不具合混入コミットの推定手法間での整合性比較と考察

北村 紗也加
京都工芸繊維大学 大学院工芸科学研究科 博士前期課程 情報工学専攻
s-kitamura@se.is.kit.ac.jp

水野 修
京都工芸繊維大学 情報工学・人間科学系
o-mizuno@kit.ac.jp

要旨

ソフトウェアの複雑さと重要性は日々増してきており，ソフトウェアの品質を高い水準で保つことが重要視されている．このような現状においてはソフトウェアの品質予測は重要な研究テーマであり，どのような手法で品質予測を行うかに注力されてきた．不具合混入コミットを推定する手法では，その評価を行うためには正解データが必要であり，不具合の正解データ *(*真値*)* として *Commit Guru* による不具合混入コミットの情報が多く利用されている．しかしながら，*Commit Guru* の不具合混入コミットの情報の正解データとしての信頼性は不明である．

本研究では，その信頼性に対する検証を行った．不具合混入コミット推定手法である *SZZ* アルゴリズムを用いて，同じ不具合データに対する結果の整合性を比較し，その結果を考察した．*Commit Guru*，および *SZZ* アルゴリズムを用いた不具合コミット推定結果の差異において検証を行った結果，*Commit Guru* の方がより優れた不具合コミット推定の結果を示し，正解データとしての可能性を示すものとなったが，その信頼性は十分であるとは言い難い．

1 はじめに

近年，ソフトウェアの重要性と複雑さは増しつつある．このような現状におかれたソフトウェア開発の現場では，ソフトウェアに混入した不具合を修正するために膨大な時間とエフォートが割かれており，またそれに伴うコストも膨大であるため [1]，ソフトウェアの品質を高い水準で保つことが重要視されている．したがって，最近のソフトウェア工学の研究ではソフトウェアの分析とソフトウェアの品質予測に焦点を当てたものが多く見受けられる [2].

ソフトウェア工学における品質予測の研究を活性化させたものとして，Śliwerski，Zimmermann，Zeller によって提案された，ソースコードやリポジトリデータを用いて不具合を混入したと推定されるコミットを発見する SZZ アルゴリズム [3] がある．SZZ アルゴリズムはバージョン管理システムにおいて変更履歴を辿り，不具合を混入したコミットを特定する．この SZZ アルゴリズムとはまた別のアルゴリズムで不具合コミットを特定するツールを公的に入手可能にしたものとして，Commit Guru [4] 等が存在する．

Commit Guru は，登録された Git リポジトリにおける全てのコミットに対し分析を行い，13 のメトリクスを計算し，不具合を混入したコミットを推定する．また，その分析結果を利用者らに無償で提供している．

Commit Guru による不具合混入コミット（以降，不具合コミットと呼称）の情報は，昨今の研究において不具合の正解データとして利用されることが多い．しかしながら，Commit Guru が推定する不具合コミットは簡易的な手法を用いており，正解データとしての信頼性は不明である．そこで，本研究では Commit Guru および SZZ アルゴリズムを用いて，分析結果の整合性を比較し，Commit Guru の正解データとしての信頼性について考察を行う．

2 準備

2.1 バージョン管理システム

バージョン管理システムとは，コンピュータ上で作成・編集されるファイルの変更履歴を管理するためのシステムである．本研究で用いた SZZ アルゴリズム，および Commit Guru はバージョン管理システムの情報を用いて不具合コミットを推定しており，バージョン管理システ

ムとして分散型バージョン管理システムである Git を使用した．

2.2 バグトラッキングシステム

バグトラッキングシステムとはプロジェクトのバグを登録し，そのバグの修正状況を追跡するシステムであり，バグの履歴管理や検索を行うことができる．本研究で用いた SZZ アルゴリズムは，バグトラッキングシステムから取得した対象プロジェクトの不具合データを用いて不具合コミットを推定しており，バグトラッキングシステムとして Bugzilla と Apache's JIRA issue tracker を使用した（表 1）．

2.3 SZZ アルゴリズム

SZZ アルゴリズムは不具合混入コミット推定手法の中で最もよく知られている手法である．

本研究で使用した SZZ アルゴリズムは，バージョン管理システムのバージョンアーカイブとバグトラッキングシステムの不具合データを用いて，次の 2.3.1 節，2.3.2 節で述べる 2 つのステップを経ることにより不具合コミットの推定を行う．

