テスト履歴に基づくfault-proneモジュール予測に向けた検討

吉行 勇人[1,a)]　大平 雅雄[1,b)]

概要：ソフトウェア開発では，不具合を取り除き信頼性を向上させるためにソフトウェアテストが行われる．効率的にテスト工数を割り当てるために不具合が含まれていそうなモジュール（fault-prone モジュール）を予測する手法が求められている．先行研究ではプロダクトメトリクスやプロセスメトリクスを利用した予測手法が提案されている．fault-prone モジュール予測はテスト工程前に適用することを想定しているため，これらの研究ではテスト工程以前に収集可能なメトリクスのみを利用している．しかし，近年ラピッドリリース方式等の複数回のリリースを繰り返す開発プロセスが広く行われており，前バージョンの開発履歴を利用できる．本研究では特に前バージョンでのテストの履歴に着目し，fault-prone モジュール予測の予測精度向上を目指す． 本稿は検討段階である予測手法について説明し，研究方針の共有を目的とする．

1. はじめに

近年，ソフトウェア開発の短納期化，低コスト化が求められている一方で，ソフトウェアが担う社会的責務は大きくなり，ソフトウェアの品質管理の重要性が高まっている．ソフトウェア開発において，プログラム製造後に製造したプログラムが正しく動作するかどうかを確かめ，不具合を発見して信頼性を向上させるために，ソフトウェアテストが行われる．市場要求を満たすソフトウェアをタイムリーに提供するためには，ソフトウェアテストにどれだけの工数をかけるかが重要となる．一般的に，テスト工数と不具合検出率は比例せず，レイリー曲線に従うことが知られている．これは，不具合の発見効率という観点では，テストにかけるべき工数の上限が存在することを意味し，工期との兼ね合いからテストにかけられる実工数にも限りがある．また，ソフトウェアの不具合は全てのモジュールに均一に存在しているわけではなく，複雑度やモジュール間の依存関係によって，不具合を含みやすいモジュールとそうでないモジュールが存在する [3]．よって，限られた工数で不具合を効率的に取り除くためには，不具合が含まれていそうなモジュール (fault-prone モジュール) から優先的にテストを行うことが有効である．

このような背景から，fault-prone モジュールを予測する手法が多く提案されている．先行研究では，ソースコードの行数や複雑度等のプロダクトメトリクス [5], [9]，ソースコードの変更履歴や過去の不具合数等のプロセスメトリクス [5], [8], [14] に基づいた予測が行われている．fault-prone モジュール予測はテスト工程の計画時に行われることを想定しているため，先行研究ではテスト工程以前に集計可能なメトリクスのみを利用している．一方で，近年，ラピッドリリースサイクルによるソフトウェア開発がいくつかのソフトウェアプロジェクトで行われている．ラピッドリリースサイクルでは，従来のリリースサイクルに比べて短いサイクルでリリースが行われるため，複数のリリースの情報が開発履歴として蓄積されやすい．特に，本研究ではリリース前に行われるソフトウェアテストの履歴に着目する．ソフトウェアテストの成否は不具合の有無と深く関係しており，fault-prone モジュールの特定に役立つ可能性がある．

そこで，本研究では，リリース時のテストの履歴を利用した fault-prone モジュール予測手法を構築し，予測精度向上を目指す．本稿は，検討段階である提案手法について説明し，研究方針の共有を目的とする．

本稿の構成は以下の通りである．続く第 2 章では，本研究で適用対象とするラピッドリリースサイクルについて述べる．第 3 章では本研究と関連する先行研究について述べる．第 4 章では本研究で提案する fault-prone モジュール予測手法について述べる．第 5 章ではまとめと今後の課題を述べる．

[1] 和歌山大学
Wakayama University
[a)] s151054@center.wakayama-u.ac.jp
[b)] masao@sys.wakayama-u.ac.jp

2. ラピッドリリースサイクル

本章では，近年多くのソフトウェアプロジェクトで行われているラピッドリリースサイクルについて説明する．

ラピッドリリースサイクルは，従来のリリースサイクルに比べて短いサイクルでリリースを行う開発サイクルである．ラピッドリリースサイクルでは，新たな機能や不具合の修正を早くユーザに提供することができ，競合製品との競争を有利に進めることができる．

