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縮化へ向けた分析

The Current State and Issues of the Bug Fixing Process in OSS Development:

An Analysis Toward Improving Time to Fix Bugs

伊原 彰紀 (Akinori Ihara)[1]　大平 雅雄 (Masao Ohira)[2]
松本 健一 (Ken-ichi Matsumoto)[3]

[1]奈良先端科学技術大学院大学 情報科学研究科 博士後期課程、
[2]奈良先端科学技術大学院大学 情報科学研究科　助教、
[3]奈良先端科学技術大学院大学 情報科学研究科　教授

［Abstract］
In recent years, a number of bugs have been reported to Open Source Software (OSS) development projects since many of OSS products have been widely used in our daily life. Especially in large OSS projects such as Apache and Mozilla, project managers have been struggling with the difficulty of improving the bug modification process in their projects. In this paper, we analyze three OSS projects to understand the current state and issues of the bug modification process. From a result of our analysis, we suggest three directions to improve the process.

［キーワード］
バグトラッキングシステム、不具合修正プロセス、修正作業時間、オープンソースソフトウェア

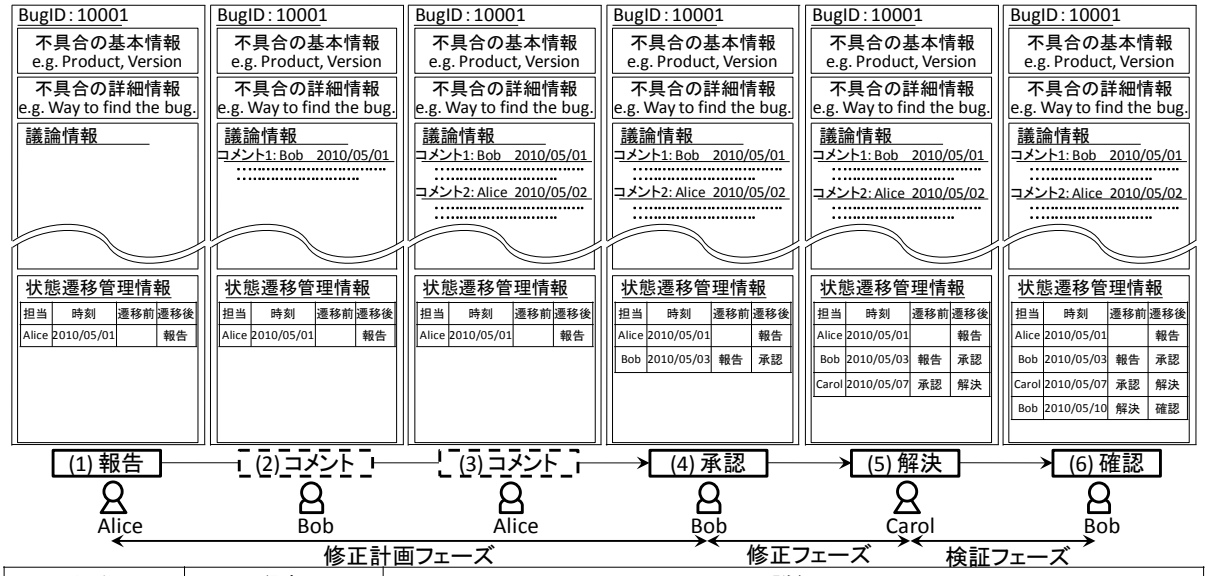
１．はじめに

オープンソースソフトウェア（OSS）の普及に伴い、近年、OSS プロジェクトに報告される不具合が急増している[2]。OSS 利用者からの不具合報告は、OSS の品質を高める上で重要な役割を担っており、OSS 開発では本来歓迎されるべきものである[3]。しかしながら、プロジェクトによっては１日に数百件の不具合報告を受けることも珍しくない[1][4]ため、報告された全ての不具合を短期間のうちに修正することは困難になりつつある[5]。

一般的な OSS 開発では、地理的に分散したボランティアの開発者同士が、メーリングリストや掲示板などの限られたコミュニケーションチャネルの中で議論を行いながら不具合の修正を行っている[6][7][8]。このような開発環境に加え、昨今の OSS 開発においては大量の不具合報告を処理する必要があるため、プロジェクト管理者が個々の不具合の内容を把握した上で適任の修正担当者を決定したり、不具合修正の進捗状況を鑑みて修正担当者毎に適宜指示やアドバイスを与えたりすることが難しくなっている。その結果、大規模な OSS を開発する多くの OSS プロジェクトで、不具合修正の長期化が問題となっている。OSS の社会的重要性は増す一方であり、不具合修正の長期化を解消するための方策が望まれている[9][10][11][12][13]。

本論文の目的は、大規模 OSS プロジェクトにおける不具合修正プロセスの課題を明らかにし、不具合修正プロセスの効率化するための方法を検討することである。そこで本論文では、3 つの OSS プロジェクト（Apache HTTP Server、Eclipse Platform、Mozilla Firefox）を対象としたケーススタディを行い、不具合管理システム(BTS: Bug Tracking System)に記録される不具合修正の進捗情報を用いて、過去の不具合修正に必要となった一連の作業内容や個々の作業時間の可視化・分析を試みる。本研究は、OSS プロジェクト管理者が自身のプロジェクトにおける不具合修正プロセスの問題発見と改善を促すことを目指している。

続く 2 章では、OSS 開発における不具合管理とその問題点について述べる。3 章で分析方法について詳述し、4 章では、ケーススタディの結果を示す。5 章では、分析結果から不具合修正プロセス改善のための方策を検討する。6 章で関連研究を紹介し、最後に 7 章で本論文のまとめと今後の課題について述べる。



行動	役割	詳細
報告 (1)	報告者	開発者やユーザがOSSに不具合を発見した時，報告者は不具合票の「不具合の基本情報」と「不具合の詳細情報」に不具合に関する情報を記載する.
コメント (2,3)	修正貢献者	開発者は報告者に不具合に関する追加情報の要求，類似・重複不具合票の提示，修正に関するアドバイスなどをする場合，コメントを書き込む.
解決 (5)	修正担当者	開発者が不具合の修正作業を行い，作業終了後に不具合が解決したことを記録する.
承認，確認(4,6)	プロジェクト管理者	報告された不具合を修正すべきかどうか，また修正後は，正しく修正されたかどうかを審査する.

