Vol.2015-SE-187 No.11
2015/3/12

オープンソースソフトウェアの進化における特定OS向け欠陥修正コミットの分析

松本　卓大[†1,a)]　大坂　陽[†1,b)]　亀井　靖高[†1,c)]　鵜林　尚靖[†1,d)]

概要：本稿では，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，その修正が特定の OS 向けにどの程度存在するのかを明らかにし，特定の OS で 発生する欠陥数やその推移の傾向について明らかにする．一般に，マルチプラットフォーム向けに開発されたソフトウェアは，各 OS 自体の進化に伴うコードの変更や追加が必要となる場合があり，この変更や追加により欠陥が発生する可能性がある．特定 OS 向け依存欠陥の特徴を検出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の効率が向上する可能性がある．そこで，本稿では 2 つのリサーチクエスチョンを実験的に検証した．Squid，Graphvis，PostgreSQL，Qt5 の 4 つのオープンソースソフトウェアを用いた結果，1) キーワードに基づく特定方法により取得した特定 OS 向けの欠陥修正コミットが全体の欠陥修正コミットに占める割合は 5.4%であった，2) Windows 向けの欠陥修正コミットは，他の OS に比べ多かった，3) OS のリリースに伴う欠陥の修正が行われており，リリース前の 2 ヶ月間の欠陥修正コミット数はその他の 2 ヶ月間の期間に比べ 2.4 倍大きい，といった知見が得られた．

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

1. はじめに

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の小さい場合では 50%，大きい場合では 90%以上かかるといわれている [7]．そのため，ソフトウェア進化に対してかかるコストを抑えることはソフトウェア開発において有用である [2][4].

ソフトウェアの進化においてコードの変更や追加を効率的に行うために，欠陥の特徴や発生箇所を明らかにする取り組みが行われている [8][10]．これらの研究では，特にマルチプラットフォームか否かについて言及されていない．しかしながら，マルチプラットフォーム向けに開発されたソフトウェアは，各 OS 自体の進化に伴うコードの変更や追加が必要となる場合があり，この変更や追加により欠陥が発生する可能性がある．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，動作が極端に遅くなる不具合や読み込み拒否の不具合が報告されている．

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（OS 依存欠陥）は，他の機能的に類似したアプリケーションでも発生する可能性がある．OS 依存欠陥の特徴を検出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の効率が向上する可能性がある．

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がどの程度であるのかを明らかにすることを目的とし，特定 OS 向けの欠陥修正コミットを分析する．具体的には，Git[*1] で管理されたオープンソースソフトウェアで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し，1)OS 依存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 OS 依存欠陥修正の数はどのように推移するのか，という 2 つのリサーチクエスチョンに取り組む．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 (Squid，Graphviz，PostgreSQL，Qt5) を用

[†1] 現在，九州大学
Presently with Kyushu Uniersity
[a)] matsumoto@posl.ait.kyushu-u.ac.jp
[b)] osaka@posl.ait.kyushu-u.ac.jp
[c)] kamei@ait.kyushu-u.ac.jp
[d)] ubayashi@ait.kyushu-u.ac.jp

[*1] ソースコードなどのファイルの変更履歴を記録，及び，追跡するための分散型バージョン管理システム

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 2 つの Research Question 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．最後に 5 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][13]．Zaidman ら [13] は，テストと関連するプロダクトコード（ソースコード）がどのように進化するのかに注目し，2 つのオープンソースソフトウェアプロジェクトに対してテストコードと関連するプロダクトコードが同時に開発されているのかを調査した．修正されているテストコードとプロダクトコードを時系列順にソートし進化の過程を可視化することで，テストコードと関連するソースコードは同時に修正されていることを示した．

McIntosh ら [9] は，ソフトウェア開発におけるビルドファイルに対するオーバーヘッドがどの程度であるのかを定量的に評価することを目的として，プロダクトファイル，テストファイル，ビルドファイルの変更回数を分析した．ワークアイテム（同一目的のために行われたコミット群）単位で分析することで，ビルドファイルがソースコードと共に修正される傾向があることを示している．

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ソフトウェア進化の過程から，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．

本研究でも従来研究と同様，ソフトウェア進化に関する理解を目的とする点で動機は同じである一方で，マルチプラットフォーム向けソフトウェアの観点から分析を行う点が新しい．