近年では，Google Chrome や Mozilla Firefox，Microsoft，Facebook 等のソフトウェアプロジェクトでラピッドリリースサイクルが採用されている．例として，図 1 に，Mozilla Firefox のリリースサイクルを示す．Mozilla Firefox では従来は 1 年ごとに行われていたメジャーバージョンリリースを，6 週間ごとに行うように変更している．開発は NIGHTLY, AURORA, BETA, RELEASE の 4 バージョンが並行して行われている．NIGHTLY は，最も活発に開発が行われ，試験的に新しい機能の追加等を行うことを目的としている．AURORA は NIGHTLY で開発されたバージョンを Web 開発者や初期ユーザ等に公開し，互換性や安定性の確認を行うことを目的としている．BETA は AURORA よりも安定したバージョンで，公式リリース前に新たに追加された機能等を試す環境として提供されている．RELEASE は公式のリリースで多くのユーザが利用する最も安定したバージョンである．各リリースの間隔は 6 週間であるため，1 つのメジャーバージョンの開発に計 4 バージョンで 24 週間かけていることになる．よって，RELEASE バージョンを開発している時点で，BETA 以前のリリース時の情報を利用することができる．また，図 1 の例では，6.0 の BETA を引き継いで 7.0 の BETA を開発しているため，同じメジャーバージョンの前バージョンだけでなく，1 つ前のメジャーバージョンの情報も利用できる．

したがって，ラピッドリリースサイクルを採用しているプロジェクトでは，従来のリリースサイクルに比べてリリース時の情報が蓄積されやすい．特にリリース時に蓄積されやすい情報として，ソフトウェアテストの情報が挙げられる．ラピッドリリースサイクルでは従来のリリースサイクルに比べてテストに対する姿勢が異なり，全てのテストをパスする必要は無く，テストをパスできなかった機能等に関しては，次回以降のバージョンに含めるような意思決定を行う方針がとられている．よって，テストが失敗したモジュールは必ずしもすぐに対応が行われるわけではなく，次バージョンの開発時に再び問題に直面する場合があると考えられる．このとき，以前のバージョンでのテスト履歴が fault-prone モジュールの特定に有用である可能性がある．

そこで，本研究では前バージョンのソフトウェアテスト

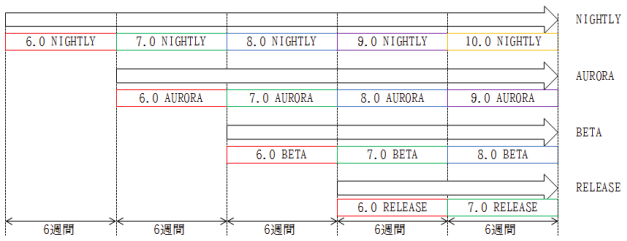

図 1 Mozilla Firefox におけるラピッドリリースサイクル

の履歴を用いて現バージョンの fault-prone モジュールを予測する手法を構築することを目的とする．複数のリリースの情報が蓄積されやすいラピッドリリースサイクルを採用しているプロジェクトは本研究の対象として適当であると言える．

3. 関連研究

本章では本研究に関連する研究について述べる．

3.1 予測モデル構築に用いられるメトリクス

先行研究で行われている fault-prone モジュール予測の説明変数として用いられるメトリクスについて説明する．メトリクスは大きく分けてプロダクトメトリクスとプロセスメトリクスに分類される．プロダクトメトリクスは，ソフトウェア開発によって生成される成果物（プロダクト）から計測されるメトリクスを指す．プロセスメトリクスは，ソフトウェア開発のプロセス上の作業等を定量化したものを指す．

3.1.1 プロダクトメトリクス

fault-prone モジュール予測で用いられる代表的なプロダクトメトリクスとして，ソースコードの行数や複雑度，クラス数，メソッド数等がある．プロダクトメトリクスは実際のソースコードがあれば集計可能であるため，適用しやすく多くの先行研究で用いられている [5], [9]．その他，モジュール間の依存関係を用いた予測 [12] や，コードクローンと不具合の有無の分析 [10] が行われている．

3.1.2 プロセスメトリクス

ソースコードの変更行や変更回数，開発者の名前，前バージョンからの追加・削除行数，過去の不具合数等のメトリクスを利用して予測を行う [5], [8], [14]．これらのメトリクスは，版管理システムや不具合管理システムから得られる．前バージョンからの追加・削除行数は code churn と呼ばれており，予測に有用であると報告されている [8]．