図 1 BTS を用いた不具合修正の流れ

２．OSS 開発における不具合管理

本章では、一般的な OSS 開発における不具合管理の方法とその問題点について述べる。

２．１ 不具合の管理

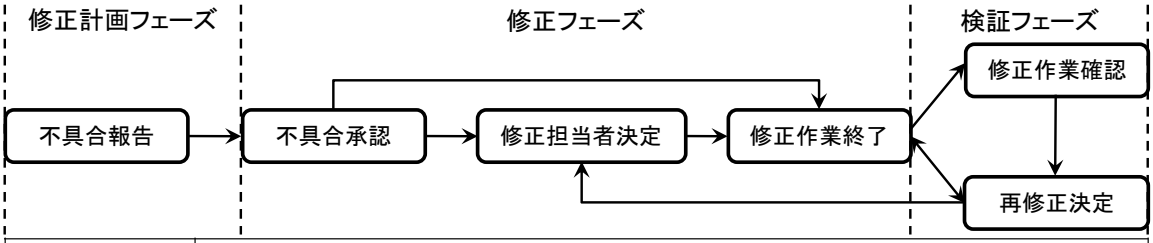
一般的な OSS 開発では、世界中に点在する開発者同士が不具合に関する情報を共有するために、Bugzilla[1]、Trac[2]、RedMine[3]などの不具合管理システム(BTS)が利用されている。BTS は、開発者あるいはユーザから報告された 1 つの不具合に対して 1 つの不具合票を作成し、個々の不具合修正の進捗を管理 (追跡) するためのシステムである。また、Web ユーザインタフェースを介して誰でも不具合の報告と修正作業進捗の閲覧が行える点に特徴がある。

図 1 に BTS を用いた不具合修正の流れを示す。各不具合票には、**不具合の基本情報**（対象プロダクトやバージョン、重要度や優先度など）、**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作業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 種類の情報が記録される。

２．２ 不具合修正プロセス

不具合修正プロセスは、不具合がプロジェクトに報告されてから正しく修正されたことが確認されるまでに必要とされる一連の作業からなる。図 2 は、Bugzilla プロジェクト[1]が提示している一般的な不具合修正プロセスの概念図である。不具合修正プロセスは、具体的な修正に向けた活動を開始する時期と、修正作業が完了し修正内容の妥当性の検証を開始する時期を境に、3 つの区間（**修正計画フェーズ、修正フェーズ、検証フェーズ**）に

[1] Bugzilla: http://www.bugzilla.org/
[2] Trac: http://trac.edgewall.org/
[3] Redmine: http://www.redmine.org/



状態	詳細
不具合報告	開発者又はユーザによって発見された不具合が報告（BTSに登録）された状態.
議論開始	BTSに登録された不具合情報に関して議論が開始された状態.
不具合承認	議論の結果，プロジェクト管理者によって修正すべき不具合として承認された状態.
修正担当者決定	プロジェクト管理者によって修正担当者が決定された状態.
修正作業終了	修正担当者によって修正作業が終了した状態.
再修正決定	不具合の修正が不十分であり，再度修正が要求された状態.
修正作業確認	プロジェクト管理者によって不具合が正しく修正されたことが確認された状態.

図 2 BTS を用いた一般的な不具合修正プロセス

分類することができる[14]。プロジェクト管理者は、この 3 つの区間それぞれで作業の進捗状況を把握する必要がある。

修正計画フェーズでは、報告された不具合に関して議論が開始されているかどうか（不具合報告が誰の目にも触れられず放置されていないかどうか）を監視しておかなければならない。修正フェーズでは、修正担当者が適切に割り当てられているかどうかや、円滑に修正作業が進められているかどうかを把握しておく必要がある。検証フェーズでは、修正担当者によって修正された作業内容の検証が滞りなく進められているかどうかや、検証作業の結果、再修正が必要と判断された不具合が放置されていないかどうかを監視しておく必要がある。

２．３ 不具合管理の課題

BTS に登録されている不具合の中には、修正作業が滞っているものも少なくない。例えば、**修正計画フェーズ**で修正に向けた議論や作業が開始されていなかったり[5][15]、**修正フェーズ**で修正担当者が決まらなかったり[9]、問題が複雑なため修正が順調に進まなかったり、**検証フェーズ**で修正内容の検証作業が開始されていなかったりする。不具合修正が長期化する原因を把握するため、不具合修正プロセスから特に改善すべき修正作業を理解する必要がある。

このような不具合の中には、優先度や重要度が高い不具合（例えば、セキュリティホールなど、早急に解決すべき不具合）が含まれていることも多く、プロジェクト管理者は修正が滞りなく進んでいるかどうかを BTS の検索機能を用いて定期的に調査している。しかしながら、修正が滞る原因となる作業を把握するためには、各々の不具合票に記載されている情報から遅れの生じている作業を相対的に比較・検討する必要があり、近年大量に報告される不具合報告を管理しなければならないプロジェクト管理者にとって大きな負担となる。

修正に要する時間の長期化は、OSS のリリース遅延やサポート遅延に繋がる。プロジェクト管理者が不具合修正プロセス中の修正が滞る原因となる作業を把握するためには、不具合修正プロセス中の各作業時間や各フェーズに要する時間、状態遷移のパターンを効率的に理解することが不可欠である。