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を分析している．修正のためにかかる時間が比較的長かった場合，その欠陥を含んでいるファイルには変更することが構造的に困難な欠陥がある可能性があるためである．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，欠陥修正にかかる時間の平均や，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 のファイルのリストアップなどを報告している．

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．

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
Squid	1996/02/22-2012/09/06	8,612	1,844
Graphviz	2004/12/23-2015/01/19	9,194	3,804
PostgreSQL	1996/07/09-2015/01/04	485,912	4,688
Qt5	2011/04/27-2014/11/12	703,281	147,399

村尾らによる研究 [14] では，ソフトウェアメトリクス値がどの程度安定しているかを計測することで，今後の開発，及び，保守過程において力を注ぐべきモジュールを特定する手法の提案している．複数のオープンソースソフトウェアに対して提案手法を適用した結果，将来的に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がわかった．

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，コミットメッセージに基づくアプローチを提案している．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い，Tian らの手法の方が精度が良いといった実験結果を報告している．

本研究では，従来の欠陥修正と比較して，OS 依存欠陥修正に着目してその欠陥の有無や欠陥数の推移を分析する点が新しい．

2.3 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) に関する研究もわずかに存在する [3]．Cusumano らによる研究 [3] では，マルチプラットフォームの難しさについて議論を行い，Netscape のマルチプラットフォーム開発から，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．分析の対象として，開発工数や開発期間などを対象としている．本研究では，特定 OS 向けのための欠陥修正の数といった，変更回数などの観点からマルチプラットフォーム向けソフトウェアの分析を行っている．

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がどの程度であるのかを明らかにすることを目的とし，特定 OS 向けの欠陥修正コミットを分析する．本研究を進めるために，2 つのリサーチクエスチョ

表 2 OS 依存欠陥修正コミットが全体に占める割合

プロジェクト名	全欠陥修正コミット	OS 依存欠陥修正コミット	OS 依存欠陥修正コミットの全体に占める割合
Squid	1,265	57	4.5%
Graphviz	2,309	107	4.6%
PostgreSQL	13,925	377	2.7%
Qt5	14,258	1,395	9.8%

表 3 OS 依存欠陥修正コミットの分類

プロジェクト名	Windows	Linux	Mac	Windows ∩ Linux	Linux ∩ Mac	Windows ∩ Mac	Windows ∩ Linux ∩ Mac
Squid	44	13	0	0	0	0	0
Graphviz	85	10	12	0	0	0	0
PostgreSQL	283	69	22	3	10	8	0
Qt5	1,006	136	189	19	10	31	4

ン (RQ) を設定した．

[RQ1] OS 依存欠陥修正はどの程度存在するのか

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，OS 自体の進化ごとにコードの変更や追加を行っている場合，その OS に依存した欠陥もある程度存在すると考えられる．そこで RQ1 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がどの程度であるかの理解のために，OS 依存欠陥修正が，全体の欠陥修正に対してどの程度存在しているのかについて調査する．

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

OS 依存欠陥が発生する要因として，ソフトウェアの進化や，各 OS 自体の進化に伴う，コードの変更や追加が考えられる．そのような要因で OS 依存欠陥修正が発生した場合，要因となる進化が発生した特定の期間の前後において，OS 依存欠陥の修正数が多く発生する可能性があると考えられる．そこで RQ2 では，ソフトウェアの進化によって OS 依存欠陥修正の数はどのように推移するかについて調べ，OS 依存欠陥修正の推移に一定の傾向がないかについて調査を行った．

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 (Squid, Graphviz，PostgreSQL，Qt5) をデータセットとして用いた．各プロジェクトの概要を表 1 に示す．

Squid. プロキシサーバ，ウェブキャッシュサーバなどに利用されるオープンソースソフトウェアである [*2]．用途としては，重複リクエストに対したキャッシュ応答によるウェブサーバの高速化などである．

Graphviz. テキスト形式で記述したグラフ表現をさまざまな形式の画像に変換するオープンソースソフトウェアである [*3]．サイズの大きなツリー構造やノード数の大きいネットワーク構造のデータを自動的に可視化する際に用いられる．

PostgreSQL. オープンソースとして配布されている，オブジェクト関係データベース管理システムである [*4]．

Qt5. C++言語で書か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である [*5]．

4.2 各 RQ の結果

[RQ1] OS 依存欠陥修正はどの程度存在するのか

アプローチ. OS 依存欠陥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 [1].