3.2 モデル構築

先行研究では線形モデルやロジスティック回帰モデル等のさまざまなモデル化手法が用いられている．一般に，あるソフトウェアモジュールにおける不具合の有無は確率的に発生すると考えられるため，多くの先行研究では

確率に基づくモデル化手法を用いている．Lessmann らは fault-prone モジュール予測における各モデル化手法の性能評価を行い，ランダムフォレスト法 [2] が最も有望であると報告している [6]．

3.3 fault-prone モジュール予測の利用評価

これまで提案された fault-prone モジュール予測手法は実データを用いた評価実験等により予測精度の評価は行われているものの，予測結果の利用による効果についての検証が十分に行われていなかった．そこで，近年では予測結果に基づいてテスト工数を割り当てたときのテスト工数削減の効果や信頼性に与える影響の評価が行われている．柿本らは TEAR モデルと呼ばれる，テスト工数割り当てとソフトウェア信頼性のモデル化を提案している [13]．また，中野らは，fault-prone モジュール予測の予測結果に基づくテスト工数割り当て（テスト戦略）手法を提案し，オープンソースソフトウェアプロジェクトを対象とした評価を行っている [15]．

これらの研究は fault-prone モジュール予測の予測精度向上によって，手法の信頼性と有用性をさらに高めることができる．

3.4 ラピッドリリースサイクルの影響分析

Khomh らは，ラピッドリリースサイクルが品質に与える影響を Mozilla Firefox プロジェクトを対象として調査している [4]．[4] では各リリース後に報告される 1 日あたりの不具合数は，従来のリリースサイクルとラピッドリリースサイクルで大きな違いが無く，ラピッドリリースサイクルでは報告された不具合のうち実際に修正が行われる不具合の数が少なくなっていることが報告されている．また，ユーザはラピッドリリースサイクルの方がソフトウェアアップデートを実行しやすい傾向がある．Mantyla らは，ラピッドリリースサイクルがソフトウェアテストに与える影響について調査している [7]．[7] では，従来のリリースサイクルでは全てのテストスイートを実行していたのに対して，ラピッドリリースサイクルではリスクの高い箇所にスコープを絞ってテストを行っていることが報告されている．

4. 提案手法

本章では本研究で提案する fault-prone モジュール予測手法について説明する．

4.1 概要

本手法は従来の fault-prone モジュール予測手法で用いられているメトリクスに加えてテスト履歴から得られるメトリクスを用いて，予測精度向上を目指す．図 2 に本手法の適用手順を示す．テスト管理システムから得られるテスト履歴メトリクスを説明変数，不具合管理システムと版管理システムから得られる不具合の有無を目的変数として予測モデルを構築する．各種変数はバージョンごとに集計し，予測対象バージョンの 1 つ前のバージョンのデータを学習データ，予測対象のバージョンのデータを予測データとしてモデル構築および予測を行う．

