マルチプラットフォーム向けソフトウェアに関する特定OS向け欠陥修正コミットの分析

松本　卓大[1,a)]　亀井　靖高[1,b)]　Shane McIntosh[2,c)]　鵜林　尚靖[1,d)]

概要：本稿では，マルチプラットフォームの進化においてかかるコストを定量的に明らかにすることを目的とする．Git 等の版管理システムにおける欠陥修正の変更（コミット）において，特定の OS 向けの修正を対象とし，各 OS 向けの修正コミットの数や修正期間，修正内容について調査を行う．本稿では 3 つのリサーチクエスチョンを実験的に検証した．データセットとして 4 つのオープンソースソフトウェア（PostgreSQL，SQLite，Qt5，Gtk）から計測したデータを用いた結果，1）特定 OS 向け欠陥修正が全体の欠陥修正に占める割合は 1.9%から 38.7%である，2）プロジェクトの進化の過程で，特定 OS 向けの欠陥修正は継続的に発生する，3）ビルドに関する修正やテストに関する修正といった，開発者の知識が必要となる修正コミットが存在する，といったマルチプラットフォーム向けソフトウェアのコストを見積もるための知見を得た．

キーワード：ソフトウェア進化，マルチプラットフォーム，オープンソースソフトウェア，OS，コスト

1. はじめに

ソフトウェアは，出荷後もユーザの要求や外部環境の変化に応じて，継続的に変化し続けなければならない [8]．このようなソフトウェアの運用及び保守（進化）は，初期のバージョンがリリースされた後も長期間にわたり継続する．ソフトウェアの進化に対してかかるコストは想定されるソフトウェアシステム全体のコスト（開発工数）の小さい場合では 50%，大きい場合では 90%以上になるといわれている [7]．ソフトウェアにかかるコストの見積もりはプロジェクトを成功させる上で重要であり，ソフトウェアの進化にかかるコストの把握はソフトウェアにかかるコストの見積もりにおける判断材料の 1 つである [9][13]．

また，ソフトウェアをマルチプラットフォーム向けに提供する場合，複数の OS に対応するために各 OS ごとに特有のファイル，コード，ライブラリが必要となることが多い．ソフトウェアの進化や対応させる各 OS 自体の進化に伴う，これらの特有のファイル，コード，ライブラリの変更や追加により欠陥が発生することが考えられる．最近の事例として， Mac OS X Yosemite 上でブラウザの Safari を動作させる場合に動作が極端に遅くなる不具合や，Photoshop で範囲選択を反転するとクラッシュするといったような不具合が報告されている [*1]．

このように，マルチプラットフォーム向けソフトウェアの進化において，特定 OS 向けに対応させるための変更や追加により発生する欠陥は，単一の OS 向けに開発されるソフトウェアでは発生しないコストである．このような特定 OS のみで発生する欠陥の修正（特定 OS 向け欠陥修正）の数，修正が発生する時期，修正内容などを明らかにすることができれば，マルチプラットフォーム向けソフトウェアの進化にかかるコストを定量的に示せることが期待できる．

そこで本稿では，マルチプラットフォーム向けソフトウェアの進化にかかるコストを定量的に明らかにすることを目的とし，特定 OS 向け欠陥修正コミットを分析する．本研究におけるコストの定義としては，ソフトウェア進化における欠陥の修正回数を基準とする．具体的な分析方法としては，Git[*2] で管理されたオープンソースソフトウェアとして公開されているデータから，欠陥修正のために行われたコードの変更や追加をコミットログから抽出する．

[1] 九州大学
Kyushu University
[2] Queen's University
[a)] 2IE15096T@s.kyushu-u.ac.jp
[b)] kamei@ait.kyushu-u.ac.jp
[c)] shanemcintosh@acm.org
[d)] ubayashi@ait.kyushu-u.ac.jp

[*1] https://discussionsjapan.apple.com/community/mac_os/os_x_yosemite
[*2] Git とは，プログラムのコードなどの変更履歴を，記録や追跡するための分散型バージョン管理システムである．

抽出した欠陥修正の中から特定 OS でのみ発生する欠陥を修正するために行われたコミットを抽出する．抽出したコミットを基に，1) 特定 OS 向け欠陥修正はどの程度存在するのか，2) ソフトウェアの進化によって特定 OS 向け欠陥修正の数はどのように推移するのか，3) 特定 OS 向け欠陥修正の修正内容はどのように分類されるか，という 3 つのリサーチクエスチョンから，マルチプラットフォームにおける進化にかかるコストを明らかにするために特定 OS 向け欠陥修正の調査を行う．リサーチクエスチョンの実施には，オープンソースソフトウェア開発プロジェクトから収集した 4 つのデータセット（PostgreSQL，SQLite，Qt5，Gtk) を用いた．

以降，2 章では，関連研究について紹介する．3 章では，本稿で取り組む 3 つのリサーチクエスチョンについて述べる．4 章では，ケーススタディについて述べる．5 章では，結果について議論を行う．6 章では，本研究の制約について述べる．最後に 7 章でまとめについて述べる．

2. 関連研究

2.1 ソフトウェア進化

これまでにも，ソフトウェア進化における研究は多く行われてきた [5][12][14].

