

現実的な設定における メソッド粒度バグ予測モデルの構築および精度評価

荻野 翔^{1,a)} 肥後 芳樹^{1,b)} 楠本 真二^{1,c)}

受付日 2021年8月2日, 採録日 2022年1月11日

概要: バグを予測する技術は品質保証に要するコストを低減できると期待されている。バグ予測はより細粒度で行えることが望ましく、メソッド粒度でのバグ予測が注目されている。これまでに、機械学習を用いて構築されたメソッド粒度バグ予測モデルについての調査が何度か行われているが、それらの実験設定には非現実的な部分がある。たとえば、モデルの目的変数として「メソッドにバグが含まれているか」ではなく「メソッドに過去にバグが含まれていたか」が用いられていたり、バグ予測のユースケースにおいて利用不可能なデータに基づいて学習用データセットが構築されていたりする。著者らはそのような非現実的な部分のない、現実的な実験設定のもとで、メソッド粒度バグ予測モデルの予測精度を計測した。その結果、F 値は平均で約 0.197、AUC は平均で約 0.745 と計測され、高精度のメソッド粒度バグ予測モデルを構築するには課題が残されていると判明した。また、説明変数算出時に参照する変更履歴の期間および学習に用いるメソッドの存在期間について適切なものを採用することで、予測精度が F 値の観点で約 15.2%、AUC の観点で約 2.9%向上することが確認できた。

キーワード: 品質保証, バグ予測, 機械学習

Evaluating Method-level Bug Prediction under Realistic Settings

SHO OGINO^{1,a)} YOSHIKI HIGO^{1,b)} SHINJI KUSUMOTO^{1,c)}

Received: August 2, 2021, Accepted: January 11, 2022

Abstract: Bug prediction is a promising technique to reduce the cost of quality assurance. Method-level bug prediction is drawing attention because bug prediction on such finer granularity is desirable. Some research on method-level bug prediction using machine learning have been conducted, but their experimental settings are partly unrealistic. For example, some of them employ “whether a bug has been fixed in the past period” as a dependent variable, and some others build a dataset based on data that cannot be obtained realistically. Therefore, we first evaluate the capability of bug prediction models built under realistic settings. The experimented models resulted in low capability (F-measure of 0.197 and AUC of 0.745 on average). Thus, there are still some issues to be solved to build a high-capability bug prediction model under realistic settings. Next, we found it effective to change (a) interval of development history referenced when calculating process metrics, and (b) period of development history referenced when collecting method for training-dataset. The trial improves the F-measure score by 15.2% and does the AUC score by 2.9%.

Keywords: quality assurance, bug prediction, machine learning

1. はじめに

近年、ソフトウェア開発の規模は増大し続けている [1]。この状況において、開発コストを低減させるための技術は欠かせない。デバッグ等に要する品質管理コストは開発コストの中でも大きな割合を占める [2] ため、品質管理コス

¹ 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology,
Osaka University, Suita, Osaka 565-0971, Japan

a) s-ogino@ist.osaka-u.ac.jp

b) higo@ist.osaka-u.ac.jp

c) kusumoto@ist.osaka-u.ac.jp

トを低減するための技術は特に重要である。

品質管理に要するコストを低減できる技術の1つとして、バグ予測が存在する。バグ予測とは、ソフトウェアを構成するモジュール（例：ソースファイル）にバグが含まれるかどうかを予測する技術である。たとえば、ある時点 T において参照可能なデータに基づいてバグ予測モデルを構築し、同時点 T で存在するバグメソッドを特定するというユースケースが考えられる。そのようにバグを含むモジュールを予測し、それらを優先的にレビュー・テストすることは効率的な品質管理を可能にし、品質管理コストの低減につながる。また、バグ予測後のレビューに要するコストを考慮するとバグ予測はより細粒度で行えることが望ましく、メソッド粒度でのバグ予測が注目されている [3], [4], [5]。

メソッド粒度でバグを予測するモデルは、主に機械学習を利用してメソッドの特徴量とバグの有無との組（レコード）の集合（学習用データセット）からバグメソッドの特徴を学習することで構築される。そして、そのモデルがどの程度正しくバグを予測できるかは別のデータセット（評価用データセット）に基づいて評価される。従来、メソッド粒度バグ予測の調査においては下記のようにデータセットが構築され、モデルの評価が行われてきた [3], [4]。

- (1) ある時点 T で存在するメソッドについてレコードを算出する。
- (2) (1) で算出されたレコードに対して交差検証を行う。
具体的には、全データを無作為に 10 個の部分集合へ分割し、そのうちの 1 個を評価用データセット、残りの 9 個を学習用データセットに振り分けてモデルの評価を行う。そのような振り分けと評価を繰り返して予測精度の総合的な評価を行う。

そのように実験用データセットを構築しモデルを評価する場合、非現実的な実験結果につながると Pascarella らは主張した [5]。なぜなら、あるメソッドのバグの有無を時刻 T において予測する際に、同時刻 T における他のメソッドでのバグの有無を予測モデルの構築に利用しているからである。彼らはその問題を解決するためにリリースバイリリースという手法を提案し、その手法でデータセットを構築した場合の予測精度を調査した。調査の結果、リリースバイリリースを採用した場合の予測精度は低く、十分な予測精度を達成するには課題が残されていると結論付けられた。

しかしながら、彼らによって構築されたデータセットの妥当性には議論の余地があると著者らは考える。Pascarella らの手法では、目的変数として「そのモジュールにバグが存在するかどうか (isBuggy)」ではなく、「そのモジュールは過去にバグが修正されたかどうか (hasBeenFixed)」が採用されており、これが現実的な設定とはいえない。なぜなら、hasBeenFixed が偽でありながら isBuggy が真であ

るようなメソッドが存在するために、hasBeenFixed を目的変数とするモデル (hasBeenFixed モデル) は正しくバグの有無を予測できない可能性があるからである。

上記のように、メソッド粒度バグ予測についての先行研究では部分的に非現実的な設定が用いられており、現実的な設定に基づいた調査はいまだなされていない。よって、著者らはまず現実的な設定のもとでメソッド粒度バグ予測の予測精度を計測した。その結果、F 値は平均で約 0.197、AUC は平均で約 0.745 と計測され、高精度のメソッド粒度バグ予測モデルを構築するには課題が残されていると判明した。また、説明変数算出時に参照する変更履歴の期間および学習に用いるメソッドの存在期間について適切なものを採用することで、予測精度が F 値の観点で約 15.2%、AUC の観点で約 2.9% 向上することが確認できた。

2. 準備

本章では、本調査と関わりが深い、目的変数・バグ予測粒度・データセットの算出方法について述べる。

2.1 目的変数

メソッド粒度バグ予測の調査においては、予測モデルの目的変数として「そのメソッドに対し過去にバグ修正が行われたかどうか (hasBeenFixed [3], [5])」・「そのメソッドにバグが存在するかどうか (isBuggy [4])」という 2 種類のメトリクスが採用されてきた。それらの目的変数の定義を以下に示す。なお、図 1 はメソッドの開発履歴および、ある時点 T におけるそれらのメソッドについて isBuggy・hasBeenFixed を算出した結果を示す。

2.1.1 hasBeenFixed

ある時点 T で存在する各メソッドについて、hasBeenFixed は以下のように算出される。

- step1** プロジェクトについて、修正済みのバグレポートとリポジトリを取得する。
- step2** バグレポートはバグが発見されたときに発行される。それぞれのバグレポートには ID が付与されている。本実験では、先行研究 [4] にならってコミットメッセージにバグレポートの ID が記載されているコミットをバグ修正コミットとし、リポジトリ内のバグ修正コミットを特定する。図 1 に描かれた開発履歴の模式図においては、プロットされた緑色の点がバグ修正コミットを意味する。
- step3** あるメソッドに対するバグ修正コミットが T 以前に存在するとき、そのメソッドを T 時点で hasBeenFixed について真であると見なす。たとえば、図 1 におけるメソッド initialize に対するバグ修正コミットが T 以前に存在するため、initialize は時点 T で hasBeenFixed について真である。
- step4** 上記の操作がすべてのバグレポートについて完了

