不具合管理システム利用時の不具合修正プロセス改善のための滞留時間分析手法の提案

伊 原 彰 紀[†1]　大 平 雅 雄[†1]　松 本 健 一[†1]

多くのオープンソースソフトウェア (OSS) プロジェクトでは不具合の情報を共有し一元管理するために不具合管理システムが用いられている．しかしながら近年，OSS が複雑化・大規模化するにつれて，報告される大量の不具合を効率的に修正することが困難になりつつある．結果的に，不具合が修正されずに残存する時間（滞留時間）が長期化するという傾向にある．不具合の滞留時間の短縮にはまず，不具合修正のプロセスのどの部分がボトルネックとなっているかを理解する必要がある．本稿では，不具合管理システム利用時の不具合の滞留時間を分析するための手法を提案する．また，Mozilla Firefox Project の不具合修正プロセスを対象としたケーススタディを行い，提案手法の有用性を確認する．

An Analysis Method for Improving a Bug Modification Process

Akinori Ihara,[†1] Ohira Masao[†1]
and Matsumoto Ken-ichi [†1]

In Open Source Software (OSS) development, a bug tracking system is often used for sharing and unifying management of bug report information. As OSS become large and complex, however, OSS developers face with a difficulty in fixing a comsiderable amont of bugs which are repored by bug reporters. As a result, bug residence time tends to be protracted. In order to cut down bug residence time, we need to understand the bottleneck in a current bug modification process of OSS. In this paper, we propose a method for analyzing bug residence time in OSS development. We also report a result of a case study which has been conducted to confirm the usufullness of the analysis method.

1. はじめに

世界各地に分散したボランティアの開発者によって開発が行われているオープンソースソフトウェア (OSS) は，近年，高品質かつ高機能なソフトウェアと進化し社会一般に広く利用されるようになった．しかし，幅広いニーズを満たすべく進化を続けた結果，OSS は大規模かつ複雑になりソフトウェアの不具合も増加傾向にある．そのため，多くの OSS プロジェクトは，不具合情報の共有と不具合管理の円滑化を目的として，Bugzilla[cite] などの不具合管理システム (Bug Tracking System: BTS) を利用している[2),3)]．

不具合管理システムは，ソフトウェア開発中に発生した不具合をデータベースに登録して情報を共有し，不具合の修正漏れの防止や不具合修正の進捗状況の把握を目的として利用される．しかし，大量に報告された不具合に対応しなければならない現状では，不具合が報告されてから修正が完了するまでの時間（本稿では滞留時間と呼ぶ）が極めて長くなってしまう場合がある[7)]．例えば，Linux カーネルにおける不具合の残存期間は平均 1.8 年間（平均 1.25 年間）であり，報告された全ての不具合が修正されるまでには非常に長い時間が必要になっている[3)]．不具合修正時間の短縮化は，OSS の利用者に望まれているだけではなく，開発者がより多くの不具合に対応するためにも近年特に重要な課題となっている．

滞留時間の短縮にはまず，不具合修正のプロセスのどの部分がボトルネックとなっているかを理解する必要がある．そこで本研究では，不具合管理システム利用時の不具合の滞留時間を分析するための手法を提案する．分析手法は，不具合管理システムに蓄積されたデータから不具合修正プロセスのボトルネックとなる部分を抽出するために，(1) 滞留時間を「未対応時間」と「修正時間」に分割し分析するための手法と，(2) 特に複雑なプロセスが存在する不具合修正開始から修正完了までの「修正時間」を詳細に分析するための手法からなる．本稿では，Mozilla Firefox Project の不具合修正プロセスを対象としたケーススタディを行い，提案手法の有用性を確認する．

以降，2 章では，関連研究について述べ本研究の背景と立場を明らかにする．3 章で一般的な OSS プロジェクトにおける不具合管理と修正プロセスについて説明し，4 章において不具合の滞留時間を分析するための手法を提案する．5 章では Mozilla Firefox Project が管理する不具合を対象としたケーススタディを行った結果を報告する．7 章でケーススタディ

