OSS開発における一般開発者の協調作業と不具合の再修正に関する一考察

林 宏徳[1]　伊原 彰紀[1]　門田 暁人[1]　松本 健一[1]

概要：近年，オープンソースソフトウェア (OSS) が官公庁，教育機関だけでなく，商用ソフトウェアの一部に利用する企業等が増加し，幅広いサービスを提供するようになっている．その一方で，ソフトウェアの高機能化にともない，不具合を修正するためには，ソフトウェアが備える各機能の専門家による協調作業が必要となりつつある．しかし，不具合が正確に修正されず，約 15%は修正作業の手戻り（再修正）が発生し，修正時間の遅延，開発コストの増大を招いている．我々はこれまで，複数のコミッターが検証作業を行うと再修正が発生しやすいことを明らかにしている．本稿では，開発者が協調的に修正作業を行った場合の再修正の分析を行った．Eclipse コミュニティを対象にケーススタディを行った結果，修正人数の増加，また，議論への参加人数が増加すると再修正が起こりやすくなることが分かった．

An Analysis of Collaboration in OSS Community for Understanding Re-opened Bug

Hayashi Hironori[1]　Ihara Akinori[1]　Monden Akito[1]　Matsumoto Ken-ichi[1]

1. はじめに

Apache，Linux をはじめとするオープンソースソフトウェア (OSS) が官公庁，教育機関等で使用されている．近年では，導入コスト，運用保守コスト削減を目的として商用ソフトウェアの一部にオープンソースソフトウェア (OSS) を利用する企業が増加し [14]，OSS が幅広いサービスを提供するようになっている．その結果，OSS に興味を持つ者が増加し，現在，膨大な不具合が日々 OSS コミュニティに報告されている [5]．

ソフトウェアの高機能化にともなって不具合修正をするために複数の機能を変更する機会が増えている．それゆえ，ソフトウェアが備える各機能の専門家による協調作業が必要不可欠となっている．しかしながら，OSS コミュニティに参加する開発者は流動的であるため，必ずしも修正すべき機能の専門家が当該不具合を修正するとは限らない．その結果，不具合の約 15%は正しく修正されず修正作業の手戻り（再修正）が発生し [9]，修正時間の遅延，開発コストの増大を招いている．再修正は開発効率を低下させる要因となるため，再修正を防ぐことが早急の課題となっている．

Shihab らは，修正完了時に，将来的に再修正を要するか否かを予測する手法を提案している．評価実験を通して，再修正を予測するためにはバグ管理システムの情報，特にコミュニケーション情報（内容，頻度，参加人数）が効果的であることを明らかにした [9]．本従来研究により，不具合修正完了時に，将来再修正が発生するか否かを予測することが可能になった．今後は，再修正の発生を防ぐための方法を解明していくことが重要である．

我々は先行研究において再修正の発生を防ぐ協調作業の方法論を確立するために，修正されたソースコードを検証する役割を担うコミッターの協調作業に注目して再修正との関係を分析した．Eclipse コミュニティを対象としたケーススタディの結果，複数の機能に影響する不具合を修正する場合，各専門性を持った開発者が検証作業を実施することで再修正を引き起こす可能性があることが示された．不具合修正における協調作業は検証にとどまらず，修正作業においても発生する．

本稿では，修正作業に取り組む開発者の協調作業を分析

[1] 奈良先端科学技術大学院大学
Takayama, Ikoma-shi, Nara 630–0192, Japan

し，再修正との関係を明らかにする．修正作業は，(1) コミュニティが不具合を確認して以降，修正が完了するまでの一連の作業，(2) 修正作業中の議論とする．本稿でも，Eclipse コミュニティを対象に，再修正される不具合と再修正されない不具合の修正数に関わる開発者数の違いを分析する．本稿より，再修正を防ぐために協調作業に要する開発者数を把握することができる．

本稿の構成は以下のとおりである．続く 2 章では関連研究について説明し，3 章で分析対象について説明する．4 章で分析方法をについて説明し，4 章でケーススタディについて説明すると共に結果を示す．5 章で考察を行い，最後に 6 章においてまとめと今後の課題について述べる．