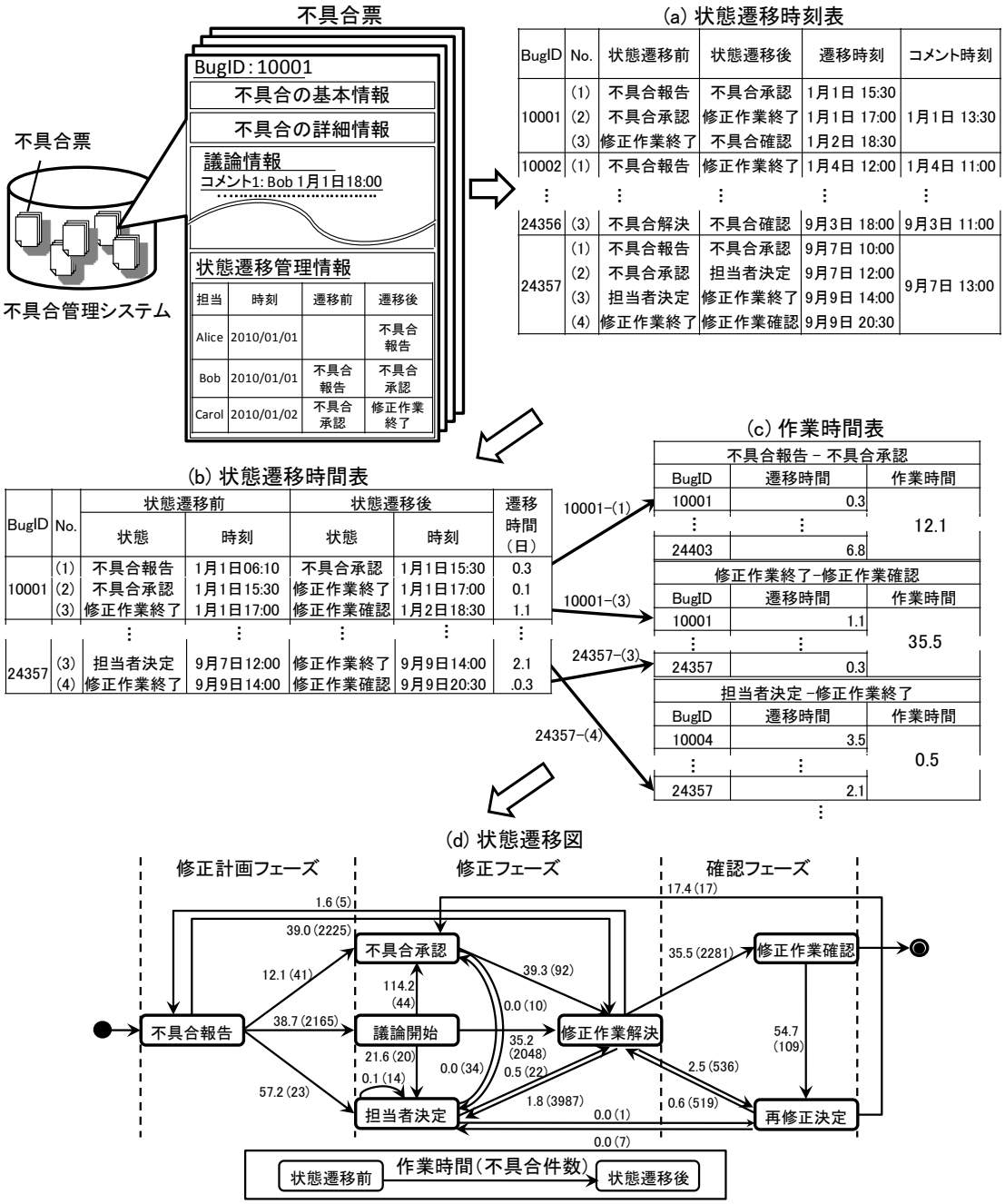


図３　状態遷移図の構築手順

３．分析方法

３．１ 概要

本論文では、BTS に記録される不具合修正の進捗情報を用いて、不具合修正プロセスを状態遷移図で表す。そして、過去の不具合修正で必要となった一連の作業内容や個々の作業時間を可視化・分析し、プロジェクトの不具合修正プロセスを効率化するための指針を検討する。分析では特に、プロジェクト特有の状態遷移や、時間を要する作業・フェーズを把握することに重点を置く。プロジェクト管理者は、本論文での分析方法を用いて自身のプロジェクトの不具合修正プロセスの問題点を発見できると期待される。分析は、BTS の不具合修正履歴から各種解析結果を付加した状態遷移図の構築と、状態遷移図を用いた不具合修正プロセスの分析から構成される。次節では、それぞれの手順について説明する。

３．２ 手順

３．２．１ 状態遷移図の構築

状態遷移図の構築手順（図 3）について説明する。図中および以下の説明では、具体例として Apache プロジェクトの BTS に実際に登録されている不具合の修正履歴情報を用いている。

1. 状態遷移時刻表の作成
   BTS に登録されている各不具合票の状態遷移管理情報から状態遷移と遷移時刻を取得する。また、議論情報から最初のコメントが投稿された時刻を取得し、状態遷移時刻表（図３-(a)）を作成する。
2. 状態遷移時間表の作成
   各不具合票の各状態遷移にかかる時間を算出し、状態遷移時間表（図３-(b)）を作成する。その時、「不具合報告」から次の状態に遷移するよりもコメントされる時刻の方が早い場合、「議論開始」を経由して、次の状態に遷移することとする。例えば状態遷移時刻表（図３-(a)）では、「不具合報告」→「不具合承認」の順で状態遷移が記録されていても、「不具合承認」より前に議論（コメント）が開始されていれば、「不具合報告」→「議論開始」→「不具合承認」の順で状態を遷移したものとする。
3. 作業時間表の作成
   すべての不具合を対象として遷移元と遷移先が同じ状態遷移対を抽出（例えば「不具合報告」→「議論開始」という状態遷移対をすべて抽出する）し、状態遷移に要した作業時間を算出する（作業時間表（図３-(c)））。各作業時間の算出には、各遷移時間の中央値を用いる（理由は後述）。
4. 状態遷移図の構築
   作業時間表（図３-(c)）に抽出された各状態と各遷移を状態遷移図として記述する。さらに、導出した各作業時間を遷移経路のエッジに記載し（括弧なし）、その遷移経路を経由した不具合の数（遷移対の数）をエッジに記載する（括弧あり）。