2.3.1 修正された箇所の特定

バージョン管理システムのバージョンアーカイブには，変更に関する情報が含まれている．バージョン間において同じ意図で行われてた変更を特定するため，バージョンアーカイブから不具合が混入されたコミットの日時，不具合を最後に修正されたコミットの日時を抽出し，バグトラッキングシステムの不具合データと照らし合わせる．抽出されたデータが不具合の修正と予測されるのであれば，この不具合を修正したであろうコミットに対し，修正したことを示す FIX タグをつける．

バージョン管理システムのバージョンアーカイブには変更の目的が欠けており，行われた修正が不具合を修正したのか，機能追加であるか判断することができない．ログメッセージのみから目的を予測することは出来る [5] が，不具合に関する簡単なレポート，不具合のステータスや解決策が格納されている不具合データを組み合わせることで，不具合予測の精度向上を図る．

[1]問題とは不具合，機能要求，改善，タスクなど不具合以外にも様々なものを指すが，本研究においては不具合のみを対象としたバグトラッキングシステムとして用いている．

2.3.2 不具合が混入された箇所の特定

FIX タグがつけられたコミットを取得する．このコミットにおいて変更されたファイルに対し，バージョン管理システムの diff コマンドを用いてコミットにおける変更箇所を抽出する．

もし変更箇所が見つからなければ FIX タグの削除を行い，変更箇所が見つかれば，その変更はどのコミットにおいて行われたのかを特定する．変更箇所に対してバージョン管理システムの blame コマンドと正規表現を用いて，その変更を行ったコミットの日時を抽出する．抽出されたデータから不具合を混入したと予測されるコミットに対し，不具合コミットであることを示す BUG タグをつける．

2.4 Commit Guru

Commit Guru は Kamei らの変更レベルの不具合予測モデル [6] を Web アプリケーションとして実装したものであり，Rosen らによって公開されている．本研究において，Commit Guru での不具合混入ラベル，メトリクスを分析データの整合性比較に用いる．Commit Guru の機能である (1) 不具合コミットのラベル付け，(2) メトリクスの測定，および (3) ダンプデータの提供について次に述べる．

2.4.1 不具合コミットのラベル付け

Commit Guru はコミットに対し，不具合を混入したコミット (Buggy) とそれ以外のコミット (Clean) を不具合混入ラベルにおいて TRUE/FALSE でラベル付けを行う．不具合コミットは不具合を修正したコミットから推定できる．

始めに，Commit Guru は Hindle らの研究 [7] におけるコミットを分類するためのキーワードリストを元に，コミットメッセージの解析を行い，各カテゴリに分類する．ここでは，' bug ', ' fix ', ' wrong ', ' error ', ' fail ', ' problem ', ' patch ' といったワードを含むコミットが不具合を修正しているであろうコミットとし，Corrective に分類する．

次に，修正を行ったコミットから不具合を混入したであろうコミットを特定する．まずバージョン管理システムの diff コマンドを用いて，どの行が修正を行ったコミットによって変更されたのかを特定する．ここで変更された行

表 1. 使用したバグトラッキングシステムとそのメトリクス

システム	システム概要	メトリクス	メトリクス概要
Bugzilla	Mozilla Foundation によって開発されたウェブベースのバグトラッキングシステム	Bug ID Opened Changed	不具合に対して一意に与えられる ID 不具合が発見された日付 不具合が最終的に修正された日付
Apache's JIRA issue tracker	Atlassian によって開発された商用の問題トラッキングシステム[1]	Issue key Created Resolved	問題に対して一意に与えられるキー 問題が作成された日付 問題が解決された日付

に対して，バージョン管理システムの annotate/blame コマンドを用いることで，それらの修正されたソースコードがどのコミットで追加されたのかを特定する．ここで見つかったソースコードを組み込んだコミットを，不具合を混入したであろうコミットとしてラベル付けする．

2.4.2 メトリクスの測定

測定されるメトリクスを表 2 に示す．Kamei らの不具合予測モデルで利用されているものと同様の数値メトリクス 13 個，および論理メトリクス 1 個から成る変更に関するメトリクスである．

2.4.3 ダンプデータの提供

分析した Git リポジトリのダンプデータは CSV 形式で提供されている．このダンプデータには，2.4.2 節において述べた 14 個の計測したメトリクスに加え，コミットハッシュやそのコミットの分類カテゴリなどが格納されている．本研究では，このダンプデータを用いて分析を行っている．

