Vol.2013-SE-181 No.6
2013/7/18

OSS開発における管理者と開発者間の社会的関係がタスク遂行に与える影響の考察

吉行　勇人[1,a)]　大平　雅雄[1,b)]

概要：OSS 開発における不具合修正タスクの割当に関する先行研究では，Anvik ら [2] の研究のように，効率的なタスク割当を支援する手法が管理者の視点から提案されている（例えば，ある不具合を修正するのに最も適した開発者を推薦する，など）．本論文では，不具合修正タスクの割当のプロセスをより深く理解するために，開発者の視点から調査する．大規模 OSS 開発プロジェクトでは，過半数の不具合修正がそのプロジェクトに参加する少数の開発者によって行われている [18]．特定の少数開発者らはしばしば，複数の管理者から割り当てられた不具合修正タスクを同時に処理する必要があるため，プロジェクト全体としては不具合修正が滞る場合も起こりえる．また，開発者個人の技術的要素だけでなく，管理者と開発者の社会的関係も，多数の不具合修正タスクに取り組む開発者のタスク優先順位に影響を与えていると考えられる．
本論文では，Eclipse Platform プロジェクトを対象に行ったケーススタディについて報告する．ケーススタディの結果，「開発者個々人の不具合修正時間は，タスクがどの管理者から割り当てられたかに依存する場合がある」ということが分かり，より効果的なタスク割当支援のためには，管理者のみではなく開発者の視点も導入する必要があることが分かった．

1. はじめに

Mozilla Firefox，Apache OpenOffice，Android といったオープンソースソフトウェア (OSS) は，商用ソフトウェアの代替肢としてだけでなく，スマートフォンや組み込みシステムのような，ソフトウェアを中核とする製品の開発期間を短縮する手段として広く使われている．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 プロジェクトでは，エンドユーザと開発者の両方から日々多数の不具合報告を受けていることが問題となっている．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もある．プロジェクトに参加する開発者は，新しい不具合報告を受けると，その不具合を再現できるかどうかや過去に同じ不具合報告がされていないか（duplicate bugs）を確認し，その不具合を修正するのに最も適した開発者に不具合修正タスクを割り当てる必要がある．

不具合修正タスクの割当プロセスを改善するために，先行研究では，タスク割当の自動化 [2, 12, 16]，重複した不具合報告の検知 [19, 21, 22]，不具合修正タスクの再割当の原因分析 [13, 14, 20] など，多数の手法が提案されている．本研究では，先行研究と同様に，不具合修正プロセスにおける効率的なタスク割当支援を目的としているが，特に不具合修正タスク割当における社会性に着目して不具合修正プロセスに関する新たな知見を導出することを目的としている．

不具合修正タスク割当に関する先行研究の多くは，Anvik ら [2] の *“who should fix this bug?”* の考え方に基づいている．これは，個々の不具合修正タスクに対して適任の開発者を特定することを指しており，主に多数のタスクを割り当てる必要のある管理者を支援するためのものである．実際，Eclipse Platform プロジェクトにおける 44%の不具合は，不具合修正を担当する開発者が変更されて（再割当が行われている）おり [16]，支援の必要性には妥当性がある．

本研究では，Eclipse Platform プロジェクトにおける不具合修正プロセスを，多大なる貢献を行う少数の開発者の視点から調査している．Anvik らの *“Who should fix this bug?”* [2] は，修正タスクを割り当てる管理者の視点から着想を得たものであるが，実際の多くの不具合修正は少数の開発者によって行われており [18]，より良いタスク割当支援を行うためには，開発者の視点から不具合修正プロセスを捉える必要があると考えたためである．

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

前述したように，特定の開発者は，複数の管理者から割り当てられた不具合修正タスクを同時に処理しなければならない．その際，開発者がどのタスクを優先して完了させるかという選択には，開発者と管理者の社会的関係が影響している可能性がある，というのが本研究の立場である．

本論文では，Eclipse Platform プロジェクトにおけるケーススタディをもとに，以下の 3 つの結論を得た．

- 管理者と開発者の社会的関係の強さ（タスク割当の実績など）は，不具合修正時間の短縮化に必ずしも寄与しない．
- どの管理者からタスクを割り当てられたかが，開発者のタスク選択の優先順位とタスク完了時間（不具合修正時間）に影響を与えている可能性がある．
- 不具合修正タスク割当支援の改善には，開発者の視点からも不具合修正プロセスを考慮する必要がある．

本論文の構成は以下の通りである．続く 2 章では，関連研究と本研究の動機について述べる．3 章では，Eclipse Platform プロジェクトにおけるケーススタディとその結果について述べる．4 章では，ケーススタディの結果とその妥当性について議論する．最後に今後の課題について述べ本論文をまとめる．