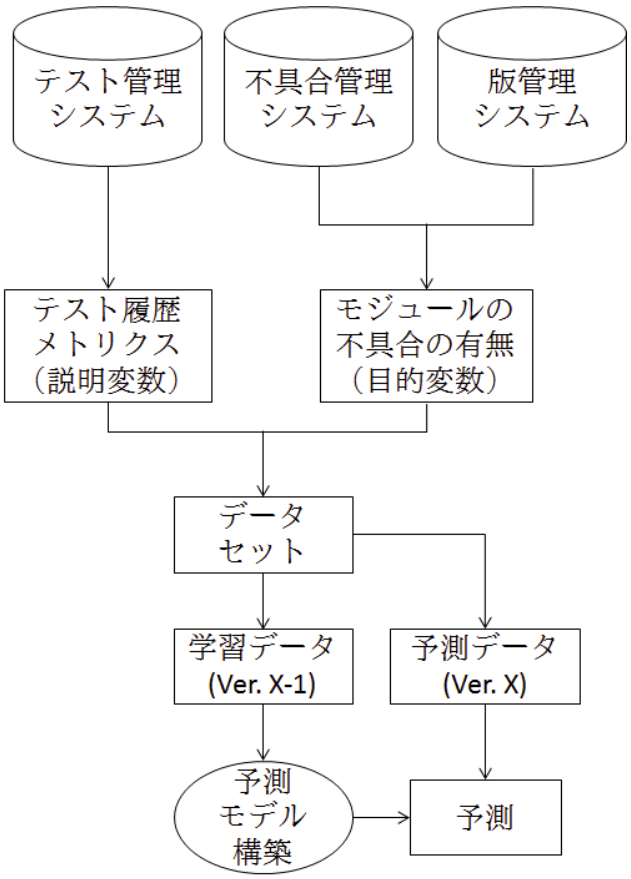

図 2 提案手法の適用手順

4.2 適用対象

本手法は，ラピッドリリースサイクル等の複数回のリリースを繰り返す開発プロセスに対して適用されることを想定している．ソフトウェアテストは主にリリース前に行われるため，リリースが頻繁に行われる環境はテスト履歴の情報が多く蓄積され，本手法の適用に必要な情報が取得しやすい環境であると言える．

本手法は，バージョン間の予測を対象とする．バージョン X の fault-prone モジュールを予測する場合，バージョン X-1 のテスト履歴を学習データ，バージョン X のデータをテストデータとして予測モデルを構築する．

予測対象となるモジュールの粒度は，大きい場合はディレクトリ単位，小さい場合はメソッド単位が考えられる．粒度が大きすぎると実際の不具合箇所の特定が困難となり，粒度が小さすぎると十分な予測精度が得られないため，適用対象に応じて調整する必要がある．本手法では代表的な