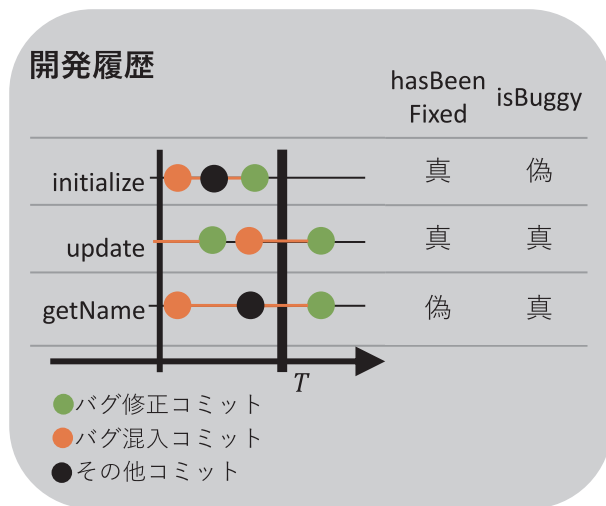


図 1 目的変数の算出例

Fig. 1 Calculations on dependent variables.

したとき、*hasBeenFixed* について真でないメソッドを *hasBeenFixed* について偽であると見なす。

2.1.2 isBuggy

ある時点 T で存在する各メソッドについて、*isBuggy* は SZZ アルゴリズム [6] を用いて以下のように算出される。SZZ アルゴリズムの実装としては Borg らによる実装を用いた [7]。

step1 *hasBeenFixed* 算出過程の step1 と同様である。

step2 *hasBeenFixed* 算出過程の step2 と同様である。

step3 メソッドのある行がバグ修正コミットにより変更されている場合、変更された行を挿入したコミットを特定し、そのコミットをバグ混入コミットと見なす。図 1 に描かれた開発履歴の模式図においては、プロットされた橙色の点がバグ混入コミットを意味する。

step4 メソッドに対するバグ修正コミットが T 以降に存在し、対応するバグ混入コミットが T 以前に存在する場合、そのメソッドを T 時点で *isBuggy* について真であると見なす。たとえば、図 1 におけるメソッド *update* に対するバグ修正コミットが T 以降に存在し、対応するバグ混入コミットが T 以前に存在するため、*update* は時点 T で *isBuggy* について真である。

step5 上記の操作がすべてのバグレポートについて完了したとき、*isBuggy* について真でないメソッドを *isBuggy* について偽であると見なす。

2.2 バグ予測粒度

バグ予測における予測粒度としては、ファイル粒度・メソッド粒度・コミット粒度の 3 つが存在する。ファイル粒度バグ予測・メソッド粒度バグ予測では、ある時点におけるファイル・メソッドにバグが存在するかを予測する。コミット粒度バグ予測では、個々のコミットに対して、バグを混入させているかどうかを予測する。コミット粒度バグ

予測には、下記の 2 つのメリットがある [8]。

- 細粒度での予測が可能である。
- 修正担当者の割当てが簡単である。

本研究では、下記の 3 つの理由からメソッド粒度でのバグ予測を対象とする。

- メソッド粒度バグ予測はファイル粒度バグ予測よりデバッグコストを低減できるとされている [4]。
- メソッド粒度予測にはある時点に存在するメソッド集合全体からバグメソッドを特定可能というコミット粒度予測に対する優位性が存在する。
- 本研究は、メソッド粒度バグ予測モデルの予測精度には課題が残るという、Pascarella らの結論の正しさを確かめることを目的の 1 つとしている。

2.3 実験用データセットの算出方法

メソッド粒度バグ予測モデルを構築し、予測精度を正しく評価するためには、学習用データセット・評価用データセットの組（実験用データセット）を実際のユースケースに基づいて構築する必要がある。本節では、従来手法とその問題点およびその問題点を解決できる手法であるリリースバイリリースについて述べる。

2.3.1 従来手法

従来のデータセット構築手法では、図 2 に示すように、ある時点に存在するメソッドについてその特徴量とバグの有無との組（レコード）を算出し、それらのレコードを学習用データセット・評価用データセットに振り分けていた [3], [4]。これでは、あるメソッドのバグの有無を時刻 T において予測する際に、同時刻 T における他のメソッドでのバグの有無を予測モデルの構築に利用していることになり、非現実的な実験結果につながると Pascarella らは主張している [5]。

2.3.2 リリースバイリリース

従来の実験用データセット構築手法の問題点を解決する手法として、リリースバイリリースと呼ばれる手法を Pascarella らは提案した。リリースバイリリースによりデータセットが構築される工程を図 3 と以下に示す。

- (1) n 番目のリリース (R_n) 時点で存在するメソッドについて、バグの有無を算出し、 R_n から R_{n-1} までの変更履歴を参照して特徴量を算出する（評価用レコード算出）。
- (2) R_{n-1} 時点で存在するメソッドについて、実際のユースケースを逸脱しないために R_n 時点で取得可能なデータのみを用いてバグの有無を算出し、 R_{n-1} から R_{n-2} までの変更履歴を参照して特徴量を算出する（学習用レコード算出）。
- (3) 評価用レコードを評価用データセットに振り分け、学習用レコードを学習用データセットに振り分ける。リリースバイリリースでは、予測時に取得できるデータ

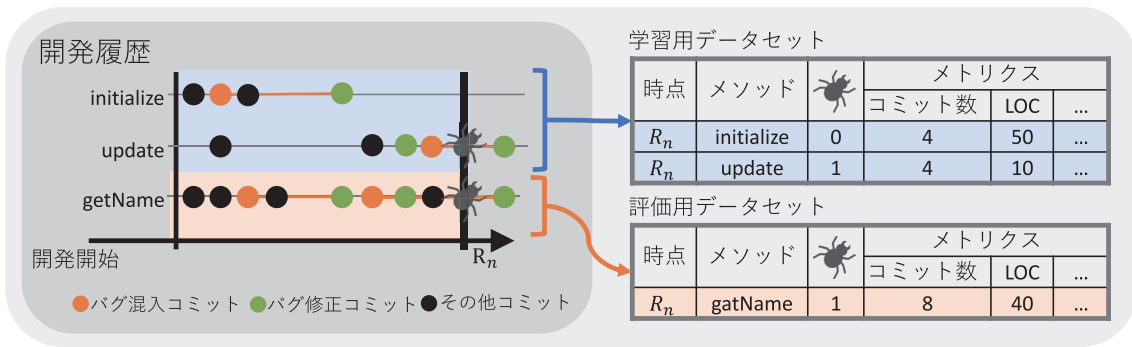


図 2 データセット構築手法（従来手法）の概要
Fig. 2 The traditional dataset-building method.

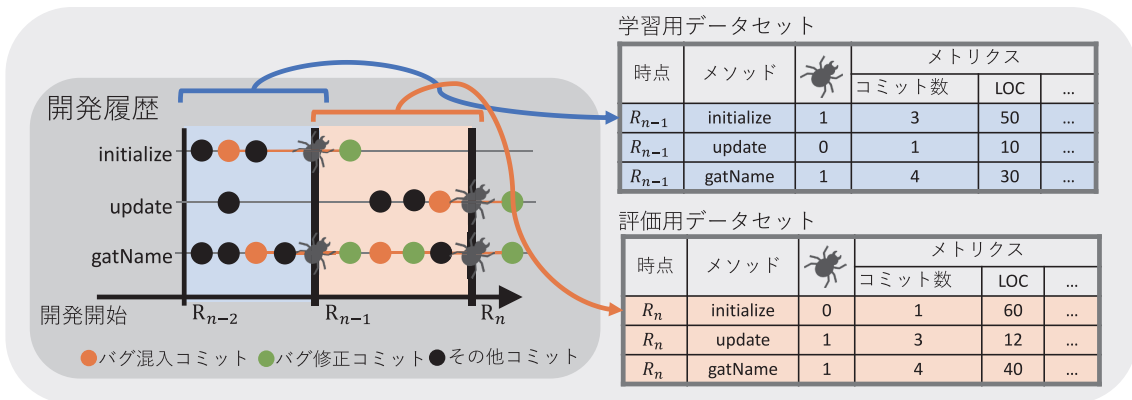


図 3 データセット構築手法（リリースバイリリース）の概要
Fig. 3 The novel dataset-building method (release-by-release).