2. 関連研究

ほとんどの先行研究では，管理者の視点から不具合修正タスクの割当を支援する手法を提案されている．不具合修正タスクの割当支援は，技術的側面と社会的側面からアプローチを分類できる．以下では，先行研究のそれぞれのアプローチと本研究の動機について説明する．

2.1 技術的側面

2.1.1 タスク割当の自動化

不具合修正タスクがある開発者に割り当てられた時，その開発者によってタスクを完了できずに，別の開発者に再度タスクが割り当てられることがある (reassigned or tossed [16])．これは，管理者がそのタスクに適任でない（知識やスキルが十分でない）開発者に割り当てを行ったことといより発生する．実際，Eclipse と Mozilla プロジェクトの 37%から 44%の不具合は，不具合修正を担当する開発者が変更されている（再割当が行われている）[16]．タスクの再割当は，不具合修正時間が増大する要因の一つとなっており，不具合修正に最も適した開発者を選び再割当を防ぐことが，不具合修正時間を削減するための効率的な方法とされている．

多くの先行研究 [1,2,8,12–14,16,17,20] が，この問題に取り組んでいる．Anvik ら [2] は，自然言語処理を用いて，過去の不具合報告から適任の開発者を推薦する手法を提案している．Jeong ら [16] は，開発者間の関係性を表すソーシャルグラフを用いて，不具合修正タスクの割当手法を提案している．他にも，複数回の再割当が発生する理由に関する分析 [14] や，再度不具合となる （reopen される）不具合報告を予測する手法 [13,20]，などが提案されている．

2.1.2 重複する不具合報告の検知

不具合を報告するユーザは，他のユーザによって報告済みの不具合あるいは修正済みの不具合を再度重複して報告することがある．これは，不具合管理システム (BTS: Bug Tracking System) には膨大な数の不具合報告が登録されているため，検索機能があるにもかかわらずすべての報告に目を通すことができないことが原因である．重複した不具合報告であることに気付かなければ，開発者らはすでに解決済みの問題を再度解決しようとするため，結果的に開発者の労力が無駄になることもある．

重複する不具合報告を防ぐために，重複不具合報告の自動検知手法がいくつか提案されている [3,19,21,22] ．例えば，Wang ら [22] は，BTS 上の不具合報告に対して自然言語処理を用いて，重複した不具合報告を検知する手法を提案している．

2.1.3 不具合報告作成支援

「良い」不具合報告は，開発者が不具合の原因を特定したり再現するのを助ける．そのため，良い不具合報告は不具合修正時間の短縮化に役立つ．しかしながら，特にエンドユーザが不具合報告をする場合，不具合を修正するために必要な情報について熟知していないため，開発者が求める情報を適切に報告できない場合がある．このような場合，開発者はユーザと何度もやり取りをしながら，不具合の特定や修正に必要な情報を聞き出す必要があり，結果的に不具合修正に時間がかかることが多い．

開発者とユーザのやり取りを改善するために，多くの先行研究 [5–7,11,15,25] では，開発者が求める不具合修正に必要な情報について，インタビューなどから分析している．例えば，Apache，Eclipse，Mozilla プロジェクトに参加する 150 人以上の開発者と 300 人以上の不具合報告者にインタビューした結果，Bettengurg ら [6,25] は，不具合を再現するための手順とスタックトレースが非常に重要な情報であると報告している．

2.2 社会的側面

2.2.1 社会的構造

ソフトウェア開発は本質的に，協調作業が求められる活動である．そのため，「良いチーム」のタスク遂行能力は，そうでないチームに比べ高いはずである．ソフトウェア開発の社会的／組織的構造を分析することにより，ソフトウェアの生産性や品質に影響を与える良い構造／悪い構造があることが分かる．

ソフトウェア開発における社会的構造を分析した先行研究は数多く存在する [4,9,10,23,24]．例えば，Bird らは，5 つのオープンソースプロジェクトのサブコミュニティを

調査し，強いつながりを持つサブコミュニティには協調性が存在することを明らかにしている [10].

2.2.2 本研究の動機：マイクロソーシャルネットワーク分析

前述の通り，不具合修正タスクの割当プロセスを技術的側面から支援する先行研究では，BTS に記録されている不具合報告に対してデータマイニング手法を適用している．先行研究の主たる目的は，不具合修正タスク割当プロセスを支援することによって不具合修正時間を削減することにあり，管理者の観点 (*"Who should fix this bug?"*) に基づいている．

また，不具合修正タスクの割当プロセスにおける社会的側面を分析した先行研究では，ソフトウェア開発にはステークホルダーの社会的関係が重要であることが示されている．しかしながら一般的に，ソーシャルネットワーク分析 (SNA) では，組織全体の社会構造しか捉えることができないため，開発者個々人の単位での社会的関係を分析するのは不向きである．

