修士学位論文

題目

ユースケースに基づくメソッド粒度バグ予測

指導教員

楠本 真二 教授

報告者

荻野 翔

令和 4 年 2 月 2 日

大阪大学 大学院情報科学研究科

コンピュータサイエンス専攻

令和 3 年度 修士学位論文

ユースケースに基づくメソッド粒度バグ予測

荻野 翔

内容梗概

バグを予測する技術はソフトウェア開発における品質保証に要するコストを低減できる．バグ予測はより細粒度で行えることが望ましく，メソッド粒度バグ予測についても研究がなされている．しかしながら，先行研究の一部は過去にバグ修正されたことがあるかどうかを予測対象にしており，他の先行研究はユースケースにおいて参照不可能なデータに基づいて予測モデルを構築している．つまり，ある時点で参照可能なデータを用いて予測モデルを構築し同時点で存在するメソッドについてバグの有無を予測するという，ユースケースに基づいた評価は未だなされていない．よって，本研究ではバグ予測の実用化を目標として，ユースケースに基づいたメソッド粒度バグ予測の評価を試みた．まず，ユースケースに基づいた実験設定のもとで予測精度を計測した結果，F 値は平均で約 0.197，AUC は平均で約 0.745 と計測され，メソッド粒度バグ予測の実用化には課題が残されていると判明した．次に，メソッド粒度バグ予測の予測精度を改善する手法を提案し，その有効性を評価した．1 つ目に，説明変数算出時に参照する変更履歴の期間，学習に用いるメソッドの存在期間を変更し，その有効性を評価した．その結果，予測精度が F 値の観点で約 15.2%，AUC の観点で約 2.9% 向上し，有効性が確認できた．2 つ目に，説明変数にメトリクスではなく時系列データを利用し，その有効性を評価した．その結果，F 値は 4.2% 低下し，AUC は 1.4% 向上し，有効性は確認できなかった．

主な用語

品質保証, バグ予測, 機械学習

目次

図目次

表目次

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，開発コストを低減す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．例えば，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 で存在するバグモジュール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 [3–5].

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，

これではユースケースにおける予測精度を評価できない．なぜなら，バグ予測のユースケースにおける予測対象は isBuggy であり，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 isBuggy を正確に予測出来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の先行研究はユースケースに基づいてメソッド粒度バグ予測を評価できているとは言えない．よって，本研究ではバグ予測の実用化を目標として，ユースケースに基づいたメソッド粒度バグ予測の評価を試みた．まず，ユースケースを反映した実験設定のもとで予測精度を計測した結果，F 値は平均で約 0.197，AUC は平均で約 0.745 と計測され，メソッド粒度バグ予測の実用化には課題が残されていると判明した．次に，メソッド粒度バグ予測の予測精度を改善しうる方法について，その有効性を評価した．1 つ目に，説明変数算出時に参照する変更履歴の期間，学習に用いるメソッドの存在期間を変更し，その有効性を評価した．その結果，予測精度が F 値の観点で約 15.2%，AUC の観点で約 2.9% 向上し，有効性が確認できた．2 つ目に，説明変数にメトリクスではなく時系列データを利用し，その有効性を評価した．その結果，F 値は 4.2% 低下し，AUC は 1.4% 向上し，有効性は確認できなかった．

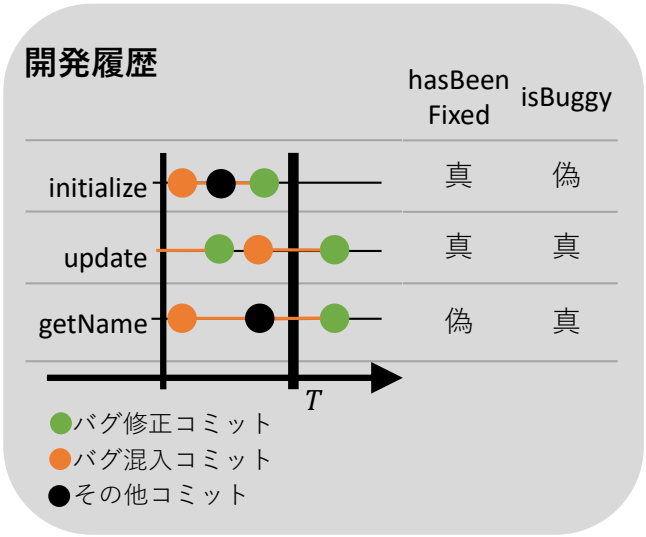

図 1　目的変数の算出例

2　準備

本章では，本研究と関わりが深い 5 つの項目について述べる．

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及び，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時に発行される．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] に倣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みなす．例えば，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　上記の操作が全てのバグレポートについて完了したとき，*hasBeenFixed* について真でないメソッドを *hasBeenFixed* について偽であるとみなす．