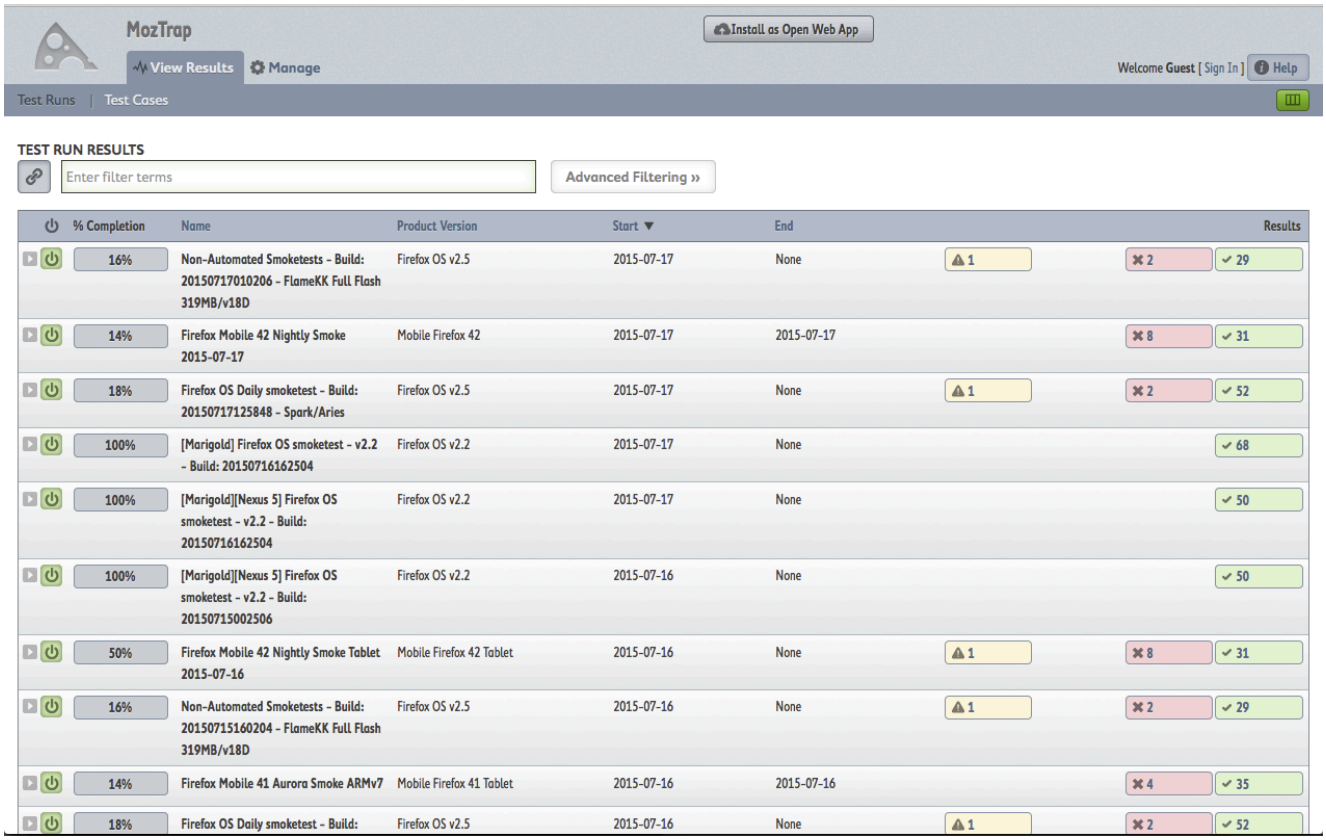

図 3　MozTrap のテスト実行履歴画面

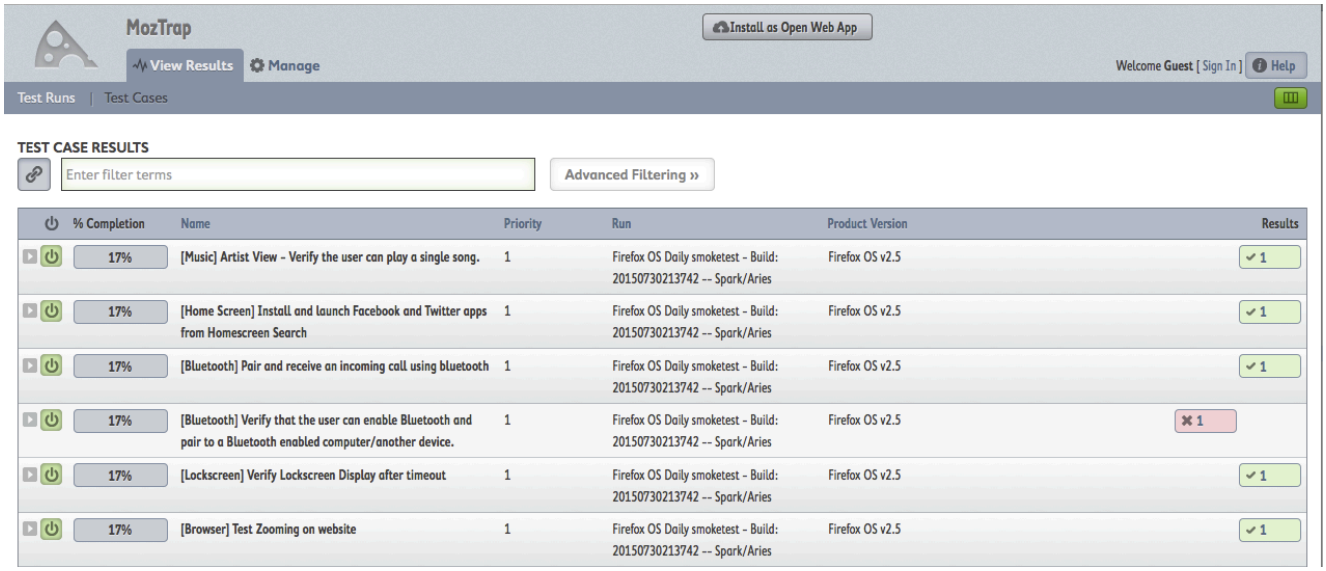

図 4　MozTrap のテストケース一覧

粒度であるソースコード単体をモジュール単位とする．

4.3　メトリクス

本手法で用いるメトリクスについて説明する．本手法で用いるメトリクスを大きく分けてプロダクトメトリクス，プロセスメトリクス，テスト履歴メトリクスに分類する．プロダクトメトリクスとプロセスメトリクスは主に先行研究で利用されているメトリクスを用いる予定である．テスト履歴メトリクスは本研究で新規に取り入れる要素である．

本研究ではテスト履歴メトリクスとして，Mozilla が開発したソフトウェアテスト管理ツール MozTrap[1] から得られるデータを用いる．MozTrap は主にシステムテストの協調作業のためのアプリケーションである．MozTrap は大きく分けてテスト実行履歴，テストケース一覧，テストケー