そこで本研究では，より細粒度でのソーシャルネットワーク分析（マイクロソーシャルネットワーク分析）を提案する．本研究のマイクロソーシャルネットワーク分析は，ソフトウェア開発に関わる個人同士の関係に着目し，その影響を考える．特に，本研究では，先行研究で中心となっていた管理者視点での考え方ではなく，新たな視点として開発者の視点での考え方を提案する．図 1 では，先行研究と本研究との視点の違いを表している．

前述の通り，過半数の不具合は，プロジェクトに参加する少数の開発者によって修正されており [18]，開発者は，様々な管理者から割り当てられた不具合修正タスクを同時に処理しなければならない．そのため，管理者が特定の不具合を修正するのに適任である開発者を見つけられたとしても，その開発者は，他の不具合修正タスクを担当しているなどして十分な時間が無く，その不具合に対してすぐには修正を始められない可能性がある．さらに，開発者の技術的要素だけでなく，管理者と開発者の社会的関係も，開発者の不具合修正タスク遂行能力に影響を与えていると考えられる．つまり，開発者が，複数の管理者からタスクを割り当てられた際に，どのタスクを優先して完了させるかという選択において，開発者と管理者のこれまでの関係性が影響している可能性がある，ということである．

管理者と開発者両方の視点から，不具合修正プロセスにおける社会的関係が不具合修正効率に与える影響を確かめるために，本研究では，以下の 2 つの Research Question を設定した．

RQ1: 管理者と開発者の社会的関係が，不具合修正時間に影響を与える

RQ2: どの管理者がタスクを割り当てたかどうかが，開発者のタスク完了時間（不具合修正時間）に影響を与える

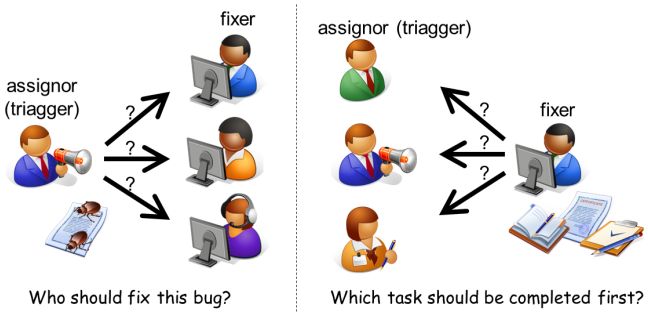

図 1 先行研究と本研究の視点の違い
Fig. 1 Difference of perspectives between existing studies and our study.

次章では，この 2 つの Research Question に答えるために行ったケーススタディについて述べる．

3. ケーススタディ

本章では，本研究で行った Eclipse Platform プロジェクトに対して行ったケーススタディについて説明する．

3.1 データセット

本ケーススタディでは, Bugzilla に記録されている Eclipse Platform プロジェクトの不具合報告をデータセットとして用いた．元となるデータは，2001 年 10 月から 2012 年 10 月までに報告された不具合報告のうち，担当した開発者／管理者が特定できた 52,593 件の不具合報告である．このデータに対して，データクリーニングとデータフィルタリングを行った．データクリーニングは，複数のメールアドレスを用いる開発者を特定する目的がある．データフィルタリングでは，複雑な要因をできるだけ排除し分析をできるだけ簡素にするために，再割当が発生せず一度で修正が完了している不具合報告のみに注目した．この結果，最終的なデータセットは 20,422 件の不具合報告となった．

このケーススタディから，前述の 2 つの Research Question に答えるための分析を行った．

3.2 社会的関係：管理者から見た視点

まず，第 1 の Research Question は，従来の，管理者から見た視点での問題である．

RQ1: 管理者と開発者の社会的関係が，不具合修正時間に影響を与える

ある管理者とある開発者のペアが，共同でタスクを遂行した経験が十分にあるならば，管理者はその開発者の持つ知識やスキルを熟知していると考えられる．したがって，管理者が適任の開発者に不具合修正タスクを依頼すれば，不具合修正時間が短縮できるはずである．もしこれが事実ならば，*"Who should fix this bug?"* という問いに対して

は，お互いを熟知している管理者–開発者ペアにタスク割り当てるという答えを得ることができる．

そこで，本ケーススタディでは，最も不具合修正タスクを割り当てた回数が多い管理者の上位 5 名を選出し，それぞれの管理者が最も多くの不具合修正タスクを依頼した開発者上位 5 名を選出した．さらに，それぞれの管理者が，それぞれの開発者にタスクを割り当てた回数と，不具合修正タスク完了にかかった日数の中央値を算出した．