Vasa らによる研究 [12] では，オブジェクト指向により開発されたソフトウェアシステムを対象として，ソフトウェアシステムのサイズや開発人数，クラス，インターフェースについて，どの箇所が複雑であり，どの箇所が安定しているかについての分析を行った．その結果，比較的コードの行数が少ないものは時間をかけて修正されるといった知見を報告している．

Kim らによる研究 [5] では，大規模なオープンソースソフトウェアの進化の過程から，リファクタリングと欠陥修正の関連性について調査した．その結果，API レベルのリファクタリングを行った後に欠陥修正の数が増えているといった知見を報告している．

村尾らによる研究 [14] では，ソフトウェアの大規模化，及び開発期間の短縮化により，ソフトウェアの全てのモジュールに注力することが困難になったことを問題としている．その解決のために，メトリックス値の恒常性という，開発及び保守過程においてどの程度ソフトウェアメトリックスの値が安定しているかを表す概念を用いて，労力を注ぐべきモジュールを特定する手法を提案している．評価実験を通して，将来問題の発生しやすいモジュールを，既存手法に比べて高い精度で特定できることを示した．

本研究では，ソフトウェア進化の中でも，特にマルチプラットフォーム向けソフトウェアの進化に着目して分析を行う．

2.2 欠陥修正

欠陥修正についての研究も，これまでに盛んに行われてきた [6][11].

Kim らによる研究 [6] では，欠陥修正の数だけではなく，欠陥が修正されるまでの時間も欠陥を分析する上で重要だとしている．ある欠陥が発見されてからその欠陥が修正されるまでにかかる時間が比較的長かった場合，その欠陥が含まれるファイルには変更が困難な構造的な欠陥がある可能性がある．ArgoUML と PostgreSQL で発生した欠陥について，その欠陥が発生してから修正されるまでの欠陥修正時間を計算し，その結果，欠陥修正時間の中央値は 200 日であることや，2 つのプロジェクト内の欠陥修正に時間を要した上位 20 のファイルなどを報告している．

Tian らによる研究 [11] では，Linux の欠陥修正パッチを自動的に特定するために，キーワードに基づく特定方法に加えて，実際のコードの変更の情報を特定に用いる方法を提案している．従来の欠陥修正を特定するキーワードに基づくアプローチと比較を行った結果，従来のキーワードに基づくアプローチに比べて Tian らの手法の方が Recall が 53.19%向上したといった知見を報告している．

本研究では，欠陥修正の中でも特定 OS 向け欠陥修正コミットに着目してその欠陥数や欠陥数の推移を分析する点が新しい．

2.3 ソフトウェアにかかるコスト

ソフトウェアにかかるコストに関する研究は，様々な観点から研究が行われてきた [10][4].

McIntosh らによる研究 [10] では，ビルドメンテナンスの観点から，ソフトウェアにかかるコストについて分析を行っている．ビルドメンテナンスにおいて開発者が負担するコストについて 1）ビルドの同時開発（例えばソースコードの変更にともなったビルドの変更はどのぐらいの頻度で必要となるのか），2）ビルドの保有率（例えば，ビルドメンテナンスを担当する開発者の割合がどの程度であるか) の 2 つの観点から分析を行い，1）ソースコードの変更率よりもビルドシステムの変更率の方が相対的に大きい，2）ビルドの専門家がいるプロジェクトとそうでないプロジェクトを比較した場合，ビルドの専門家がいるプロジェクトの方がそうでないプロジェクトに比べ，ビルドに従事する開発者の割合が 20%以上小さい，つまり，ビルドの専門家がいれば，新たにビルドの学習を行う開発者の人数を小さくできるといった知見を報告している．

Yujuan らによる研究 [4] では，インフラストラクチャを自動で構築するためのコードである Infrastructure-as-code (IaC) ファイルに焦点をあてて，ソフトウェアにかかるコストについて分析を行っている．IaC は新しい種類のソースコードであり，専門的なプログラミングの知識を必要とする．IaC ファイルの数，修正頻度，修正規模の観点から

分析を行い，1）プロジェクトに含まれる IaC ファイルは 3.85%から 11.11%である，2）IaC ファイルのファイルサイズは同一命令のためのテストファイルやビルドファイルに比べ大きい，3）IaC ファイルの 28%は月ごとに変更が行われているといった知見を報告している．

本研究では，マルチプラットフォーム向けのソフトウェア特有の変更に焦点をあて，その観点からソフトウェアにかかるコストの調査を行う．

2.4 マルチプラットフォーム向け開発

マルチプラットフォーム（またはクロスプラットフォーム）に関する Cusumano らによる研究 [3] では，マルチプラットフォーム向けソフトウェア開発の難しさについて議論を行っている．Netscape のマルチプラットフォーム開発から，開発工数や開発期間を対象として分析を行い，マルチプラットフォーム開発の行われていない製品に比べて，コストの増加や性能低下といった知見を報告している．

