「マルチメディア，分散，協調とモバイル
(DICOMO2014) シンポジウム」 平成26年7月

オープンソースプロジェクト特性に基づく バグ収束過程の理解

坂口 英司[1] シェード ルアンワン[2] 伊原 彰紀[1] アーノン ルンサワン[2] 松本 健一[1]

概要：近年，多くのソフトウェア開発企業でオープンソースソフトウェア（OSS）を利用した事業が増加している．無償で利用でき，拡張性の高い OSS は，企業が求める開発コスト削減に大きく貢献している．しかしながら，OSS 開発の特徴から，リリース直後の OSS は高品質とは必ずしも言えず，企業はどのタイミングの OSS を利用すべきかの意思決定は容易ではない．本論文では，バグ修正曲線に基づき，OSS の品質が安定する時期（修正が完了するバグの増加が逓減する時期）を理解し，更に，バグの修正完了日が遅れる理由を分析する．GIMP プロジェクトを対象に分析を行った結果，バグ報告数にソフトウェアの安定期が関係していることが示唆された．

Understanding Bug Convergence Process in Open Source Projects

Sakaguchi Hideshi[1] Shade Ruangwan[2] Akinori Ihara[1] Arnon Rungsawang[2]
Matsumoto Ken-ichi[1]

1. はじめに

近年，IT システムの開発コスト削減等を目的として多くの企業がオープンソースソフトウェア (OSS) の利用を検討している．独立行政法人 情報処理推進機構 (IPA) の調査によると商用ソフトウェア開発企業が抱える事業の約 4 割は OSS を利用しており，年々 OSS の需要が高まっている [13]．その一方で，企業に OSS の知識を持つ者が少なく，緊急時に OSS プロジェクトからサポートを受けられないなど，OSS の導入に不安を抱える企業も少なくない．

リリース直後に多数のバグが報告される OSS 開発の特徴を鑑みると，後々の保守コストを下げるために企業は未修正バグが少ない OSS を利用することが望ましい．しかし，明確な雇用形態を持たない OSS プロジェクトでは，リリース後の未修正バグを誰が，いつ修正完了するか明らかではないため，ソフトウェアの品質が安定する時期（安定期）を予測にすることは容易ではない．

従来研究では修正完了時期を予測するための研究が数多く行われている [12][7][4]．従来研究では 1 つのバグが修正完了するまでの時間を予測することを目的としているため，多くはバグの内容，ソースコードの複雑さ，修正担当者の技術力などから修正時間を予測している．これらの研究では，リリースされたソフトウェアが抱える全てのバグの修正を考慮した研究ではないため，ソフトウェアの安定期を予測することは目的としていない．

本論文では，OSS のリリース後に未修正のバグ数が逓減し，収束する過程をバグ修正曲線を作成し，ソフトウェアの安定期を分析する．バグ修正曲線とは，横軸に日付，縦軸に修正完了バグ数を描く曲線であり，プロジェクトの進捗状況の確認に用いることができる．また，本論文では，ソフトウェアが抱えるバグの修正が遅延する原因を理解するために，従来研究で挙げられているバグ修正時間に影響する要因（プロダクトの特徴，バグ修正者の特徴，プロジェクト管理状況）によってリリース後，早い段階で修正されたバグ（早期修正バグ）と，修正完了日が遅れたバグ（長期修正バグ）に違いがあるかを分析する．

バグ修正曲線と類似するグラフとして，ソフトウェア開

[1] 奈良先端科学技術大学院大学
Nara Institute Science and Technology
Takayama, Ikoma-shi, Nara 630–0192, Japan
[2] カセサート大学
Kassesart University
Phahonyothin Rd., Ladyao, Chatuchak, Bangkok 10900, Thai

発企業で，一般的に使用されているソフトウェアの信頼度成長曲線はテストケースの実行に対する累積バグの発見数をとったグラフがある．本論文では OSS 開発を対象としており，リリースされてからバグが発見されることが多く，それらがいつ修正され，ソフトウェアがいつ安定期をむかえるかを理解することが本論文の課題である．