その結果を，表 1 ならびに表 2 に示す．

表 1 は，割当総数上位 5 名の管理者の割当総数と修正日数の中央値を表している．表 1 から，管理者 A_a と管理者 A_d が割り当てた不具合修正タスクは，短時間（1 時間以下）で完了していることが分かる．一方，管理者 A_b が割り当てた不具合修正タスクは，約 26 日もの時間がかかっていることがわかる．

表 2 は，割当総数上位 5 名の管理者が，タスクを依頼した回数の多い開発者上位 5 名に割り当てた回数と，不具合修正にかかった日数の中央値を表している．表 2 から，管理者 A_a と管理者 A_d，管理者 A_e は，上位 5 名の開発者の不具合修正日数が，表 1 で示した全体の不具合修正日数と比べて同じくらいか，それ以下であった．

それに対して，管理者 A_b と管理者 A_c は，上位 5 名の開発者に割り当てたときの不具合修正日数が，表で示した全体の不具合修正日数に比べて多かった．例えば，管理者 A_c の，全体の不具合修正日数の中央値は 8.27 日であったが，管理者 A_a がタスクを依頼した開発者 5 名のうち，全体の不具合修正日数の下回ったのは $F11$ のみであった．つまり，タスクを依頼した回数が多く開発者のスキルをよく知っていたとしても，必ずしも不具合修正時間に影響を与えるものであるとは限らないことが分かった．また，表 1 から，タスクを割り当てた総数と不具合修正時間との間には，相関があるとは言えないことも分かった．

3.3 社会的関係：開発者から見た視点

第 2 の Research Question は，本研究の主たる目的である．

RQ2: どの管理者がタスクを割り当てたかどうかが，開発者のタスク完了時間（不具合修正時間）に影響を与える

表 2 では，同一人物を含む 25 人の開発者が登場するが，これらの開発者は，表 2 に登場する 5 名の管理者以外の管理者からも，不具合修正タスクを依頼されている．タスクを依頼された管理者によって，開発者の不具合修正タスク遂行能力が異なっていれば，Anvik らの *“Who should fix this bug?”* だけでなく，*“Who should assign this bug?”* という考え方が必要であることが分かるはずである．そこで，様々な管理者にタスクを依頼されたときの管理者のタスク完了時間（不具合修正時間）を開発者毎に比較する．

表 1 タスク割当総数上位 5 名の管理者のタスク割当総数と修正日数
Table 1 Total number of assignments by the top 5 assignors and median days to complete the assignments

管理者	割当総数	修正日数
A_a	1,529	0.03
A_b	1,500	26.09
A_c	1,343	8.27
A_d	1,186	0.00
A_e	895	8.08

表 2 タスク依頼回数上位 5 名の管理者がそれぞれの開発者にタスクを依頼した回数と修正日数
Table 2 Top 5 assignors' task assignments and median days to be fixed by assignees (fixers)

管理者 → 開発者	タスク依頼回数	修正日数
$A_a \rightarrow F1$	290	0.00
$A_a \rightarrow F2$	224	0.00
$A_a \rightarrow F3$	242	0.02
$A_a \rightarrow F4$	457	0.04
$A_a \rightarrow F5$	187	0.08
$A_b \rightarrow F6$	428	20.54
$A_b \rightarrow F7$	288	26.09
$A_b \rightarrow F8$	138	27.33
$A_b \rightarrow F9$	158	35.04
$A_b \rightarrow F10$	103	85.10
$A_c \rightarrow F11$	262	2.93
$A_c \rightarrow F12$	266	13.88
$A_c \rightarrow F13$	71	15.26
$A_c \rightarrow F14$	64	17.12
$A_c \rightarrow F15$	64	28.81
$A_d \rightarrow F16$	180	0.00
$A_d \rightarrow F17$	270	0.00
$A_d \rightarrow F18$	92	0.00
$A_d \rightarrow F19$	255	0.00
$A_d \rightarrow F20$	91	1.00
$A_e \rightarrow F21$	45	2.94
$A_e \rightarrow F22$	53	4.15
$A_e \rightarrow F23$	89	4.71
$A_e \rightarrow F24$	144	5.50
$A_e \rightarrow F25$	77	5.62

まず，最も不具合修正タスクを完了した回数が多い開発者の上位 5 名を選出し，それぞれの開発者に最も多くの不具合修正タスクを依頼した管理者上位 5 名を選出した．次に，それぞれの開発者が，それぞれの管理者からタスクを依頼された回数と，不具合修正タスク完了にかかった日数の中央値を算出した．