取得した欠陥修正コミットからさらに OS のキーワードを含んでいるか否かで，特定 OS 向けの欠陥修正コミットを判断する．キーワードとして Windows，Linux，Mac の 3 つのキーワードをコミットメッセージに含んでいるかで特定を行った．ただし，Mac というキーワードに関してはコミットメッセージの中に Macaddres，Macro，Machine，人名に “Mac” を含むものなども Mac のコミットとして判定されていたので，これらのキーワードを含むコミットは除外した．コミットメッセージの中に Windows というキーワードを含んでいた場合は Windows に分類，コミットメッセージの中に Windows と Linux の二つのキーワードを含んでいた場合は Windows ∩ Linux に分類するというような方法を用いて，それぞれのグループのコミット数を調査した．

結果と考察. 全ての欠陥修正コミット数と OS 依存欠陥修正コミット数について表 2 に示す．OS 依存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

[*2] http://www.squid-cache.org/
[*3] http://www.graphviz.org/
[*4] http://www.postgresql.org/
[*5] http://qt-project.org/qt5

表 4 本手法によって検出された OS 依存欠陥修正コミットのメッセージ例

メッセージ文	解釈
We don't normally expect any such failures, but on Windows it can happen with some anti-virus or backup software that lock files without FILE_SHARE_DELETE. 中略 there's no such locking issues on other platformsflag.	Windows で発生する予期せぬ障害を修正している．他のプラットフォーム上では，このような問題は発生しない．
Fix line end mishandling in pg_upgrade on Windows. (中略) On non-Windows platforms, this change has no	Windows での pg_upgrade(PostgresSQL をアップグレードするもの) を修正している．他のプラットフォームではこのような問題は発生しない．

める割合は 9.8%と，約 1 割が OS に依存した欠陥修正コミットであった．4 つのプロジェクトにおける，OS 依存欠陥修正コミットが，全体の欠陥修正コミットに占める割合の平均は 5.4%であった．本結果より，マルチプラットフォーム向けソフトウェアの進化には 5.4%程度の追加の開発コストがかかることが示唆された．

OS 依存欠陥分類の詳細を表 3 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)Windows ユーザの総数が大きく，ユーザからの障害報告が多くなされているためではないかと考えられる．

キーワードに基づく特定方法によって取得した欠陥修正コミットが OS 依存欠陥修正コミットであるか確かめるために，取得したいくつかの欠陥修正コミットのコミットメッセージを目視によって確認した．コミットメッセージの結果を表 4 に示す．メッセージ内容から，キーワードに基づく特定方法によって取得された欠陥修正コミットが OS 依存欠陥修正コミットであることが明らかになった．

> OS 依存欠陥修正のためのコミットが全ての欠陥修正コミットに占める割合は 5.4%であり，Windows 向けの欠陥修正コミットが他の OS 向けの欠陥修正コミットに比べて大きいことがわかった．

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

アプローチ．欠陥修正コミットの日付を抽出し，各 OS 向けの欠陥修正コミット数の推移を調査した．欠陥修正コミットは，キーワードに基づく特定方法で抽出したコミットを対象とした．ただし，今回は 2 つ以上の OS キーワードを含むコミットについては，それぞれに 1 コミットとした．例えば，OS キーワードに Windows，Linux を含む場合，Windows 向けのコミット数を 1 つ追加し，Linux 向けのコミット数も 1 つ追加する．

結果と考察．欠陥修正コミットの推移について Squid の結果を図 1(a) に，Graphviz の結果を図 1(b) に，PostgreSQL の結果を図 1(c) に，Qt5 の結果を図 1(d) に示す．図の横軸が時系列，左の縦軸が OS 依存欠陥修正のコミット数，右の縦軸が全ての欠陥修正コミットの数である．