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, 本論文で用いるバグ修正曲線について述べる．4 章では，バグ修正曲線の分析方法とバグ修正時間に影響を与える要因の分析方法について述べる．章では，GIMP プロジェクトが管理するバグを対象としたケーススタディを行った結果を報告する．6 章でケーススタディを通じて得られた知見から OSS の安定期について考察を行う．最後に 7 章で本稿のまとめと今後の課題を述べる．

2. 関連研究

本章では，本論文の位置づけを説明するために，従来研究で行われたバグ修正時間に関する研究，及び，バグ修正曲線に類似する信頼度成長曲線について述べる．

2.1 バグ修正時間の関連研究

従来研究では，バグ報告内容（機能名，バグ発生環境，等）を用いて，バグ修正時間を予測する研究が行われてきた [3][4][7][12]．Weiss ら [7] は，類似するバグは修正時間も同程度かかるという仮説を示し，バグの修正時間を予測する手法を提案している．また，Hewett ら [4] は，バグ報告時に記録されるコンポーネントや優先度などの情報を用いた予測モデルを提案している．

近年では，バグ修正者の情報を用いることで，予測精度の向上が試みられている [1][2][5][6]．Bhattacharya ら [1] は，修正者の評判に基づくバグ修正時間の予測モデルを構築している．また，Marks ら [5] は，バグが一定期間内（3 ヵ月，1 年，3 年）に修正されるか否かの予測を行うことで予測精度の向上を達成している．

従来研究では，1 件のバグ修正にかかる時間を，バグの特徴，修正者の特徴を用いて修正時間を予想しているが，刻一刻と変化する OSS プロジェクトでは，プロジェクトのタスク量なども考慮した上で修正時間を検討する必要がある．特に，OSS プロジェクトには，日々多くのバグ報告が寄せられており，Mozilla プロジェクトでは，一日に数百件報告されることもある [9]．また，OSS プロジェクトでは開発者の流動性が高く，新たに参加する開発者もいれば，去っていく開発者も存在する．従って，プロジェクトが抱えるバグ修正にかかる時間は，プロジェクトが抱える未修正バグ数や，貢献者の人数にも影響すると考えられる．従って，本論文では，修正時間に影響する要因としてプロダクトの特徴，バグ修正者の特徴，プロジェクトの管理状況からソフトウェアの安定期を分析する．

2.2 ソフトウェアの信頼度成長曲線

ソフトウェア信頼性は「ソフトウェアが，規定の環境の下で，意図する期間中に，ソフトウェア故障が発生することなく動作することができる性質や度合」と定義 [11] されており，従来ソフトウェア信頼度成長モデル（SRGM: Software Reliability Growth Model）が提案されている．SRGM は，デバッグ期間中の単位時間あたりに発見された欠陥数，単位プログラムコードあたりの平均欠陥数などをもとに，「テスト開始前にソフトウェア内に存在する欠陥数の期待値」や「ある時点でのソフトウェア内の残存欠陥数の期待値」を予測する手法である．SRGM はソフトウェアがリリース前に，プロジェクトが抱える未発見欠陥に対して，どれだけ欠陥が残存しているのかを評価する手法であり，本論文が対象とする，リリース後にソフトウェアが安定する時期を理解するための研究とは目的が異なる．

3. バグ修正曲線

バグ修正曲線は，ソフトウェアに発見されたバグが修正されていく過程を分析するためのグラフである．横軸をバグの修正完了日，縦軸を修正完了バグ数とし，ソフトウェアリリース後に修正が早く完了したバグをグラフ下部にプロットする．曲線から，プロジェクトが修正完了バグ数を把握し，修正状況，未修正バグの収束過程をモニタリングすることができる．

本論文では，バグ修正曲線を作成するために，OSS プロジェクトが使用しているバグ管理システムに記録された情報を用いる．バグ管理システムは，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バグ情報（バグの深刻度，再現方法）をデータベースに登録し，バグの修正状況（進捗状況，修正されたソースコード，開発者からのコメント）などを一元管理するシステムである．バグ修正曲線は，各バグが修正完了した日付，バグが再現されるバージョン名を取得し，バージョン別に未修正のバグ数が逓減し，収束する過程を示す．