なお、手順 5. において中央値を用いる理由は主に、OSS 開発での不具合修正における次の 3 つの特徴に依拠する．1 つ目は，プロジェクトへの自由な参加・離脱を認める OSS 開発では、不具合修正を行う開発者のスキルにばらつきが大きい、すなわち、開発者のスキルを統制することが難しいことである。2 つ目は、各開発者のプロジェクト参加理由の違い（業務の一部、余暇の時間を使った趣味、プログラミングやプロジェクトベース開発の学習・体験のためなど）により、開発者が OSS 開発に充てられる時間にばらつきが大きいことである。最後に、軽微な修正に留まる場合から、プロダクト全体に影響を及ぼすものまで、個々の不具合そのものの複雑度・難易度にばらつきが大きいという特徴がある。

これらの特徴から、OSS の不具合修正において必要とされる各作業時間は、一般的に正規分布に従わず偏った分布になる。1 つの不具合を修正するのに数年かかるようなケースも含まれ、外れ値によって平均値が直観的に解釈できないほど大きくなることが少なくないため、サンプルの代表性を示す際に平均値を用いるのは妥当ではない。そのため、十分な数の不具合票を保持する大規模プロジェクトを対象とした分析では、中央値を用いてサンプルの傾向を調べることが有効となる。ただし、小規模プロジェクトを分析対象とする場合には，5.2 節で述べるように、中央値のみで作業時間を解釈することは難しいため追加の分析が別途必要となる。

本手法で構築した状態遷移図は、図 2 に示す状態遷移図と異なる点がある。それは、 Bugzilla プロジェクトが提示する不具合修正プロセスと異なるプロセスを経由することを OSS プロジェクトが可能にしているためである。例えば、OSS プロジェクトでは、管理者が不具合報告を行う場合、不具合承認を経由せずに直接担当者を決定する。また、報告者自身が修正を行う場合、「不具合報告」から直接「修正作業終了」に遷移することがある。また、図 2 には「議論開始」という状態が定義されていない。しかし実際の OSS プロジェクトでは、「議論開始」を起点として不具合修正のための各種作業が開始されることが多いことから、本研究では BTS の議論情報に記録されている最初のコメント時刻を用いて「議論開始」状態を定義し、状態の 1 つとして用いている。

３．２．２ 状態遷移図を用いた分析

状態遷移図を用いて、不具合修正を遅延させる原因となる作業やフェーズを特定する手順を述べる。

（1） 不具合修正プロセスの把握：状態遷移図を用いて、各フェーズに存在する状態遷移を把握する。例えば図 3 の Apache プロジェクトの状態遷移図には、図 2 で示した一般的な不具合修正プロセスには存在しない状態遷移（「不具合報告」→「不具合解決」など）が存在することが見て取れる。

（2） 各フェーズ中で時間を要する作業の把握：状態遷移図を用いて、フェーズ毎に多く不具合で実施される

作業を把握し、その作業に要する時間を把握する。一つの状態から複数の遷移がある場合、対象とする状態から多くの不具合が遷移している作業に着目する。例えば図 3 の状態遷移図の「不具合報告」の場合、「不具合報告」の次の状態は 4 つ（「不具合解決」、「不具合承認」、「議論開始」、「修正担当者決定」）のいずれかであり、合計 32,642 件（2225+41+2165+23=4454 件）の不具合が関係する。その不具合の内、約 50%（2225÷4454≒0.500）が「不具合解決」へ、約 49%（2165÷4454≒0.486）が「議論開始」へ遷移することが分かる。また、状態遷移にそれぞれの 40 日、61 日かかっていることが分かる。このように、分析 2 では、多くの不具合が経由する状態遷移を見つけるために、各状態から次の状態に遷移する割合を用いて分析する。また、修正担当者の変更（「修正担当者決定」→「修正担当者決定」）や再修正（「不具合解決」→「再修正決定」）のように、修正が滞る直接的な原因となる作業も、各作業が実施される不具合件数と各作業に要する時間を用いて分析する。

（3） 最も時間を要するフェーズの把握：手順(2)で挙げた各フェーズの時間を要する作業を比較し、その中で最も時間を要している作業を特定しそのフェーズを把握する。

4. ケーススタディ

本論文では、BTS を利用している 3 つの大規模な OSS プロジェクトを対象にケーススタディを行った。本章ではケーススタディを行った結果を報告する。

4. 1 分析対象データ

本論文では、多くの不具合を管理している 3 つの OSS プロジェクト（Apache、 Eclipse Platform、 Mozilla Firefox)を対象にケーススタディを行った。対象プロジェクトに関するソフトウェアの種類と不具合件数を表 1 に示す。これら 3 つの OSS プロジェクトは、（1）多くの開発者・ユーザを有しており社会的に大きな影響を持つこと、（2）開発・運営スタイルが他の OSS プロジェクトの模範となっており対象プロジェクトから得られる知見の一般性を期待できること、（3）それぞれ利用されるドメインと利用者層が大きく異なるため比較により OSS プロジェクトの多面性や多様性をある程度捉える事ができること、（4）不具合管理のために Bugzilla を利用しており分析結果の比較が可能なこと、(5）プロジェクト立ち上げから相当な時間経過がありプロジェクト運営が安定していること（得られるデータの意味が時期によって大きく変わらないこと)、（6）知見の一般性を高めるに足る十分なデータ（不具合報告数）が得られること、を主な理由として選定した。

4. 2 分析結果

本節では、各プロジェクトの分析結果を詳細に説明する。図４、５、６はそれぞれ、Apache HTTP Server、Eclipse Platform、Mozilla Firefox プロジェクトの不具合修正プロセスを状態遷移図として可視化したものである。