†1 奈良先端科学技術大学院大学 情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology

を通じて得られた知見から提案手法の有効性について考察を行う．最後に 8 章で本稿のまとめと今後の課題を述べる．

2. 関 連 研 究

2.1 不具合修正の問題点

近年，OSS の進化に関する研究が盛んに行われている[4),12)]．OSS の進化を計測するために，Wang らソフトウェア中に存在している不具合の数や修正された障害の数など指標として用いている．提案指標のケーススタディとして，Linux ディストリビューションである Ubuntu を対象とした分析を行った結果，報告されている不具合のうち，修正されている不具合は 2 割程度に過ぎず，修正されていない不具合の中には修正担当者の割当てすら行われていない不具合が数多く存在していることが明らかとなった．

Wang らの分析結果は，不具合の修正開始までに相当な時間がかかっていることを示唆するものであるが，具体的にどれくらいの時間を要しているかについては明らかにしていない．本研究で提案する分析手法は，不具合の修正開始までにどれくらいの時間を要しているのか，また，修正開始から完了までにどれくらいの時間を要しているのかを区別して明らかにするためのものである．

2.2 OSS 開発における不具合の滞留時間

OSS 開発における不具合の滞留時間についての代表的研究には，Mockus らの研究[8)]，Herraiz らの研究[5)] がある．

Mockus らは OSS 開発が成功する理由を調査するために，2 つの有名な大規模プロジェクトである Apache プロジェクトと Mozilla プロジェクトの分析を行っている[8)]．利用者にとって不具合が素早く修正されることが重要であることから，調査項目の一つとして不具合の滞留時間を分析している．分析の結果，カーネルやプロトコルに関するモジュールや広い範囲で利用される機能を持つモジュールに存在する不具合は滞留時間が短いことが分かった．また，優先度別の滞留時間は P1，P3 に指定された不具合のうち 50%が約 30 日以内，P2 に指定された不具合のうち 50%が約 80 日以内，P4，P5 に指定された不具合のうち 50%は滞留時間が約 100 日以内であることが分かった．

Herraiz らは重要度や優先度の段階が複数あり複雑であることを問題に挙げ，不具合報告者が適切な重要度や優先度を容易に判断し指定できるようにすることを目的として，分類項目の簡略化のために重要度別，優先度別の滞留時間の違いを統合開発環境の一つである Eclipse を対象に分析した[5)]．重要度別の滞留時間を分析した結果，重要度が高い不具合は低い不具合に比べて滞留時間が短いことが分かった．また，重要度，優先度ともに簡略化が可能であることが分かった．

Mockus らの研究，Herraiz らの研究では滞留時間を導出し分析を行っているが，不具合の滞留時間を短縮するためには，不具合の修正プロセスのの部分で時間を要しているかを明らかにする必要がある．本稿で提案する手法は，修正プロセスを細かく分割することで，修正プロセスのボトルネックとなっている部分を抽出する点に特徴がある．

3. OSS プロジェクトにおける不具合管理と修正プロセス

本章では，不具合管理システムを利用している一般的な OSS プロジェクトの不具合管理プロセスについて説明し，本稿で用いる用語を定義する．

3.1 不具合管理システム利用時の修正プロセス

多くの OSS プロジェクトは，不具合の管理を行うために不具合管理システムを利用して

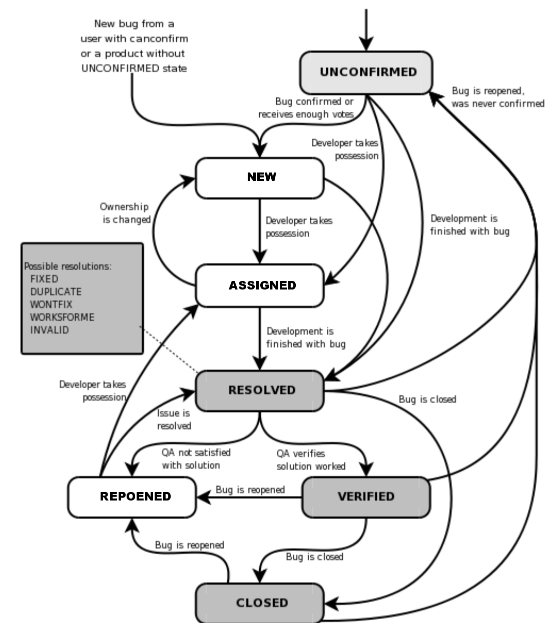

図 1 Bugzilla 利用時の不具合修正プロセス[10)]