2.5 マン・ホイットニーの U 検定

マン・ホイットニーの U 検定 (MannWhitney U test) とは，ノンパラメトリック検定のひとつであり,『対応のない 2 群が同じ分布の母集団から構成されている』とする帰無仮説に基づいて検定する．

データサイズ n_1 のデータ D_1，およびデータサイズ n_2 のデータ $D_2(n_1 \leq n_2)$ から成る 2 つのサンプルデータ群に対し，両郡のデータの値が小さい順に順位を割り当て，D_1, D_2 のそれぞれの順位和 R_1, R_2 を求める．同じ値のデータが存在した場合には，それらが異なると考えた場合の順位の平均値を割り当てる．このとき，D_1, D_2 の順位和の合計は次のようになる．

$$R_1 + R_2 = \frac{N(N+1)}{2} \tag{1}$$

これより，両郡のデータサイズ，順位和を用いて，次の式を用いて統計量を求める．

$$U_1 = R_1 - \frac{n_1(n_1+1)}{2} \tag{2}$$

$$U_2 = R_2 - \frac{n_2(n_2+1)}{2} \tag{3}$$

式 (2)，式 (3) によって求めた U_1, U_2 において小さい方の値を U とする．U は有意性を調べるときに用いられる．サンプルサイズが大きい場合，U は正規分布に近似できる．このとき，z は次式で表される．

$$z = \frac{U - m_U}{\sigma_U} \tag{4}$$

ここで，m_U および σ_U は U の平均および標準偏差であり，有意性を調べることができる標準正規偏差である．m_U および σ_U は次の式で表される．

$$m_U = \frac{n_1 n_2}{2} \tag{5}$$

$$\sigma_U = \sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}} \tag{6}$$

タイ値が存在した場合，次の式で σ の補正を行う．

$$\sigma_{\text{corr}} = \sqrt{\frac{n_1 n_2}{12}\left((n+1) - \sum_{i=1}^{k} \frac{{t_i}^3 - t_i}{n(n-1)}\right)} \tag{7}$$

ここで，$n = n_1 + n_2$ であり，t_i はランク i を共有するタイ値の数であり，k はランクの数である。

本研究においては，Python の scipy パッケージを用いてこのマン・ホイットニーの U 検定を使用した．

3 実験

3.1 研究設問

先に述べたとおり，不具合混入の正解データ (真値) を作成する手法として，SZZ アルゴリズムや Commit Guru

表 2. メトリクス一覧

名前	定義	説明	関連研究
la	変更によって追加されたコード行数	追加されるコード行数が多いほど不具合が混入しやすい.	コード行数の相対的なメトリクスは欠陥モジュールにおいて効果の高い指標である [8,9].
ld	変更によって削除されたコード行数	削除されるコード行数が多いほど不具合が混入しやすい.	
lt	変更前の総コード行数	大きいファイルほど，そのファイル内の変更は不具合を混入している可能性が高い.	大きいファイルほど不具合を多く含む [10].
ns	変更されたサブシステム数	修正したサブシステム数が多い変更ほど，不具合が多くなりやすい.	変更における不具合の確率はサブシステム数が増えるにつれ高くなる [11].
nd	変更されたディレクトリ数	修正したディレクトリ数が多い変更ほど，不具合が多くなりやすい.	変更されたディレクトリの数が多いほど，不具合を混入する機会が増える [11].
nf	変更されたファイル数	関係するファイル数が多い変更ほど，不具合が多くなりやすい.	モジュール内のクラス数はリリース後における欠陥が現れやすい特徴である [12].
ndev	修正されたファイルを変更したことがある人数	コーディングデザインの違いの観点から，ファイルの開発に携わった人数が多いほど不具合が混入しやすい.	ファイルに携わった人数が多いほど多くの不具合を含んでいる [13].
age	修正された全てのファイルにおいて前回の変更から経過した平均日数	直近の変更ほど，不具合を混入する原因になりやすい.	昔の変更より最近の変更の方がより不具合に寄与する [14].
nuc	修正されたファイルにおける変更の個数	以前の変更を探し出す必要があるため，nuc が大きいほど不具合を混入しやすい.	修正したファイルの範囲が広いほど，複雑さが増す [15].
exp	変更を行った開発者の総コミット数	経験豊富な開発者ほど不具合を混入しにくい.	プログラマの経験は不具合の確率を著しく低下させる [11].
rexp	age によって重み付けされた exp	最近そのファイルに対しよく変更を行なっている人物はそのファイルに対しての知識が深いと考えられるため，不具合を混入しにくい.	
sexp	変更されたサブシステムに対するこれまでの変更の数	サブシステムについてよく知っている人物は不具合を混入しにくい.	
entrophy	変更が多くのファイルに渡って行われているほど大きくなる値	ファイルに跨って分散している多くの変更を探し出す必要があるため，高いエントロピーを持つ変更は不具合が多くなりやすい.	分散した変更は不具合を混入しやすい [15, 16].
fix	変更が不具合を修正したかどうか (TRUE / FALSE)	不具合修正は初期実装における欠陥を意味するため，その近辺で欠陥が存在する可能性が高い.	不具合を修正した変更は新機能を実装した変更よりも不具合を混入しやすい [17,18].