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みなす．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みなす．例えば，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　上記の操作が全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みなす．

2.2　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 [8]．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．

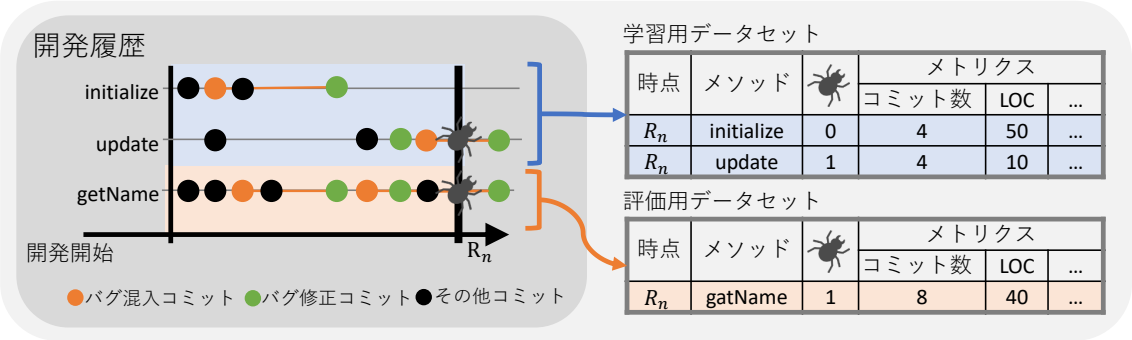

図 2　データセット構築手法 (従来手法) の概要

- 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一つとしている．

2.3　データセット構築手法

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及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. プロジェクトのリリース時点を特定する．具体的には，プロジェクトのバージョンを X.Y.Z(例: 1.2.1) の形式で表現するセマンティックバージョニングにおけるメジャーバージョン（Y および

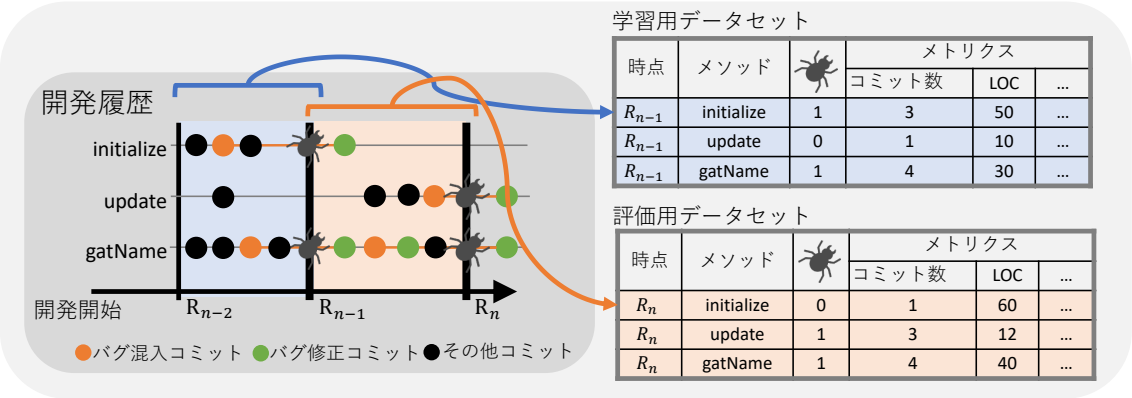

図 3　データセット構築手法 (リリースバイリリース) の概要

Z が 0 であるリリース）をリリースとして利用する．

2. 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} までの変更履を参照して特徴量を算出する (評価用レコード算出)．
3. 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)．
4. 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．

2.4　評価指標

バグ予測モデルの予測精度を評価する指標を以下に示す．

$$\text{再現率} = \frac{TP}{FN + TP}$$
$$\text{適合率} = \frac{TP}{FP + TP}$$
$$\text{F 値} = \frac{2 \times \text{再現率} \times \text{適合率}}{\text{再現率} + \text{適合率}}$$
$$\text{ROC-AUC} = \text{ROC 曲線の下側の面積}$$

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，縦軸をバグメソッドのうち検出できた割合 (True Positive Rate(FPR))，横軸をバグのないメソッドのうち誤検出した割合 (False Positive Rate(FPR)) とするグラフである．