その結果を，表 3 と表 4 に示す．

表 3 は，タスク遂行回数上位 5 名の開発者の，タスク遂行回数と，修正日数の中央値を表している．表 3 より，開

発者 F_a と開発者 F_d は，1 日以下でタスクを完了している一方で，開発者 F_c は，タスク遂行に 20 日以上かかっていることがわかる．

表 4 は，タスク遂行総数上位 5 名の開発者が，いままでにタスクを依頼された回数の多い管理者上位 5 名にタスクを依頼された回数と不具合修正にかかった日数の中央値を表している．表 4 より，タスクを依頼した管理者によって，表 3 で示した全体の修正日数よりも修正日数が短い場合も長い場合もあった．例えば，表 3 では，開発者 F_b の全体の修正日数は 8.17 であるが，管理者 $A9$ にタスクを依頼されたときは，全体の修正日数以上かかっている．同様に，開発者 F_e に対して，管理者 $A22$ がタスクを依頼した場合も，全体の修正日数よりも長い時間がかかっている．さらに，開発者 F_b における管理者 $A9$，開発者 F_e における管理者 $A22$ は，ともに，それぞれの開発者に対して最も多くのタスクを依頼した管理者である．

また RQ1 と同様に，タスク遂行回数と修正日数に相関があるとは言えなかった．開発者は，よく知っている管理者によってタスクを依頼されたときに良いパフォーマンスを発揮する訳ではないが，特定の管理者にタスクを依頼された場合は良いパフォーマンスを発揮することがあることが分かった．

4. 考察

本章では，ケーススタディにより得られた結果と，本研究の妥当性のについてそれぞれ考察する．

4.1 不具合解決総数 vs. 時間効率

Research Question2 で述べた通り，全体の修正時間と比べて，不具合修正に長い時間がかかっているペアが存在することが分かった．

例えば，表 2 を見ると，管理者 A_c（表 4 における $A9$）は，開発者 $F12$（表 4 における F_b）に，もっとも多くタスクを依頼している．また，表 4 を見ると，開発者 F_b は，管理者 $A9$ から，もっとも多くタスクを依頼されている．よって，このペアは，互いに知識やスキルを熟知していると考えられるが，このペアの不具合修正日数を見ると，全体の不具合修正日数よりも，時間がかかっている．

図 2 は，管理者 A_c がタスクを依頼したことのある開発者計 24 名と，開発者 F_b にタスクを依頼したことがある管理者計 18 名の不具合修正日数を表している．図 2 の上部をみると，開発者 F_b は，管理者 A_c がタスクを依頼したことのある開発者のうち，不具合修正日数が 9 番目に短い．また，図 2 の下部をみると，管理者 A_c は，開発者 F_b にタスクを依頼したことのある管理者のうち，不具合修正日数が 12 番目に短い．管理者 A_c と開発者 F_b のペアは，十分な回数不具合を修正しているが，不具合修正には他のペアよりも長い時間がかかっており，効率的なペアであるとは言えない．

表 3 タスク遂行総数上位 5 名の開発者の割当総数と修正日数
Table 3 Total number of bugs assigned to the top 5 fixers and median days to fix the bugs

開発者	タスク遂行総数	修正日数
F_a	1,231	0.17
F_b	1,088	8.17
F_c	952	20.06
F_d	901	0.13
F_e	864	13.18

表 4 タスク遂行回数上位 5 名の開発者がそれぞれの管理者からタスクを依頼された回数と修正日数
Table 4 Top 5 fixers' assigned bugs and median days to fixe the bugs

開発者 ← 管理者	タスク遂行回数	修正日数
$F_a \leftarrow A1$	457	0.04
$F_a \leftarrow A2$	127	0.16
$F_a \leftarrow A3$	120	0.21
$F_a \leftarrow A4$	171	0.78
$F_a \leftarrow A5$	156	0.94
$F_b \leftarrow A6$	80	3.50
$F_b \leftarrow A7$	144	5.50
$F_b \leftarrow A8$	154	6.01
$F_b \leftarrow A9$	266	13.88
$F_b \leftarrow A10$	69	36.06
$F_c \leftarrow A11$	89	6.97
$F_c \leftarrow A12$	428	20.54
$F_c \leftarrow A13$	71	21.97
$F_c \leftarrow A14$	171	22.00
$F_c \leftarrow A15$	115	39.29
$F_d \leftarrow A16$	290	0.00
$F_d \leftarrow A17$	255	0.00
$F_d \leftarrow A18$	159	4.13
$F_d \leftarrow A19$	67	7.89
$F_d \leftarrow A20$	52	8.04
$F_e \leftarrow A21$	44	5.44
$F_e \leftarrow A22$	288	26.09
$F_e \leftarrow A23$	12	38.67
$F_e \leftarrow A24$	118	60.38
$F_e \leftarrow A25$	72	61.15