4. 2. 1 Apache HTTP Server プロジェクト

修正計画フェーズでは、「不具合報告」から次の状態に遷移する不具合の約 98%（(2225+2165)÷(2225+41+2165+23)≒0.986)が「修正作業終了」または「議論開始」に遷移している。また、**修正フェーズ**では、不具合の多くは「議論開始」後、直接「修正作業終了」に 35.2 日で遷移している。**検証フェーズ**では、約 81%（2281÷(2281+519+3+5)≒0.812)の不具合が約 35 日で正しく修正されたと判断（「修正内容確認」）されており、約 19%(519÷(2281+519+3+5)≒0.185)の不具合は約 1 日で再修正が必要と判断されている。しかしながら、正しいと判断された不具合のうち約 5%（109÷2281≒ 0.047)は再修正が求められている。

以上のことから、**Apache HTTP Server プロジェクトでは、多くの不具合は各フェーズに同時間（約 40 日）かかっている**ことを把握することができた。

表 1 ケーススタディの対象

プロジェクト	Apache HTTP Server	Eclipse Platform	Mozilla Firefox
種類	Web サーバソフトウェア	統合ソフトウェア開発環境	Web ブラウザ
不具合件数	4,923	26,113	63,652
データ取得期間	2003/1～2008/12	2001/10～2009/11	2003/1～2008/12

４．２．２ Eclipse Platform プロジェクト

修正計画フェーズでは、「不具合報告」から次の状態に遷移する不具合の約 82%（(10567+10472)÷(4756+2+10567+10472)≒0.816)が「議論開始」または「修正担当者決定」に遷移しており、それぞれに遷移するまでに 6.0 日かかっている。**修正フェーズ**では、多くの不具合は「修正担当者決定」後、「修正作業終了」に至っている。しかしながら、始めに割り当てられた担当者が適切でないため再度修正担当者を決定していることが多く（「修正担当者決定」からの自己ループ：10,105 回)、修正担当者の再決定には「修正担当者決定」後の「修正作業終了」に要する時間（4.0 日）よりも多くの時間がかかっている（4.9 日）ことが見て取れる。**検証フェーズ**では、約 72%(2281 ÷(2281+536+351)≒0.720)の不具合は 35.5 日で正しく修正されたと判断（「修正内容確認」）されており、約 17%(536 ÷(2281+536+351)≒ 0.169)の不具合は 2.5 日で再修正が必要と判断されている。しかしながら、正しいと判断された不具合のうち約 13%(289 ÷ (2281+1) ≒0.126)は再修正が求められている。

以上のことから、**Eclipse Platform プロジェクトでは、検証フェーズの「修正内容確認」に遷移するまでの時間が最も長い**ことを把握することができた。

４．２．３ Mozilla Firefox プロジェクト

修正計画フェーズでは、不具合報告から次の状態に遷移する不具合の約 92%((18806 + 34843)÷(11 + 18806 + 1749 + 34843 + 2758)≒ 0.922)が「修正作業終了」、または「議論開始」に遷移しており、「修正作業終了」に遷移する場合は約 5 日、「議論開始」に遷移する場合は約 11 日かかっている。**修正フェーズ**では、多くの不具合は「議論開始」後に直接「修正作業終了」に遷移しており、約 13 日かかっている。「修正担当者決定」の自己ループから、修正担当者の再決定は頻繁に行われていない(1,781 回)が、担当者が変更された場合は約 1 カ月要することが見て取れる。**検証フェーズ**では、修正解決後、次の状態に遷移する不具合のうち約 72%（11326÷(11326+1267+219+2951)≒0.719)は正しく修正されたと判断（「修正内容確認」）されており、約 8%（1267÷(11326+1267+219+2951)≒0.080)は再修正が必要と判断されている。Mozilla Firefox プロジェクトは不具合の検証に約 2 日かかっている。Mozilla Firefox プロジェクトでは、正しいと判断された不具合のうち、再修正が求められている不具合は 1%未満(101÷(11326+11+37)≒0.009)であった。

以上のことから、**Mozilla Firefox プロジェクトでは、修正計画フェーズや修正フェーズに長い時間を要している**ことを把握することができた。

４．２．４ まとめ

表 2 はケーススタディの結果をまとめたものである。表中の各セルには、多くの不具合に実施される作業(1)、とその作業時間(2)を示す。最下行には、不具合修正プロセスの中で最も時間を要するフェーズを示す。

表 2 より、プロジェクトそれぞれで不具合解決までに必要な作業内容が大きく異なることが見て取れる。また、各フェーズを比較することで改善を要するフェーズも確認することができる（Apache の場合は、すべてのフェーズにおいて１つの不具合を解決するために１カ月以上要しているため、すべてのフェーズでプロセス改善が必要とされていると解釈できる)。ただし、不具合管理のルールやポリシーはプロジェクトにより異なるため、各作業で想定（あるいは許容）される作業時間は一様に定義することはできない。最終的には、プロジェクト管理者が状態遷移図を確認し、自身のプロジェクトにとってボトルネックとなっている作業を特定する必要がある。