が知られているものの，手法間ので正解データの違いについては検証が進んでいない.

本研究では次に示す研究設問（Research Question）を設け，検証を行う.

- **RQ1: Commit Guru と SZZ アルゴリズムにおいて，それぞれの不具合混入コミット推定の結果にはどのような特徴が存在するか.**
- **RQ2: Commit Guru と SZZ アルゴリズムにおいて，不具合混入コミット推定性能の差はどの程度か.**

3.2 対象プロジェクト

本実験では，表 3 に示すプロジェクトを対象に分析を行った.

3.3 準備

使用したバグトラッキングシステムの不具合データを表 4，Commit Guru より得られた対象プロジェクトのダンプデータの詳細を表 5 に示す.

以降において，指定したコミット群を表 6，表 7 のように呼称する．また，(BUG_{szz}, $CLEAN_{szz}$)，(BUG_{guru}, $CLEAN_{guru}$) のペアを，それぞれ CF_{szz}，CF_{guru} と呼称す

表 3. 対象プロジェクト

Project	機能	リポジトリ URL
Log4j	Java のロギングツール	https://github.com/apache/log4j
OpenJPA	Java Persistance API 仕様のオープンソース実装	https://github.com/apache/openjpa.git
Camel	Java ベースのオープンソース統合フレームワーク	https://github.com/apache/camel
HBase	列指向データベース管理システム	https://github.com/apache/hbase.git

表 4. 使用した不具合データ

Project	Bug-tracking System	Resloved	Resolution	Issue Type	status	# of bugs
Log4j	Bugzilla	≤ 2018-01-22	FIXED	–	RESOLVED, VERIFIED, CLOSED	305
OpenJPA		≤ 2018-01-17				1162
Camel	JIRA	≤ 2018-01-22	–	bug	closed	1457
HBase		≤ 2018-01-29				5466

表 5. 得られたダンプデータ

Project	# of Total Data	Buggy rate	Date dumped
Log4j	3275	45.6%	2018/01/17
OpenJPA	4864	11.1%	2018/01/10
Camel	30739	20.7%	2018/01/10
HBase	14745	31.9%	2018/01/23

表 6. 推定結果に基づくコミット群 (1 手法)

Method	Classify as	
	Buggy	Clean
SZZ	BUG_{szz}	$CLEAN_{szz}$
Commit Guru	BUG_{guru}	$CLEAN_{guru}$

表 7. 推定結果に基づくコミット群 (2 手法)

Method		Commit Guru	
	Classify as	Buggy	Clean
SZZ	Buggy		$DIFF_{szz}$
	Clean	$DIFF_{guru}$	$BOTH_{clean}$

表 8. SZZ アルゴリズムから得られた結果

Project	BUG tags (commits)	Buggy rate (Weighting)
Log4j	531 (296)	9.0% (16.2%)
OpenJPA	3156 (1301)	26.8% (64.9%)
Camel	4473 (2171)	7.1% (14.6%)
HBase	4257 (2408)	16.5% (28.9%)

る．

4 結果

SZZ アルゴリズムの実行結果を表 8 に示す．SZZ アルゴリズムは 1 つのコミットに対し複数の BUG タグを付けるため，Buggy rate として BUG タグが付けられたコミット数と，コミットについた BUG タグ数で重み付けしたコミット数の 2 つを算出した．

5 考察

5.1 RQ1: Commit Guru と SZZ アルゴリズムにおいて，それぞれの不具合混入コミット推定の結果にはどのような特徴が存在するか．

不具合コミットはなんらかの形で不具合を混入していないコミットとの間に差が生じていると考えられる．ゆえに，不具合コミット推定手法においても，推定された不具合コミットとそれ以外のコミットには差が生じていると思われる．ここでは，Commit Guru，および SZZ アルゴリズムの不具合コミットの推定結果，すなわち BUG_{szz}，BUG_{guru} において，それらを比較した際にどのような特徴が得られるかについて考察する．