そこで，前述のペアとの比較として，同一の開発者で，タスクを依頼した管理者の異なるペアに着目する．表 2 を見ると，管理者 A_e（表 4 における $A7$）は，開発者 $F24 = F12$（表 4 における F_b）に，もっとも多くタスクを依頼している．また，表 2 を見ると，開発者 F_b は，管理者 $A7$ から，3 番目に多くタスクを依頼されている．管理者 A_e とペアは，前述のペアに比べてタスクを遂行した回数は少ないが，不具合修正日数を見ると，全体の不具合修正日数よりも短い時間でタスクを完了している．

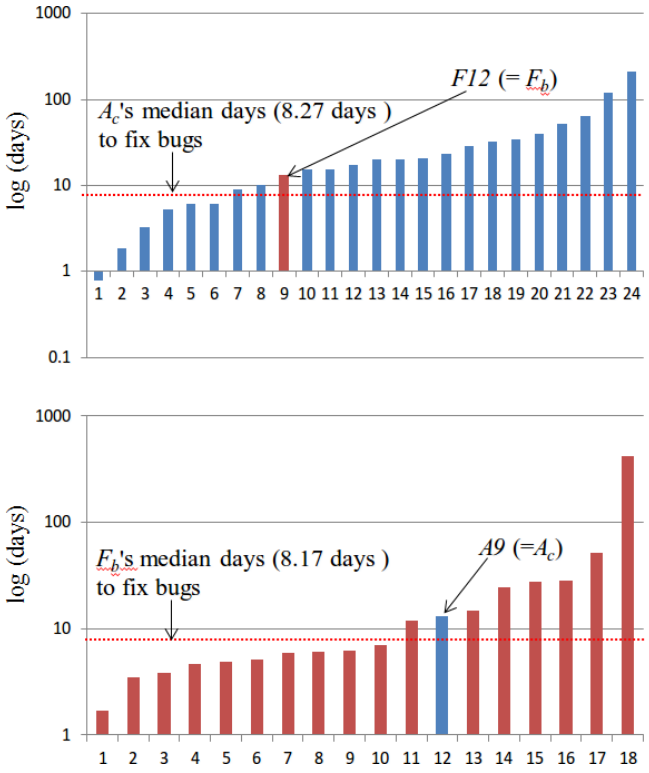

図 2　A_c (= $A9$) と F_b (= $F12$) のそれぞれの不具合修正タスク遂行能力

Fig. 2　Overall performance of A_b (= $A9$) and fixer F_b (= $F12$).

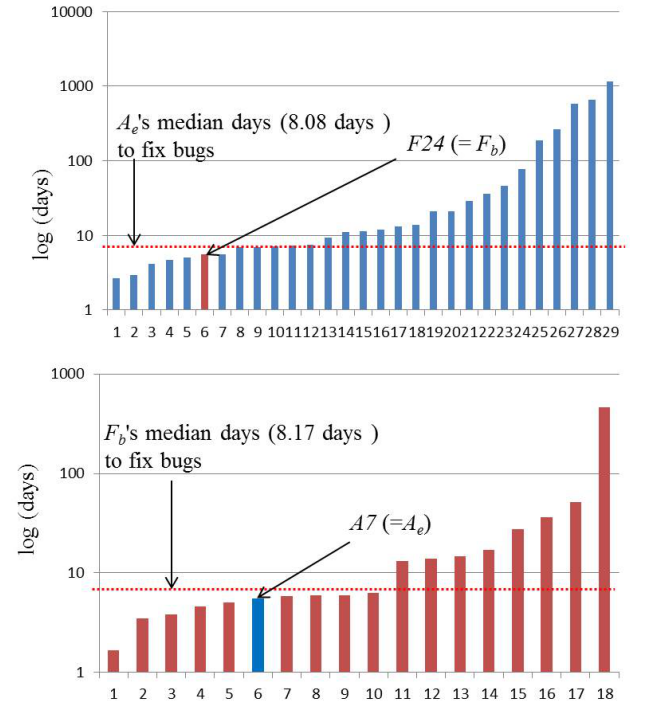

図 3　A_e (= $A7$) と F_b (= $F24 = F12$) のそれぞれの不具合修正タスク遂行能力

Fig. 3　Overall performance of A_e (= $A7$) and fixer F_b (= $F24 = F12$).