表 1　不具合が修正されるまでの状態

状態	詳細
不具合報告	不具合管理システムに不具合の詳細を報告する
未承認不具合登録	不具合管理システムに不具合の詳細を登録する
不具合承認	不具合であることを承認する
修正担当者決定	修正の担当者が決定し，修正が開始される
不具合解決	不具合の修正が完了する
再修正決定	不具合が十分に修正されず，再修正を決定する
修正確認	不具合が修正されたかを検証する

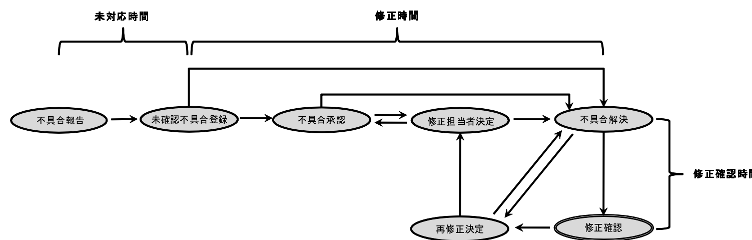

図 2　不具合の修正プロセスと滞留時間

いる．不具合管理システムは，ソフトウェア開発中やリリース後に利用者などから報告された不具合をデータベースに登録し，不具合修正の状況などを一元管理するシステムである．不具合管理システムを導入することで，OSS プロジェクトは，不具合の修正漏れを防止したり，不具合修正の計画立案の補助（どのモジュールの不具合を重点的に修正すべきか，など）などに役立てることができる．

代表的な不具合管理システムには Bugzilla[1]，Trac[11]，Mantis[6]，RedMine[9] などがある．実際の不具合修正プロセスは，**図 1** のような複雑なプロセスとなる場合が多いが，いずれの不具合管理システムにおいても大まかには，不具合が発見されてから修正が完了するまでの不具合修正プロセスには**表 1** のような状態があり，**図 2** のような順序で不具合修正の状態が変化するものとして捉える事ができる．

3.2　未対応時間と修正時間

不具合管理システムに報告された不具合には，アサイン（修正担当者の割当）されない状態のまま放置されているものが数多く存在することが分かっている [cite]．図 2 に示すように本稿では，不具合滞留時間のうち，不具合が報告されてから修正が開始されるまでの時間を未対応時間，また，修正作業に取り掛かってから修正が完了するまでに要した時間を修正時間と定義し，滞留時間を大きく二つに分類する．なお，全ての不具合に修正担当者がアサインされる訳ではないので，修正作業がどの担当者によっていつ開始されたかどうかを判別することが困難な場合がある．本稿では修正に関する議論が開始された時刻を修正が開始する時刻と考え，不具合報告後，最初にコメントが投稿された時刻を修正開始時刻と考える．

3.3　修 正 時 間

OSS プロジェクトにより修正プロセスは多少異なる場合があるが，不具合管理システムに報告される不具合は図 1 のようなプロセスで修正される．修正時間の中でどの部分に長い時間を要しているかを知るためには，ある状態からある状態へ遷移するのにかかる時間を分析する必要がある．特に本稿では，不具合が登録されてから修正が完了するまでの作業と，修正が完了してから修正が本当に完了しているか確認する作業に要する時間をそれぞれ分析する．

4.　OSS プロジェクトを対象とした不具合滞留時間分析手法

本章では，不具合管理システム利用時の不具合滞留時間を分析し，滞留時間を長期化させている要因を特定するための分析手法について述べる．分析手法は，不具合管理システムに蓄積されたデータから不具合修正プロセスのボトルネックとなる部分を抽出するために，(1) 滞留時間を「未対応時間」と「修正時間」に分割し分析するための手法と，(2) 特に複雑なプロセスが存在する不具合修正開始から修正完了までの「修正時間」を詳細に分析するための手法からなる．以下に，それぞれの分析手法について説明する．

4.1　未対応時間と修正時間の分割に基づく滞留時間分析

不具合管理に関する先行研究から，不具合管理システムに報告された不具合には，修正が開始されない状態のまま放置されているものが数多く存在することが分かっている[12]．つまり，不具合が報告されてから修正に取り掛かるまでの時間（本稿では未対応時間と呼ぶ）が長いことが，滞留時間を長期化させる要因の一つになり得る．

