Vol.2013-GN-88 No.6
Vol.2013-SPT-5 No.6
2013/5/16

OSS開発におけるコミッターによる協調作業の一考察

林 宏徳[1,a)]　伊原 彰紀[1,b)]　門田 暁人[1,c)]　松本 健一[1,d)]

概要：昨今，ソフトウェア開発の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ている．OSS 導入時には，対象プロジェクトのリリース後における保守体制を理解することが重要であり，OSS 開発の不具合修正プロセスを分析する研究が進んでいる．不具合修正プロセス中のコミッターによるレビュー作業は，ソフトウェアをリリースする直前の作業であり，ソフトウェアの品質に最も影響を与える．したがって，コミッターの不十分なレビュー作業は開発の手戻り（再修正）を引き起こし作業効率の低下に繋がる．そこで本稿はコミッターのレビュー作業による OSS 品質への影響について考察する．

1. はじめに

昨今，ソフトウェア開発の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ている．OSS 利用者にとっては，継続してサポートが受けられる事が重要であるため，導入時にプロジェクトの保守体制を理解する必要がある．従来研究では，OSS プロジェクトの保守体制を評価する技術として，不具合修正プロセスを分析する手法 [1][4][5] が提案されている．特に，不具合修正プロセス中の主要開発者（コミッター）によるレビュー作業は，ソフトウェアをリリースする直前の作業であり，ソフトウェアの品質に最も影響を与える作業といっても過言ではない．コミッターの不十分なレビュー作業は開発の手戻り（再修正）を引き起こし作業効率の低下に繋がる．

再修正は一般的に，変更されたファイルが的確にレビューされていれば防ぐことができる．しかしながら，プロジェクトに報告される不具合のうち，約 15%の不具合は再修正が発生している [6]．これまでの研究では，不具合の再修正を防ぐために，修正直後に再修正が引き起こされるか否かを予測する技術が提案されている [6]．当該技術では，不具合修正プロセスにおける再修正の有無を 4 つの要因（作業内容，不具合報告，不具合修正，修正担当者）を用いて判断しており，開発者の協調作業を考慮していない．しかし，OSS 開発には多くの開発者間の協調作業が求められており，その作業形態が再修正に影響していることも十分に考えられる．例えば，協調作業における意思疎通に問題がある場合，レビューの見落とし等が起こる可能性等が考えられる．

また近年，複数の機能に影響する不具合修正には，複数人のコミッターによるレビュー作業が行われている．複数人のコミッターによる協調作業は，意思疎通に問題があると，レビューの見落とし等が発生すると考えられる．

本稿では，コミッターの協調作業に着目し，その必要性と，再修正との関係について調査する．これにより，OSS の品質をコミュニティからも判断できるようになると期待される．

2. 対象とする協調作業

本章では，OSS 開発の不具合修正プロセスにおいて本研究で対象とする協調作業を説明する．図 1 は一般的な OSS の不具合修正プロセス [4] を示す．このプロセスでは先ず，(a) 報告者は不具合を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) 修正者により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．本稿では特に，コミッターと呼ばれる開発者を対象とし，分析を行う．コミッターは主にレビューやリポジトリへのアクセスを担当しており，コミュニティの中でも重要な役割を担っている．また，不具合の修正のために多くのソースコードを変更する場合には，複数のコミッターによる協調作業が必要であると考えられる．プロジェクトの意思決定に関わるコミッター間の協調作業を分析対象とすることで，OSS の保守体制を見極めることができると考えられる．

[1] 奈良先端科学技術大学院大学
Takayama-chou, Ikoma-shi, Nara 101–0062, Japan
[a)] hironori-ha@is.naist.jp
[b)] akinori-i@is.naist.jp
[c)] akito-m@is.naist.jp
[d)] masumoto@is.naist.jp

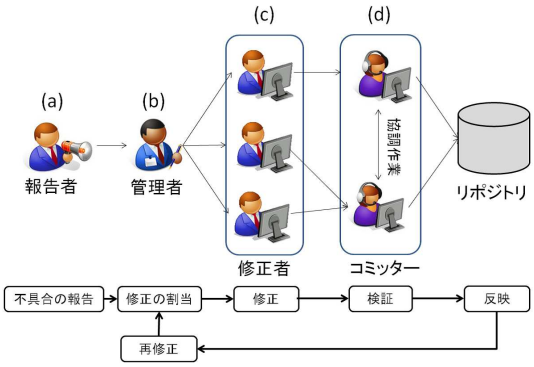

図 1 代表的な不具合修正プロセス

3. 実験

本章では，OSS 開発におけるコミッターの協調作業を 3 つの観点から検証する．対象は Eclipse プロジェクトのバグ管理システム Bugzilla に 2001 年から 2009 年の間に報告された約 193,000 件の不具合報告のうち，この報告は SZZ アルゴリズム [7] によりソースコードに関連付けされた 15,651 件の不具合を取り扱う．

RQ1: コミッターの知識領域はどの程度あるのか.