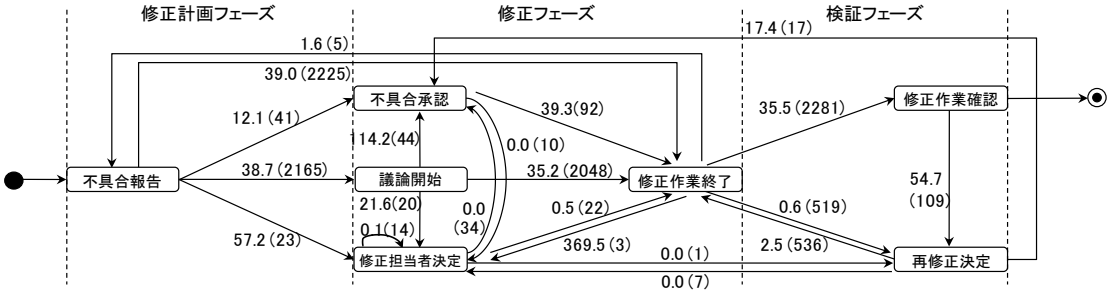


図 4 Apache プロジェクトの不具合修正プロセス

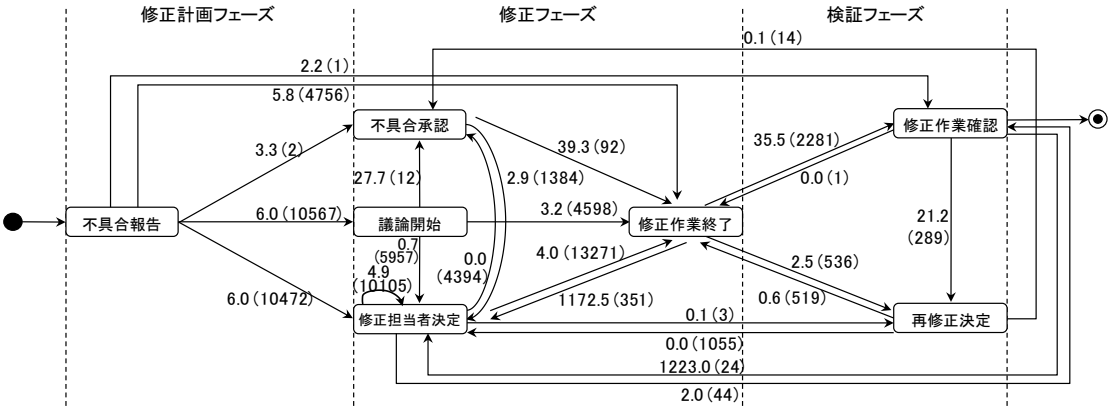


図 5 Eclipse Platform プロジェクトの不具合修正プロセス

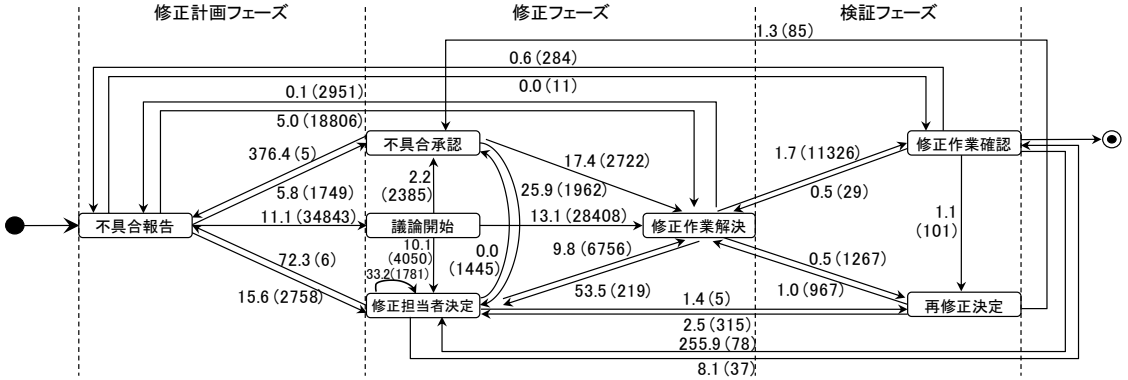


図 6 Mozilla Firefox プロジェクトの不具合修正プロセス

表 2 ケーススタディの結果

	Apache HTTP Server	Eclipse Platform	Mozilla Firefox
修正計画フェーズ	(1)　不具合報告→修正作業終了、 不具合報告→議論開始 (2)　約 40 日	(1)　不具合報告→議論開始、 不具合報告→修正担当者決定 (2)　約 6 日	(1)　不具合報告→修正作業終了、 不具合報告→議論開始 (2)　約 11 日
修正フェーズ	(1)　議論開始→修正作業終了 (2)　約 40 日	(1)　修正担当者決定→修正担当者決定、 修正担当者決定→修正作業終了 (2)　約 4 日	(1)　議論開始→修正作業終了 (2)　約 13 日
検証フェーズ	(1)　修正作業終了→修正内容確認 (2)　約 35 日	(1)　修正作業終了→修正内容確認 (2)　約 35 日	(1)　修正作業終了→修正内容確認 (2)　約 2 日
最も改善が必要とされるフェーズ	すべてのフェーズ	検証フェーズ	修正計画フェーズ、 修正フェーズ

５．考察

５．１ 不具合修正プロセス改善のための方策

本節では、ケーススタディの結果を踏まえ、不具合修正プロセスにおける各フェーズの作業時間を短縮するための方策を検討する。

修正計画フェーズでは、3 つのプロジェクト全てにおいて、不具合報告後に議論開始状態に遷移する不具合票が数多く存在することを確認した。特に Mozilla Firefox プロジェクトでは、全体の約 60%の不具合が、報告後、議論開始に遷移していた。その原因の一つには、報告者が不具合票に記載する内容と、開発者が求める情報が異なっていることが多く[10][12]、プロジェクト管理者や修正者が不具合に関する追加情報を求めるため議論開始に遷移していることが考えられる。Mozilla Firefox は Apache HTTP Sever や Eclipse Platform に比べてライトユーザが多く、開発経験のないユーザからの不具合報告が多いため、開発者が不具合修正のために必要としている情報が不具合票に記載されていないことが考えられる。ライトユーザの多いプロジェクトでは特に、修正計画フェーズにかかる時間の短縮に向けて、報告内容として必要な情報に漏れがないように、不具合票の報告フォーマット(不具合の再現方法やスクリーンショットを添付する仕組みなど)をあらかじめ用意しておく必要がある。