4. 分析方法

本章では，3 章で説明したバグ修正曲線を用いて（分析 1）ソフトウェアで発見されるバグの収束時期（ソフトウェアの品質が安定する時期），及び，(分析 2) プロジェクト特性による修正完了時期の違いの分析を理解するための分析方法を述べる．

4.1 分析 1: ソフトウェア安定時期の分析

OSS を利用する企業は，高品質なソフトウェアの利用を求めるため，未修正バグが収束する時期を把握する必要がある．しかしながら，3 章で説明したバグ修正曲線は，現

表 1 分析 2 で用いるメトリクス

要因	メトリクス名	説明
プロダクト	CountLineCode	ソースコード行数
	AvgCyclomatic	サイクロマチック複雑度
	CountDecFunction	関数の数
修正者	BugFixers	バグ報告されてから修正が完了するまでに修正対象ファイルを変更した開発者数
	AuthorsExperience	バグ修正者が当該バグ以外に修正した経験のあるバグ数
プロジェクト	ExistedBugsInOpen	バグ報告された時点における未修正バグ数（最終的に修正されなかったバグも含む）.
	CommitterBeforeOpen	修正対象ファイルに変更を加えた開発者数.

在の収束状況を把握することは可能であるが，残りどれだけのバグが発見され，修正されるか明らかではない．本論文では，バグ管理システムに登録されている，バグ修正記録を用いてバグ修正曲線を作成し，過去にリリースされたバージョンのソフトウェア安定期を分析する．

次に，リリースから時間が経過するにつれ修正されるバグ数が減少していく特徴から，修正完了バグ数の標準偏差を用いてソフトウェア安定時期を理解する．具体的には，7 日前時点の修正完了バグ数，6 日前までの修正完了バグ数，5 日前までの修正完了バグ数，というように 7 日前から順に修正完了バグ数を計測し，7 つの値（修正完了バグ数）の標準偏差 SD を導出し，SD の推移を分析する．リリースからの経過時間が長くなるにつれ，修正完了されるバグ数は減少することから SD の値も小さくなるため，SD の値からソフトウェア安定期を見積もることができると期待できる．

4.2 分析 2: 修正完了時期の違いの分析

従来研究で，バグの修正時間に関する研究は数多く行われているが，プロジェクトの特徴を考慮した研究はほとんど行われていない．本論文では，短い時間で修正されたバグ（早期修正バグ）と，長い時間かかって修正されたバグ（長期修正バグ）に関して，プロダクトの特徴，バグ修正者の特徴，プロジェクト管理状況の違いを分析する．プロダクトの特徴（プロダクト），バグ修正者の特徴（修正者），プロジェクト管理状況（プロジェクト）における具体的なメトリクスは表 1 に示す．プロダクトの特徴として，ソースコードが複雑であるほど，ソースコード理解に時間がかかり，且つ，修正後に複数人による検証作業が求められるため [8]，修正完了日が遅れると考えられる．また，修正担当者数が変更されるほど，バグに対する知識が浅い者が担当することになり，修正時間が絶対的に長期化するが [10]，各バージョンにおいても修正完了日が遅くなる考えられる．また，プロジェクト管理状況として，バグの報告数や，修正する担当者が少ないほど修正完了日が遅れると考えられる．本論文では，早期修正バグと長期修正バグにおける各メトリクス値に統計的有意差があるかを確認し，未修正バグが収束時期に影響を及ぼす要因を明らかにする．

表 2 GIMP の歴史

バージョン	変更内容
1.x.x	専門家に使用を勧められる十分な安定化を果たした.
2.0.x	機能は 1.x.x 系列とインタフェースはほとんど変更されていないが，ソースコードの管理，変更を容易に実現するために，ほとんどのソースコードが書き換えられた.
2.2.x	相互運用性と標準のサポート，ショートカットエディター，プレビュー機能などが追加された.
2.4.x	外観の刷新，ブラシ機能，選択ツールなどが追加された.
2.6.x	ユーザインタフェースの改善，手書きツール機能の追加，ブラシ感度の調整が行われた.