2. 関連研究

2.1 不具合の再修正に関する研究

ソフトウェアの保守業務はソフトウェア開発におけるコストの大半を占めるため，ソフトウェア工学の分野では不具合修正に関する研究が盛んに行われている．例えば，ソフトウェアの不具合の混入箇所を特定する研究 [12] や修正担当者の決定方法に関する研究 [2] などが挙げられる．不具合修正プロセスの一連の作業において，特に修正されたソースコードの社会への公開は注意が必要である．Shihab らは，不具合を引き起こすソースコードが社会へ公開されることを防ぐために，4 つの観点（作業習慣，不具合の特徴，不具合修正，作業者）から，どの変数が再修正と深い関係を持つかを明らかにした [9]．その結果，不具合管理システムにおける開発者間のコミュニケーション情報が再修正に影響していることが分かった．また，調査した変数を用いて再修正を予測するモデルを構築し，84%~ 90%の予測精度を実現した．

しかしながら，再修正の発生を防ぐために開発者が具体的に取るべき行動については触れていないため，今後は再修正を防ぐための開発者の行動指針を提示する必要がある．

2.2 ソフトウェア開発における協調作業の研究

OSS 開発は地理的に分散した開発者がインターネット上で協調作業をする事で成り立っており [3][15]，その意思疎通・協調作業が OSS コミュニティを成功に導く一つの要因と考えられる [7] ．これまで OSS 開発における協調作業を分析する研究が進められている．

Aberu らは OSS コミュニティにおける開発者間のコミュニケーションが OSS の品質に与える影響を調査した [1]．その結果，一般開発者間の議論が増加する程ソフトウェアに不具合が混入しやすくなることが明らかとなった．更に調査結果を元に開発者間のコミュニケーションに基づく不具合混入の予測モデルを構築した．

Jeong らは不具合修正を依頼する管理者と実際に修正作業を行う修正者のコミュニケーションに着目し，作業依頼にかかる時間を調査した [6]．その結果，適切な修正者に仕事が割り当てられるまでに数多くの不適切な依頼が行われており，実際に修正作業に取り掛かるまでに膨大な時間を要している事が明らかになった．Jeong らは修正依頼の効率化のため，マルコフモデルを用いて修正の依頼・被依頼の関係を構造的に解析し，可視化を行った．また，構造的に得られた関係を用いて予測モデルを構築する事で，不具合の修正担当者を推薦するシステムを提案した．

従来研究で，開発者間の協調作業がソフトウェアの品質や効率に関係すると示されているが，再修正への影響について調査した研究はない．本稿では，再修正の発生を防ぐことを目的とし，開発者間の協調作業を分析する．

3. OSS 開発における不具合管理

本章では，分析の対象とする OSS 開発における不具合の管理と不具合修正における協調作業について説明する．

3.1 不具合管理システム

不具合管理システムは，開発者もしくは利用者から報告された 1 つの不具合に対して 1 つの不具合票を作成し，各々の不具合修正の進捗を管理するためのシステムである．一般的な OSS 開発では，不具合に関する情報を共有するために，Bugzilla [4] や Redmine [8]，Trac [11] 等の不具合管理システムが利用されている．図 1 に不具合管理システム Bugzilla の一画面を示す．また，不具合管理システムの多くは，Web サーバー上で動作し，Web ブラウザ経由で誰でも不具合の報告と修正についての議論が行える点に特徴がある．主に不具合管理システムから得られる情報は，不具合の基本情報（対象プロダクトやバージョン，修正の重要度や優先度），不具合の詳細情報（不具合の内容や不具合を再現するための手順），議論情報（不具合の内容や修正に関して行われる議論），状態遷移管理情報（不具合修正処理の状態管理を行うために記録される情報）の 4 種類がある [13]．Shihab らの研究においても再修正に寄与する要因を調査する上で不具合管理システムを調査しており，主に不具合の (1) 基本情報と (2) 詳細情報，(3) 議論に関してはその量のみに着目し，状態遷移管理情報に関しては着目していない．本稿では協調作業を調査するため，コミュニティが不具合を確認して以降，修正が完了するまでの一連の作業（修正プロセス作業）における協調作業，修正作業中に議論する開発者の協調作業と再修正との関係を分析する．