各メトリクスの傾向 Commit Guru によって測定されたメトリクスは，表 2 において示した通り，そのコミットが不具合コミットとなるリスクの減少要因または増加要因と考えられている．ここでは，CF_{szz}，CF_{guru} において，Clean とされたコミットの中央値に対する Buggy なコミットの大小関係を各メトリクスごとにオッズを用いて評価し，各メトリクスがリスクの減少要因，増加要因のどちらとして働く傾向があるかを分析した．

その結果，オッズそのものの値に差はあるものの，ほとんどのメトリクスに対し両手法を通じて同様の働きが得られ，それは Buggy と推定されたコミットに対し期待されるものであった．

表 9. 外れ値検出の結果

project		LOF	method isolationForest	oneClassSVM
Log4j	# of commits	29	33	42
	BUG_{guru}	0.40%	0.70%	0.64%
	BUG_{szz}	0.18%	0.34%	0.24%
	$DIFF_{guru}$	0.21%	0.37%	0.40%
	$DIFF_{szz}$	0.00%	0.00%	0.00%
OpenJPA	# of commits	41	49	50
	BUG_{guru}	0.29%	0.56%	0.29%
	BUG_{szz}	0.35%	0.66%	0.43%
	$DIFF_{guru}$	0.062%	0.041%	0.021%
	$DIFF_{szz}$	0.12%	0.14%	0.16%
Camel	# of commits	271	308	309
	BUG_{guru}	0.09%	0.27%	0.28%
	BUG_{szz}	0.07%	0.16%	0.19%
	$DIFF_{guru}$	0.06%	0.14%	0.12%
	$DIFF_{szz}$	0.04%	0.03%	0.03%
HBase	# of commits	128	148	151
	BUG_{guru}	0.27%	0.75%	0.49%
	BUG_{szz}	0.11%	0.38%	0.27%
	$DIFF_{guru}$	0.20%	0.38%	0.24%
	$DIFF_{szz}$	0.03%	0.01%	0.02%

外れ値 先述した通り，不具合コミットはなんらかの形で不具合を混入していないコミットとの間に差が生じていると考えられるため，1 クラス SVM (One Class SVM), アイソレーションフォレスト (Isolation Forest), LOF (Local Outlier Factor) の 3 手法を用いて外れ値検出を行った．

各コミット群において，検出された外れ値をどの程度含んでいるか，その結果を表 9 に示す．全手法において，BUG_{guru} の方が BUG_{szz} よりも外れ値と判断されたコミットを含んでいた．

分類カテゴリ 不具合コミットの分類カテゴリの観点から，Commit Guru と SZZ アルゴリズムはどのような傾向があるかを分析した．結果を表 10 に示す．

その結果，BUG_{szz} は'Merge' に分類されたコミットを含まなかったことに対し，BUG_{guru} はわずかではあるが，'Merge' に分類されたコミットを含んでいた．また，BUG_{szz} と BUG_{guru} には' Corrective' に分類されたコミットに対して顕著な差が見られた．$DIFF_{szz}$ において Feature Addition に分類されたコミットの割合が高くなっているプロジェクトも存在するが，そもそもの不具合コミット推定の数量が違うため，ここにおいて比較するには疑念の余地が残る．

5.2 RQ2: Commit Guru と SZZ アルゴリズムにおいて，不具合混入コミット推定性能の差はどの程度か.

全コミットは，表 7 のように分類される．不具合コミット推定性能の差を調べるため，ここでは手法間で推定結果が異なるコミット $DIFF_{guru}$ および $DIFF_{szz}$ の比較を行った．また，それとともに Clean と推定されたコミット群にどれだけ差がないかを分析するため，$BOTH_{clean}$ に対して，$CLEAN_{guru}$, $CLEAN_{szz}$ の比較を行った．

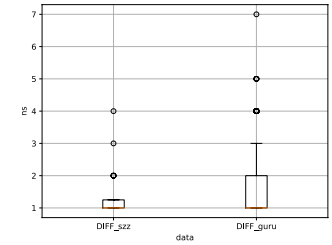
メトリクスの傾向 $DIFF_{guru}$，$DIFF_{szz}$ ともに，RQ1 より判断された Buggy と推定されたコミットに期待される傾向に対して，おおよそ沿った傾向を持っていた．また，Buggy と推定されたコミットに期待される傾向により近い傾向をもつコミット群は $DIFF_{szz}$ であった．