そこで，不具合の滞留時間を未対応時間と修正時間の 2 つに分割し，それぞれに要した時間を分析する．滞留時間を未対応時間と修正時間に分割することで，滞留時間を長期化させる次の以下の 3 種類の不具合を抽出することができる．

a　未対応時間のみが長い不具合
b　修正時間のみが長い不具合
c　未対応時間と修正時間が共に長い不具合

これらの 3 種類の不具合のうちどの不具合が最も多いかを OSS プロジェクトの運営・管理者が把握することで，不具合修正をより効率的に行うための対策（例えば [i] の割合が多いプロジェクトでは，不具合への早期対応を開発者に呼びかけるなど）を講じることができるようになると考える．

4.2　状態遷移に基づく修正時間分析

不具合の滞留時間を未対応時間と修正時間に分割することで抽出される不具合のうち，前述の [b] や [c] に分類された不具合が，図 2 示した状態をどのように遷移して修正されたのか，また個々の状態遷移にどれくらいの時間を要しているのかを分析することは，修正プロセスをより効果的に改善するのに役立つと考える．特に本研究では，修正作業そのものに要する時間（未承認不具合登録から不具合解決までの時間）と，修正作業完了後の確認作業に要する時間（不具合解決から修正確認までの時間）に着目する．

4.2.1　修正作業に要する時間

不具合の実際の修正作業では，図 2 より 3 つの不具合修正のパターン（状態遷移のパターン）が存在する．

- 修正担当者が決定されてから不具合が解決される場合（修正担当者決定→不具合解決）
- 不具合と承認された後，修正担当者が決定されずに不具合が解決される場合（不具合承認→不具合解決）
- 不具合と認められず，かつ，修正担当者が決定されずに解決される場合（未承認不具合登録→不具合解決）

これらの分類により，修正時間に最も影響している状態を特定することができる．また，修正担当者を割り当てた場合と割り当てない場合で修正時間に違いがあるかどうかなども知ることができる．

4.2.2　修正確認に要する時間

不具合の修正作業完了後（不具合解決の後），修正が正しく行われたかを確認する作業が存在する．修正確認に関する状態遷移のパターンは，図 2 より 2 つに分類することができる．

- 不具合の修正が確認された場合（不具合解決→修正確認）
- 不具合が十分に修正されていなかった場合（不具合解決→再修正決定）

これらの分類により，修正作業の確認作業で，改めて修正が必要な場合と正しい修正が確認できた場合で，不具合の修正時間にどれくらいの違いがあるのかを知ることが期待できる．

4.3　分析対象外のデータ

不具合管理システムは，不具合の報告だけでなく機能追加要求なども同様のフォームを用いて報告できるため，一種のタスク管理として利用することが可能である．本分析手法は不具合の滞留時間の分析を目的としていることから，機能追加に関するデータは分析対象としない．また，不具合管理システムに登録された不具合の中でも，修正が行われなかった不具合も存在する．例えば，不具合が報告されたが，開発者による確認で不具合と見なされなかった場合や，不具合であるが今後修正される予定がない場合などである．不具合として承認されず，かつ，実際に修正が行われなかった不具合データも本分析手法の対象外とする．

表 2　Firefox の分析対象データの概要

分析対象不具合数	2376 件
データ取得開始日	2001 年 1 月 1 日
データ取得終了日	2008 年 12 月 10 日

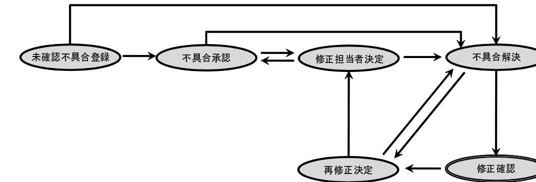

図 3　Firefox プロジェクトにおける不具合の修正プロセス

5.　ケーススタディ

本章では 4 章述べた分析手法を用いて Firefox プロジェクトを対象としてケーススタディを行う．Firefox の分析対象データの概要を**表 2** に示す．分析対象とする Firefox のバージョンは 1.0 系列，2.0 系列，3.0 系列とする．Firefox では**図 3** のプロセスで修正を行っている．