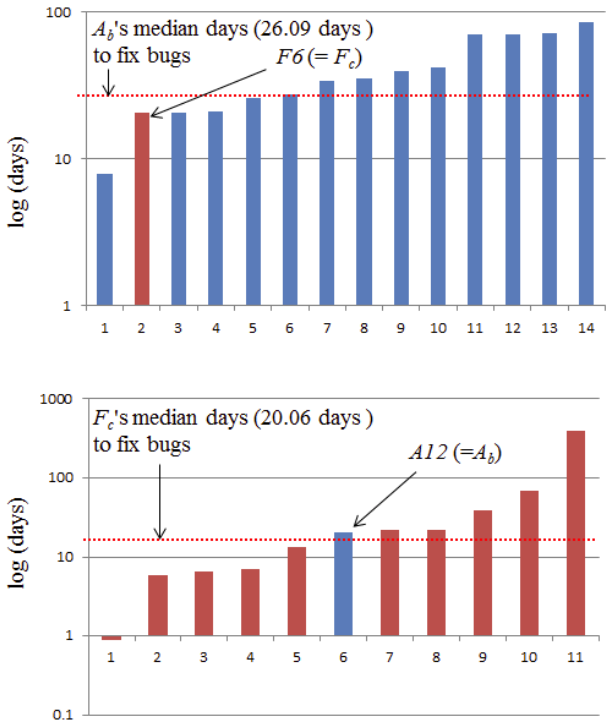

図 4　A_b (= $A12$) と F_c (= $F6$) のそれぞれの不具合修正タスク遂行能力

Fig. 4　Overall performance of A_b (= $A12$) and fixer F_c (= $F6$).

図 3 は，管理者 A_e がタスクを依頼したことのある開発者計 29 名と，開発者 F_b にタスクを依頼したことがある管理者計 18 名の不具合修正日数を表している．図 3 の上部を見ると，開発者 F_b は，管理者 A_e がタスクを依頼したことのある開発者のうち，不具合修正日数が 6 番目に短い．また，図 3 の下部をみると，管理者 A_e は，開発者 F_b にタスクを依頼したことのある管理者のうち，不具合修正日数が 6 番目に短い．よって，同じ開発者でも，タスクを依頼した管理者によって発揮するパフォーマンスに差があることが分かる．

また，別のペアについても分析を行った．表 2 を見ると，管理者 A_b（表 4 における $A12$）は，開発者 $F6$（表 4 における F_c）に，もっとも多くタスクを依頼しており，なおかつ不具合修正日数も全体と比べて短い．また，表 4 を見ると，開発者 F_c は，管理者 $A12$ から，もっとも多くタスクを依頼されているが，不具合修正日数は全体よりも長い．よって，管理者 A_b にとっての開発者 F_c は，何度もやり取りした経験があり不具合修正日数も短いので，信頼してタスクを依頼しがちだが，開発者 F_c は，管理者 A_b に依頼されたタスクを優先してこなしているわけではないと考えられる．

図 4 は，管理者 A_b がタスクを依頼したことのある開発者計 14 名と，開発者 F_c にタスクを依頼したことがある管理者計 11 名の不具合修正日数を表している．図 4 の上部を見ると，開発者 F_c は，管理者 A_b がタスクを依頼したこ

とのある開発者のうち，不具合修正日数が 2 番目に短く，全体の修正日数よりも短い．また，図 4 の下部をみると，管理者 A_b は，開発者 F_c にタスクを依頼したことのある管理者のうち，不具合修正日数が 6 番目に短いが，全体の修正日数よりも長い．このように，管理者からみると優秀な開発者であると考えられているが，開発者から見ると，その管理者から依頼されたタスクを優先してこなしているわけではなく，開発者の能力を発揮しきれていない場合もある．

これらのことから，不具合修正に適任の開発者を選ぶ（*"Who should fix this bug?"*）だけである現在の不具合修正タスク割当支援は十分でないと考えられる．より効率的な不具合修正タスク割当支援には，開発者の能力を最も発揮させる管理者を選ぶ，*"Who should assign this bug?"* という考え方（開発者から見た視点）が必要である．

4.2 妥当性の検証

本研究では，分析を簡潔にするために，Eclipse Platform プロジェクトの不具合報告のうち，再割当が発生していない不具合報告のみに注目した．その結果，データセットに偏りが生じてしまい OSS 開発の本質を捉えることができていない可能性がある．そのため，今後，より価値のある分析結果を得るためには，フィルタリングの基準を見直し再度追加分析する必要がある．

また，今回は Eclipse Platform プロジェクトについてのみ分析を行った．本プロジェクトは，大規模かつすでに成功したプロジェクトで，ケーススタディを行うのに十分な数の不具合報告が既に記録されている．しかし，本プロジェクトには，IBM 社に雇用されている開発者が多数存在し，Eclipse が IBM 社の OSS プロジェクトとしてスタートした時点から開発に多大な貢献を行っている．このような特殊な環境から得られた本研究の結果は，他の OSS コミュニティに対して適用できない可能性がある．

また，本研究では，不具合報告の内容（優先度や重要度，議論の内容など）を分析に用いなかった．より正確な結果を得るためには，今後はこれらの情報も用いて分析をする必要がある．

5. まとめと今後の課題

本研究では，不具合修正タスク割当プロセスを，社会的視点から分析した．特に，開発者視点での考え方が不具合修正タスク割当の効率改善に有益であることが分かった．本研究で得られた知見は以下の 3 つである．