本研究では，マルチプラットフォーム向けソフトウェアにかかるコストを明らかにするために，特定の OS のための修正を分析対象として，大規模なデータに基づく定量的な分析を行っている．

3. Research Question

本稿では，マルチプラットフォーム向けソフトウェアの進化にかかるコストを定量的に明らかにすることを目的として，特定 OS 向け欠陥修正コミットの調査を行う．我々は，特定 OS 向け欠陥修正コミットの調査にあたって 3 つのリサーチクエスチョン（RQ）を設定した．RQ1 では，特定 OS 向け欠陥修正はどの程度存在するかについて調査する．RQ2 では，ソフトウェアの進化によって特定 OS 向け欠陥修正の数はどのように推移するかについて調査する．RQ3 では，特定 OS 向け欠陥修正はどのような修正が多いのかについて調査する．

[RQ1] 特定 OS 向け欠陥修正はどの程度存在するのか

特定 OS 向け欠陥修正が，全体の欠陥修正に対して，どの程度存在しているのかを明らかにすることができれば，単一の OS 向けのソフトウェアと比較して追加で発生するコストがどの程度であるかを明らかにすることができる．RQ1 では，特定 OS 向け欠陥修正が全体の欠陥修正に対してどの程度存在するのか，またそれぞれの OS 向けの欠陥修正はどの程度存在するのかについて調査する．

[RQ2] ソフトウェアの進化によって特定 OS 向け欠陥修正の数はどのように推移するのか

ソフトウェアの進化に対してかかるコストは想定されるソフトウェアシステム全体のコストの小さい場合では 50%，大きい場合では 90%以上になるといわれている [7]．継続的な進化の中で特定 OS 向け欠陥修正がどのように発生しているかを明らかにすることは，将来のソフトウェア進化の過程で特定 OS 向け欠陥修正がどの程度発生するかを予測する上で有用であると考えられる．RQ2 では，ソフトウェアの進化によって特定 OS 向け欠陥修正の数がどのように推移していくのかについて調査する．

表 1 対象データセット

プロジェクト	期間	コミット数	ファイル数
PostgreSQL	1996/07/09-2015/01/04	485,912	4,688
SQLite	2000/3/29-2015/4/22	14,681	1,283
Qt5	2011/04/27-2014/11/12	703,281	147,399
Gtk	1997/1/2-2015/03/24	54,204	4,147

表 2 ビルド時の実行環境

OS	シリーズ	バージョン
Windows	Windows7	Home Premium
Linux	CentOS	6.6
Mac	OS X	9.5

[RQ3] 特定 OS 向け欠陥修正の修正内容はどのように分類されるか

開発者の人数や開発者の知識はソフトウェアのコストを見積もるための要因の 1 つとして含まれている [13]．欠陥にはいくつかの種類があり，欠陥の種類によって求められる知識は異なる．RQ3 では，特定 OS 向け欠陥修正の修正内容をソースコード，GUI，テスト，ビルド，軽微な修正の 5 つのグループに分類する．

McIntosh らの研究では [10]，ビルドの専門家がいるプロジェクトとそうでないプロジェクトを比較した場合，ビルドの専門家がいるプロジェクトの方がそうでないプロジェクトに比べ，ビルドに従事する開発者の割合が 20%以上小さいといった知見を報告している．つまり，ビルドの専門家がいれば，新たにビルドについて学習を行う開発者の人数を小さくできる．それぞれの修正に対する開発者の人数の割合はマルチプラットフォーム向けソフトウェアの進化に対してかかるコストに影響を及ぼす．マルチプラットフォーム向けソフトウェアの進化に対してかかるコストを明らかにする上で，特定 OS 向け欠陥修正にはどのような修正が多いのかについて明らかにすることは有用であると考えられる．

4. ケーススタディ

4.1 データセット

マルチプラットフォーム向けに開発された，Git の閲覧履歴のある 4 つのオープンソースソフトウェア（PostgreSQL，SQLite，Qt5，Gtk）を分析対象のプロジェクトとして用いた．これらのプロジェクトは 2 つの機能的に類似したグループに属している．SQLite と PostgreSQL はデータベース管理のプロジェクトである．Qt5，Gtk は GUI のツールキットを持つプロジェクトである．これらのプロジェクト

表 3 取得した特定 OS 向け欠陥修正コミット

プロジェクト名	特定 OS 向け欠陥修正コミットの全体に占める割合	全欠陥修正コミット	特定 OS 向け欠陥修正コミット	キーワードに基づき特定したコミット	ビルドに基づき特定したコミット
PostgreSQL	2.7%	13,925	370	370	2
SQLite	1.9%	4,045	75	75	0
Qt5	38.7%	14,258	5,520	2,309	4,010
Gtk	8.5%	10,236	866	435	567