表 3 GIMP プロジェクトで修正されたバグ数

バージョン番号	1.x.x	2.0.x	2.2.x	2.4.x	2.6.x
バグ報告数	1,777	621	1,917	900	1049
修正完了バグ数	814	200	398	159	261
バグ修正完了率	45.8%	32.2%	20.8%	17.7%	17.0%

5. ケーススタディ

本章では，GIMP プロジェクトのバグ修正履歴を用いて，3 章述べたバグ修正曲線を作成し，バグ修正の効率化に影響する 3 つの観点で（プロダクトの特徴，バグ修正者の特徴，プロジェクト管理状況）修正完了日が遅れるバグの要因を分析する．

5.1 対象プロジェクト

本論文では，大規模プロジェクトで多くのユーザが利用している GIMP を対象にケーススタディを行う．GIMP は，これまでにバージョン 1.0，2.0，2.2，2.4，2.6，2.8 系列をリリースしており，本論文では最新の 2.8 系列以外を対象とする．それぞれのバージョンの更新内容の概略を 2 [*1] に示す．また，表 3 は，各バージョンの変更行数，発見されたバグ数，修正完了済みのバグ数を示す．GIMP プロジェクトでは，報告されたバグのうち約 46%〜17%のバグが修正されており，最近リリースされたバージョンほど修正されるバグの割合は低下している．