有意差 $BOTH_{clean}$ に対して，$CLEAN_{guru}$, $CLEAN_{szz}$ それぞれをマン・ホイットニーの U 検定を用いて比較した結果，Log4j, Camel, HBase は $CLEAN_{guru}$, OpenJPA は $CLEAN_{szz}$ の方がより多くのメトリクスにおいて有意差が存在しなかった．

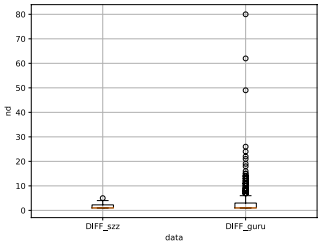
外れ値 全手法において，外れ値と推定されたコミットをより含んでいるのは Log4j, Camel, HBase においては $DIFF_{guru}$，OpenJPA においては $DIFF_{szz}$ であった．

また，箱ひげ図を用いて $DIFF_{szz}$，$DIFF_{guru}$ における各メトリクスごとの値の分布を評価したとき（図 1），$DIFF_{guru}$ の方が外れ値が明らかに多く，第一四分位点，第三四分位点の間が大きく開いていることがわかる．Buggy と推定されるコミットは，Clean と推定されるコミットと比較すると何らかの異常な特徴が現われると考えられる．そのため，比較的まとまりがある $DIFF_{szz}$ は $DIFF_{guru}$ よりも Buggy であるコミットが含まれている可能性は低いと考えられる．ここではページ数の関係より，コミット数が少ない Log4j のみを示したが，コミット数が比較的多い他のプロジェクトにおいても同様の結果が得られた．

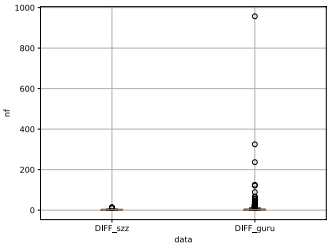
以上より，推定コミット数の多さにも関わらず，Buggy と推定されたコミットに期待される傾向におおよそ沿っており，外れ値をより含んでいる $DIFF_{szz}$ の方が不具合コミット推定結果において優れていると考えられる．また，Clean と推定されたコミット間にほとんどのプロジェクトにおいて有意差が存在しなかったことから，Clean と推定されたコミット群はある程度似通っていると考えられる．このため，その中に含んでいる Buggy と推定されるコミット数は少ないと考えられる．以上より，不具合コミットである可能性が高いものをより推定でき