6.　分 析 結 果

6.1　未対応時間と修正時間

滞留時間，未対応時間，修正時間の各統計量を**表 3** に示す．また，滞留時間毎の不具合数の分布と未対応時間毎の不具合数の分布と修正時間の不具合数の分布を**図 4** に示す．左図のヒストグラムが滞留時間に関するグラフで，横軸を滞留時間とする．中央のヒストグラムが未対応時間に関するグラフで横軸を未対応時間とする．右図のヒストグラムが未対応時間に関するグラフで横軸を修正時間とする．ヒストグラムのデータ区間は 1 日である．滞留時間，未対応時間，修正時間が非常に長い (90 日以上) の不具合については 1 つの階級に

表 3　Firefox における不具合の滞留時間

	滞留時間	未対応時間	修正時間
平均 (日)	144.14	7.85	136.29
中央値 (日)	54.88	0.02	47.05
標準偏差	225.85	37.10	221.27
分散	50987.71	1375.96	48940.58
最大値 (日)	2415.40	478.38	2376.38
最小値 (日)	0.00	0.00	0.00

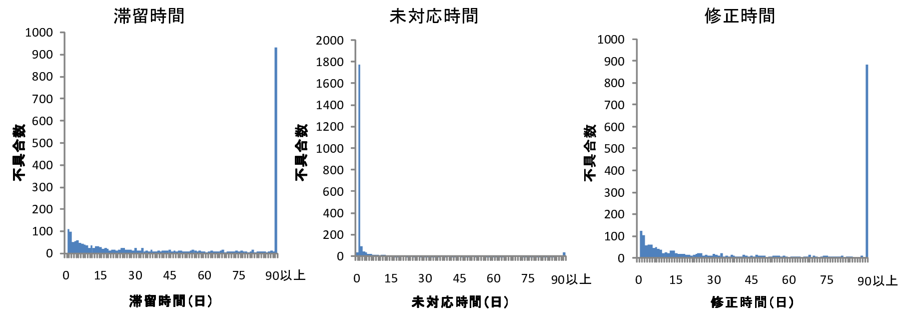

図 4　滞留時間と未対応時間と修正時間の分布

まとめている．未対応時間の中央値が 0.05 日であり，図 4 の未対応時間の分布より，1800 件程度 (全体の約 75%) の不具合は 1 日以内に修正が開始されていることが分かる．一方，修正は中央値が 10.04 日であり，図 4 の修正時間の分布より，450 件程度 (全体の約 20%) の不具合は 1 日以内に修正が開始されているが，3 か月以上の時間を要している．全体的に対応時間が短く，修正時間の長い不具合が多く存在していることから，滞留時間短縮のためには修正作業の見直しが必要であると考えられる．

しかしながら，修正時間が短く，未対応時間に長い時間を要している不具合も存在していることが分かった．このような不具合は実際には短い時間で修正が可能であるにも関わらず長い時間放置されていた不具合であることが考えられ，滞留時間短縮のための一つの改善点であると言える．

6.2　修 正 時 間

図 3 で示した Firefox において，修正過程において不具合の状態が遷移した件数を**図 6** に示す．括弧内の数字は分析データ 2376 件に対する割合である．また，修正過程において状