3.2 不具合修正プロセス

不具合修正プロセスは，不具合が報告されてから正しく修正された事を確認するまでの一連の作業からなる．図 2 に代表的な不具合修正プロセスを示す [6]．このプロセスでは先ず，(a) 報告者（利用者もしくは開発者）が不具合を

図 1 不具合管理システムの一画面

表 1 不具合管理システムで得られる情報

区分	内容例
(1) 不具合の基本情報	対象プロダクト 修正の優先度 修正の重要度
(2) 不具合の詳細情報	不具合の内容 不具合を再現する方法 欠陥のある箇所
(3) 議論情報	開発者による発言 発言者 発言日時
(4) 状態遷移管理情報	不具合報告記録 不具合修正記録 CC リスト管理記録

バグ管理システムに登録し，(b) 次に管理者が修正者に不具合の修正依頼をする．(c) 修正者が不具合を修正し，その後修正された不具合は，(d) コミッターによりレビューされた後リポジトリに反映される．ここで，コミッターのレビューが適当でないと不完全なファイルがリポジトリに反映され，再修正につながるため，先行研究ではコミッター間の協調作業に着目して分析を行った．

4. 分析方法

本章では，分析に用いるデータの収集，整形及び分類方法と，分析の手順について述べる．

4.1 修正プロセス作業の分析

先行研究においては，不具合修正に関わるコミッターの人数と再修正の生起確率に関係性が見られた．本稿では，再修正が発生する不具合修正に関わる開発者（コミッターに限らない）の人数の分析を行う．状態遷移管理情報の分析では，図 2 に示される全ての開発者を対象に，不具合が報告されてから一度リポジトリに変更が更新されるまでの一連の作業を協調作業とみなす．また，状態遷移管理情報において，複数回に渡って修正が加えられた記録を持つ不具合を再修正が行われた不具合とみなす [9]．

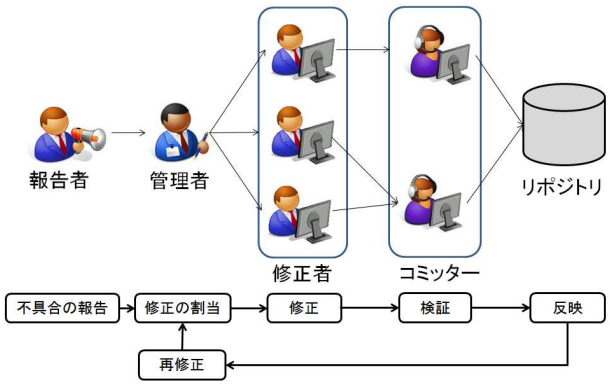

図 2 代表的な不具合修正プロセス

4.2 議論の分析

不具合の修正のために開発者同士で議論を行う．不具合が複数の機能に影響を及ぼす場合，開発者は各専門家からのフィードバックを受けることが重要である．しかしながら，多くの開発者が意見を出すと議論が発散することが考えられ，必ずしも多くの開発者が議論に参加することがよいとは限らないと考える．

また，各不具合票には CC リストと呼ばれる機能があり，報告された不具合が影響する機能に携わる開発者は CC リストに登録される．不具合の情報が新しく追加されたとき，CC リストに登録されている開発者に連絡が届くシステムになっている．CC リストに登録されている開発者は，不具合票に議論の追加など新しい情報が追加されたことをリアルタイムに把握することができるため，共同開発者予備軍とみなすことができる．