修正フェーズでは、繰り返し修正担当者が変更されるため時間を要することがある。それはプロジェクト管理者がプロジェクト参加者の専門性を把握できていないため、適切な修正担当者を決定することが困難であることが大きな原因となっている[9][16]。本ケーススタディにおいても、各プロジェクトの状態遷移図における状態「修正担当者決定」の自己ループ回数、また作業時間を分析した結果、Eclipse Platform プロジェクトでは繰り返し修正担当者が変更されており（10105 回、4.9 日）、Mozilla Firefox プロジェクトでは修正担当者の決定に時間を要している（33.2 日）ことを確認することができた。Eclipse Platform プロジェクトには、多くの IBM 社員が開発に携わっていることが知られており[22]、社内メンバ間で担当者の依頼を容易に行うことができるため、担当者の変更が繰り返し行われたと考えられる。一方、Mozilla Firefox プロジェクトの場合、世界中に開発者が多く存在しているため、適切な担当者を見つけることが困難と指摘されており[9]、修正担当者の決定に時間を要すると考えられる。OSS 開発において適切な修正担当者を決定するための方法は、ソフトウェア工学の分野において研究が進んでおり、目的に応じて利用できる。例えば、過去の修正担当者の変更履歴を用いて適切な修正担当者を自動的に決定する手法[9]や、開発者が版管理システムや BTS に記述した文章内容を基に開発者の専門知識を把握し、新しく報告された不具合の修正に適切な開発者を決定する方法[16]などがある。しかしながら、それらの手法が OSS プロジェクトに実際に適用された事例がほとんどなく、不具合修正担当者の自動決定手法の導入と効果の検証が今後の課題である。

検証フェーズでは、不具合の再修正によって解決日が先延ばしになってしまうことがある。本ケーススタディの結果から、Apache HTTP Server プロジェクトでは不具合の修正確認後、再修正と判断されるまでに約 2 ヶ月かかっていることが分かった。検証フェーズの長期化は、再修正が必要と判断されたとき、以前にどのような修正を行ったか思い出す必要がある。不具合解決後、時間の経過に伴い、以前の修正内容を思い出すことが困難となるため、再修正不具合発見の遅れは修正時間の長期化に繋がる。これまで、再修正される不具合を予め把握するために、不具合票に記載される情報を用いて再修正が必要となる不具合を予測するための予測モデルが提案されている[17]。再修正が必要となる可能性の高いモジュールをあらかじめ見積もっておくことで、プロジェクト管理者は再修正が必要と予測される不具合を優先的に検証し、修正者担当候補者へ早急に再修正の依頼を行うことができる。

５．２ 制約

本論文で行った不具合情報に基づく解析では、不具合票の状態が正確に記録されていることが重要である。ケーススタディで対象とした OSS プロジェクトでは不具合報告や修正などの状態の変更に関する規則が各プロジェクトの Web ページに記載されているため、比較的正確に記載されているが、プロジェクトによっては、担当者が変更されているにも拘わらず、状態の変更が行われていない場合などがある。本論文が分析の対象としたプロジェクト以外の管理者が自身のプロジェクトで同様の分析を行う際には、状態の変更が正しく行われているか確認をしてから利用する必要がある。

また、本論文で扱った各作業時間は、実際の作業に要した時間とは必ずしも一致しない場合がある。例えば、修正は終えているがコミットを翌日に行った場合、実質の作業を行った時間と不具合票に記録される作業時間に時間差が生じる。今後、OSS プロジェクトで Hackystat[18]のような開発者の行動履歴を取得するツールが利用されれば、BTS に記録される時間と実質の作業を行った時間との時間差は小さくなると考えられるが、現在はその

ような正確な情報を取得することができないため、本論文では、不具合票に作業の完了を記録した時を作業の終了時刻としている。

本論文では、統計処理に足る不具合データを保持する大規模 OSS プロジェクトを対象としているため、不具合修正プロセスにおける各作業時間の算出には中央値を用いた。しかしながら、小規模 OSS プロジェクトのように十分な不具合票を保持していない場合、プロジェクトに関わる開発者も限られており、開発者のスキルなどに依存することが予想されるため中央値を用いることは適切でないと考えられる。小規模プロジェクトを対象とする場合は、開発者個々の生産性から不具合の修正作業に時間を要する原因を調査するなど他の方法を用いて分析する必要があり、本研究の今後の課題としたい。

６．関連研究

OSS の機能拡張が進むにつれ、ソフトウェアは大規模かつ複雑になり、開発者数は増加し、不具合報告も増加している。そのため、プロジェクト管理者は効率的な不具合修正を実現するために、ソフトウェアの開発状況、不具合の修正状況、プロジェクトに参加する共同開発者などを把握することが容易ではなくなっている。近年、OSS 開発の現状を容易に把握するために、数多くの研究が行われている。特に、不具合修正の効率的な実施につながる研究が多く報告されている[13][19][20][21]。

Macro ら[21]はプロジェクト管理者がソフトウェア中の不具合の偏在傾向や、リファクタリングの効果などを把握するために、ソースコードの変更と発見される不具合の関係を可視化するシステムを提案している。具体的には、各ファイルのリビジョン履歴と当該ファイルに発見された不具合の修正状況の関係、各モジュールのコミット数の変化と不具合報告数の変化を可視化している。Macro らは OSS プロジェクトを対象にケーススタディを行い、システムの有用性を確認している。Macro らの研究は、不具合修正プロセス中の修正フェーズにおける開発者の活動と不具合の混入について把握するための研究であり、不具合修正プロセス中の全てのフェーズを俯瞰的に把握する本研究とは異なる。

その他にも、Sarma ら[13]は、ソフトウェア開発における開発者の技術的課題（モジュールの依存関係の把握やタスクの把握など）と特に分散開発で抱えている社会的課題（モジュール管理者の把握や適任の修正担当者の把握など）を解決するための可視化ツール Tesseract を提案し、開発者のインタビューにより有用性を確認している。Samra らの研究も、本論文と同様、不具合修正の効率化を目的の一つとした研究である。Samra らの研究は、作業の内容を分析することによって課題を把握する手法であるが、本論文では不具合修正プロセス中の各作業時間から修正が滞る原因となる作業を把握することを目的に分析を行った。

７．おわりに

本論文では不具合の修正状況を俯瞰的に把握するために、不具合修正プロセス中で時間を要する作業やフェーズを把握するために BTS を利用している 3 つの大規模な OSS プロジェクトを対象に分析を行った。本研究で用いた分析方法により、不具合修正プロセス中で最も時間を要するフェーズや作業、担当者の変更回数、再修正要する不具合の件数などを網羅的かつ俯瞰的に把握できることが分かった。本分析方法を用いることにより、OSS プロジェクト管理者は、プロジェクトに報告された大量の不具合票を１つ１つ確認することなく、修正が頻繁に滞る原因となる作業を容易に把握することができるため、不具合修正プロセスの改善のために自身のプロジェクトで有効な方策を（例えば、5.1 節で挙げた方策の中から）選択しやすくなるものと考える。