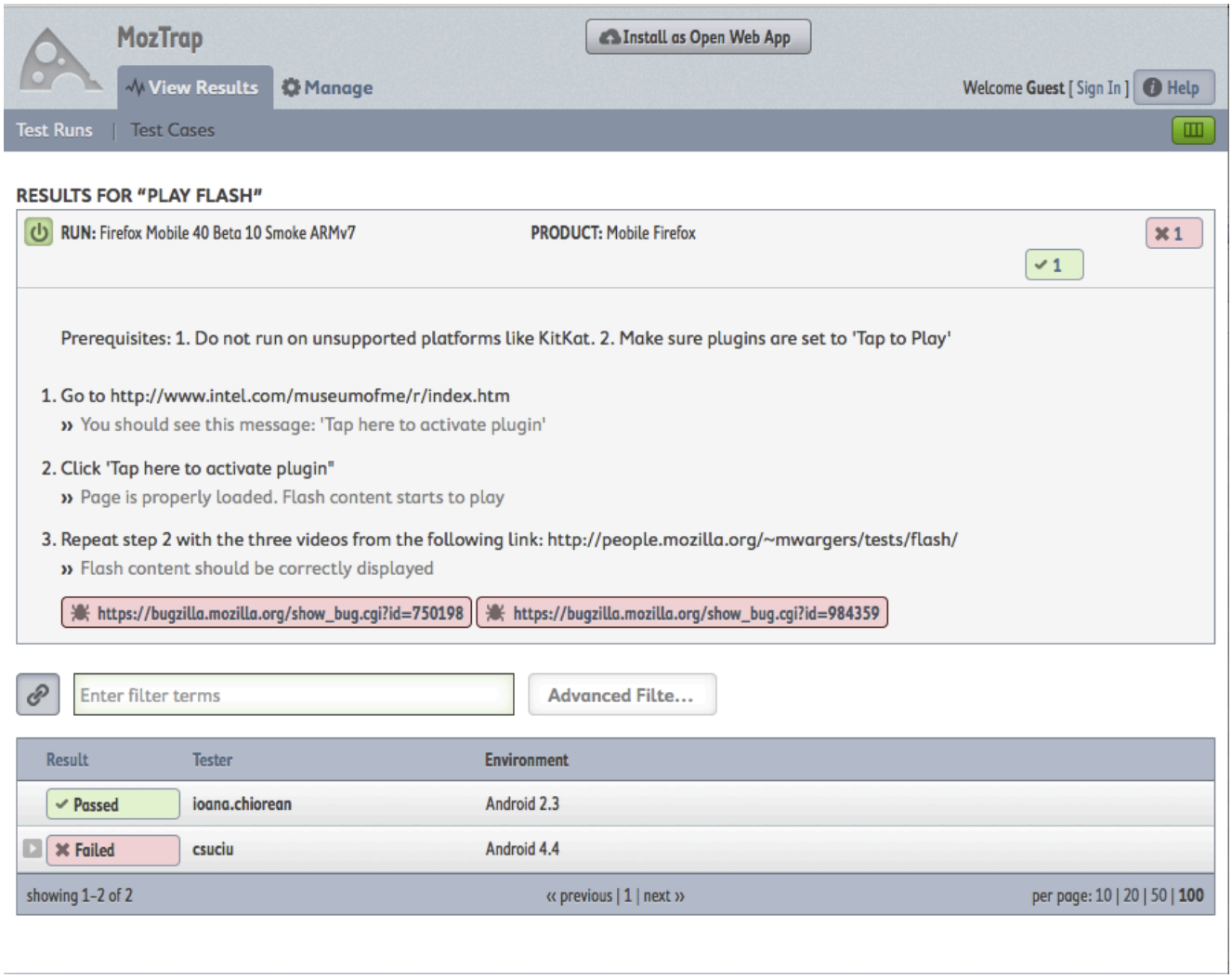

図 5　MozTrap のテストケース詳細画面

表 1　テスト履歴メトリクス

メトリクス名	尺度	説明
テストケース実行回数	比例	テストケースを実行した回数
テスト成功数	比例	テストケースの実行が成功した回数
テスト失敗数	比例	テストケースの実行が失敗した回数
テストステップ数	比例	テストケースに含まれるステップの数
リリースまでの日数	比例	そのテストケースを実行してからリリースされるまでの日数
テスト実行者数	比例	テストケースを実行した人数
テスト実行者	名義	テストケースを実行した人のユーザ名