本稿では，再修正が発生する議論参加者，及び，CC リスト登録者を計測する．

5. ケーススタディ

本章では，Eclipse コミュニティにおける開発者の協調作業の分析について述べる．

5.1 分析対象データ

ケーススタディの対象は Eclipse プロジェクトの不具合

管理システム Bugzilla に 2001 年から 2009 年の間に報告された約 193,000 件の不具合のうち，SZZ アルゴリズム [10] によりコミットログに関連付けされた 15,651 件の不具合を取り扱う．コミットログとはリポジトリが更新される際の記録のことであり，どのソースコードがいつ誰によって更新されたかという情報を持つ．これを不具合管理システムの不具合報告に関連付ける事で，コミッターの活動を含めた詳細な情報を得ることができる．

5.2 修正プロセス作業の分析結果

図 3，図 4 にそれぞれ再修正が行われていない場合と再修正が行われた場合において，修正作業に参加した開発者数を示す．また，基本統計量を表 2 に示す．この 2 つの集合において，マン・ホイットニーの U 検定による有意な差 (有意水準 5%) がみられた．

この結果から，再修正が起きない不具合修正はその半数以上が 1 人もしくは 2 人の開発者によって行われるが，再修正が起きる不具合修正はその多くが少なくとも 2 人以上の作業者によって行われていることが分かる．従って，不具合修正作業に関わる開発者が 3 人以上となる場合は，再修正が起きる可能性が高いと判断できる．

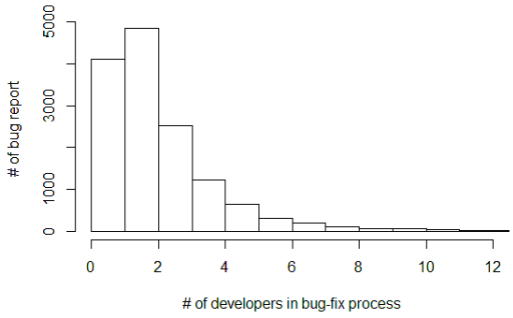

図 3 不具合修正の作業をする開発者の数（再修正なし）

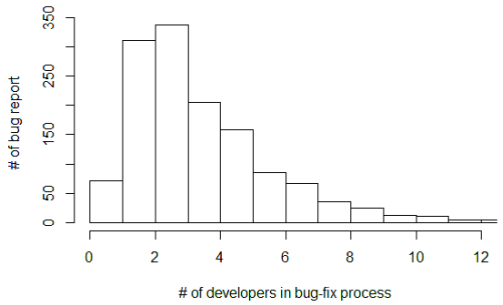

図 4 不具合修正の作業をする開発者の数（再修正あり）

表 2 基本統計量

	最小値	中央値	平均値	最大値	分散
再修正なし	1.00	2.00	2.55	76.0	6.65
再修正有り	1.00	3.00	4.14	35.0	8.44

5.3 議論の分析結果

前章で述べた通り，議論情報の分析について，議論に参加した開発者数，メーリングリストに登録された開発者数の 2 つの観点から調査した結果について以降で述べる．

5.3.1 議論に参加した開発者数

図 5，図 6 にそれぞれ再修正が行われていない場合と再修正が行われた場合の議論に参加した開発者数を示す．また，基本統計量を表 3 に示す．

この 2 つの集合において，マン・ホイットニーの U 検定による有意な差 (有意水準 5%) がみられた．再修正が起きない不具合修正の議論は 2 人以下によって行われることが多いが，再修正が起きた不具合修正の議論はその多くが 1~ 6 人によって行われている．結果より，3 人以上が議論に参加し始めた段階で再修正に注意して作業を行うべきであるという知見が得られる．