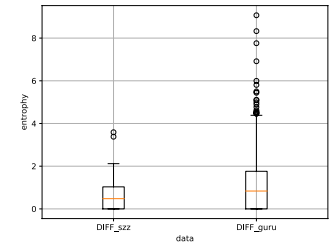
(a) ns



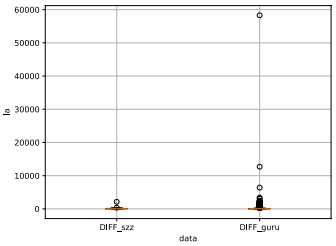
(b) nd



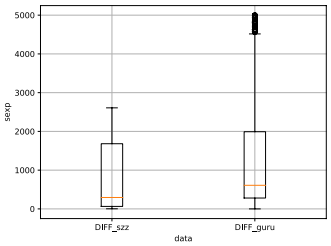
(c) nf



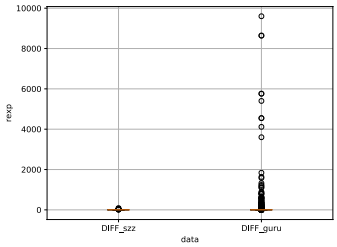
(d) entrophy



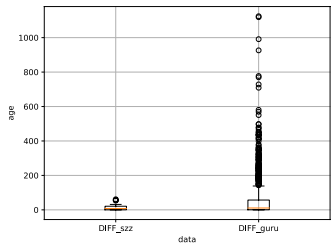
(e) la



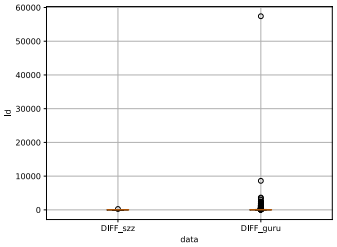
(f) sexp



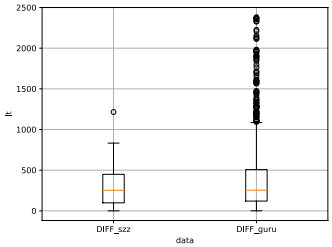
(g) rexp



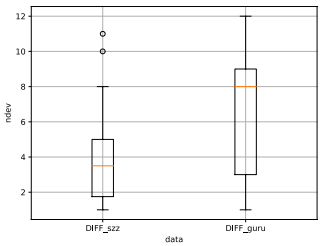
(h) age



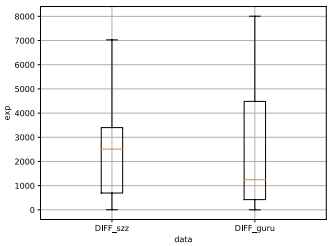
(i) ld



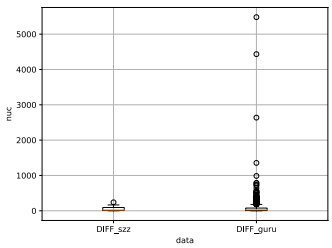
(j) lt



(k) ndev



(l) exp



(m) nuc

図 1. SZZ アルゴリズムと CommitGuru の間でバグ推定の結果が異なるものにおけるメトリクス値の分布の比較 (Log4j プロジェクト)

表 10. 各コミット群における分類カテゴリの割合

project	commits group	# of commits	None	Feature Addition	Corrective	Non Functional	Preventative	Perfective	Merge
Log4j	BUG_{guru}	1495	29.70%	23.08%	34.45%	6.76%	4.62%	1.40%	0.00%
	BUG_{szz}	297	28.62%	25.93%	30.64%	6.73%	7.41%	0.67%	0.00%
	$DIFF_{guru}$	1218	29.72%	22.82%	35.06%	6.90%	3.94%	1.56%	0.00%
	$DIFF_{szz}$	20	15.00%	50.00%	15.00%	15.00%	5.00%	0.00%	0.00%
OpneJPA	BUG_{guru}	540	58.89%	15.19%	14.63%	1.11%	7.96%	2.22%	0.00%
	BUG_{szz}	1301	62.18%	14.60%	11.84%	4.53%	5.30%	1.54%	0.00%
	$DIFF_{guru}$	216	50.93%	12.96%	23.15%	1.39%	10.65%	0.93%	0.00%
	$DIFF_{szz}$	977	61.51%	13.92%	12.79%	5.73%	5.02%	1.02%	0.00%
Camel	BUG_{guru}	6360	57.89%	19.47%	16.84%	1.18%	3.57%	1.02%	0.03%
	BUG_{szz}	2171	61.72%	20.68%	12.76%	1.01%	2.49%	1.34%	0.00%
	$DIFF_{guru}$	4970	57.32%	18.71%	17.85%	1.27%	3.86%	0.95%	0.04%
	$DIFF_{szz}$	781	64.92%	18.05%	11.91%	1.28%	2.43%	1.41%	0.00%
HBase	BUG_{guru}	4698	56.85%	15.11%	18.69%	1.60%	5.26%	2.38%	0.11%
	BUG_{szz}	2408	60.09%	15.49%	15.49%	2.12%	4.44%	2.37%	0.00%
	$DIFF_{guru}$	3072	54.33%	15.66%	20.21%	1.43%	5.86%	2.34%	0.16%
	$DIFF_{szz}$	782	56.91%	18.41%	14.83%	2.56%	5.12%	2.17%	0.00%

るのは Commit Guru であると考えられる．しかしながら，プロジェクトによって結果が非常に左右されており（OpenJPA プロジェクト），表 10 より Commit Guru は Merge と推定されるコミットを Buggy として推定している．このため，Commit Guru の情報を正解データとして利用するとしても，そのデータのみを十二分に信頼するのではなく，様々な手法を用いて検証を行った方が良いと考えられる．

6 結言

ソフトウェアの複雑さと重要性が日々増してきている昨今，ソフトウェアの品質予測は重要な研究テーマであり，その研究は活発化している．その中で，不具合の正解データとして用いられることの多い Commit Guru の不具合混入コミットの情報に対する信頼性は不確かなものであった．そこで，本研究では Commit Guru の不具合混入コミットの情報における正解データとしての信頼性検証のため，不具合混入コミット推定手法である SZZ アルゴリズムとの結果の整合性を比較し，考察を行なった．その結果，Commit Guru の不具合混入コミットの情報は不具合の正解データとして可能性を示すものの，その信頼性は十分であるとは言い難い．

参考文献

[1] G. Tassey, “The economic impacts of inadequate infrastructure for software testing,” Technical report, National Institute of Standards and Technology, 2002.