2.5 精度目標

本研究では，バグ予測技術が実用的であると判断する最低限の基準を ROC-AUC=0.78 と設定し，これを精度目標とする．以下にその根拠を述べる．

ソフトウェア開発において，品質保証に費やせるコストは有限であり，かつプロジェクトごとに様々である [1]．よって，品質保証に費やせるコストに応じて，バグ予測に対する下記のスタンスが存在すると考えられる．

- 品質保証に多量のコストを費やせるため，多少の誤検出を許してでも，大半のバグを検出したい．
  このスタンスを満足させるには，少なくとも「FPR が 0.5 の時，TPR が 0.9 以上」を満たす必要があるとする．なぜなら，9 割以上のバグを検出できるとしても，バグのないメソッドの半分近くを誤検知することを許してしまうと後のデバッグ作業が非常に困難なためである．
- 品質保証に少量のコストしか費やせないため，多少のバグの見逃しを許してでも，誤検出を減らしたい．
  このスタンスを満足させるには，少なくとも「FPR が 0.1 の時，TPR が 0.5 以上」を満たす必要があるとする．なぜなら，バグのないメソッドの 1 割程度を誤検知することを許してしまうと後のデバッグ作業が困難であり，かつ本条件は半分近くものバグメソッドを見逃すことを許容しているためである．

実用的なバグ予測モデルは上記のスタンスを満足させるべきである．上記のスタンスを満足させるモデルの ROC 曲線は図 4 のようになり，この ROC 曲線の AUC は 0.78 である．よって，これをバグ予測技術が実用的であると判断する最低限の基準とし，当面の精度目標とする．

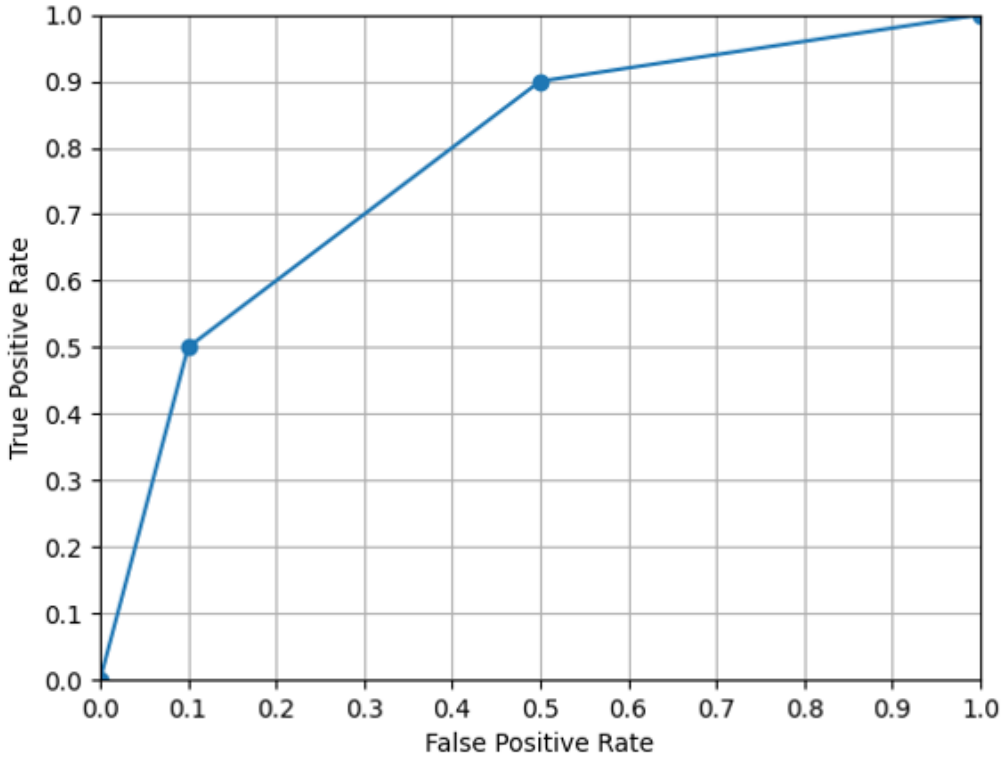

図 4　品質管理に対する 2 つのスタンスを満足させる ROC 曲線

3　Research Questions

本研究では，下記の 4 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確かめることである．それにより，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か.

RQ2 の目的は，以下の 2 つである.

- メソッド粒度バグ予測モデルのユースケースにおける予測精度を知ること.
- 目的変数の両者についてモデルの予測精度を比較し，バグ予測の難しさを明らかにすること.