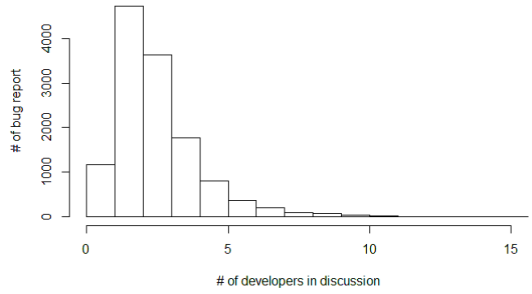

図 5 不具合修正の議論に参加した開発者の数（再修正なし）

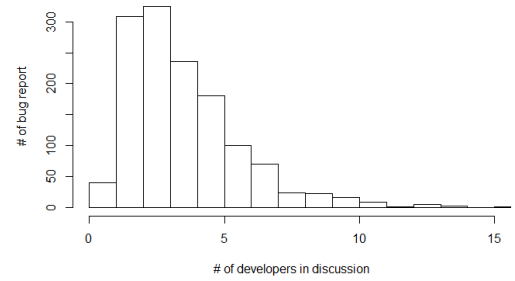

図 6 不具合修正の議論に参加した開発者の数（再修正あり）

表 3 基本統計量

	最小値	中央値	平均値	最大値	分散
再修正なし	1.00	3.00	3.03	44.0	3.39
再修正有り	1.00	4.00	4.12	31.0	6.46

5.3.2 CC リストに登録された開発者数

図 7，図 8 にそれぞれ再修正が行われていない場合と再修正が行われた場合の CC リストに登録された開発者の数を示す．また，基本統計量を表 4 に示す．

この 2 つの集合において，マン・ホイットニーの U 検定による有意な差 (有意水準 5%) がみられた．両者の結果において，CC リスに誰も登録されないまま不具合修正が

完了する場合が見られた．特に，再修正が起こらない不具合修正ではCCを使用しない場合が最多数となった．つまり一人で修正を完了させて，他の開発者からの助けを求める必要がないため，再修正が起きていないことが示唆される．一方，再修正が起きた不具合は登録者1人の場合が最も多いことが分かった．他人から変更に気づくように登録しているのは，他人からの修正，及び，フィードバックを期待したことが考えられる．しかし，実際他人からの修正やフィードバックがなかったために，再修正になったと考える．

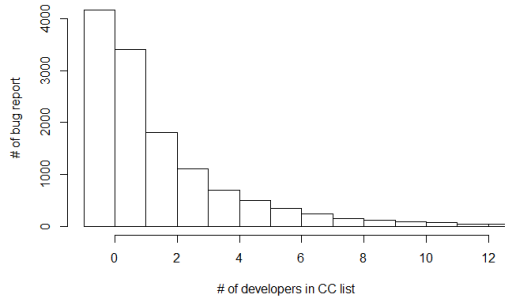

図 7　不具合修正における CC リストの登録者数（再修正なし）

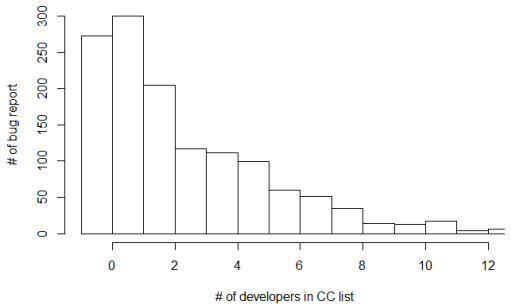

図 8　不具合修正における CC リストの登録者数（再修正あり）

表 4　基本統計量

	最小値	中央値	平均値	最大値	分散
再修正なし	0.00	1.00	2.62	76.0	11.6
再修正有り	0.00	2.00	3.28	36.0	16.6

6. 考察

本章ではケーススタディの結果を元に再修正を防ぐ方法について議論し，本稿の制約について述べる．

6.1 再修正を避けるには

前章では多くの開発者が修正作業や修正に関する議論に関わる場合において再修正が起きやすくなるという結果が得られた．その理由としては，再修正が起きる不具合は問題が複雑であるためその修正に多くの協調作業を必要とするからであると考えられる．したがって，前章の知見より再修正が起きやすいと判断された場合の不具合修正にはレビューやテストを重点的に行うことで修正内容を充分検証するべきであると考えられる．

6.2 制約

前章で検証した4つのパラメータが不具合修正プロセスにおいて，いつ判断材料として使用できるかについて述べる．

不具合の修正作業に関わる開発者の人数，議論に参加した人数と議論における発言回数に関しては，修正されたソースファイルがリポジトリに更新される段階で初めて正確に把握することができる．修正作業の途中で開発者の人数が3人を超えた場合は再修正が起こる可能性が高いと判断できるが，2人以下の場合には不具合修正プロセスが一度終了するまで再修正に関する判断ができない．