のみを用いてバグ予測モデルを構築し、そのバグ予測モデルを用いて予測時に存在するメソッドについてバグの有無を判定しており、従来手法の問題は発生しない。

3. Research Questions

本研究では、下記の3つの Research Question (RQ) について調査した。

RQ1. *hasBeenFixed* を正確に予測できるモデルは *isBuggy* を正確に予測できるのか。

RQ1 の目的は、*hasBeenFixed* を目的変数として予測するモデルが *isBuggy* を正確に予測できないことを、すなわち *isBuggy* と *hasBeenFixed* との分布が異なることを確かめることである。これにより、下記のことが示せる。

- *hasBeenFixed* モデルについての結論 [5] は *isBuggy* モデルにはあてはまらない。
- 先行研究 [5] の RQ2 と本研究の RQ2 は差異が存在し、本研究の RQ2 を調査する意義は存在する。

RQ2. *isBuggy* を目的変数としたモデルは *hasBeenFixed* を目的変数としたモデルより予測精度が低いのか。

RQ2 の目的は、以下の2つである。

- 現実的な設定におけるメソッド粒度バグ予測モデルの予測精度を知ること。
- *isBuggy* モデルと *hasBeenFixed* モデルについて予測

精度を評価する対照実験を行い、現実的な設定のもとでバグを予測することの難しさを明らかにすること。

RQ3. 現実的な設定におけるメソッド粒度バグ予測において、(1) プロセスメトリクス算出時に参照する変更履歴の期間・(2) 学習に用いるメソッドの存在期間についての変更は予測精度改善に有効か。

RQ3 の目的は、現実的な設定のもとでメソッド粒度バグ予測の予測精度を改善することである。その方法の1つとして、リリースバイリリースがかかえる下記の2つの問題を解決することを考えた。

- (1) リリース間隔に一貫性がない場合、学習対象のバグメソッドと予測対象のバグメソッドとの間で変更履歴についての特徴が異なってしまう。その解決策は、メトリクス算出時に参照する変更履歴の期間として、ある時点から N コミット前まで・ある時点から N カ月前までといった固定長の期間を用いることである。
- (2) バグ予測モデルを構築するための学習データとして、予測時を基準として直近のリリース時点に存在するメソッドから算出されるレコードしか利用しておらず、それより過去のメソッドから算出できるレコードを利用していない。その解決策は、過去のメソッドから算出できるレコードも学習データとして利用することである。

表 1 対象プロジェクト
Table 1 Target projects.

プロジェクト名	開発期間	リリース数	コミット数	バグレポート数	メソッド数*	バグがあるメソッドの割合*
cassandra	4,529 日	3	26,172	5,582	15,893	8.0 %
egit	4,318 日	5	6,601	2,554	5,017	9.2 %
jgit	4,318 日	5	8,298	746	8,542	2.1 %
linuxtools	4,491 日	8	10,767	2,201	11,524	2.1 %
poi	7,116 日	3	10,881	2,673	19,408	16.3 %
realm-java	3,217 日	6	8,676	802	5,690	1.3 %
sonar-java	3,247 日	7	7,675	1,180	5,931	2.8 %
wicket	6,151 日	4	21,049	2,889	38,625	1.5 %

*各リリース時についての平均値

しかしながら、具体的にプロセスメトリクス算出時にどの程度の長さの期間を参照すればよいのか、どの程度古いメソッドを学習に用いればよいのかは不明である。よって、それら 2 つの設定について候補となる値をあげ、それぞれの組合せ（設定パターン）を採用した場合について予測精度を評価し、精度が最も高くなる設定パターンを特定する。また、そのとき従来手法と比較して予測精度がどの程度改善されたかを評価する。

4. 実験設定

本章では、各実験に共通する設定について述べる。

4.1 対象プロジェクト

本研究における対象プロジェクトは、下記の 4 つの条件を満たすプロジェクトから無作為に選択された 8 つのプロジェクトである。その概要を表 1 に示す。

- 開発履歴（Git リポジトリ）を取得可能。
- プログラミング言語として Java を用いている。
- セマンティックバージョンングを採用している。
- メジャーバージョンの 3 回以上連続したリリースが確認できる。

4.2 リリース

リリースバイリリースという手法を用いて、学習用データセット・評価用データセットを構築する。そのためには、プロジェクトについてリリース時点を特定する必要がある。我々は、Pascarella らによって提案された手法を用いて、セマンティックバージョンングを採用しているプロジェクトについてリリースを特定する。セマンティックバージョンングとは、プロジェクトのバージョンを X.Y.Z（例：1.2.1）の形式で表現する手法である。我々はセマンティックバージョンングにおけるメジャーバージョン（Y および Z が 0 であるリリース）をリリースとして利用する。

4.3 目的変数

本調査で評価実験を行う予測モデルの目的変数は has-

表 2 説明変数（コードメトリクス）

Table 2 Independent variables (code metrics).

メトリクス名	概要
FanIn	そのメソッドを参照するメソッドの数
FanOut	そのメソッドが参照するメソッドの数
LocalVar	ローカル変数の数
Parameters	引数の数
CommentRatio	ソースコードに対するコメントの割合（行単位）
CountPath	実行可能経路の数
Complexity	サイクロマティック数
execStmt	実行可能ステートメントの数
maxNesting	ネスト数の最大値

BeenFixed および isBuggy である。それぞれの定義については、準備の章で述べたため省略する。

4.4 説明変数

我々は Giger らにより定義されたコードメトリクス・プロセスメトリクス [3] を説明変数として採用する。つまり、各メソッドに対して算出されるこれらのメトリクスに基づいて、そのメソッドのバグの有無を予測する。それぞれの概要を表 2、表 3 に示す。

4.5 学習アルゴリズム

学習アルゴリズムとしては、多数の先行研究で採用されており [3], [4], [5], [9], 比較的高い精度で予測できている *Random Forest* (RF) [10] を採用した。その実装としては、scikit-learn [11] による実装を用いた。

4.6 ハイパーパラメータの最適化

ハイパーパラメータの最適化（パラメータチューニング）は機械学習を用いて構築されるモデルの精度向上に有効である [12]。本研究では、交差検証を取り入れて下記のようにハイパーパラメータチューニングを行った。

- (1) 学習用データセット S を構成するレコードを 5 等分することで、学習用データセットのサブセット S_1, S_2, S_3, S_4, S_5 を得る。

- (2) ハイパーパラメータ最適化アルゴリズムの一種であるベイズ最適化 [13] の実装である optuna [14] によって、評価対象のハイパーパラメータ H が与えられる。なお、探索対象のハイパーパラメータとその探索範囲は表 4 に示す。
- (3) 学習用データセットのサブセットの 1 つをバリデーション用データセット V として選び、残りをハイパーパラメータチューニングにおける学習用データセット S' とする。このとき S' , V の組合せは合計 5 パターン

表 3 説明変数 (プロセスメトリクス)

Table 3 Independent variables (process metrics).

メトリクス名	概要
MethodHistories	コミット回数
Authors	そのメソッドを編集した人数
StmtAdded	追加されたステートメントの総数
MaxStmtAdded	追加されたステートメントの最大値
AvgStmtAdded	追加されたステートメントの平均値
StmtDeleted	取り除かれたステートメントの総数
MaxStmtDeleted	取り除かれたステートメントの最大値
AvgStmtDeleted	取り除かれたステートメントの平均値
Churn	StmtAdded - StmtDeleted
MaxChurn	Churn の最大値
AvgChurn	Churn の平均値
Decl	メソッド宣言の変更回数
Cond	条件文の変更回数
ElseAdded	else 文の追加回数
ElseDeleted	else 文の削除回数

表 4 ハイパーパラメータと探索範囲

Table 4 Target hyperparameters and the range of values.