RQ3. 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の変更は予測精度改善に有効か.

RQ3 の目的は，リリースバイリリースが抱える下記の 2 つの問題を解決することで，メソッド粒度バグ予測の予測精度が改善されるかを調査することである.

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 ヶ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.

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挙げ，それぞれの組み合わせ (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したかを評価する.

RQ4. メソッド粒度バグ予測において，説明変数としての時系列データの利用は精度改善に有効か.

ファイル粒度バグ予測において，メトリクスではなく時系列データを説明変数に利用することで予測精度が改善した例が多数報告されている [9–11]．当該手法はメソッド粒度バグ予測においても有効であると予想されるが，調査は未だなされていない．よって，RQ4 ではメソッド粒度バグ予測における当該手法の有効性を調査する.

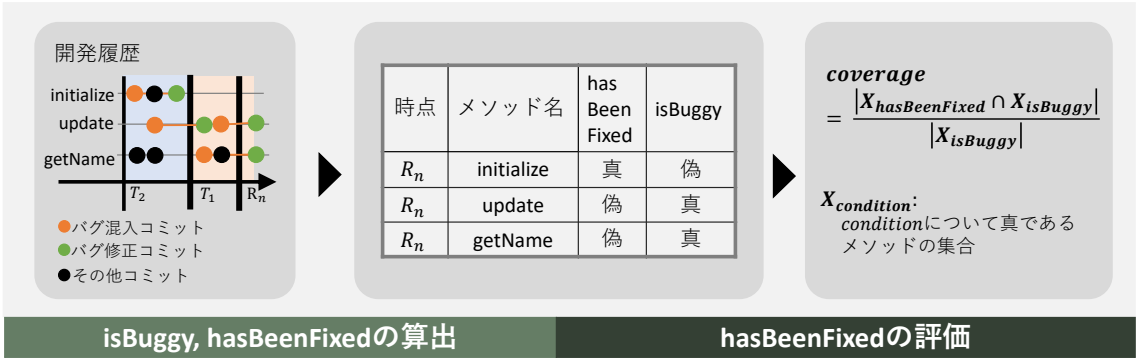

図 5　実験の概要 (RQ1)

4　RQ1

RQ1 では，*isBuggy* と *hasBeenFixed* との分布が異なることを，つまり 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を確かめる．実験工程を図 5 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* 及び *hasBeenFixed* を算出する．
2. *isBuggy*・*hasBeenFixed* の分布の重なりを評価する．その値が基準値以下であれば， *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とわかる．

4.1　実験設定

4.1.1　評価指標

hasBeenFixed の分布と *isBuggy* の分布の一致度を評価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．例えば，図 5 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．本調査では，カバレッジの計測値が 0.5 以下であれば，それぞれの分布が異なるとみなす．

4.1.2 対象プロジェクト

本調査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ニ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 実験結果

表 2 は *isBuggy*・*hasBeenFixed* の分布の一致度合い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．

表 1　対象プロジェクト

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

表 2　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

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

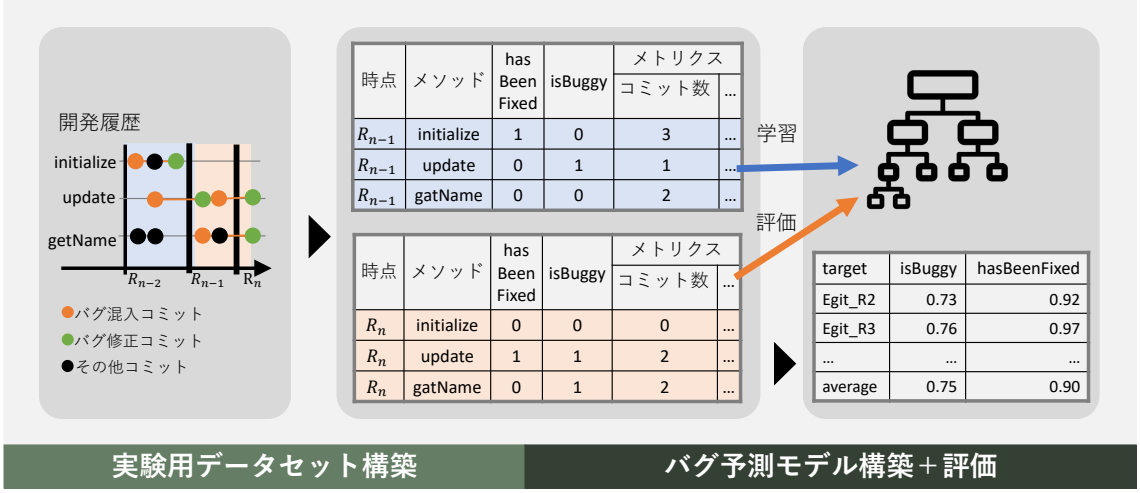

図 6　実験の概要 (RQ2)