[*1] GIMP の歴史: http://docs.gimp.org/ja/gimp-introduction-history.html

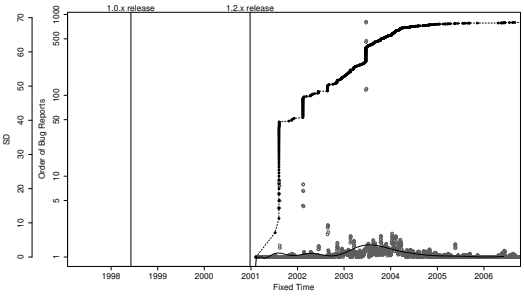

図 1 バージョン 1.x.x のバグ修正曲線と SD

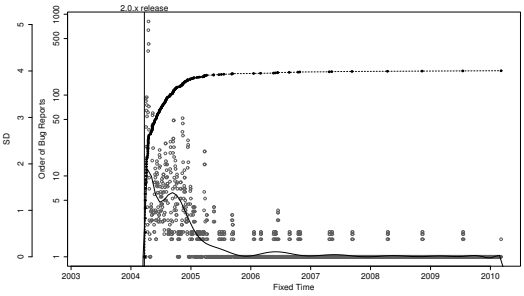

図 2 バージョン 2.0.x のバグ修正曲線と SD

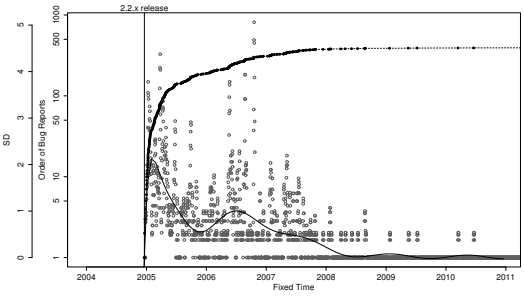

図 3 バージョン 2.2.x のバグ修正曲線と SD

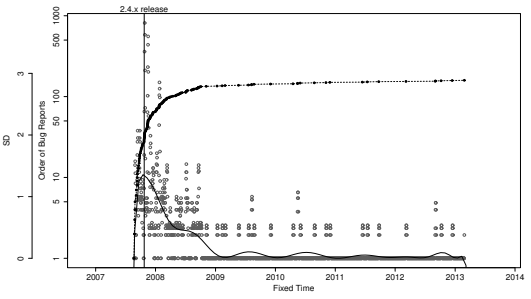

図 4 バージョン 2.4.x のバグ修正曲線と SD

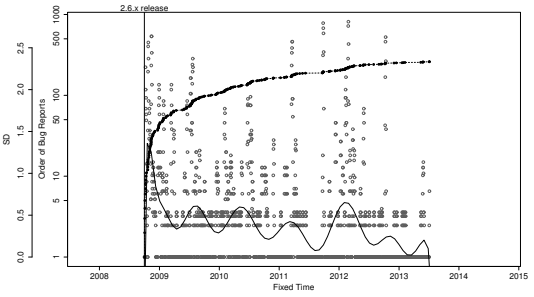

図 5 バージョン 2.6.x のバグ修正曲線と SD

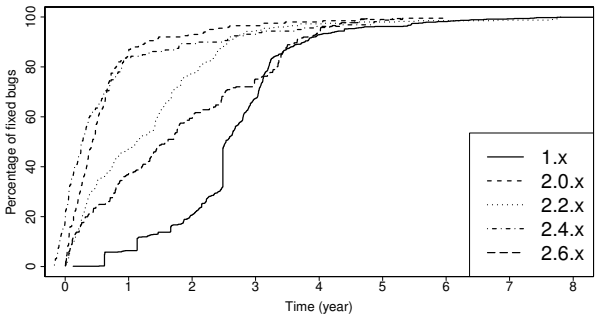

図 6 リリース後の経過時間によるバグ修正完了率の推移

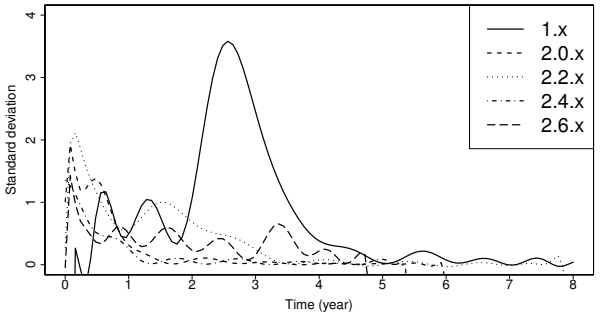

図 7 リリース後の経過時間による SD の推移

5.2 分析結果

5.2.1 分析 1: ソフトウェア安定時期の分析

GIMP プロジェクトにおける各バージョンのバグ修正曲線を図 1〜5 に示す．横軸は時間，左側の縦軸は SD，右側の縦軸は修正完了バグ数を対数軸で示す．黒丸は各バグの修正完了時点，白丸は SD，曲線は SD の近似曲線を表す．

バージョン 1 系列を除いて，多くの場合，リリースして 1 年以内に 50%以上のバグが修正されている．バージョン 2.0，2.4 系列は 2.2，2.6 系列に比べて，リリースから時間が経過するにつれ黒点の間隔が広くなっており，リリース直後の約 1 年で多くのバグが修正されている．図 6 はリリース後の経過時間による修正完了バグの割合を示す．横軸はリリース後の経過時間，縦軸は修正完了バグ数の割合を示す．バージョン 2.0，2.2 系列は，80%バグが修正完了するまでにそれぞれリリース後 232 日，195 日，90%のバグが修正完了するまでに 422，262 日要している．一方で，バージョン 2.2，2.6 系列はリリースは，80%バグが修正完了するまでにそれぞれリリース後 611 日，897 日，90%のバグが修正完了するまでに 775 日，1086 日かかっていることが分かった．表 3 より，バージョン 2.0，2.2 系列のバグ報告数は，2.4，2.6 系列よりも少なく，バグ報告数がソフトウェアの安定期が遅延する一要因である可能性がある．

GIMP プロジェクトではバグ管理システム Bugzilla をバージョン 1.2 をリリース頃に使い始め，バージョン 1.0 と区別していない．従って，本論文ではバージョン 1.x.x