[2] E. Shihub, “An exploration of challenges limiting pragmatic software defect prediction,” PhD thesis, Queen’s University, 2012.

[3] J. Śliwerski, T. Zimmermann, and A. Zeller, “When do changes induce fixes?,” Proceedings of the 2005 International Workshop on Mining Software Repositories, pp.1–5, Proc. of 2nd International workshop on Mining software repositories, ACM, New York, NY, USA, 2005.

[4] C. Rosen, B. Grawi, and E. Shihab, “Commit guru: Analytics and risk prediction of software commits,” Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, pp.966–969, ESEC/FSE 2015, ACM, New York, NY, USA, 2015.

[5] A. Mockus and L.G. Votta, “Identifying reasons for software changes using historic databases,” Proceedings of the International Conference on Software Maintenance (ICSM’00), pp.120–130, ICSM ’00, IEEE Computer Society, Washington, DC, USA, 2000.

[6] Y. Kamei, E. Shihab, B. Adams, A.E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, “A large-scale empirical study of just-in-time quality assurance,” IEEE Trans. Softw. Eng., vol.39, no.6, pp.757–773, June 2013.

[7] A. Hindle, D.M. German, and R. Holt, “What do large commits tell us?: A taxonomical study of large commits,” Proceedings of the 2008 International Working Conference on Mining Software Repositories, pp.99–108, Proc. of 5th International workshop on Mining software repositories, ACM, New York, NY, USA, 2008.

[8] R. Moser, W. Pedrycz, and G. Succi, “A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction,” Proceedings of the 30th International Conference on Software Engineering, pp.181–190, Proc. of 30th International Conference on Software Engineering, ACM, New York, NY, USA, 2008.

[9] N. Nagappan and T. Ball, “Use of relative code churn measures to predict system defect density,” Proceedings of the 27th International Conference on Software Engineering, pp.284–292, ACM, New York, NY, USA, 2005.

[10] A.G. Koru, D. Zhang, K.E. Emam, and H. Liu, “An investigation into the functional form of the size-defect relationship for software modules,” IEEE Transactions on Software Engineering, vol.35, pp.293–304, 2009.

[11] A. Mockus and D.M. Weiss, “Predicting risk of software changes,” Bell Labs Technical Journal, vol.5, no.2, pp.169–180, aug 2002.

[12] N. Nagappan, T. Ball, and A. Zeller, “Mining metrics to predict component failures,” Proceedings of the 28th International Conference on Software Engineering, pp.452–461, Proc. of 28th International Conference on Software Engineering, ACM, New York, NY, USA, 2006.

[13] S. Matsumoto, Y. Kamei, A. Monden, K.-i. Matsumoto, and M. Nakamura, “An analysis of developer metrics for fault prediction,” Proceedings of the 6th International Conference on Predictive Models in Software Engineering, pp.18:1–18:9, PROMISE ’10, ACM, New York, NY, USA, 2010.

[14] T.L. Graves, A.F. Karr, J.S. Marron, and H.P. Siy, “Predicting fault incidence using software change history,” IEEE Trans. Software Eng., vol.26, pp.653–661, 2000.

[15] A.E. Hassan, “Predicting faults using the complexity of code changes,” Proceedings of the 31st International Conference on Software Engineering, pp.78–88, Proc. of 31st International Conference on Software Engineering, IEEE Computer Society, Washington, DC, USA, 2009.

[16] M. D’Ambros, M. Lanza, and R. Robbes, “An extensive comparison of bug prediction approaches,” 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), pp.31–41, 2010.

[17] P.J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, “Characterizing and predicting which bugs get fixed: An empirical study of microsoft windows,” Proceedings of the 32th International Conference on Software Engineering (ICSE), pp.495–504, ACM, May 2010.

[18] R. Purushothaman and D.E. Perry, “Toward understanding the rhetoric of small source code changes,” IEEE Transactions on Software Engineering, vol.31, pp.511–526, 2005.