パラメータ	探索範囲
RF モデルを構成する決定木の数	2~256
決定木の深さの最大値	2~256
決定木の葉ノード数の最大値	2~256
葉ノードを構成するサンプルの最小数	2~256
ノードを構成するサンプルの最小数	2~256

- 存在する。それぞれの S' , V の組合せについて、 H に基づいてモデルの構築・予測精度の計測を行い、各組合せでの予測精度の平均値を H に対する評価とする。
- (4) (2), (3) を AMD Ryzen 9 3950X を搭載したコンピュータを利用して 10 時間が経過するまで繰り返し、評価が最も高いハイパーパラメータを最適なハイパーパラメータとして採用する。

4.7 評価指標

バグ予測モデルを評価する指標を以下に示す。

$$\text{再現率} = \frac{TP}{FN + TP}$$

$$\text{適合率} = \frac{TP}{FP + TP}$$

$$F \text{ 値} = \frac{2 \times \text{再現率} \times \text{適合率}}{\text{再現率} + \text{適合率}}$$

$$ROC-AUC = ROC \text{ 曲線の下側の面積}$$

TP はバグが存在すると予想され、実際にバグが存在したメソッドの個数である。 FN はバグが存在しないと予想され、実際にはバグが存在したメソッドの個数である。 FP はバグが存在すると予想され、実際にはバグが存在しなかったメソッドの個数である。ROC 曲線は、縦軸を TP 、横軸を FP とするグラフである。

5. 実験

5.1 RQ1

RQ1 では、hasBeenFixed を目的変数として予測するモデルが isBuggy を正確に予測できないことを、すなわち isBuggy と hasBeenFixed との分布が異なることを確かめる。実験工程を図 4 と以下に示す。

- (1) 対象プロジェクトの各リリース時点に存在するメソッドについて、isBuggy および hasBeenFixed を算出する。
- (2) isBuggy に対する hasBeenFixed のカバレッジの値を計測する。計測されたカバレッジの値が 0.5 以下であ

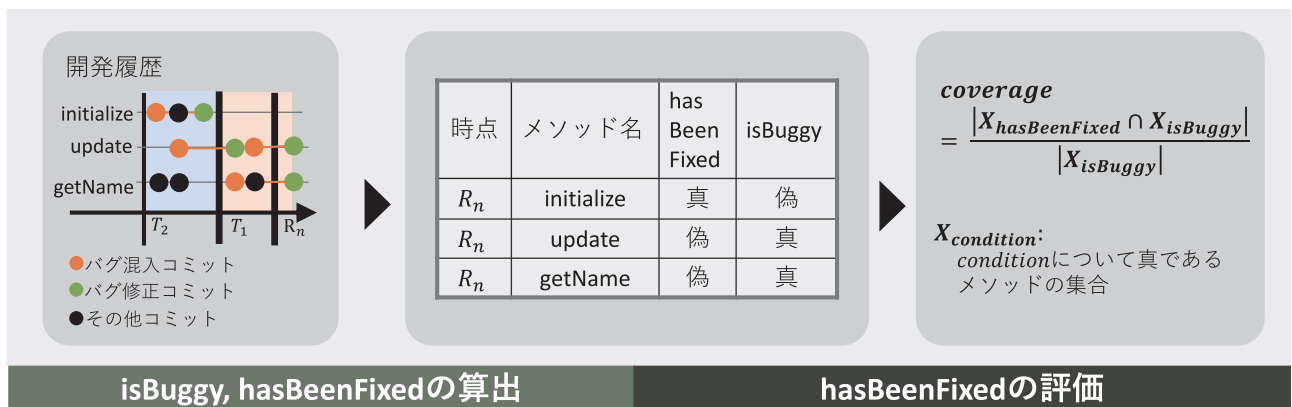


図 4 実験の概要 (RQ1)

Fig. 4 Summary of experiment (RQ1).

れば, *hasBeenFixed* の分布と *isBuggy* の分布は異なり, *hasBeenFixed* を目的変数として予測するモデルが *isBuggy* を正確に予測できないことが分かる.

5.1.1 評価指標

我々は, *hasBeenFixed* の分布と *isBuggy* の分布が異なるかを調査するための評価指標として, 以下にカバレッジを定義する.

$$\text{カバレッジ} = \frac{|X_{hasBeenFixed} \cap X_{isBuggy}|}{|X_{isBuggy}|}$$

$X_{hasBeenFixed}$ は, *hasBeenFixed* について真であるメソッドの集合である. $X_{isBuggy}$ は *isBuggy* について真であるメソッドの集合である. たとえば, 図 4 にあるメソッド *initialize* は R_n 時点で *hasBeenFixed* について真であり, *isBuggy* について偽であるため, *initialize* は R_n 時点で $X_{hasBeenFixed}$ の要素であり, $X_{isBuggy}$ の要素ではない.

5.1.2 実験結果

表 5 は *hasBeenFixed* がバグメソッドを捕捉できた割合であるカバレッジを対象プロジェクトの各リリース時点について算出した結果を示す. 全 38 件のリリース中, 35 件でカバレッジの値は 0.5 を下回っており, 平均的にカバレッジの値は 0.5 を大きく下回っている. よって, *hasBeenFixed* の分布と *isBuggy* の分布は異なり, *hasBeenFixed* を目的変数として予測するモデルが *isBuggy* を正確に予測できないことが確認できた.

5.2 RQ2

RQ2 では, 各対象プロジェクトの各リリースについて *hasBeenFixed* モデルおよび *isBuggy* モデルを構築し, それらの予測精度を比較する. 実験工程を図 5 と以下に示す.

表 5 対象プロジェクトの各リリース時点についてのカバレッジ

Table 5 Coverage on each target project & release.

プロジェクト	リリース	バグメソッド数	カバレッジ
cassandra	R1	803	0.375
cassandra	R2	1239	0.337
cassandra	R3	1715	0.232
egit	R1	221	0.656
egit	R2	254	0.350
egit	R3	532	0.265
egit	R4	658	0.612
egit	R5	734	0.451
jgit	R1	72	0.181
jgit	R2	140	0.143
jgit	R3	171	0.111
jgit	R4	248	0.109
jgit	R5	269	0.093
linuxtools	R1	446	0.058
linuxtools	R2	272	0.121
linuxtools	R3	209	0.115
linuxtools	R4	273	0.205
linuxtools	R5	239	0.427
linuxtools	R6	142	0.169
linuxtools	R7	100	0.210
linuxtools	R8	2	1.000
poi	R3	2584	0.091
poi	R4	2500	0.322
realm-java	R1	74	0.243
realm-java	R2	82	0.134
realm-java	R3	248	0.020
realm-java	R4	36	0.083
realm-java	R5	22	0.000
realm-java	R6	6	0.000
sonar-java	R1	16	0.000
sonar-java	R2	53	0.057
sonar-java	R3	151	0.139
sonar-java	R4	394	0.142
sonar-java	R5	194	0.253
sonar-java	R6	194	0.180
wicket	R7	370	0.359
wicket	R8	184	0.196
wicket	R9	32	0.188
平均値		15879	0.254

