp. 337 –345 (2008).
[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 (online), DOI: 10.1145/1985441.1985472 (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] 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).
[7] Lamkanfi, A., Demeyer, S., Giger, E. and Goethals, B.: Predicting the severity of a reported bug, *7th IEEE Working Conference on Mining Software Repositories (MSR)*, pp. 1–10 (2010).
[8] Park, J.-W., Lee, M.-W., Kim, J., won Hwang, S. and Kim, S.: CosTriage: A Cost-Aware Triage Algorithm for Bug Reporting Systems, *AAAI* (Burgard, W. and Roth, D., eds.), AAAI Press (2011).
[9] Runeson, P., Alexandersson, M. and Nyholm, O.: Detection of Duplicate Defect Reports Using Natural Language Processing, *Proceedings of the 29th international conference on Software Engineering (ICSE '07)*, pp. 499–510 (2007).
[10] Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W. M., Ohira, M., Adams, B., Hassan, A. E. and ichi Matsumoto, K.: Studying Re-opened Bugs in Open Source Software, *Empirical Software Engineering*, pp. 1–38 (2012).
[11] Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W. M., Ohira, M., Adams, B., Hassan, A. E. and Matsumoto, K.: Predicting Re-opened Bugs: A Case Study on the Eclipse Project, *17th Working Conference on Reverse Engineering (WCRE'10)*, pp. 249–258 (2010).
[12] 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).
[13] Syer, M. D., Adams, B., 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)* (2011).
[14] Tian, Y., Lo, D. and Sun, C.: DRONE: Predicting Priority of Reported Bugs by Multi-factor Analysis, *Proceedings of the 29th IEEE International Conference on Software Maintenance (ICSM '13)*, pp. 200–209 (2013).
[15] 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).

[16] 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).

[17] 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. 52, No. 2, pp. 933–944 (2013).

[18] 吉行勇人，大平雅雄，戸田航史：OSS 開発における管理者と修正者の社会的関係を考慮した不具合修正時間予測，コンピュータソフトウェア ((to appear)).

[19] 柏祐太郎，大平雅雄，阿萬裕久，亀井靖高：大規模 OSS 開発における不具合修正時間の短縮化を目的としたバグトリアージ手法，情報処理学会論文誌 ((to appear)).