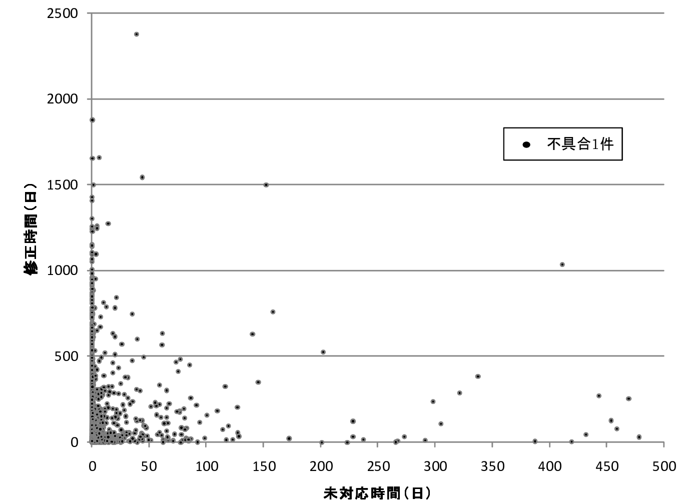

図 5　未対応時間と修正時間の関係

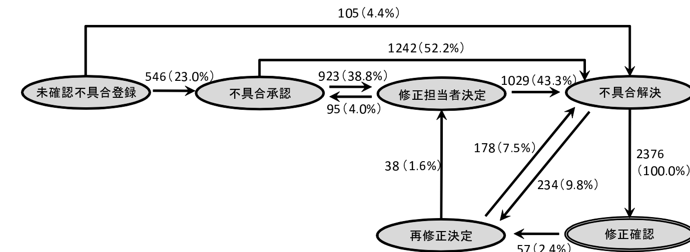

図 6　各修正プロセスにおける状態遷移件数 (数字は各状態間を遷移する不具合の件数，括弧内の数字は分析データ数に対する割合)

態間に要した日数の中央値を示した図を**図 7** に示す．

修正時間において，どのプロセスで長い時間を要しているかを分析した結果，不具合承認から不具合解決に要する時間や不具合解決から修正確認に要する時間が長いことが分かった．

図 6 と図 7 より修正作業と修正確認に要する時間の分析結果をそれぞれ報告する．

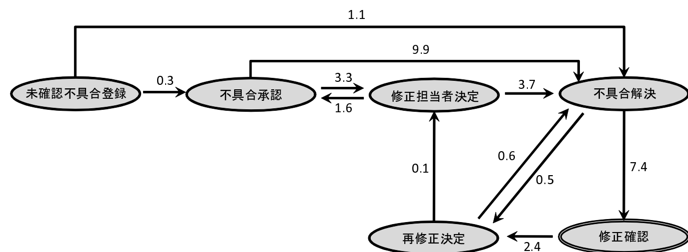

図 7 各修正プロセスに要する時間 (数字は各状態間に要する日数の中央値)

6.2.1 修正作業に要する時間

不具合と認められず，修正担当者が決定されずに解決された (未承認不具合登録→不具合解決) 不具合は 105 件，不具合と認められた後，修正担当者が決定されずに解決された (不具合承認→不具合解決) 不具合は 1242 件，修正担当者が決定されてから解決された (修正担当者解決→不具合解決) 不具合は 1029 件であった．不具合のうち未確認の不具合が修正完了の状態に遷移することは少なく，多くは不具合と認められた後に修正完了，もしくはアサインしてから修正が完了していることが分かった．実際に 3 つの修正プロセスに要する時間を図 7 より分析する．未確認の不具合が完了する場合は全体の中で修正完了までの時間が最も早いことが分かった．一方，修正担当者が決定されずに修正が完了される不具合は 9.9 日であり，修正担当者が決定されてから修正が完了する不具合は 7.0 日 (不具合承認→修正担当者解決→不具合解決のプロセスで要する時間の和) であることから，修正担当者が決定される方が修正に要する時間は早いことが分かった．

6.2.2 修正確認に要する時間

不具合の約 10%は少なくとも一度は再度修正を行っていることが分かった．不具合の修正が確認される (不具合解決→修正確認) ために要する時間は完全に修正が行われていない (不具合解決→再修正決定) と判断された場合の方が確認作業に要する時間が短いことが分かった．図 7 より修正プロセス間の時間で不具合と認められた後に修正完了するプロセス (不具合承認→不具合解決) の次に長い時間を要している箇所が不具合の修正が確認されるプロセスであることから，滞留時間短縮のためには修正確認作業の効率化が一つの改善点であると考えられる．

7. 考　　察

7.1 未対応時間と修正時間

分析手法を適用した結果，未対応時間の長い 309 件 (13%) の不具合は修正が開始されるまでに 1 週間以上の時間を要していたことが分かった．また，修正時間のみが長い 1921 件 (81%) の不具合は修正が解決されるまでに 1 週間以上の時間を要していたことが分かった．

これらの結果は滞留時間を分類したことで得られたことから，本稿で提案した分析手法は修正状態を把握する際に有用であると言える．

これまで，Herraiz らの研究[5] では滞留時間の違いから重要度や優先度の簡略化を行っていた．しかしながら，(a) 未対応時間のみが長い不具合や (b) 修正時間のみが長い不具合が存在するため，滞留時間のみで重要度や優先度の簡略化を行うのは適切ではないと考える．不具合を重要度や優先度別に未対応時間と修正時間を導出することで，より正確な簡略化が実現されると考えられる．