を用いることにより，それぞれの RQ において得られた結果がそのプロジェクト独自の結果なのか，機能的に類似するプロジェクトでは似たような結果が得られるのかを確認する．各プロジェクトの概要を表 1 に示す．

PostgreSQL. C 言語で実装された関係データベース管理システムである [*3]．サーバで実行される処理のまとまりを関数として定義できるといった特徴がある．

SQLite. C 言語で実装された関係データベース管理システムである [*4]．軽量なデータベースであり，アプリケーションに組み込むなどして利用される．

Qt5. C++言語で実装されたアプリケーションユーザーインターフェースフレームワークである [*5]．

Gtk. C 言語で実装されたクロスプラットフォームの GUI ツールキットである [*6]．

4.2 実行環境

特定 OS 向け欠陥修正コミットの特定方法の 1 つにビルドを用いる．ビルドは各 OS の差のみではなく，OS のシリーズやバージョンによっても参照されるファイルに違いが生じる可能性がある．今回の研究における Windows, Linux，Mac でのビルド時の実行環境を表 2 に示す．

4.3 各 RQ の結果

[RQ1] 特定 OS 向け欠陥修正はどの程度存在するのか

アプローチ. 特定 OS 向け欠陥修正がどの程度存在するか調査するために，Git のコミットログから，欠陥を修正するために行われたコミットを収集した（以降，欠陥修正コミットとする）．今回は，コミットログのメッセージの中に bug，fix（大文字，小文字を問わない）のキーワードを含むものを欠陥修正コミットとした [2].

欠陥修正コミットに対して，キーワードに基づく特定方法とビルドに基づく特定方法によって特定 OS 向け欠陥修正の数について調査を行った．

キーワードに基づく特定方法. この特定方法では取得した欠陥修正コミットから，さらにコミットメッセージ中に OS のキーワードを含んでいるものを特定 OS 向けの欠陥修正コミットとした．OS のキーワードとして Windows, Linux，Mac の 3 つのキーワードを用いた．ただし，Mac に関しては OS キーワードに Mac を使用した場合，コミットメッセージの中に Macaddres，Macro，Machine，人名に “Mac” を含むコミットが多く取得された．これらのキーワードを含むコミットを除外するために，Mac では正規表現として “Mac[\t]” と “MacOS” を用いた．

表 4 特定 OS 向け欠修正コミットの分類

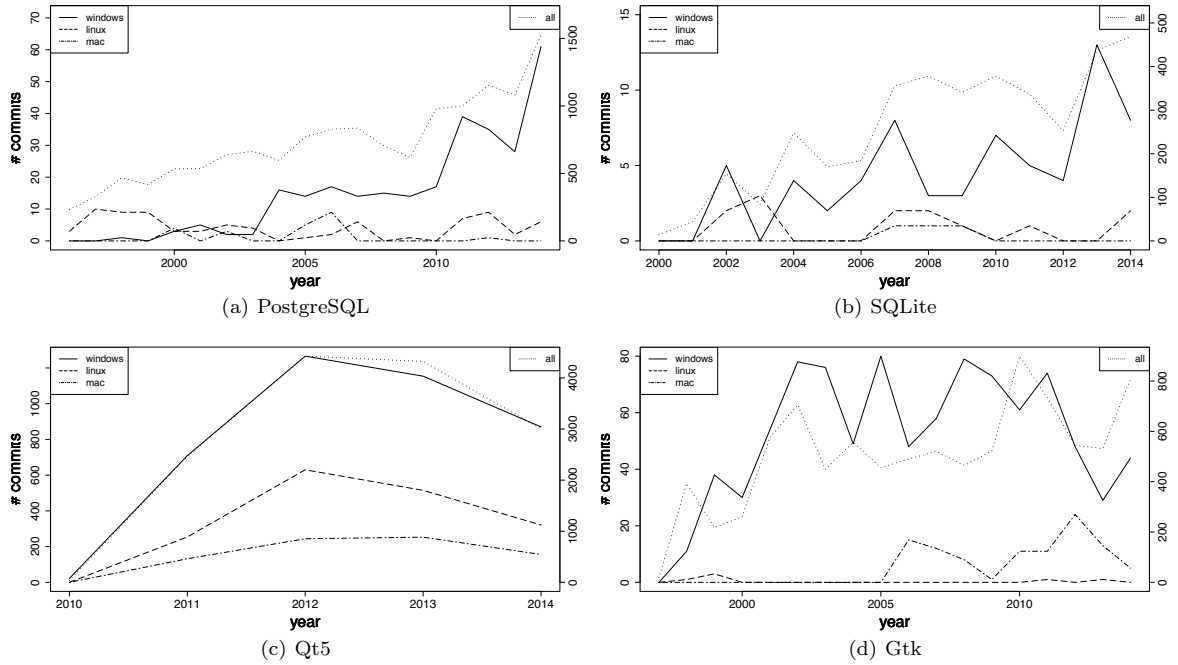
プロジェクト名	Windows	Linux	Mac
PostgreSQL	284	80	11
SQLite	62	11	3
Qt5	4,056	1,613	564
Gtk	766	8	100