ス詳細の 3 つのページから構成される．図 3 に MozTrap のテスト実行履歴の画面イメージを示す．テスト実行履歴画面には，実行されたテストスイートごとに，テストスイート名，テスト対象のプロダクト名およびバージョン，テスト実行日，テストに失敗または成功したテストケース数等が表示される．図 4 に MozTrap のテストケース一覧の画面イメージを示す．テストケース一覧画面には，テストケース名，優先度，そのテストケースを実行したテストスイート名，実行の成否等が表示される．図 5 に MozTrap のテストケースの詳細画面を示す．テストケースの詳細画面には，テストケースのステップ，テストの成否，実行環境，実行者等が記録されている．テストケースは実行環境等を設定して複数回実行される場合もあり，各実行に対してテスト結果が記録される．また，そのテストケースと関連する不具合が報告されている場合，不具合管理システムのリンクが記録されている．

MozTrap から集計可能なメトリクスを表 1 に示す．

各メトリクスは，モジュールごとに算出する．そのためには，各テストケースと各モジュールを紐付ける必要がある．テストケースには，対応する不具合管理システムに報

告された不具合番号が記録されており，4.4 節で述べる不具合情報の抽出手法の過程で，不具合番号とソースコードの紐付けを利用してテストケースとソースコード（モジュール）の紐付けを行う．

4.4 不具合情報の抽出

本手法の予測モデルの目的変数である各モジュールの不具合の有無は，Śliwerski らによって提案された SZZ アルゴリズム [11] を用いて抽出する．SZZ アルゴリズムはいつどのモジュールに不具合が混入したかを特定するアルゴリズムで，先行研究で広く利用されている．版管理システムと不具合管理システムに記録された情報を元に，どの時点のモジュールに不具合が存在しているかを特定する．

SZZ アルゴリズムの手順は以下の通りである．

(1) **不具合修正のための変更の抽出**

版管理システムに記録されたソースコードの変更履歴の中から，不具合を修正するための変更を抽出する．版管理システムには主に変更の理由や内容を記録するためのコミットメッセージと呼ばれる機能がある．多くのソフトウェアプロジェクトでは，版管理システムのコミットメッセージに不具合管理システムの不具合番号を記載することで，どの不具合を修正するための変更であるかを記録している．具体的には，Mozilla プロジェクトではコミットメッセージに “Bug XXXXXX” の記述を入れることで管理している．そこで，同様の記述のあるコミットメッセージが含まれる変更を不具合修正のための変更として抽出する．

(2) **不具合の原因箇所の特定**

版管理システムのログには各変更でどのファイルのどの行を変更したかが記録されている．まず，手順 1 で抽出した不具合修正のための変更時に，変更されたファイルおよび行には不具合が存在していたと仮定する．次に，版管理システムの機能である annotate や blame コマンドを用いて，その行がいつの時点で追加，編集された行であるかを特定し，不具合の原因の候補とする．このとき，空行やコメントの変更は不具合と直接関係無いとして，不具合の原因候補に含めない．

(3) **整合性の確認**

手順 2 で特定された，不具合の原因候補とした行は，全ての行が不具合の原因であるとは限らない．そこで，その行が追加，編集された日付が対応する不具合の報告日よりも前であるかどうかを確認する．そうであった場合は，報告された不具合の原因であるとして，その時点のファイルに不具合が存在していたとラベル付けする．そうでなければ，誤判定として不具合の原因ではないとし，その時点のファイルには不具合が存在していなかったとする．

4.5 モデル化手法

不具合の含有は確率的に発生すると考えられるため，fault-prone モジュール予測に適したモデル化手法は，判別分析かつ確率を表すことができるものである．そこで，本手法ではロジスティック回帰分析およびランダムフォレスト法を用いる．

ロジスティック回帰分析は，2 クラスの標本を判別する判別関数にロジスティック関数を用いて式 1 のように表現する．

$$P(y|x_1,...,x_p) = \frac{1}{1+e^{-(\alpha_1\times_1+...+\alpha_p\times_p+\beta)}} \qquad (1)$$

ここで，$y \in \{0,1\}$ は 2 値のクラスを表す目的変数，x_i は目的変数，α_i は回帰係数，$P(y|x_1,...,x_p)$ は説明変数 x_i に対して y が 1 のクラスに属する確率である．本手法の場合，あるモジュールに不具合が含まれる確率 $P(y|x_1,...,x_p)$ の値が閾値を超えた場合に，そのモジュールに不具合が含まれると判定する．閾値は一般的には 0.5 とすることが多いが，モデルのチューニング方針によって調整する必要がある．