表 4 分析 2 の結果

観点	メトリクス名	統計的有意差のあるバージョン				
		1.x.x	2.0.x	2.2.x	2.4.x	2.6.x
プロダクト	CountLineCode	**	**	–	**	–
	AvgCyclomatic	**	**	–	–	–
	CountDecFunction	**	**	–	**	–
修正者	BugFixers	**	**	**	**	**
	AuthorsExperience	–	–	–	–	–
プロジェクト	ExistedBugsInOpen	**	**	**	**	**
	CommitterBeforeOpen	**	**	**	**	–

と記している．バージョン 1 系列は，特定の時期にまとめて修正完了している時期があるが，修正が終わっていてもシステムに登録されている情報を更新しておらず，管理者がまとめてバグの状態を完了と書き換えた可能性があるため言及しなかった．プロジェクト開始当初はバグの修正管理があまりできていないため，ソフトウェアの安定期を見積もることは難しいと示唆される．

次に，リリース後にソフトウェアの残存バグ数を把握することは容易ではないため，本論文では SD の推移からソフトウェア安定時期を見積もることができるかを分析する．図 7 は，リリース後の経過時間による SD の推移を示す．リリース後の 1 年間は SD の増減が激しいが，1 年〜2 年を過ぎると，バージョン 1 系列を除いて SD が 0.5 未満で安定することから，バージョンアップが少なく安定したソフトウェアが提供されていることが示唆される．

5.2.2 分析 2: 修正完了時期の違いの分析

本論文では早期修正バグを修正完了バグの中でリリース後，早期に修正された 20%のバグとし，長期修正バグを修正完了時期が遅い 20%のバグとする．バグ修正時間に影響する 3 つの観点（プロダクトの特徴，バグ修正者の特徴，プロジェクト管理状況）に関するメトリクス値が早期修正バグと長期修正バグで優位な統計的有意差（有意水準 5%）を確認するためにマンホイットニーの U 検定を行い，結果を表 4 に示す．統計的有意差が確認された場合「**」，確認されなかった場合「–」と表す．また，特徴的な結果を得た，CountLineCode（プロダクト），CountDecFunction（プロダクト），BugFixers （修正者），ExistedBugInOpen（プロジェクト）については，早期修正バグと長期修正バグの分布の違いを図 8〜図 11 に示す．横軸は各バージョンと長期修正バグを long，早期修正バグを short と示す．縦軸は各メトリクス値を示す．

表 4 より，BugFixers （修正者）と ExistedBugInOpen（プロジェクト）に関しては全てのバージョンで早期修正バグと長期修正バグに統計的有意差を確認した．一方で，AuthorsExperience（修正者）ではいずれのバージョンにおいても統計的有意差を確認できなかった．BugFixers（修正者）は，早期修正バグの中央値は 1.0 系列を除いて 1 人，長期修正バグの中央値が 2〜3 人であった．また，