ビルドに基づく特定方法. 初めに，それぞれの OS の環境下でそれぞれのプロジェクトのビルドを行う．それぞれの OS ごとのビルド実行時に参照されるファイルを MAKAO[1] というツールを用いて取得する．それぞれの OS で参照されるファイルの比較を行い特定の OS でのみ参照されるファイルを抽出する．例えばそれぞれの OS でのビルド時の参照ファイルとして Windows では *f1*, *f4*, *f5*, *f7*, Linux では *f2*, *f4*, *f6*, *f7*, Mac では *f3*, *f5*, *f6*, *f7* を参照されたとする．その場合，特定 OS のみで参照されたファイルとして抽出されるファイルは Windows のみで参照された *f1*, Linux のみで参照された *f2*, Mac のみで参照された *f3* となる．この方法により抽出されたファイルの欠陥修正コミットを，この特定方法では特定 OS 向け欠陥修正コミットとした．

2 つの特定方法を用いる理由. ビルドに基づく特定方法では，特定の OS でのみ参照されるファイルを基準に特定 OS 向け欠陥修正コミットの特定を行っている．しかし，共通で参照されるファイルであっても特定の OS でのみ読み込まれるコード行が存在する可能性がある．この場合，特定の OS でのみ読み込まれるコード行を修正している欠陥修正コミットも本来は特定 OS 向け欠陥修正コミットである．

このような特定 OS 向け欠陥修正コミットを抽出するためにキーワードに基づく特定方法を用いる．キーワードに基づく特定方法の場合，修正したファイルに関わらずコミットを行った人が特定 OS の欠陥を修正したということをコミットメッセージに記載していれば特定 OS 向け欠陥修正コミットとして取得することができる．つまり，ビルドに基づく特定方法では OS の差異をファイル単位で管理しているものを抽出し，キーワードに基づく特定方法ではビルドに基づく特定方法だけでは抽出できなかった，OS

[*3] http://www.postgresql.org/
[*4] http://www.sqlite.org/
[*5] http://qt-project.org/qt5
[*6] http://www.gtk.org/



(a) PostgreSQL (b) SQLite

(c) Qt5 (d) Gtk

図 1 各プロジェクトの特定 OS 向け欠陥修正コミット数の推移

の差異を行単位で管理しているものを抽出する．以上の理由から，この 2 つの特定方法によって特定 OS 向け欠陥修正コミットの特定を行った．

結果と考察．取得した特定 OS 向け欠陥修正コミットと全ての欠陥修正コミットに占める割合について表 3 に示す．GUI 系のプロジェクトでは，ビルドに基づく特定方法により取得した特定 OS 向け欠陥修正コミットの数が大きく，そうでないプロジェクトにおいてはビルドに基づく特定方法により取得した特定 OS 向け欠陥修正コミットの数は小さかった．これはそれぞれの OS での GUI の差異に関する特有の修正はファイル単位で管理しているのではないかと考えられる．また，特定 OS 向け欠陥修正コミット数の大きいプロジェクトでは，全ての欠陥修正に占める割合は 38.7%と，全ての欠陥修正コミットのうち約 4 割が OS に依存した欠陥修正コミットであるという結果が得られた．

また，特定 OS 向け欠陥修正の分類の結果を表 4 に示す．どのプロジェクトにおいても Linux と Mac に比べて，Windows の欠陥修正コミット数が大きい傾向にある．これは，1）今の Windows の構造が Linux と Mac の構造とは異なっているため，2）対象とした OSS の主要な開発 OS が Posix 互換 OS であったためではないかと考えられる．実際のソースコードの修正のいくつかを目視で確認したところ，Windows の 32bit 版のための修正や，Cygwin[*7] のための修正といった，Windows のための修正が行われていた．

[*7] Cygwin は Windows 上で動作する UNIX ライクな環境の 1 つである．

表 5 リリース日前後 2 ヶ月とその他の 2 ヶ月の Windows 向け欠陥修正コミット数

プロジェクト名	リリース日前	リリース後	その他の期間
PostgreSQL	6.6	1.2	2.8
Gtk	9.0	6.3	6.8

特定 OS 向け欠陥修正が全体の欠陥修正に占める割合は 1.9%から 38.7%であった．Windows 向けの欠陥修正コミットが他の OS の欠陥修正コミットに比べて大きかった．

[RQ2] ソフトウェアの進化によって特定 OS 向け欠陥の数はどのように推移するのか

アプローチ．RQ1 で取得した特定 OS 向け欠陥修正コミットから，コミットが行われた日付をコミットログから抽出し，各 OS 向けの欠陥修正コミット数の推移を調査した．2015 年は，5 月現在で 12 月分のデータを得ることができないので分析対象外とした．