5　RQ2

RQ2 では，各対象プロジェクトの各リリースについて *hasBeenFixed* モデル及び *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 6 と以下に示す．

1. 比較対象である，目的変数の *hasBeenFixed*・*isBuggy* について，以下の操作を行う．
   (a) 実験用データセットを対象プロジェクト・リリースごとに構築する．データセット構築手法は 5.1.3 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．

5.1　実験設定

5.1.1　目的変数

本調査における目的変数は isBuggy 及び hasBeenFixed である．説明は 2 章を参照のこと．

5.1.2　説明変数

本調査では 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 3，表 4 に示す．

5.1.3 実験用データセット

リリースバイリリースに従って学習用データセット及び評価用データセットを構築する．図 7 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

表 3　説明変数 (コードメトリクス)

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	コメント行数 / LOC
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

表 4　説明変数 (プロセスメトリクス)

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

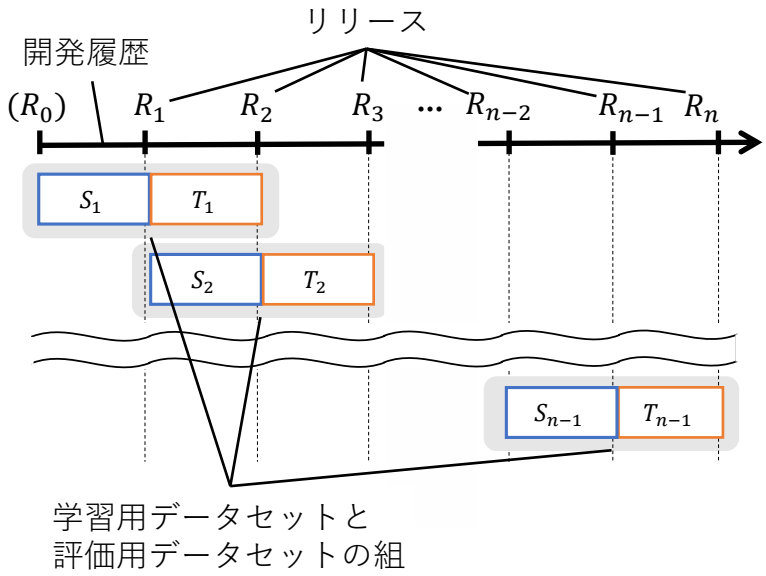

図 7　n 個のリリースが存在する場合に算出される実験用データセット

ト S_{n-1} に振り分ける．なお，多重共線性対策は行わない [12]．

- 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) [13]．例えば，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の件数が同じになるまで，isBuggy について真であるレコードを無作為に復元抽出する [14]．

5.1.4　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 [3–5, 15]，比較的高い精度で予測できている *Random Forest*(RF) [16] を採用した．その実装としては，scikit-learn [17] による実装を用いた．

5.1.5　ハイパーパラメータチューニング

ハイパー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 [18]. 本調査では，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チューニング手法の一種，ベイズ最適化 [19] の実装である optuna [20] 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 5 に示す.
3. 学習用データセットのサブセットの一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 パターン存在する．それぞれの S', V の組み合わせについて，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.

5.1.6　評価指標

本調査における評価指標は再現率, 適合率, F 値，ROC-AUC である．説明は 2.4 節を参照のこと.

5.1.7　精度目標

本調査における精度目標は ROC-AUC=0.78 である．説明は 2.5 節を参照のこと.

表 5　ハイパーパラメータと探索範囲

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

5.1.8 対象プロジェクト

RQ1 と同様に，8 種類の Java プロジェクトを対象とした．表 1 を参照のこと．

5.2 実験結果

5.2.1 バグ予測精度の平均値

表 6 は，対象プロジェクトの各リリース時点において *isBuggy* モデル・*hasBeenFixed* モデルを構築し予測精度を評価した結果を示す．*isBuggy* モデルは *hasBeenFixed* モデルと比べて F 値 (平均値) が約 53.2% 低く，AUC(平均値) が約 17.6% 低い．よって，*isBuggy* の予測は *hasBeenFixed* の予測より難しいと言える．また，*isBuggy* モデルの AUC は平均で約 0.745 であり，これは精度目標（AUC=0.78）を満たさない．よって，メソッド粒度バグ予測の実用化には課題が残ると言える．