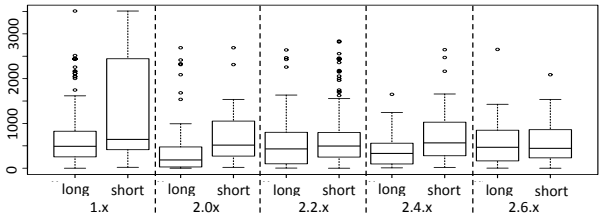

図 8 早期修正バグと長期修正バグの CountLineCode の分布

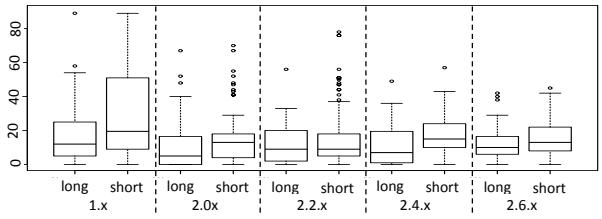

図 9 早期修正バグと長期修正バグの CountDecFunction の分布

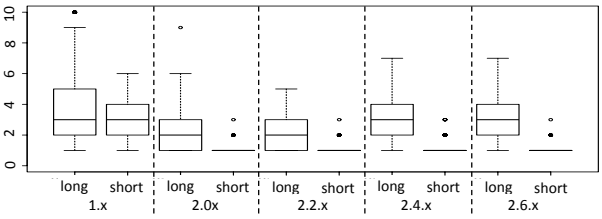

図 10 早期修正バグと長期修正バグの BugFixers の分布

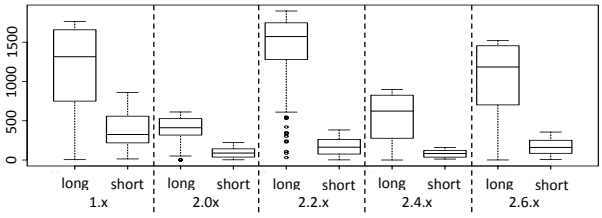

図 11 早期修正バグと長期修正バグの ExistedBugInOpen の分布

ExistedBugInOpen（プロジェクト）は，早期修正バグの中央値が 83 件〜325 件，長期修正バグの中央値が 411 件〜1572 件であった．これらのメトリクスと安定期までの時間との関係を分析することで，相関を分析することで，ソフトウェアの安定期を予測できると期待される．しかし，本論文では GIMP のみ，且つ，5 つのバージョンのみを対象としているため，相関の分析結果を議論するために十分なデータセットが準備できていないため，今後の課題とする．

また，リリースからソフトウェアの安定期までの時間が

短かった 2.0，2.4 系列では CountLineCode（プロダクト），CountDecFunction（プロダクト）で早期修正バグと長期修正バグに統計的有意差が確認できたが，安定期までの時間が長かった 2.2，2.6 系列では統計的有意差が確認できなかった．2.0，2.4 系列の早期修正バグの CountLineCode（プロダクト）は 185 行，330 行である一方，2.2，2.6 系列のではそれぞれ 431 行，467 行であった．また，2.0，2.4 系列の早期修正バグの CountDecFunction（プロダクト）は 13，15 である一方，2.2，2.6 系列のではそれぞれ 9，13 であった．以上より，早期修正バグの CountLineCode，CountDecFunction を分析することで，ソフトウェアの安定期が遅延するか否かを予測が可能になると期待される．

6. 議論

6.1 OSS 安定時期の予測

従来研究では，各バグの修正時間を予測を目的としていた．それに対して，本論文ではリリース後のバグの修正完了日に着目しており，未修正のバグ数が逓減し，収束するソフトウェアの安定期を理解することを目的として分析を行った．GIMP を対象にケーススタディを行ったバグ修正曲線がバージョンにより異なる曲線であり，特に，バグ報告数の多いバージョンが安定期が遅延する一要因であることが示唆された．各バグの修正時間を予測する場合とは異なり，プロジェクトが抱えるバグの修正はプロジェクトの開発状況，進捗が要因となり得る．その他にも，開発者の参加数，ソースコードの変更行数などが考えられる．ソフトウェア安定期に関係すると示唆されるメトリクスの検討は今後の課題とする．

6.2 制約

本論文では，バグ管理システムに登録された情報と，修正を要するソースコードを紐づけるために，ソースコードにコミットされるときに登録されるコメントにバグの ID が記されたバグのみ対象とした．開発者がソースコードを変更する際に ID を記さない場合もあり，実際は，本論文で対象としたソースコードよりも多くのソースコードがバグ修正のために変更された可能性もある．現在，バグとソースコードの紐付けの技術が数々提案されているため，今後の技術発展によって，より正確な結果が得られると期待される．

本論文では大規模プロジェクト，且つ，開発が開始され長期間バージョンアップを継続している GIMP プロジェクトを対象にケーススタディを行った．未修正バグの収束過程は，分析 2 において着目した修正時間に影響を及ぼす要因（プロダクトの特徴，バグ修正者の特徴，プロジェクトの特徴）によってそれぞれ異なるため，必ずしも分析 2 の結果が他のプロジェクトに適用できるとは限らない．今後，他のプロジェクトでも同様の実験を行い，本分析結果について検討する．