- 管理者と開発者の社会的関係の強さ（タスク割当の実績など）は，不具合修正時間の短縮化に必ずしも寄与しない．
- どの管理者からタスクを割り当てられたかが，開発者のタスク選択の優先順位とタスク完了時間（不具合修正時間）に影響を与えている可能性がある．
- 不具合修正タスク割当支援の改善には，開発者の視点からも不具合修正プロセスを考慮する必要がある．

今後は，他の OSS プロジェクトに対しても分析を行い，新たなタスク割当手法や不具合修正時間の予測モデルの構築を行う．

謝辞

本研究の一部は，文部科学省科学研究補助金 (基盤 (B):23300009) および (基盤 (C):24500041) による助成を受けた．

参考文献

[1] Anvik, J., Hiew, L. and Murphy, G. C.: Coping with an open bug repository, *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange (eclipse '05)*, pp. 35–39 (2005).

[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] Bettenburg, N. and Hassan, A. E.: Studying the Impact of Social Structures on Software Quality, *Proceedings of the 2010 IEEE 18th International Conference on Program Comprehension*, ICPC '10, Washington, DC, USA, IEEE Computer Society, pp. 124–133 (online), available from ⟨http://dx.doi.org/10.1109/ICPC.2010.46⟩ (2010).

[5] Bettenburg, N., Just, S., Schröter, A., Weiß, C., Premraj, R. and Zimmermann, T.: Quality of bug reports in Eclipse, *Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange (Eclipse '07)*, pp. 21–25 (2007).

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

[7] Bettenburg, N., Premraj, R., Zimmermann, T. and Kim, S.: Extracting structural information from bug reports, *Proceedings of the 2008 international working conference on Mining software repositories*, pp. 27–30 (2008).

[8] Bhattacharya, P. and Neamtiu, I.: 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).

[9] Bird, C., Gourley, A. and Devanbu, P.: Detecting Patch Submission and Acceptance in OSS Projects, *Proceedings of the 4th International Workshop on Mining Software Repositories (MSR'07)*, No. 26 (2007).

[10] Bird, C., Pattison, D., D'Souza, R., Filkov, V. and Devanbu, P.: Latent Social Structure in Open Source Projects, *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'08)*, pp. 24–35 (2008).

[11] Breu, S., Premraj, R., Sillito, J. and Zimmermann, T.: 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).

[12] Cubranic, D. and Murphy, G. C.: Automatic bug triage using text categorization, *Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering (SEKE2004)*, pp. 92–97 (2004).

[13] Guo, P. J., Zimmermann, T., Nagappan, N. and Murphy, B.: 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) - Volume 1*, pp. 495–504 (2010).

[14] Guo, P. J., Zimmermann, T., Nagappan, N. and Murphy, B.: "Not my bug!" and other reasons for software bug report reassignments, *Proceedings of the ACM 2011 conference on Computer supported cooperative work (CSCW'11)*, pp. 395–404 (2011).

[15] Hooimeijer, P. and Weimer, W.: Modeling bug report quality, *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering (ASE '07)*, pp. 34–43 (2007).

[16] 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).

[17] Matter, D., Kuhn, A. and Nierstrasz, O.: Assigning bug reports using a vocabulary-based expertise model of developers, *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories (MSR '09)*, pp. 131–140 (2009).

[18] Mockus, A., Fielding, R. T. and Herbsleb, J. D.: Two Case Studies of Open Source Software Development: Apache and Mozilla, *ACM Transactions on Software Engineering and Methodology*, Vol. 11, No. 3, pp. 309–346 (2002).

[19] 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).

[20] Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W., Ohira, M., Adams, B., Hassan, A. 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).

[21] Sun, C., Lo, D., Wang, X., Jiang, J. and Khoo, S.-C.: A discriminative model approach for accurate duplicate bug report retrieval, *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE '10) - Volume 1*, pp. 45–54 (2010).

[22] Wang, X., Zhang, L., Xie, T., Anvik, J. and Sun, J.: 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).

[23] Weißgerber, P., Neu, D. and Diehl, S.: Small patches get in!, *Proceedings of the 5th International Working Conference on Mining Software Repositories (MSR'08)*, pp. 67–76 (2008).

[24] Zanetti, M. S., Scholtes, I., Tessone, C. J. and Schweitzerx, F.: Categorizing bugs with social networks: a case study on four open source software communities, *Proceedings of 35th International Conference on Software Engineering (ICSE'13)*, pp. 1032–1041 (2013).

[25] Zimmermann, T., Premraj, R., Bettenburg, N., Just, S., Schroter, A. and Weiss, C.: What Makes a Good Bug Report?, *IEEE Transactions on Software Engineering (TSE)*, Vol. 36, No. 5, pp. 618–643 (2010).