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

		isBuggy モデル				hasBeenFixed モデル			
プロジェクト名	学習→評価	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

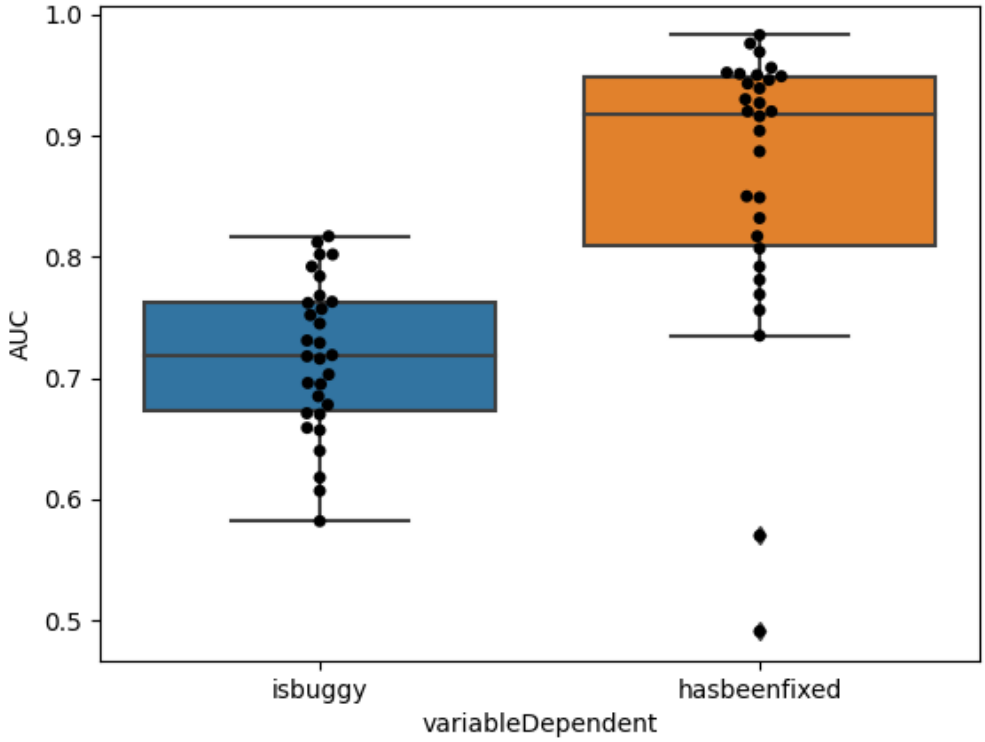

図 8　実験結果 (予測精度の分布)

5.2.2　バグ予測精度の分布

図 8 は *isBuggy* モデル・*hasBeenFixed* モデルの予測精度の分布を箱ひげ図とスウォームプロットにより表現している．*isBuggy* の予測において，精度目標（AUC=0.78）をクリアしたプロジェクト・リリース時点が存在する．一方で，予測精度が 0.6 に満たないプロジェクト・リリース時点も存在し，これはバグの有無をランダムに出力した場合と同程度の精度である．つまり，メソッド粒度バグ予測において予測精度はプロジェクト・リリースごとに様々で，予測がうまくいくプロジェクト・リリース時点とそうでないものがある．