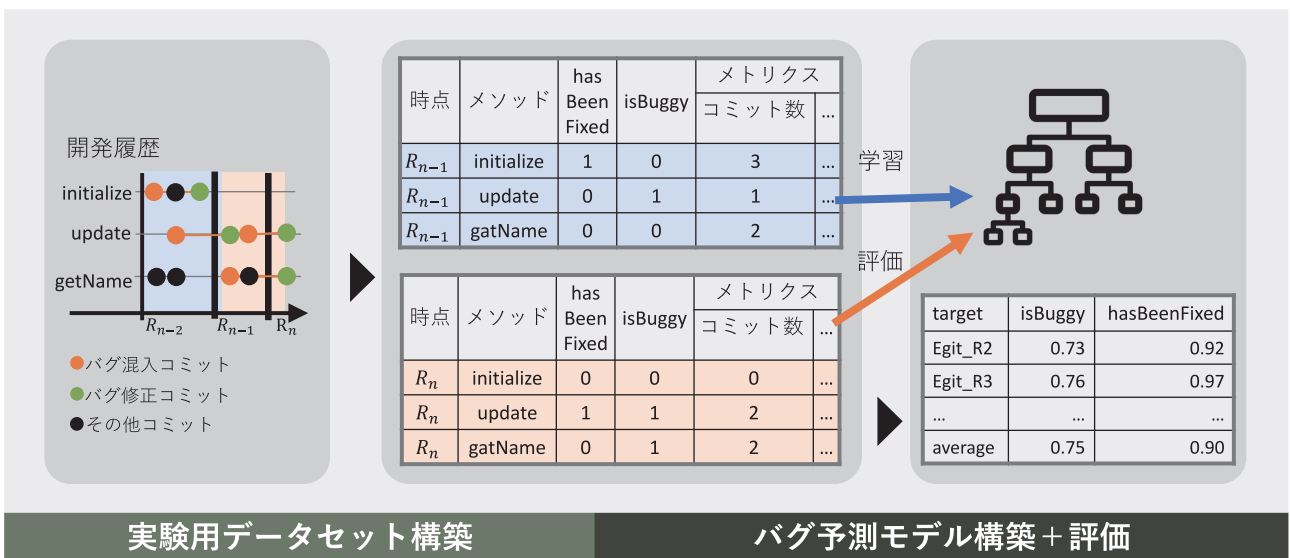


図 5 実験の概要 (RQ2)

Fig. 5 Summary of experiment (RQ2).

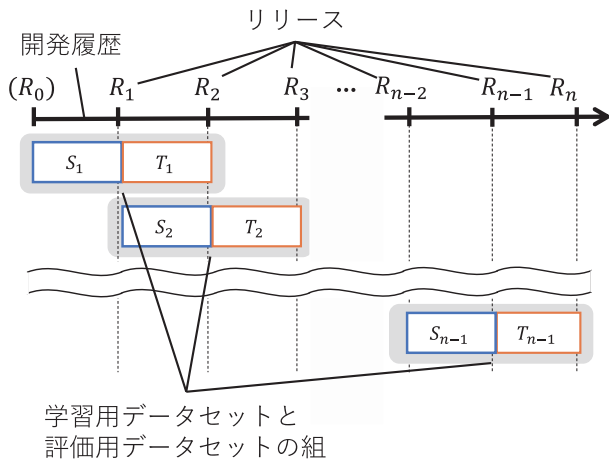


図 6 n 個のリリースが存在する場合に算出される実験用データセット

Fig. 6 Calculated datasets on the project that has n releases.

- (1) 比較対象である目的変数の *hasBeenFixed*・*isBuggy* のそれぞれについて、以下の工程を行う。
 - (a) その目的変数を採用した場合の実験用データセットを対象プロジェクトの各リリースごとに構築する。データセット構築手法は 5.2.1 項で述べる。
 - (b) 実験用データセットごとにバグ予測モデルを構築し、予測精度を評価する。
- (2) 各目的変数を採用した場合の予測精度 (平均値) を比較評価する。

5.2.1 データセット構築

リリースバイリリースに従って学習用データセットおよび評価用データセットを構築する。図 6 に示すように、プロジェクトの R_n について、学習用データセット S_{n-1} と評価用データセット T_{n-1} との組 (実験用データセット D_{n-1}) が算出される。プロジェクトの R_n についての実験用データセット D_{n-1} は下記のように算出される。

- (1) R_n 時点で存在する各メソッドについて、説明変数と目的変数の組 (レコード) を下記のように算出し、評価用データセット T_{n-1} に振り分ける。
 - プロセスメトリクスを、 R_{n-1} から R_n までの開発期間を参照して算出する。
 - プロダクトメトリクスを、 R_n 時点のソースコードを参照して算出する。
 - バグの有無を、実験時に参照可能な開発期間すべてを参照して算出する。
- (2) R_{n-1} 時点で存在する各メソッドについて、レコードを下記のように算出し、学習用データセット S_{n-1} に振り分ける。なお、ランダムフォレストは多重共線性に対し頑健であるため、多重共線性対策は行わない [15]。
 - プロセスメトリクスを、 R_{n-2} から R_{n-1} までの開発期間を参照して算出する。
 - プロダクトメトリクスを、 R_{n-1} 時点のソースコード

を参照して算出する。

- バグの有無を、開発開始時から R_n までの開発期間を参照して算出する。
- (3) 目的変数の値についてレコード数に大きく偏りがある学習用データセットから構築されたモデルは、予測対象を多数派の目的変数の値へしか分類しないという問題が存在する (class-imbalance problem)。たとえば、目的変数である *isBuggy* について偽であるレコードが多数派のデータセットに基づいてモデルを構築した場合、そのモデルへどのような説明変数を入力しても *isBuggy* について偽であると分類されてしまう。この問題を回避するために、学習用データセット S_{n-1} に含まれる *isBuggy* について真であるレコードの件数と *isBuggy* について偽であるレコードが同じ件数になるまで、*isBuggy* について真であるレコードを無作為に復元抽出する [16]。

5.2.2 実験結果

対象プロジェクトの各リリース時点について、*isBuggy* モデル・*hasBeenFixed* モデルをそれぞれ構築した。予測精度を評価した結果を表 6 に示す。

isBuggy モデルの F 値は平均で約 0.197、AUC は平均で約 0.745 と低い値が計測された。よって、現実的な設定のもとで高精度のメソッド粒度バグ予測モデルを構築するには課題が残るといえる。

また、*isBuggy* モデルの F 値 (平均値) は *hasBeenFixed* モデルより約 53.2% 低く、AUC については約 17.6% 低い。よって、非現実的な設定と比較して、現実的な設定のもとでバグを予測することは難しいといえる。

5.3 RQ3

RQ3 では、データセット構築手法をリリースバイリリースから部分的に変更した場合の予測精度の変化を調査する。具体的には、プロセスメトリクス算出時に参照する変更履歴の期間 (interval)・学習に用いるメソッドの存在期間 (period) を変更した場合の予測精度を評価する。RQ3 の実験工程を図 7 と以下に示す。

- (1) interval の値と period の値の組合せ (設定パターン) について、以下の操作を行う。
 - (a) その設定パターンを採用した場合の実験用データセットを対象プロジェクトの各リリースごとに構築する。データセット構築手法は 5.3.3 項で述べる。
 - (b) 実験用データセットごとにモデルの構築・精度評価を行い、その設定パターンを採用した場合の予測精度の平均値を算出する。
- (2) 各設定パターンを採用した場合の予測精度 (平均値) を比較し、精度が最も高くなる設定パターンを特定する。また、そのときリリースバイリリースと比較して

表 6 対象プロジェクトの各リリースについての予測結果
Table 6 Prediction capability on each target project & release.