結果と考察．欠陥修正コミット数の推移について，各プロジェクトごとの欠陥修正コミット数の推移を図 1 に示す．横軸がプロジェクトがリリースされた年，左の縦軸がそれぞれの OS 向けの欠陥修正コミット数，右の縦軸が全ての欠陥修正のためのコミット数である．グラフからどのプロジェクトにおいても，プロジェクトがリリースされてから，ソフトウェアが進化していく中で，特定 OS 向け欠陥修正コミットが一定量存在していることがわかる．このことか

表 6 修正内容分類のためのグループ

グループ名	内容
ソースコード	ソースコードの修正（GUI の修正を含まない）
GUI	ソースコードの修正の中でもボタンやメニューの不具合に関する修正といった，ユーザサイドから確認することができる欠陥の修正
テスト	テストコードの修正やテストケースの追加といったテストに関する修正
ビルド	configure の修正，コンパイルエラーの修正，ビルドの修正といったようなビルドに関連する修正
軽微な修正	タイポの修正やコメントの修正といったようなドキュメントの修正
その他	Revert コミットなどどれにも属さないコミット

表 7 それぞれの OS 向けの修正内容

OS 名	プロジェクト名	ソースコード	GUI	テスト	ビルド	軽微な修正	その他	合計
Windows	PostgreSQL	79	-	1	17	2	1	100
	SQLite	19	-	18	21	2	2	62
	Qt5	56	17	4	16	1	6	100
	Gtk	46	31	1	8	4	10	100
Linux	PostgreSQL	55	-	3	19	3	0	80
	SQLite	5	-	1	4	1	0	11
	Qt5	55	18	4	15	1	7	100
	Gtk	3	0	1	3	0	0	7
Mac	PostgreSQL	3	-	0	4	2	2	11
	SQLite	0	-	0	0	1	2	3
	Qt5	22	60	3	10	0	5	100
	Gtk	43	33	1	16	5	2	100

らマルチプラットフォーム向けソフトウェアの進化において特定 OS のための修正に対するコストは継続的にかかりつづけると考えられる．また，各 OS ごとにコミット数が大きくなる期間には差異があった．これは，OS 側に進化が生じた際に発生したのではないかと考えられる．

そこで，OS の進化に伴って特定 OS 向け欠陥修正コミットの数がどのように変化するかについて調査を行った．調査の対象としては，他の OS に比べて欠陥修正コミット数の大きい Windows について，プロジェクトの期間が長い PostgreSQL と Gtk を対象とした．調査の方法としては，OS の各シリーズのリリース日を調査し，リリース日前の 2 ヶ月とリリース日後の 2 ヶ月の平均コミット数，その他の期間の 2 ヶ月ごとの平均コミット数を調査した．それぞれの期間での Windows 向けの欠陥修正コミット数の平均を表 5 に示す．リリース前の期間ではその他の期間に比べ欠陥修正コミットが多く行われている．一方，リリース後の期間では他の期間に比べて欠陥修正コミットが少ない．これは，OS のリリースに伴って欠陥が発生する可能性のある箇所を事前に修正しているのではないかと考えられる．

> プロジェクトの進化の過程で，特定 OS 向けの欠陥修正は継続的に発生する．OS のリリースに伴う欠陥の修正が行われており，その修正が，OS がリリースする前の 2 ヶ月の間で多く行われているプロジェクトがあった．

[RQ3] 特定 OS 向け欠陥修正の修正内容はどのように分類されるか

アプローチ. 取得した特定 OS 向け欠陥修正コミットのコミットメッセージの内容を実際に目視によって確認していき，それぞれのコミットがどのような欠陥の修正であったかをいくつかのグループに分類した．目視によって確認するコミットとしては，それぞれの OS において，その OS 向けの欠陥修正コミットが 100 件未満の場合はすべてのコミットを確認し，100 件以上の場合はランダムに 100 件のコミットを選びそのコミットを確認した．分類を行うためのグループについての概要を表 6 に示す．ソースコードの修正と GUI に関する修正のどちらに分類するのかの判断が難しい修正はすべてソースコードの修正に分類した．

結果と考察. それぞれの OS での修正内容の分類結果について表 7 に示す．それぞれのプロジェクトにおいてグループごとのコミットの数に差異はあるが，特定 OS 向け欠陥修正コミットの数が 100 件を超えているものは，どの OS においても，それぞれのグループに分類されるコミットが存在した．つまり，今回分類したそれぞれのグループの知識が開発者に求められるということである．

実際の修正内容の概要を表 8 に示す．それぞれのグループにおいて，GUI では，メニューの修正（表 8 の GUI の 1），テキスト描画の修正（表 8 の GUI の 3），ページサイズの修正（表 8 の GUI の 4）などが確認された．テストでは，自動テストの修正表（表 8 のテストの 1），テストスク