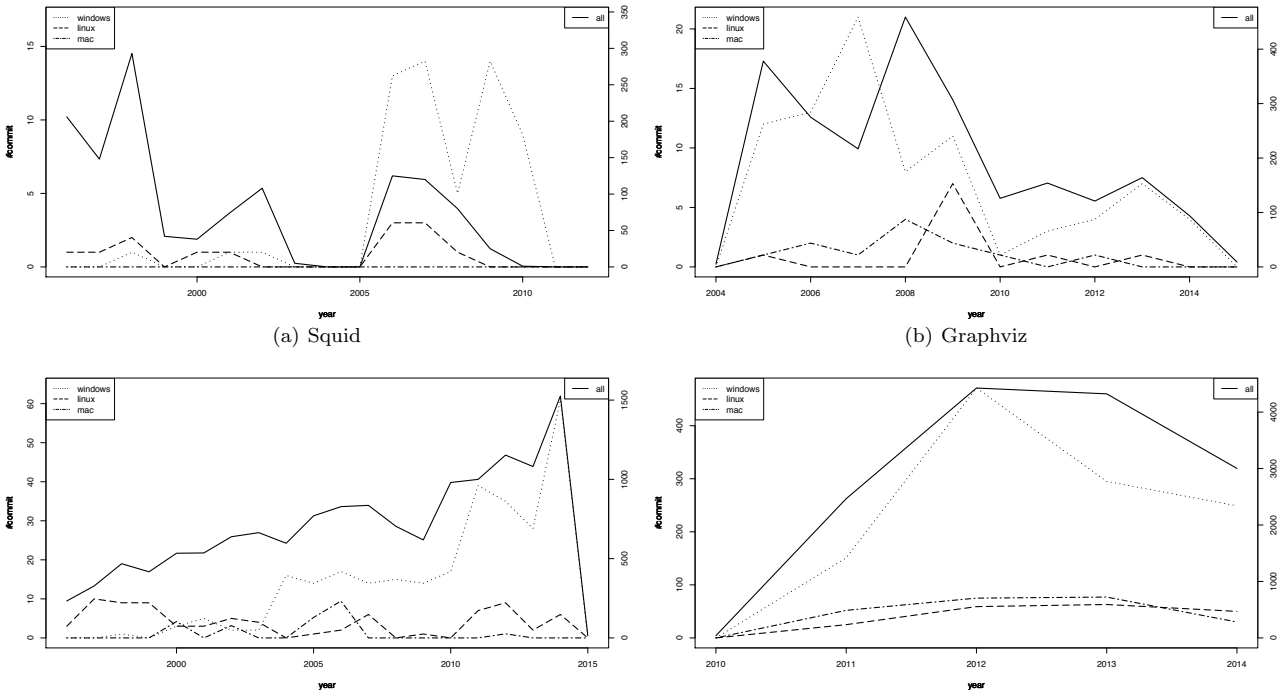
どのプロジェクトでも，欠陥修正コミット数が大きい特定の時期が存在した．また各 OS ごとにコミット数が大きくなる時期には差異があった．これは，OS 側に進化が生じた際に発生したのではないかと考えられる．そこで，他の OS に比べ欠陥修正コミット数の大きい Windows について，Windows の全てのシリーズのリリース期間を含んでいる PostgreSQL を対象に，それぞれのシリーズのリリース日の前後の期間を対象とし調査を行った．調査の方法としては，リリース前の 2 ヶ月とリリース後の 2 ヶ月，その他の期間の 2 ヶ月の平均を調査した．調査対象とした Windows シリーズのリリース日を　図 2 に示す．図の横軸は時系列で目盛幅は 3 ヶ月，縦軸はコミット数である．図に点線で示された箇所がリリース日であり，網掛けで示された期間が対象としたリリース前後の期間である．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数が 2.4 倍大きく行われている．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 0.43 倍小さい．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．

リリース期間のコミットが OS のリリースに伴って発生した欠陥を修正しているのか確かめるために，リリース期間のコミットを手動で確認した．WindowsXP のリリース日のコミットメッセージを表 6 に示す．表 6 から，Windows の環境が変化したことにより発生した欠陥を修正したとされるコミットメッセージが確認された．

> OS のリリースに伴う欠陥の修正が行われており，リリース前の 2 ヶ月間の欠陥修正コミット数はその他の 2 ヶ月間の期間に比べ 2.4 倍大きい．

5. 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



(a) Squid (b) Graphviz

(c) PostgreSQL (d) Qt5

図 1 OS 依存欠陥修正コミットの推移

表 6 WindowsXP 64bit リリース日のコミットメッセージ

コミット	メッセージ文	解釈
commit1	Fix breakage of LINUX_PROFILE code due to recent Windows changes.	最近の Windows の変化に伴って発生した LINUX_PROFILE 破損の修正.