また、プロジェクトに応じて、不具合修正時間の長期化に影響しているフェーズは異なっていた。しかし、3 つのプロジェクト共に、不具合についての議論がなされた後に実際の修正が開始される場合が多く、修正計画フェーズでは議論に時間を要することが分かった。修正計画フェーズでは、報告者と修正者の合意形成が重要であるため、今後、2 者間の情報伝達を支援することも必要である。また、各プロジェクトが抱える不具合全てを対象として本研究では分析を行ったが、不具合は修正内容によって解決時間は異なる[2]。そのため今後は、プロジェクト管理者が各作業時間を正確に把握するために、OSS の種類や特徴、不具合の特徴による各作業時間について分析を行うことが課題である。

本論文で対象とした OSS プロジェクトは、現在我々の社会に広く流通しているオープンソースソフトウェアの一部に過ぎない。その他のプロジェクトも同様に、広く流通したが故に報告される大量の不具合と対峙せざるを得ないというジレンマに苦しんでいる。我々人類の英知の結晶ともいえるオープンソースソフトウェアという知的財産をさらに発展させていくために、本研究がその一助となることを切に願っている。

謝辞

本研究の一部は、文部科学省「次世代 IT 基盤構築のための研究開発」の委託に基づいて行われた。また、本研究の一部は、文部科学省科学研究補助費（基盤 B：課題番号 23300009，若手 B：課題番号 22700033）による助成を受けた。

［参考文献］

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

[2] P. Hooimeijer and W. Weimer.：Modeling bug report quality, In Proceedings of the twenty-second IEEE/ACM international conference on Automated Software Engineering (ASE’07), pp.34-43, 2007.

[3] E.S. Raymond.： The cathedral and the bazaar: Musings on linux and open source by an accidental revolutionary, Sebastopol, CA, O’Reilly Media, Oct. 1999.

[4] N. Bettenburg, R. Premraj, T. Zimmermann, and S.Kim.： Duplicate bug reports considered harmful... really?, In Proceedings of the 24th IEEE International Conference on Software Maintenance (ICSM’08), pp.337-345, 2008.

[5] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler.：An empirical study of operating systems errors, In Proceedings of the eighteenth ACM Symposium on Operating Systems Principles (SOSP’01), pp.73-88, 2001.

[6] 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, pp.153-158, 2009.

[7] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann.： Information needs in bug reports: improving cooperation between developers and users, In Proceedings of the 2010 ACM conference on Computer Supported Cooperative Work (CSCW’10), pp.301-310, 2010.

[8] 飯尾淳, 清水浩之, 谷田部智之, 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, 三菱総合研究所所報, no.46, pp.52-65, 2006.

[9] 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), pp.111-120, 2009.

[10] N. Bettenburg, S. Just, A. Schr ¨oter, C. Weiss, R. Premraj, and T. Zimmermann.：What makes a good bug report?, In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE’08), pp.308-318, 2008.

[11] X. Wang, L.Zhang, T. Xie, J. Anvik and J. Sun.：An approach to detecting duplicate bug reports using natural language and execution information, In Proceedings of the 30th International Conference on Software Engineering (ICSE’08), pp. 461-470, 2008.

[12] S. Just, R. Premraj, and T. Zimmermann.： Towards the next generation of bug tracking systems, In Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC’08), pp.82-85, 2008.

[13] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb.：Tesseract: Interactive visual exploration of socio-technical relationships in software development, In Proceedings of the 31st International Conference on Software Engineering (ICSE’09), pp.23-33, 2009.

[14] A. Ihara, M. Ohira, and K. Matsumoto.：An analysis method for improving a bug modification process in open source software development, In Proceedings of the joint International and annual ERCIM Workshops on Principles of Software Evolution and software Evolution workshops (IWPSE-Evol2009), pp.135-144, 2009.

[15] Y. Wang, D. Guo, and H. Shi.：Measuring the evolution of open source software systems with their communities, ACM SIGSOFT Software Engineering Notes, vol.32, pp.1-10, 2007.

[16] G. Canfora and L. Cerulo.：Supporting change request assignment in open source development, In Proceedings of the 2006 ACM Symposium on Applied Computing (SAC’06), pp.1767-1772, 2006.

[17] E. Shihab, A. Ihara, Y. Kamei, W.M. Ibrahim, M. Ohira, B. Adams, A.E. Hassan, and K. Matsumoto.: Predicting re-opened bugs: A case study on the eclipse project, In Proceedings of the 17th Working Conference on Reverse Engineering (WCRE'10), pp.249-258, 2010.

[18] P.M. Johnson, H. Kou, J.M. Agustin, Q. Zhang, A. Kagawa, and T. Yamashita.: Practical automated process and product metric collection and analysis in a classroom setting: Lessons learned from hackystat-UH, In Proceedings of the 2004 International Symposium on Empirical Software Engineering (ISESE'04), pp.136-144, 2004.

[19] D.M. German, A. Hindle, and N. Jordan.: Visualizing the evolution of software using softchange, Proceedings of the 16th Internation Conference on Software Engineering and Knowledge Engineering (SEKE'04), pp.336-341, 2004.

[20] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu.: Fair and balanced?: bias in bug-fix datasets, 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), pp.121-130, 2009.

[21] M. D'Ambros and M. Lanza.: Software bugs and evolution: A visual approach to uncover their relationship, In Proceedings of the 10th European Conference on Software Maintenance and Reengineering (CSMR'06), pp.229-238, 2006.

[22] 小山 貴和子：OSS 開発における時差の分析 ：OSS 開発者の情報交換への影響, 奈良先端科学技術大学院大学情報科学研究科修士論文 (2010).

(2011 年 10 月 13 日受理)