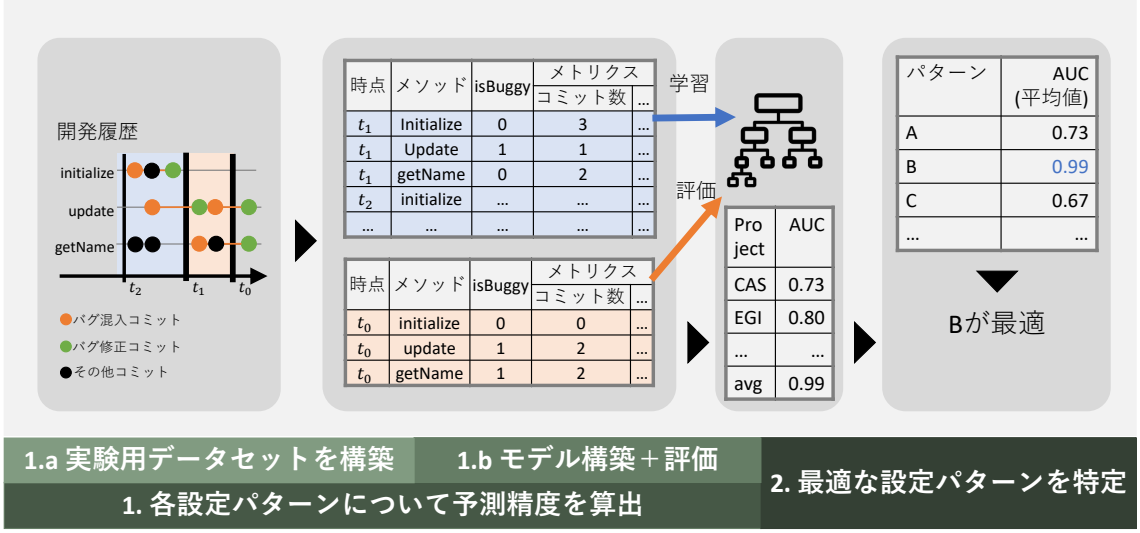

図 9　実験の概要 (RQ3)

6　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 9 と以下に示す．

1. interval の値と period の値の組み合わせ (設定パターン) について，以下の操作を行う．

   (a) 実験用データセットを対象プロジェクト・リリースごとに構築する．データセット構築手法は 6.1.5 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予測精度がどの程度改善したかを評価する．

6.1 実験設定

6.1.1 目的変数

本調査における目的変数は isBuggy である．説明は 2.1 章を参照のこと．

6.1.2 説明変数

本調査における説明変数は RQ2 のものと同様である．説明は 5.1.2 項を参照のこと．

6.1.3 説明変数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 ヶ月前まで，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 6.1.5 項で述べる．

- N コミット (ただし，N の候補値は 500, 1000, 1500, 2000, 2500 である．)
- N ヶ月 (ただし，N の候補値は 1, 2, 3, 6, 12 である．)

6.1.4 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，過去の時点に存在した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 6.1.5 項で述べる．

- 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．)

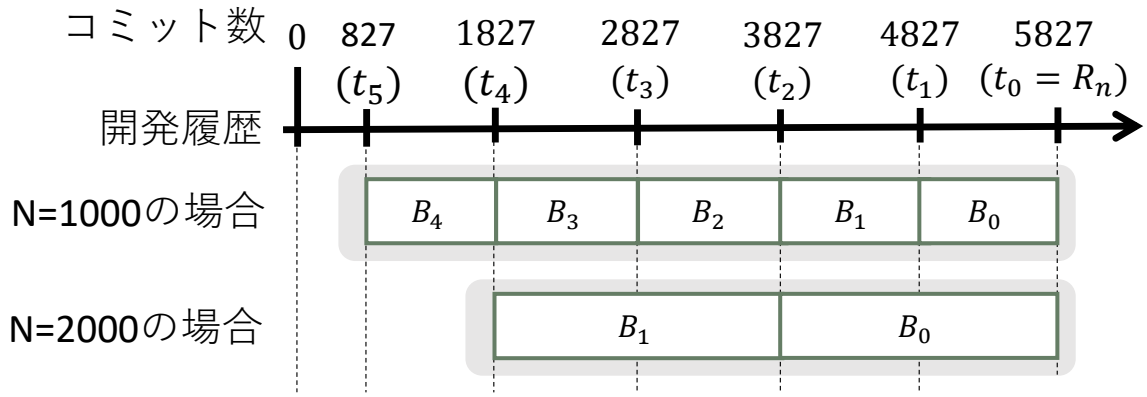

図 10　算出されるデータブロックの例 (interval = N コミット)

6.1.5　実験用データセット

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 ヶ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 10 に示す．

   (a) R_n 時点を t_0 とおく．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

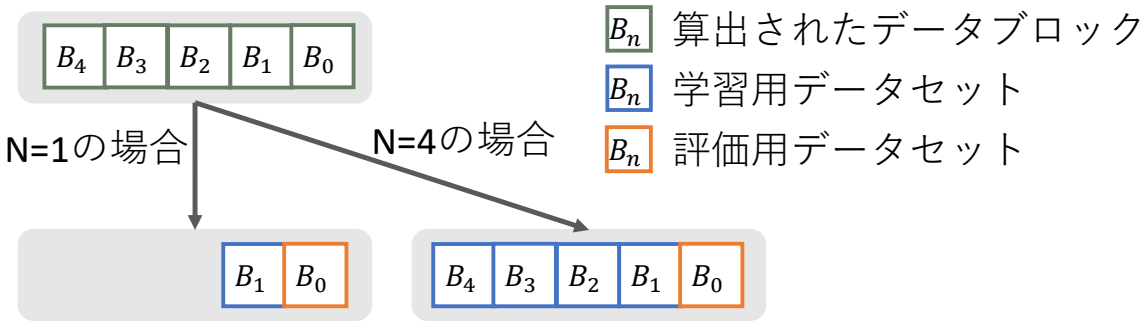

図 11　データブロックの振り分け (period = last($N/5$))