プロジェクト名	学習→評価	isBuggy モデル				hasBeenFixed モデル			
		適合率	再現率	F 値	AUC	適合率	再現率	F 値	AUC
cassandra	R1 → R2	0.207	0.634	0.312	0.762	0.319	0.740	0.446	0.887
cassandra	R2 → R3	0.178	0.718	0.285	0.802	0.259	0.905	0.403	0.930
egit	R1 → R2	0.465	0.640	0.538	0.784	0.160	0.451	0.237	0.491
egit	R2 → R3	0.502	0.557	0.528	0.763	0.661	0.968	0.786	0.969
egit	R3 → R4	0.389	0.658	0.489	0.792	0.845	1.000	0.916	0.956
egit	R4 → R5	0.233	0.598	0.336	0.757	0.697	0.928	0.796	0.927
jgit	R1 → R2	0.189	0.516	0.276	0.703	0.193	0.588	0.291	0.92
jgit	R2 → R3	0.155	0.566	0.243	0.719	0.067	0.965	0.125	0.916
jgit	R3 → R4	0.104	0.559	0.175	0.678	0.153	0.764	0.255	0.943
jgit	R4 → R5	0.045	0.730	0.085	0.671	0.036	0.957	0.070	0.781
linuxtools	R1 → R2	0.058	0.604	0.105	0.716	0.146	0.806	0.247	0.807
linuxtools	R2 → R3	0.043	0.407	0.077	0.685	0.151	0.226	0.181	0.849
linuxtools	R3 → R4	0.085	0.594	0.149	0.659	0.396	0.523	0.451	0.920
linuxtools	R4 → R5	0.102	0.773	0.181	0.817	0.310	0.791	0.445	0.950
linuxtools	R5 → R6	0.082	0.502	0.141	0.729	0.290	0.965	0.446	0.983
linuxtools	R6 → R7	0.018	0.337	0.033	0.670	0.153	0.938	0.264	0.976
poi	R3 → R4	0.107	0.678	0.185	0.607	0.353	0.933	0.512	0.769
realm-java	R1 → R2	0.108	0.460	0.175	0.718	0.534	0.348	0.422	0.792
realm-java	R2 → R3	0.042	0.218	0.070	0.657	0.097	0.500	0.163	0.832
realm-java	R3 → R4	0.021	0.366	0.039	0.618	0.081	0.359	0.132	0.756
realm-java	R4 → R5	0.010	0.684	0.020	0.640	0.126	0.472	0.199	0.949
realm-java	R5 → R6	0.002	0.500	0.004	0.582	0.022	0.087	0.035	0.735
sonar-java	R1 → R2	0.075	0.535	0.132	0.695	0.500	0.145	0.225	0.570
sonar-java	R2 → R3	0.128	0.656	0.214	0.752	0.088	0.544	0.152	0.817
sonar-java	R3 → R4	0.116	0.523	0.190	0.731	0.080	0.820	0.146	0.850
sonar-java	R4 → R5	0.075	0.736	0.136	0.812	0.477	0.673	0.559	0.951
sonar-java	R5 → R6	0.044	0.886	0.084	0.802	0.147	0.980	0.256	0.946
wicket	R7 → R8	0.032	0.503	0.061	0.696	0.296	0.586	0.393	0.939
wicket	R8 → R9	0.007	0.656	0.014	0.768	0.127	0.970	0.224	0.952
平均値		0.117	0.63	0.197	0.745	0.280	0.850	0.421	0.904

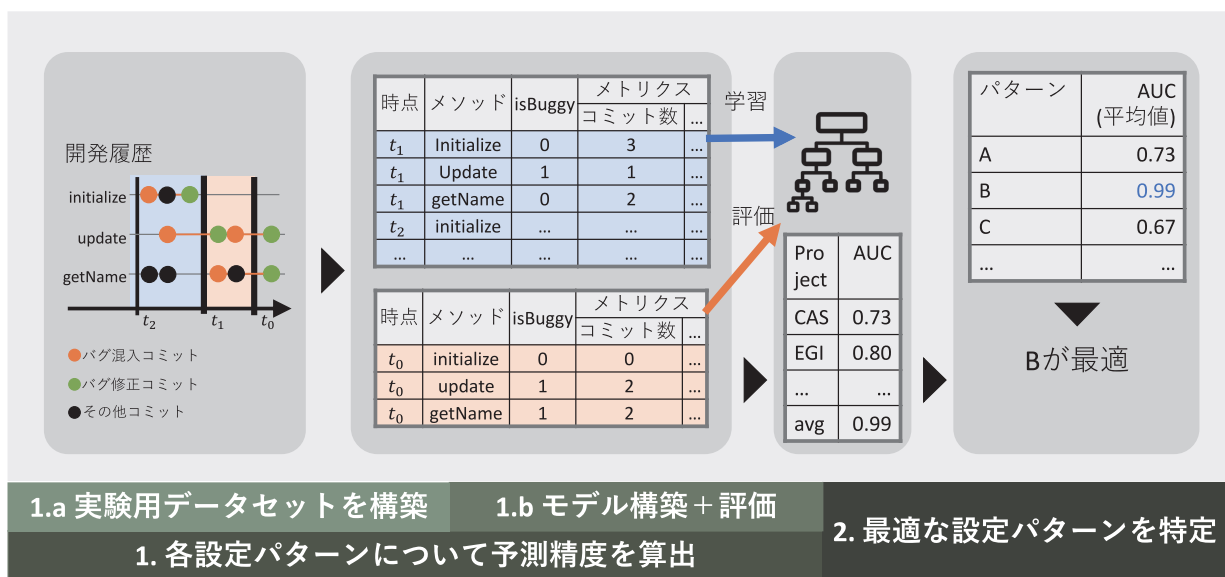


図 7 実験の概要 (RQ3)
Fig. 7 Summary of the experiment (RQ3).

予測精度がどの程度改善されたかを評価する。

5.3.1 プロセスメトリクス算出時に参照する変更履歴の期間 (interval)

リリースバイリリースでは、あるリリースからその直前のリリースまでの変更履歴を参照して、メソッドについてのレコードを算出する。このとき、リリース間隔に一貫性がなければ、学習対象のバグメソッドの特徴と予測対象のバグメソッドの特徴が異なり、予測精度が悪くなると考えられる。一方で、ある時点から N カ月前まで、ある時点から N コミット前までといった固定長の期間を参照する場合、そのような問題は起こらないと思われる。しかしながら、どの程度の期間を参照すればよいのかは不明であるため、本実験では下記の 10 パターンについて調査する。各パターンを採用した場合の具体的なデータセット算出工程は 5.3.3 項で述べる。

- N コミット (ただし、 N の候補値は 500, 1,000, 1,500, 2,000, 2,500 である.)
- N カ月 (ただし、 N の候補値は 1, 2, 3, 6, 12 である.)

5.3.2 学習に用いるメソッドの存在期間 (period)

リリースバイリリースは、 R_n 時点で存在するメソッドを評価用とした場合、学習用に用いるメソッドは直前のリリースである R_{n-1} 時点で存在するメソッドのみである。リリースバイリリースのように直前のバグメソッドだけではなく、昔のバグメソッドについての特徴も学習することで予測精度が向上する可能性がある。

しかしながら、どの程度古いメソッドを学習に用いればよいのかは不明であるため、本実験では下記の 5 パターンについて調査する。なお、ある時点で存在するメソッドについて算出されたレコードの集合をデータブロックと定義し、各パターンを採用した場合の具体的なデータセット算出過程は 5.3.3 項で述べる。

- $\text{last}(N_1/5)$: N_2 個存在するデータブロックのうち、最新の $N_2 * (N_1/5)$ 個 (ただし、小数点以下の数値は切り上げる。 N_1 の候補値は 1, 2, 3, 4, 5 である)。

5.3.3 実験用データセット構築

本実験では、対象とする interval · period · プロジェクト · リリース R_n に基づいて、以下の手順で学習用データセット S_n と評価用データセット T_n との組 (実験用データセット D_n) を構築する。

(1) 対象プロジェクトから、以下の条件を満たすリリースを特定し、それを評価対象リリース R_n とおく。

- 過去に 1 回以上のリリースを経ている。
- 過去に 5,000 回以上のコミットを経ている。
- 開発開始から 24 カ月以上経過している。

(2) interval に従ってデータブロックを算出する。算出例を図 8 に示す。

(a) R_n 時点をとく。 t_0 から interval だけ前の時点 t_1 を特定し、 t_1 から interval だけ前の時点 t_2 を

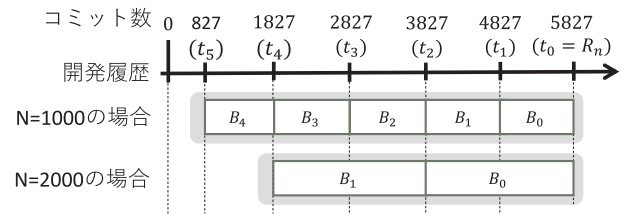


図 8 算出されるデータブロックの例 (interval = N コミット)

Fig. 8 Datablock calculation (interval = N commits).