メーリングリストの登録人数に関しては修正作業が進むにつれて増えることも減ることもある点に注意が必要である．したがって，メーリングリストの情報に関しては，修正プロセスが一度終了するまで再修正の判断材料として利用することは難しい．

7. おわりに

先行研究において我々は，コミッターという一部の開発者の協調作業と再修正との関係を分析し，それらに関係があることを明らかにした．本稿では，開発者の協調作業と再修正の関係を分析し，より有用な知見を得ることを目指した．得られた知見を以下に示す．

- 不具合修正に関わる開発者の数が増えると再修正のリスクが高まる．特に，1人で不具合修正に取り組む場合には問題はなく，3人以上になると再修正が発生しやすくなる．
- 不具合管理システムにおける議論（議論に参加している人数，発言回数，メーリングリストに登録されている人数）も再修正の判断材料となる．特に，議論に3人以上が参加すると再修正が起こりやすくなると判断して良い．

最後に以下を今後の展望とする．

- 今回のケーススタディで対象としたコミュニティはEclipseのみである．一般的な議論をするためには，対象コミュニティを増やした上での分析が必要がある．
- 本稿で取り扱った議論の量や発言者数等は不具合修正終了時点において初めて正確に把握できるものである．不具合修正開始直後や修正途中における再修正に関する知見を得ることを目指す．
- 本稿ではコミュニケーションの形態についての定量的な値と再修正との関係を分析したが，コミュニケーションの内容については触れていない．より詳細な分析をするため，議論における発言の内容などを分析

する．

謝辞

本研究の一部は，文部科学省科学研究補助費（若手 B：課題番号 25730045）による助成を受けた．

参考文献

[1] R. Abreu and R. Premraj. How developer communication frequency relates to bug introducing changes. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution and software evolution (IWPSE-Evol'09) workshops*, pages 153–158, 2009.

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

[3] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann. Information needs in bug reports: improving cooperation between developers and users. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW'10)*, pages 301–310, 2010.

[4] Bugzilla. http://www.bugzilla.org/.

[5] P. Hooimeijer and W. Weimer. Modeling bug report quality. In *Proceedings of the 22nd International Conference on Automated Software Engineering (ASE'07)*, pages 34–43, 2007.

[6] G. Jeong, S. Kim, and T. Zimmermann. Improving bug triage with bug tossing graphs. In *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)*, pages 111–120, 2009.

[7] S. Matsumoto, Y. Kamei, M. Ohira, and K. Matsumoto. A comparison study on the coordination between developers and users in foss communities. In *Socio-Technical Congruence (STC 2008)*, number 8, pages 1–9, May 2008.

[8] RedMine. http://www.redmine.org/.

[9] E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K. Matsumoto. Studying re-opened bugs in open source software. In *Journal of Empirical Software Engineering*, 2012.

[10] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? In *Proceedings of the 2nd International Workshop on Mining software repositories (MSR '05)*, pages 1–5, 2005.

[11] Trac. http://trac.edgewall.org/.

[12] J. Zhou, H. Zhang, and D. Lo. Where should the bugs be fixed? - more accurate information retrieval-based bug localization based on bug reports. In *Proceedings of the International Conference on Software Engineering (ICSE'12)*, pages 14–24, 2012.

[13] 松. 健. 伊原 彰紀, 大平 雅雄. OSS 開発における不具合修正プロセスの現状と課題：不具合修正時間の短期可へ向けた分析. In **情報処理学会論文誌**, volume 6, pages 1–12, 2011.

[14] I. 情報処理推進機構）. 第 3 回オープンソースソフトウエア活用ビジネス実態調査 (2009 年度調査）. 2009.

[15] 飯尾淳, 清水浩之, 谷田部智之, and 比屋根一雄. 東アジア it 市場のオープンソースソフトウェアによる活性化. In **三菱総合研究所所報**, number 46, pages 52–65, 2006.