表 8　確認した実際の修正のメッセージ

	ソースコード	GUI	テスト	ビルド
1	Changes multihead reorganizing code for win32 support	fix menu coloration	Fix QSGBorderImage and QSGAnimatedImage autotests	disable USE_MMX for msvc build cause he assembler doesn't fit and is out
2	Fix handling of restricted processes for Windows Vista	Fixed position of menu gutter when using a custom widget action	Fix up test use QFINDTESTDATA for shadow build	Fix compilation with MinGW
3	Change the way pg_basebackup's tablespace mapping is implemented	Fixed text drawing in OpenGL 2 paint engine	Fix pg_upgrade test script's line end handling on Windows	fix angle build under msys
4	Fix annoying beeping introduced by Mac IME	Fix winPageSize() on Mac	Fix mmap1.test so that it passes on windows as well as unix	Fix contrib/sepgsql and contrib/xml2 to always link required libraries
5	Fix the SQLITEMIXEDENDIAN64BITFLOAT option so that it works on goofy linux kernels that employ CONFIG_FPE_FASTFPE.	Cocoa: Fix menu popup	Fix test case numbering in backup_malloc.tes	Fix char2wchar/wchar2char to support collations properly. This change removes the bogus assumption that all locales selectable in a given database have the same wide-character conversion method

リプトの修正（表 8 のテストの 3），テストケースの修正（表 8 のテストの 5）などが確認された．ビルドでは，ビルドの修正（表 8 のビルドの 2），コンパイルの修正（表 8 のビルドの 2），ライブラリのリンクの修正（表 8 のビルドの 4）などが確認された．

それぞれのグループに分類される特定 OS 向け欠陥修正コミットが存在しており，グループごとに開発者の知識が必要となる．

5. 議論

この章では，それぞれの RQ で得られた結果がマルチプラットフォーム向けソフトウェアのコストの見積もりにどのように影響を与えるかについて考察を述べる．

[RQ1] 特定 OS 向け欠陥修正はどの程度存在するのか

RQ1 の結果から，マルチプラットフォーム向けソフトウェアの開発が単一の OS 向けの開発に比べて，余計にマルチプラットフォーム向けの工数がかかることがわかる．プロジェクトによって 1.9%から 38.7%と幅があるものの，マルチプラットフォーム向けの開発プロジェクトにはそのプロジェクトに応じた追加の開発工数を見積もることが望ましい．

[RQ2] ソフトウェアの進化によって特定 OS 向け欠陥の数はどのように推移するのか

RQ2 の結果から，RQ1 で得られたマルチプラットフォーム向けの工数が，プロジェクトの進化のとの時期でかかるのかがわかる．プロジェクトの進化の過程で，特定 OS 向けの欠陥修正は継続的に発生することから，ソフトウェアの進化の期間に応じてマルチプラットフォーム向けソフトウェアに対してコストが増加し続けると考えられる．

また，OS のリリースに伴う欠陥の修正が行われており，その修正が，OS がリリースする前の 2 ヶ月の間で多く行われていることから，マルチプラットフォーム向けソフトウェアはソフトウェア自体の機能の改善や拡張とは別に，OS のリリースに対応するための追加のコストが発生する場合があると考えられる．

[RQ3] 特定 OS 向け欠陥修正の修正内容はどのように分類されるか

RQ3 の結果から，RQ1 で得られたマルチプラットフォーム向けの工数のうち，人員をどういった比率で用意すればいいかがわかる．表 6 に示したグループに分類される特定 OS 向け欠陥修正コミットが存在していたことから，開発者の人数を考える上で，それぞれのグループに属する欠陥修正コミットの数に応じて，それぞれのグループに関する知識を持った開発者を割り当てる必要があると考えられる．

6. 本研究の制約

構造的妥当性．特定 OS 向け欠陥修正コミットを取得するにあたってビルドに基づく特定方法とキーワードに基づく特定方法を用いた．ビルドに基づく特定方法ではファイル単位での OS の差異を取得することを目的とし，キーワー

ドに基づく特定方法では行単位の OS の差異を取得することを目的とした．しかし，キーワードに基づく特定方法ではコミッターが OS キーワードをコミットメッセージに含めたかどうかに依存するため，全ての特定 OS 向け欠陥修正コミットを取得できていない可能性がある．

また，本研究では，全ての欠陥修正コミットに対する特定 OS 向け欠陥修正コミットの割合によって，単一の OS 向けのソフトウェアと比較してマルチプラットフォーム向けソフトウェアが追加で発生するコストを定義している．それぞれの欠陥修正にかかった実際のコストは考慮していないので，欠陥修正コミット数とコストの関係は必ずしも明確ではない．

内的妥当性．特定 OS 向け欠陥修正コミットを取得する方法の 1 つとしてビルドに基づく特定方法を用いた．ビルドを行う場合，ビルドを行う前にどのような設定でビルドを行うかについて，configure ファイルなどを用いて設定を行う．この場合，configure でのオプションの差異などによってもビルド時に参照するファイルに差が生じる場合がある．本研究では，configure のオプションとしてプラットフォームを選択するオプション以外は全て同じオプションを用いた．