ランダムフォレスト法は，回帰木を用いて集団学習を行う手法であり，2001 年に Breiman によって提案された [2]．モデル構築用のデータセットに対して繰り返しランダムサンプリング（復元抽出）を行い，得られたサンプル群から多数の回帰木を構築する．そして，各回帰木の出力の平均を最終的な予測結果として得る．従来の集団学習ではモデル構築時に全ての説明変数を用いていたのに対して，ランダムフォレスト法では無作為に選択された説明変数を用いてモデルを構築する．ランダムフォレスト法は，[6] で最も高い性能であることが報告されており，多くの先行研究で用いられている．

また，ロジスティック回帰分析とランダムフォレスト法はどの説明変数がモデル構築に寄与したかを算出することができ，本手法で新たに提案したメトリクス群の有意性を確かめることができる．

5. まとめと今後の課題

本稿ではテスト履歴を利用した fault-prone モジュール予測手法を提案した．本手法は，ラピッドリリースサイクル等の複数回のリリースを繰り返すリリースサイクルを対象として，前バージョンリリース時のテスト履歴から得られるメトリクスを用いて，現バージョンの fault-prone モジュールを予測する手法である．

本稿で提案した手法は検討段階であり，今後は本手法の有用性を確かめる実験を行う予定である．実験では，MozTrap に記録されている Mozilla プロジェクトの実際のテスト履歴を用いて，予測モデルを構築し，評価を行う．

謝辞 本研究の一部は，文部科学省科学研究補助金（基盤 (C): 15K00101）による助成を受けた．

参考文献

[1] : Mozilla MozTrap, https://moztrap.mozilla.org/.

[2] Breiman, L.: Random Forests, *Machine Learning*, Vol. 45, pp. 5–32 (2001).

[3] Glass, R. L.: *Facts and Fallacies of Software Engineering*, Addison-Wesley (2003).

[4] Khomh, F., Dhaliwal, T., Zou, Y. and Adams, B.: Do faster releases improve software quality? An empirical case study of Mozilla Firefox, *Proceedings of the 9th International Workshop on Mining Software Repositories (MSR '12)*, pp. 179–188 (2012).

[5] Kim, S., Whitehead, E. and Zhang, Y.: Classifying Software Changes: Clean or Buggy?, *IEEE Transactions on Software Engineering (TSE)*, Vol. 34, No. 2, pp. 181–196 (2008).

[6] Lessmann, S., Baesens, B., Mues, C. and Pietsch, S.: Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings, *IEEE Transactions on Software Engineering*, Vol. 34, No. 4, pp. 485–496 (2008).

[7] Mantyla, M., Khomh, F., Adams, B., Engstrom, E. and Petersen, K.: On Rapid Releases and Software Testing, *Proceedings of the 29th IEEE International Conference on Software Maintenance (ICSM '13)*, pp. 20–29 (2013).

[8] Nagappan, N. and Ball, T.: Use of Relative Code Churn Measures to Predict System Defect Density, *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pp. 284–292 (2005).

[9] Nagappan, N., Ball, T. and Zeller, A.: Mining Metrics to Predict Component Failures, *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)*, ICSE '06, pp. 452–461 (2006).

[10] Rahman, F., Bird, C. and Devanbu, P.: Clones: What is that smell?, Proceedings of the 7th International Workshop on Mining Software Repositories (MSR ' 10), pp. 72–81 (2010).

[11] Śliwerski, J., Zimmermann, T. and Zeller, A.: When Do Changes Induce Fixes?, *Proceedings of the 2nd International Workshop on Mining Software Repositories (MSR '05)*, MSR '05, pp. 1–5 (2005).

[12] Zimmermann, T. and Nagappan, N.: Predicting Defects Using Network Analysis on Dependency Graphs, *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, pp. 531–540 (2008).

[13] 柿元　健，門田暁人，亀井靖高，まつ本真佑，松本健一，楠本真二：Fault-prone モジュール判別におけるテスト工数割当てとソフトウェア信頼性のモデル化，情報処理学会論文誌，Vol. 50, No. 7, pp. 1716–1724 (2009).

[14] 畑　秀明，水野　修，菊野　亨：開発履歴メトリクスを用いた細粒度な Fault-prone モジュール予測，情報処理学会論文誌，Vol. 53, No. 6, pp. 1635–1643 (2012).

[15] 中野大輔，門田暁人，亀井靖高，松本健一：バグモジュール予測を用いたテスト工数割り当て戦略のシミュレーション，コンピュータソフトウェア，Vol. 31, No. 1, pp. 118–128 (2014).