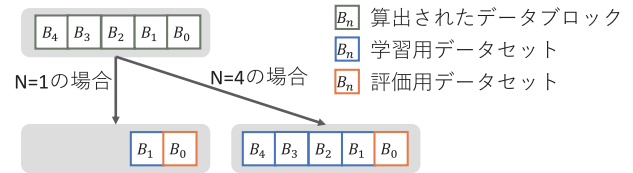


図 9 データブロックの振り分け (period = $\text{last}(N/5)$)

Fig. 9 Datablock distribution (period = $\text{last}(N/5)$).

特定する、のように再帰的に参照時点を特定する。

(b) 参照時点 t_k で存在するメソッドについて、下記のようにメトリクスとバグの有無との組 (レコード) を算出し、それらのレコードをデータブロック B_k とする。

- プロセスメトリクスを、 t_{k+1} から t_k までの開発期間を参照して算出する。
- プロダクトメトリクスを、 t_k 時点のソースコードを参照して算出する。
- 参照時点 t_0 で存在するメソッドについては、実験時に参照可能な開発期間すべてを参照してバグの有無を算出する。それ以外のメソッドについては、開発開始時から t_0 までの開発期間を参照してバグの有無を算出する。

(3) 算出されたデータブロックを、period に従って評価用データセット T_n · 学習用データセット S_n に振り分ける。振り分けの例を図 9 に示す。

(a) データブロック B_0 を評価用データセット T_n に振り分ける。

(b) $\text{last}(N_1/5)$ が採用されている場合、 B_0 以外のデータブロック N_2 個のうち最新の $N_2 * (N_1/5)$ 個 (小数点以下切り上げ) を学習用データセット S_n に振り分ける。たとえば、データブロックとして B_1, B_2, B_3, B_4 が存在し、 $\text{last}(1/5)$ が採用されている場合、最新の $4 * (1/5) = 0.8 \approx 1$ 個である B_1 のみが学習用データセットに振り分けられ、 $\text{last}(4/5)$ が採用されている場合、最新の $4 * (4/5) = 3.2 \approx 4$ 個であるすべてのデータブロックが学習用データセットに振り分けられる。なお、ランダムフォレストは多重共線性に対し頑健であるため、多重共線性対策は行わない [15]。

(4) 目的変数の値についてレコード数に大きく偏りがある

学習用データセットから構築されたモデルは、予測対象を多数派の目的変数の値へしか分類しないという問題が存在する (class-imbalance problem). たとえば、目的変数である *isBuggy* について偽であるレコードが多数派のデータセットに基づいてモデルを構築した場合、そのモデルへどのような説明変数を入力しても *isBuggy* について偽であると分類されてしまう. この問題を回避するために、学習用データセット S_{n-1} に含まれる *isBuggy* について真であるレコードの件数と *isBuggy* について偽であるレコードが同じ件数になるまで、 S_{n-1} に含まれる *isBuggy* について真であるレコードから無作為に復元抽出する [16].

5.3.4 実験結果

各設定パターンに基づいて、バグ予測モデルを構築した場合の予測精度の平均値を表 7 に示す. 最も予測精度 (F 値・AUC) が高くなるのは interval として 12 カ月を、period として last(5/5) を採用したときであった. このとき、F 値は約 0.191 であり、リリースバイリリースを用いた場合と比較して約 15.2%の精度向上が見られた. また、AUC は約 0.773 であり、約 2.9%の精度改善が見られた.

6. 妥当性の脅威

6.1 内的妥当性

6.1.1 メトリクス算出アルゴリズムの妥当性

説明変数であるコードメトリクス・プロセスメトリクスおよび目的変数である *isBuggy*・*hasBeenFixed* を算出するツールは、調査にあたって一から実装した. Giger らによるメトリクスの説明 [3] に従って実装したものの、正確な値が算出できていない可能性がある.

また、目的変数の算出については、2つの問題がある.

- (1) *isBuggy* は SZZ アルゴリズムを用いて算出されており、SZZ アルゴリズムには精度の面で改善の余地がある [17], [18]. SZZ アルゴリズムはバグ修正に基づいて、そのバグが混入した時点を特定する. 過去の調査では、その特定精度が約 77%であった [17].
- (2) バグが顕在化していないメソッドを、バグが存在しないメソッドと見なしてしまっている. 本研究では、実際に行われたバグ修正に基づいて、バグメソッドを特定している. よって、バグが潜在しているメソッドが存在し、それらのメソッドの特徴をバグが存在しないメソッドの特徴としてモデルが学習してしまっている恐れがある.

6.1.2 パラメータチューニングの不足

パラメータチューニングは機械学習を用いて構築されたモデルの精度向上に有効である [12]. 本調査では各モデルの構築時に 10 時間のパラメータチューニングを行ったが、より多くの時間を費やせば、より高精度のバグ予測モデルが構築される可能性がある.

表 7 採用した設定パターンに対する予測精度 (AUC について降順)
Table 7 Prediction capability on each setting (descending order on AUC).

interval	period	適合率	再現率	F 値	AUC
12 カ月	last(5/5)	0.114	0.596	0.191	0.773
6 カ月	last(1/5)	0.101	0.666	0.175	0.772
6 カ月	last(3/5)	0.108	0.631	0.184	0.770
12 カ月	last(4/5)	0.113	0.598	0.190	0.766
6 カ月	last(2/5)	0.105	0.623	0.180	0.765
6 カ月	last(4/5)	0.103	0.621	0.176	0.764
12 カ月	last(2/5)	0.107	0.599	0.182	0.763
2000 コミット	last(5/5)	0.104	0.600	0.177	0.762
12 カ月	last(3/5)	0.107	0.603	0.181	0.762
6 カ月	last(5/5)	0.110	0.610	0.187	0.761
2000 コミット	last(3/5)	0.105	0.578	0.178	0.758
3 カ月	last(4/5)	0.104	0.635	0.178	0.758
2000 コミット	last(4/5)	0.106	0.588	0.180	0.757
1500 コミット	last(5/5)	0.102	0.618	0.175	0.756
1000 コミット	last(3/5)	0.099	0.624	0.171	0.755
2000 コミット	last(1/5)	0.100	0.636	0.172	0.755
2 カ月	last(4/5)	0.099	0.640	0.171	0.755
1 カ月	last(1/5)	0.101	0.645	0.174	0.754
3 カ月	last(5/5)	0.100	0.636	0.172	0.754
2000 コミット	last(2/5)	0.099	0.606	0.171	0.753
2500 コミット	last(4/5)	0.101	0.590	0.173	0.753
2 カ月	last(2/5)	0.102	0.598	0.174	0.753
1000 コミット	last(5/5)	0.099	0.615	0.171	0.752
2500 コミット	last(2/5)	0.101	0.596	0.173	0.752
3 カ月	last(2/5)	0.098	0.616	0.169	0.752
2 カ月	last(3/5)	0.101	0.632	0.175	0.751
1500 コミット	last(4/5)	0.103	0.603	0.176	0.750
2500 コミット	last(3/5)	0.101	0.581	0.172	0.750
2500 コミット	last(5/5)	0.105	0.577	0.177	0.750
リリースバイリリース		0.092	0.652	0.162	0.750
3 カ月	last(3/5)	0.104	0.593	0.177	0.750
1 カ月	last(3/5)	0.099	0.624	0.170	0.748
2 カ月	last(5/5)	0.098	0.631	0.170	0.748
1000 コミット	last(4/5)	0.100	0.617	0.171	0.747
500 コミット	last(3/5)	0.101	0.599	0.172	0.747
1000 コミット	last(2/5)	0.098	0.603	0.169	0.746
1500 コミット	last(2/5)	0.098	0.612	0.169	0.746
1500 コミット	last(3/5)	0.102	0.600	0.175	0.746
2500 コミット	last(1/5)	0.098	0.605	0.169	0.746
1 カ月	last(4/5)	0.096	0.633	0.167	0.746
1 カ月	last(5/5)	0.099	0.618	0.170	0.746
1500 コミット	last(1/5)	0.097	0.616	0.168	0.745
500 コミット	last(2/5)	0.098	0.599	0.168	0.744
12 カ月	last(1/5)	0.103	0.592	0.176	0.744
2 カ月	last(1/5)	0.095	0.628	0.165	0.744
1000 コミット	last(1/5)	0.100	0.602	0.172	0.743
3 カ月	last(1/5)	0.094	0.598	0.163	0.743
500 コミット	last(4/5)	0.099	0.595	0.170	0.741
1 カ月	last(2/5)	0.095	0.619	0.164	0.741
500 コミット	last(5/5)	0.099	0.591	0.169	0.740
500 コミット	last(1/5)	0.090	0.592	0.157	0.728