7. おわりに

本論文では，OSS リリース後にソフトウェアの品質が安定する時期（安定期）を理解することを目的に，OSS である GIMP がリリース後に未修正のバグ数が逓減し，収束する過程を表すバグ修正曲線を作成し，ソフトウェアの安定期を分析した．そして，従来研究で挙げられているバグ修正時間に影響する要因（プロダクトの特徴，バグ修正者の特徴，プロジェクト管理状況）によって短い時間で修正されたバグ（早期修正バグ）と，長い時間かかって修正されたバグ（長期修正バグ）に違いがあるかを分析した．分析からの知見を以下に示す．

- バグ報告数が多いほど，ソフトウェアの安定期が遅延することが示唆された．
- BugFixers（修正者）と ExistedBugInOpen（プロジェクト）から修正完了日が遅れるバグを特定できることが分かった．また，CountLineCode，CountDecFunction を分析することで，ソフトウェアの安定期が遅延するか否かを理解できると示唆された．

ソフトウェアの安定期を知ることは，コスト削減を目的として OSS を利用するソフトウェア開発企業にとって重要であるため，精度の高いソフトウェア安定時期の予測技術が求められている．本論文が，ソフトウェア安定時期の予測のための一助となることを期待する．

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] Pamela Bhattacharya, Iulian Neamtiu, “Bug-fix Time Prediction Models: Can We Do Better?”, Proceedings of the 8th Working Conference on Mining Software Repositories (MSR’10), pages. 207–210, 2011.

[2] Emanuel Giger, Martin Pinzger, Harald Gall, “Predicting the Fix Time of Bugs”, Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering (RSSE’10), pages. 52–56, 2010.

[3] Nguyen Duc Anh, Daniela S. Cruzes, Reidar Conradi, Claudia Ayala, “Bug-fix Time Prediction Models: Can We Do Better?”, Proceedings of the 7th International Conference on Predictive Models in Software Engineering (Promise’11), pages. 1–10, 2011.

[4] Rattikorn Hewett, Phongphun Kijsanayothin, “On modeling software defect repair time”, Journal of Empirical Software Engineering, vol. 14, no. 22, pages. 165–186, 2009.

[5] Lionel Marks, Ying Zou, Ahmed E. Hassan, “Studying the Fix-time for Bugs in Large Open Source Projects”, Proceedings of the 7th International Conference on Predictive Models in Software Engineering (Promise’11), pages. 523–526, 2010.

[6] Anbalagan Parasanth, Vouk Mladen, “On Predicting the Time ttake to Correct Bug Reports in Open Source

Projects”, Proceedings of the International Conference on Software Maintenance (ICSM’09), pages. 1–8, 2011.
[7] Cathrin Weiss, Rahul Premraj, Thomas Zimmermann, Andreas Zeller, “How Long Will It Take to Fix This Bug?”, Proceedings of the 4th International Workshop on Mining Software Repositories (MSR’07), pages. 1–8, 2007.
[8] Kazuki Hamasaki, Raula Gaikovina Kula, Norihiro Yoshida, A. E. Camargo Cruz, Kenji Fujiwara, Hajimu Iida, “Who does what during a code review? datasets of OSS peer review repositories”, Proceedings of the 10th International Workshop on Mining Software Repositories (MSR’13), pages. 49–52, 2013.
[9] Pieter Hooimeijer, Westley Weimer, “Modeling bug report quality”, Proceedings of the 22nd International Conference on Automated Software Engineering (ASE’07), pages 34–43, 2007.
[10] Gaeul Jeong, Sunghun Kim, Thomas Zimmermann, “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), pages. 111-120, 2009.
[11] 山田茂, 高橋宗雄, “ソフトウェアマネジメントモデル入門”, 共立出版株式会社, 1993.
[12] 正木仁, 大平雅雄, 伊原彰紀, 松本健一 “OSS 開発における不具合割当パターンに着目した不具合修正時間の予測”, 情報処理学会論文誌, vol. 54, no. 2, pages.933-944, 2013.
[13] 独立行政法人 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)”, 2010.