[**動機**] RQ1 では協調作業の要因として，コミッターの知識領域について調査する．コミッターの知識領域が全網羅的である場合，彼らは知識を補うために協調作業をする必要がなく，その他に協調作業の理由があると判断できる．

[**手順**] 本稿ではコミッターの知識領域を開発者が関わったことのある機能[*1]として定義する．知識領域内に存在する機能数をコミッター毎に調べることで，コミッターのプロジェクト理解を定量的に示す．

[**結果**] 図 2 にコミッターの持つ機能数の度数分布を示す．この表から，およそ 50%のコミッターの知識領域は 1~ 2 機能程度であるということが分かった．したがって，不具合修正が複数機能に影響する場合，コミッターの協調作業が必要とされる可能性が高くなると予想される．

RQ2: 不具合修正には何人のコミッターが必要か.

[**動機**] 不具合修正が 1 機能の変更のみで完了する場合，コミッターの知識領域が限定的であったとしても，協調作業は必要ではない．そのため，RQ2 では不具合修正の影響範囲と修正に関与するコミッターの人数を調査する．

[**手順**] 不具合修正のために必要な機能数で分類する．次に，それぞれの不具合修正に必要なコミッターの人数を調査し，不具合修正を図 3 に示されるような 4 種類の不具合修正パターンに分類する．

[*1] 本稿では，機能をシステムを構成するルートディレクトリ直下のサブディレクトリの数と定義する．これにより，Eclipse project の機能は “org.eclipse.team.ui”, “org.eclipse.team.core”, “org.eclipse.swt” 等の 89 機能に分割される．

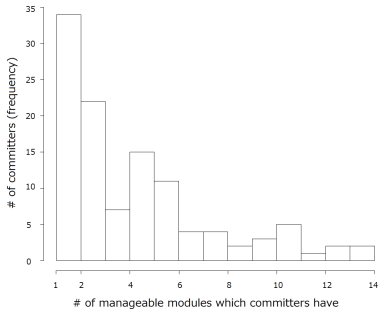

図 2 コミッターの取り扱う機能数

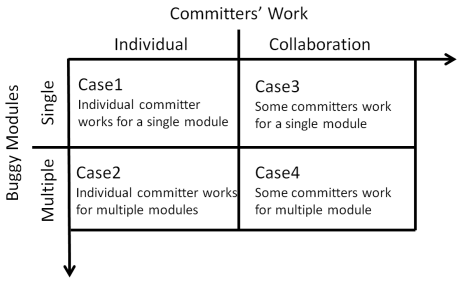

図 3 不具合修正パターン

Case1 コミッターが協調作業することなくひとつの機能を取り扱う場合を示す．

Case2 協調作業をすることなく複数機能を取り扱う場合を示す．

これらの場合，コミッターは不具合修正に必要な知識を全て有していると言える．

Case3 コミッターは協調作業をしながらひとつの機能を取り扱う場合を示している．この場合，不具合修正に必要な知識が少ないにもかかわらず，コミッターの知識が不足していると見なすことができ，我々の仮説では Case 3 は Case 1 よりも稀なケースとなる．

Case4 はコミッターが協調作業をしながら複数機能を取り扱う場合である．この場合に関しても，我々の仮説に基づけば Case 2 よりも多くなるはずである．

[**結果**] 各 Case が全体に占める数と割合を表 1 に示す．この結果から，ひとつの機能における不具合修正においては，個人による作業が協調作業のおよそ 5 倍となる事が分かった．一方，複数機能の不具合修正においては，協調作業の割合が個人による作業の割合のおよそ 2 倍という結果となった．加えて，図 4 に欠陥を含む機能が 1 機能の修正に必要とされるコミッターの人数，図 5 に欠陥を含む機能数が複数の修正に必要とされるコミッターの人数をそれぞれ示す．欠陥を含む機能数が単数の場合，平均 1.28 人のコミッターが必要とされ，複数の場合，平均 2.18 人が必要とされる．また，マン・ホイットニーの U 検定では，これらの分布に有意差（有意水準 5%）が認められた．

これらの結果から，不具合修正が複数の機能の変更を必要とする場合にコミッターの協調作業が起こるということが分かる．

表 1 バグ修正の割合

		committers' work		total
		individual	collaborative	
buggy modules	single	(Case1) 75.68% 11,845	(Case3) 15.72% 2,460	91.40% 14,305
	multiple	(Case2) 2.63% 411	(Case4) 5.97% 935	8.60% 1,346
total		78.30% 12,256	21.70% 3,395	15,651