6.2 外的妥当性

6.2.1 対象言語

本調査では Java で開発されているプロジェクトを対象としており、他の言語で記述されたプロジェクトに対しては、本調査で得られた知見があてはまらない可能性がある。

6.2.2 対象プロジェクト

本調査では 4.1 節に記載された条件を満たす、比較的大規模な OSS プロジェクトのみを実験対象としており、そのほかのプロジェクトに関しては本調査で得られた知見があてはまらない可能性がある。

7. おわりに

本研究が最終目的として見据えているのは、バグ予測の実用化であり、本研究はその第一歩である。

本研究では、まず先行研究でバグ予測モデルの目的変数として採用された *hasBeenFixed* の妥当性を調査した。その結果、*hasBeenFixed* が補足できたバグメソッドの割合は 0.5 を大きく下回っているため、*hasBeenFixed* はバグ予測モデルの目的変数として妥当ではないと判明した。次に、現実的な設定のもとでメソッド粒度バグ予測を評価した。その結果、予測精度は F 値が平均で約 0.197、AUC は平均で約 0.745 とかなり低く、現実的な設定のもとでバグ予測を行うには課題が残されていると判明した。また、説明変数算出時に参照する変更履歴の期間および学習に用いるメソッドの存在期間について適切なものを採用することで、予測精度が F 値の観点で約 15.2%、AUC の観点で約 2.9% 向上することが確認できた。

現在、我々は現実的な設定に基づいたメソッド粒度バグ予測の予測精度を改善する方法を模索している。特に、ヒューリスティックに基づいて算出されたメトリクスを説明変数として利用するのではなく、深層学習を用いてソースコードやコミット履歴から統計的に抽出される特徴量を説明変数として利用することを考えている。

謝辞 本研究は JSPS 科研費 JP20H04166, JP21K18302 および JP21K11829 の助成を得て行われた。

参考文献

- [1] 情報処理推進機構：ソフトウェア開発分析データ集 (2020)。入手先 (<https://www.ipa.go.jp/files/000085879.pdf>)。
- [2] Britton, T., Jeng, L., Carver, G., Cheak, P. and KKatzenellenbogen, T.: Increasing software development productivity with reversible debugging, available from (<https://undo.io/media/uploads/files/Undo-ReversibleDebugging-Whitepaper.pdf>)。
- [3] Giger, E., D'Ambros, M., Pinzger, M. and Gall, H.: Method-level bug prediction, *International Symposium on Empirical Software Engineering and Measurement*, pp.171–180 (2012)。
- [4] Hata, H., Mizuno, O. and Kikuno, T.: Bug prediction based on fine-grained module histories, *Proc. International Conference on Softw. Eng.*, pp.200–210 (2012)。
- [5] Pascarella, L., Palomba, F. and Bacchelli, A.: Re-evaluating method-level bug prediction, *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp.592–601 (2018)。
- [6] Sliwerski, J., Zimmermann, T. and Zeller, A.: When do changes induce fixes? *SIGSOFT Software Engineering Notes*, Vol.30, No.4, pp.1–5 (2005)。
- [7] Borg, M., Svensson, O., Berg, K. and Hansson, D.: Szz unleashed: An open implementation of the szz algorithm—featuring example usage in a study of just-in-time bug prediction for the jenkins project, *Proc. 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*, pp.7–12 (2019)。
- [8] Fukushima, T., Kamei, Y., McIntosh, S., Yamashita, K. and Ubayashi, N.: An empirical study of just-in-time defect prediction using cross-project models, *Proc. 11th Working Conference on Mining Software Repositories, MSR 2014*, pp.172–181, Association for Computing Machinery (2014)。
- [9] Lessmann, S., Baesens, B., Mues, C. and Pietsch, S.: Benchmarking classification models for software defect prediction: A proposed framework and novel findings, *IEEE Trans. Softw. Eng.*, Vol.34, No.4, pp.485–496 (2008)。
- [10] Liaw, A. and Wiener, M.: Classification and regression by randomforest, *R news*, Vol.2, pp.18–22 (2002)。
- [11] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E.: Scikit-learn: Machine learning in Python, *Journal of Machine Learning Research*, Vol.12, pp.2825–2830 (2011)。
- [12] 尾崎嘉彦, 野村将寛, 大西正輝：機械学習におけるハイパラメータ最適化手法：概要と特徴, 電子情報通信学会論文誌 D, Vol.103, No.9, pp.615–631 (2020)。
- [13] Moćkus, J.: On bayesian methods for seeking the extremum, *Optimization Techniques IFIP Technical Conference Novosibirsk*, pp.400–404 (1975)。
- [14] Akiba, T., Sano, S., Yanase, T., Ohta, T. and Koyama, M.: Optuna: A next-generation hyperparameter optimization framework, *Proc. 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp.2623–2631 (2019)。
- [15] Dormann, C.F., Elith, J., Bacher, S., Buchmann, C., Carl, G., Carré, G., García Marquéz, J.R., Gruber, B., Lafourcade, B., Leitão, P.J., Münkemüller, T., McClean, C., Osborne, P.E., Reineking, B., Schröder, B., Skidmore, A.K., Zurell, D. and Lautenbach, S.: Collinearity: a review of methods to deal with it and a simulation study evaluating their performance, *Ecography*, Vol.36, No.1, pp.27–46 (2013)。
- [16] Chawla, N.V.: *Data Mining and Knowledge Discovery Handbook*, pp.875–886, Springer (2010)。
- [17] Williams, C. and Spacco, J.: Szz revisited: Verifying when changes induce fixes, *Proc. 2008 Workshop on Defects in Large Software Systems*, pp.32–36 (2008)。
- [18] da Costa, D.A., McIntosh, S., Shang, W., Kulesza, U., Coelho, R. and Hassan, A.E.: A framework for evaluating the results of the szz approach for identifying bug-introducing changes, *IEEE Trans. Softw. Eng.*, Vol.43, No.7, pp.641–657 (2017)。



萩野 翔

令和 2 年大阪大学基礎工学部情報科学学科卒業。現在、同大学大学院情報科学研究科博士前期課程在学中。ソフトウェアリポジトリマイニングに関する研究に従事。



肥後 芳樹 (正会員)

2002 年大阪大学基礎工学部情報科学科中退。2006 年同大学大学院博士後期課程修了。2007 年同大学院情報科学研究科コンピュータサイエンス専攻助教。2015 年同准教授。博士 (情報科学)。ソースコード分析、特にコード

ドクローン分析、リファクタリング支援、ソフトウェアリポジトリマイニングおよび自動プログラム修正に関する研究に従事。電子情報通信学会、日本ソフトウェア科学会、IEEE 各会員。



楠本 真二 (正会員)

1988 年大阪大学基礎工学部卒業。1991 年同大学大学院博士課程中退。同年同大学基礎工学部助手。1996 年同講師。1999 年同助教。2002 年同大学大学院情報科学研究科助教授。2005 年同教授。博士 (工学)。ソフト

ウェアの生産性や品質の定量的評価に関する研究に従事。電子情報通信学会、IEEE、JFPUG 各会員。