7.2 修 正 時 間

7.2.1 修正に要する時間

分析手法を適用した結果，不具合と認められず，かつ，修正担当者が決定されずに解決される場合の修正に要する時間が最も短いことが分かった．

また,「修正担当者が決定されてから不具合が解決される場合」と「不具合と承認された後，修正担当者が決定されずに不具合が解決される場合」では，修正担当者が決定した場合の方が修正解決が早いことが分かった．そのため，不具合は修正担当者を決定した場合の方が修正が早く解決されること分かった．

修正プロセスの間隔を詳細に分析することで Wang らに研究で修正担当者の有無が滞留時間が長期化する要因であると指摘を裏付けすることが可能となった．

これらの結果は修正プロセスを分類したことで得られたことから，本稿で提案した分析手法は滞留時間を短縮するための修正手法を分析の際に有用であると言える．

7.2.2 修正確認に要する時間

これまでの研究では，修正確認時間も滞留時間として分析されていたことから修正確認にどれだけの時間を要しているかは明らかにされていなかった．しかし，再修正決定に遷移する不具合数を導出することで，全体の 10%程度再修正が行われたことが分かった．さらに，状態の遷移に要する時間を導出することで，修正プロセスに要する時間のうち修正が確認される作業時間に長い時間を占めていることが分かった．これらの結果は修正確認作業を分類

したことで得られたことから，本稿で提案した分析手法は修正確認作業の状態を把握する際に有用であると言える．

8. おわりに

本稿では不具合の滞留時間の短縮を目的として不具合修正のどの部分がボトルネックとなっているかを理解するために不具合管理システム利用時の不具合の滞留時間を分析するための手法を提案した．ケーススタディとして Mozilla Firefox プロジェクトで管理されている不具合の修正に要する時間を詳細に分析したところ，以下の知見を得ることができた．

- 未対応時間と修正時間を分割して分析することで修正状態を把握することができた
- 修正プロセスを分類したことで滞留時間を短縮するための修正手法を明らかにすることができた
- 修正確認作業を分類したことで修正確認作業の状態を把握することができた

しかしながら，各不具合の特徴 (重要度，優先度，不具合の内容，不具合が発見された箇所など) を考慮していない．不具合の特徴により修正プロセスは異なり，長い時間を要するプロセスは異なると考えられる．今後は各不具合の特徴を考慮し，不具合の特徴に合わせた効率的な修正活動の支援を行いたい．

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（若手 B：課題番号 20700028）による助成を受けた．および公益信託マイクロソフト知的財産研究助成基金による助成を受けた．

参考文献

1) Bugzilla: http://www.bugzilla.org/.
2) Canfora, G. and Cerulo, L.: Impact Analysis by Mining Software and Change Request Repositories, *In Proceedings of the 11th IEEE International Software Metrics Symposium*, IEEE Computer Society, p.29 (2005).
3) Chou, A., Yang, J., Chelf, B., Hallem, S. and Engler, D.: An empirical study of operating systems errors, *In Proceedings of the eighteenth ACM symposium on Operating systems principles*, ACM, pp.73–88 (2001).
4) Godfrey, M.W. and Tu, Q.: Evolution in Open Source Software: A Case Study, *In Proceedings of the International Conference on Software Maintenance*, pp.131–142 (2000).
5) Herraiz, I., German, D.M., Gonzalez-Barahona, J.M. and Robles, G.: Towards a simplification of the bug report form in eclipse, *In Proceedings of the 2008 international working conference on Mining software repositories*, pp.145–148 (2008).
6) Mantis: http://www.mantisbt.org/.
7) Michail, A. and Xie, T.: Helping users avoid bugs in GUI applications, *In Proceedings of the 27th international conference on Software engineering*, ACM, pp. 107–116 (2005).
8) 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).
9) RedMine: http://www.redmine.org/.
10) The Bugzilla Team: The Bugzilla Guide–3.3.4 Development Release, http://www.bugzilla.org/docs/3.4/en/html/Bugzilla-Guide.html.
11) Trac: http://trac.edgewall.org/.
12) Wang, Y., Guo, D. and Shi, H.: Measuring the evolution of open source software systems with their communities, *SIGSOFT Softw. Eng. Notes*, Vol.32, No.6, p.7 (2007).