RQ1 と RQ2 の結果から，コミッターの取り扱う事のできる機能数は少ないため，それらを補うために協調作業が行われると示唆される．

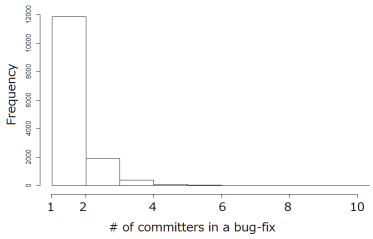

図 4 単機能の修正に必要なコミッターの人数

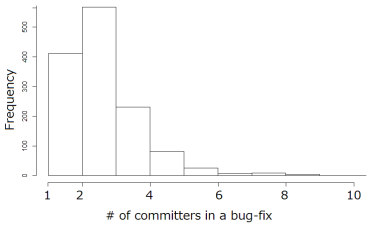

図 5 複数機能の修正に必要なコミッターの人数

RQ3: コミッターの協調作業は成功しているのか.
[**動機**] コミッターの協調作業と再修正の関係について明らかにする事で，ソフトウェアの品質をコミュニティから判断できるか否かを結論付ける．
[**手順**] 不具合修正の難易度は修正が必要な機能数にも依存し，再修正の起こる割合にも影響すると考えられる．したがって，RQ3 においても修正が必要な機能数を RQ2 で示した図 3 を考慮し，再修正が起こる確立を計算する．但し，ここで再修正とはバグ管理システム上の修正の過程で “fixed(修正済み)” の状態に 2 回以上遷移した不具合とする [3].
[**結果**] 表 2 に再修正の起きた件数とその割合を示す．Case1，Case2 と Case3，Case4 の比較より，コミッターによる協調作業が行われると再修正が起こる可能性が高くなることがわかる．

表 2 不具合修正の作業形態

		committers' work	
		individual	collaborative
buggy modules	single	(Case1) 6.41% 759	(Case3) 15.98% 393
	multiple	(Case2) 11.44% 47	(Case4) 16.36% 153

4. 考察

従来研究 [2] では開発者の知識はプロジェクト全体の10%以内に限られていることが明らかになっている．コミッターの場合においても，RQ1 の結果より同様の結果が得られた．コミッターは一般開発者に比べ参加期間が長く，プロジェクトへの理解が深いとされているが [9][8]，全体に対して把握している知識領域は一般開発者と同等であることが分かった．

また，各 RQ より，不具合修正が複数機能に及ぶ場合，コミッター間の協調作業が求められ，協調作業が行われると再修正が起こりやすくなるという事が分かった．この結果から，再修正の有無を知るためには，ソースコードの凝集度と修正に関わるコミッターの人数を調べれば良いと言える．従来の研究では主にソースコードの特徴量によってソフトウェアの品質が判断されてきたが [6]，この結果から，OSS コミュニティもソフトウェアの品質を判断する材料になり得る．

5. おわりに

本稿では，OSS の保守体制を理解する判断材料のひとつとして，不具合修正プロセスにおけるコミッターの協調作業を調査した．検証の結果，コミッターの協調作業は互いの知識を補うために必要であり，その協調作業はソフトウェアの品質に関わるという事が確認できた．

最後に以下を今後の展望とする．

- 再修正とコミッター間との協調作業の関係だけでなく，協調作業とソースコードの特徴量との関係について調査する．
- 再修正とコミッター間の協調作業の関係について，具体的な原因を言及する．
- 得られた協調作業と再修正の関係から，再修正の予測モデルの構築を試みる．
- コミッターの知識に基づく担当者の推薦モデルの構築を試みる．
- 対象プロジェクトを Eclipse project だけでなく，Android project や Mozilla Firefox project 等にも拡張し，一般的な議論を目指す．

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．
また，本研究の一部はオウル大学のインターンシッププ

ログラムで行われたものである．受け入れて頂いたMakku Oivo氏に深謝する．

参考文献

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

[2] Cui, X., Beaver, J., Stiles, E., Pullum, L., Klump, B., Treadwell, J. and Potok, T.: The Swarm Model in Open Source Software Developer Communities, *Proceedings of the 2nd International Conference on Social Computing*, pp. 656–660 (2011).

[3] Ihara, A., Ohira, M. and Matsumoto, K.: Differences of Time between Modification and Re-Modification: An Analysis of A Bug Tracking System, *Proceedings of the 3rd International Workshop on Knowledge Collaboration in Software Development (KCSD'09)*, pp. 17–22 (2009).

[4] Jeong, G., Kim, S. and Zimmermann, T.: 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 (online), DOI: http://doi.acm.org/10.1145/1595696.1595715 (2009).

[5] Matsumoto, S., Kamei, Y., Monden, A., Matsumoto, K. and Nakamura, M.: An Analysis of Developer Metrics for Fault Prediction, *Proceedings of the 6th International Conference on Predictive Models in Software Engineering (PROMISE'10)*, pp. 1–9 (2010).

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

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

[8] Ye, Y., Yamamoto, Y. and Kishida, K.: Understanding knowledge sharing activities in free/open source software projects: An empirical study, *Proceedings of the Asia Pacific Software Engineering Conference*, pp. 472–481 (2004).

[9] Ye, Y. and Kishida, K.: Toward an Understanding of the Motivation Open Source Software Developers, *in Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*, pp. 419–429 (2003).