外的妥当性．今回に用いたプロジェクトは，まず機能的に似たプロジェクトを 2 つずつ用いており，それぞれの機能ごとにプロジェクトのサイズも異なっている．しかしながら，対象としたプロジェクトはオープンソースソフトウェアの中の一部のプロジェクトであり，今回の結果は全てのプロジェクトに対する一般性はないかもしれない．

7. まとめ

本稿では，マルチプラットフォーム向けソフトウェアの開発，保守においてかかるコストを定量的に明らかにすることを目的として，3 つのリサーチクエスチョンを設定し，特定 OS 向けの欠陥修正コミットの分析を行った．4 つのオープンソースソフトウェアに対して 3 つのリサーチクエスチョンを実施した結果，特定 OS 向け欠陥修正に関して，マルチプラットフォーム向けソフトウェアのコストを見積もるための下記の知見を得た．

- 特定 OS 向け欠陥修正が全体の欠陥修正に占める割合は 1.9%から 38.7%であった．
- GUI 系のプロジェクトではファイル単位での特定 OS に対する修正が大きかった．
- Windows 向けの欠陥修正コミットは，他の OS に比べ多かった．
- プロジェクトの進化の過程で，特定 OS 向けの欠陥修正は継続的に発生する．
- OS のリリースに伴う欠陥の修正が行われており，その修正が，OS がリリースする前の 2 ヶ月の間で多く行われているプロジェクトが存在した．
- RQ3 の表 6 に示した各修正内容のグループに属するコミットが存在し，グループごとに開発者の知識が必要となる．

謝辞．本研究の一部は，JSPS 科研費（若手 A：課題番号 15H05306）による助成を受けた．

参考文献

[1] Bram Adams, Herman Tromp, Kris De Schutter, and Wolfgang De Meuter. Makao. In *Proc. of the International Conference on Software Maintenance (ICSM)*, pp. 517–518, 2007.

[2] Adrian Bachmann, Christian Bird, Foyzur Rahman, Premkumar Devanbu, and Abraham Bernstein. The missing links: bugs and bug-fix commits. In *Proc. of the International Symposium on Foundations of Software Engineering (FSE)*, pp. 97–106, 2010.

[3] Michael A Cusumano, David B Yoffie, 青山幹雄. 第 12 回 インターネット時代のソフトウェア開発戦略: Netscape のクロスプラットフォーム開発に学ぶ (ソフトウェア新時代). 情報処理, Vol. 40, No. 4, pp. 418–423, 1999.

[4] Yujuan Jiang and Bram Adams. Co-evolution of infrastructure and source code – an empirical study. In *Proc. of the Working Conference on Mining Software Repositories (MSR)*, 2015.

[5] Miryung Kim, Dongxiang Cai, and Sunghun Kim. An empirical investigation into the role of api-level refactorings during software evolution. In *Proc. of the International Conference on Software Engineering(ICSE)*, pp. 151–160, 2011.

[6] Sunghun Kim and E James Whitehead Jr. How long did it take to fix bugs? In *Proc. of the Working Conference on Mining Software Repositories (MSR)*, pp. 173–174, 2006.

[7] Jussi Koskinen. Software maintenance costs. *Information Technology Research Institute, ELTIS-Project University of Jyväskylä*, 2003.

[8] Meir M Lehman. Programs, life cycles, and laws of software evolution. *In Proc. of the IEEE*, Vol. 68, No. 9, pp. 1060–1076, 1980.

[9] Isa Maleki, Laya Ebrahimi, and Farhad Soleimanian Gharehchopogh. A hybrid approach of firefly and genetic algorithms in software cost estimation. 2014.

[10] Shane McIntosh, Bram Adams, Thanh HD Nguyen, Yasutaka Kamei, Ahmed E. Hassan. An empirical study of build maintenance effort. In *Proc. of the International Conference on Software Engineering(ICSE)*, pp. 141–150. ACM, 2011.

[11] Yuan Tian, Julia Lawall, and David Lo. Identifying linux bug fixing patches. In *International Conference on Software Engineering(ICSE)*, pp. 386–396, 2012.

[12] Rajesh Vasa, J-G Schneider, and Oscar Nierstrasz. The inevitable stability of software change. In *Proc. of the International Conference on Software Maintenance(ICSM)*, pp. 4–13, 2007.

[13] Corinne C Wallshein and Andrew G Loerch. Software cost estimating for cmmi level 5 developers. *Journal of Systems and Software*, Vol. 105, pp. 72–78, 2015.

[14] 村尾憲治, 肥後芳樹, 井上克郎. ソフトウェアメトリックス値の変遷に基づいた注力すべきモジュールを特定する手法の提案. 電子情報通信学会論文誌 D, Vol. 91, No. 12, pp. 2915–2925, 2008.