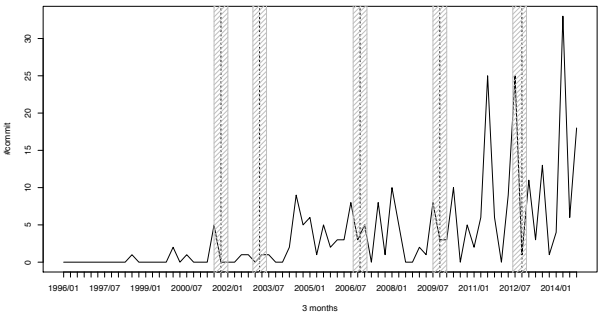


図 2 Windows 向け欠陥修正コミットとリリース日

表 5 リリース日前後の Windows 向け欠陥修正コミット数

期間	2 ヶ月の平均コミット数
リリース前	6.6
リリース後	1.2
その他の期間	2.8

進化にかかるコストがどの程度であるのかを明らかにすることを目的として，2 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 2 つのリサーチクエスチョンを実施した結果，OS 依存欠陥に関して下記の知見を得た．

- キーワードに基づく特定方法により取得した特定 OS 向けの欠陥修正コミットが全体の欠陥修正コミットに占める割合は 5.4%であった．
- Windows 向けの欠陥修正コミットは，他の OS に比べ多かった．
- OS のリリースに伴う欠陥の修正が行われており，リリース前の 2 ヶ月間の欠陥修正コミット数はその他の 2 ヶ月間の期間に比べ 2.4 倍大きい．

謝辞 本研究の一部は，日本学術振興会 科学研究費補助金（若手 A：課題番号 24680003，挑戦的萌芽：課題番号 25540026）による助成を受けた．

参考文献

[1] Adrian Bachmann, Christian Bird, Foyzur Rahman, Premkumar Devanbu, and Abraham Bernstein. The missing links: bugs and bug-fix commits. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pp. 97–106, 2010.

[2] Rajiv D Banker and Sandra A Slaughter. A field study of scale economies in software maintenance. *Management science*, Vol. 43, No. 12, pp. 1709–1725, 1997.

[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] Michael E Fagan. Advances in software inspections. In *Pioneers and Their Contributions to Software Engineering*, pp. 335–360. 2001.

[5] Miryung Kim, Dongxiang Cai, and Sunghun Kim. An empirical investigation into the role of api-level refactorings during software evolution. In *Proceedings of the 33rd International Conference on Software Engineering*, pp. 151–160, 2011.

[6] Sunghun Kim and E James Whitehead Jr. How long did it take to fix bugs? In *Proceedings of the 2006 international workshop on Mining software repositories*, pp. 173–174, 2006.

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

[8] Paul Luo Li, James Herbsleb, Mary Shaw, and Brian Robinson. Experiences and results from initiating field defect prediction and product test prioritization efforts at abb inc. In *Proceedings of the 28th international conference on Software engineering*, pp. 413–422, 2006.

[9] Shane McIntosh, Bram Adams, Thanh HD Nguyen, Yasutaka Kamei, and Ahmed E Hassan. An empirical study of build maintenance effort. In *Proceedings of the 33rd international conference on software engineering*, pp. 141–150. ACM, 2011.

[10] Osamu Mizuno, Shiro Ikami, Shuya Nakaichi, and Tohru Kikuno. Fault-prone filtering: Detection of fault-prone modules using spam filtering technique. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, pp. 374–383, 2007.

[11] Yuan Tian, Julia Lawall, and David Lo. Identifying linux bug fixing patches. In *Software Engineering (ICSE), 2012 34th International Conference on*, pp. 386–396, 2012.

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

[13] Andy Zaidman, Bart Van Rompaey, Serge Demeyer, and Arie Van Deursen. Mining software repositories to study co-evolution of production & test code. In *Software Testing, Verification, and Validation, 2008 1st International Conference on*, pp. 220–229. IEEE, 2008.

[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.