に振り分ける．振り分けの例を図 11 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）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．例えば，データブロックとして B_1, B_2, B_3, B_4 が存在し，last(1/5) が採用されている場合，最新の $4 * (1/5) = 0.8 \fallingdotseq 1$ 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，last(4/5) が採用されている場合，最新の $4 * (4/5) = 3.2 \fallingdotseq 4$ 個である全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，多重共線性対策は行わない [12].

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) [13]．例えば，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の件数が同じになるまで，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 [14].

6.1.6　学習アルゴリズム

本調査における学習アルゴリズムは RQ2 のものと同様である．説明は 5.1.4 項を参照のこと．

6.1.7　ハイパーパラメータチューニング

本調査におけるハイパーパラメータチューニングのプロセスは RQ2 のものと同様である．説明は 5.1.5 項を参照のこと．

6.1.8　評価指標

本調査における評価指標は再現率, 適合率, F 値，ROC-AUC である．説明は 2.4 節を参照のこと．

6.1.9　精度目標

本調査における精度目標は ROC-AUC=0.78 である．説明は 2.5 節を参照のこと．

6.1.10　対象プロジェクト

RQ1・RQ2 と同様に，8 種類の Java プロジェクトを対象とした．表 1 を参照のこと．

6.2　実験結果

6.2.1　予測精度比較（精度が最高のパターン vs リリースバイリリース）

表 7 は，各設定パターンについて予測精度の平均値を示している．最も予測精度 (F 値・AUC) が高くなるパターンは interval として 12 ヶ月を，period として last(5/5) を採用したものであった．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% の精度改善が見られた．また，AUC は約 0.773 であり，約 2.9% の精度改善が見られた．この差が統計的に有意かどうか，ウィルコクソンの符号順位検定を用いて検定したところ，p 値 $= 0.004 < 0.05$ であり，有意と判定された．しかしながら，精度目標（AUC=0.78）は達成されなかった．

6.2.2　実行時間比較（精度が最高のパターン vs リリースバイリリース）

精度が最高となるパターンを採用した場合・リリースバイリリースを採用した場合について実行時間を比較した結果を表 8 に示す．ハイパーパラメータチューニングに長時間かつ固定期間を費やしているため，両者に大きな差はみられない．また，バグ予測は自動かつバックグラウンドで進行することを考慮すると，両者ともに現実的な実行時間に収まっていると考える．

6.2.3　period に対する精度比較

図 12 は縦軸を予測精度（AUC)，横軸を period の候補値，つまり過去データを利用する割合とするグラフである．過去データを利用する割合と予測精度（AUC）の間に強い相関 (r=0.88) が見られるた

め，昔のバグメソッドについての特徴も学習させることは予測精度改善に有効といえる．

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)

interval	period	適合率	再現率	F 値	AUC
12 ヶ月	last(5/5)	0.114	0.596	0.191	0.773
6 ヶ月	last(1/5)	0.101	0.666	0.175	0.772
6 ヶ月	last(3/5)	0.108	0.631	0.184	0.770
12 ヶ月	last(4/5)	0.113	0.598	0.190	0.766
6 ヶ月	last(2/5)	0.105	0.623	0.180	0.765
6 ヶ月	last(4/5)	0.103	0.621	0.176	0.764
12 ヶ月	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 ヶ月	last(3/5)	0.107	0.603	0.181	0.762
6 ヶ月	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 ヶ月	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 ヶ月	last(4/5)	0.099	0.640	0.171	0.755
1 ヶ月	last(1/5)	0.101	0.645	0.174	0.754
3 ヶ月	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 ヶ月	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 ヶ月	last(2/5)	0.098	0.616	0.169	0.752
2 ヶ月	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 ヶ月	last(3/5)	0.104	0.593	0.177	0.750
1 ヶ月	last(3/5)	0.099	0.624	0.170	0.748
2 ヶ月	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 ヶ月	last(4/5)	0.096	0.633	0.167	0.746
1 ヶ月	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 ヶ月	last(1/5)	0.103	0.592	0.176	0.744
2 ヶ月	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 ヶ月	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 ヶ月	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

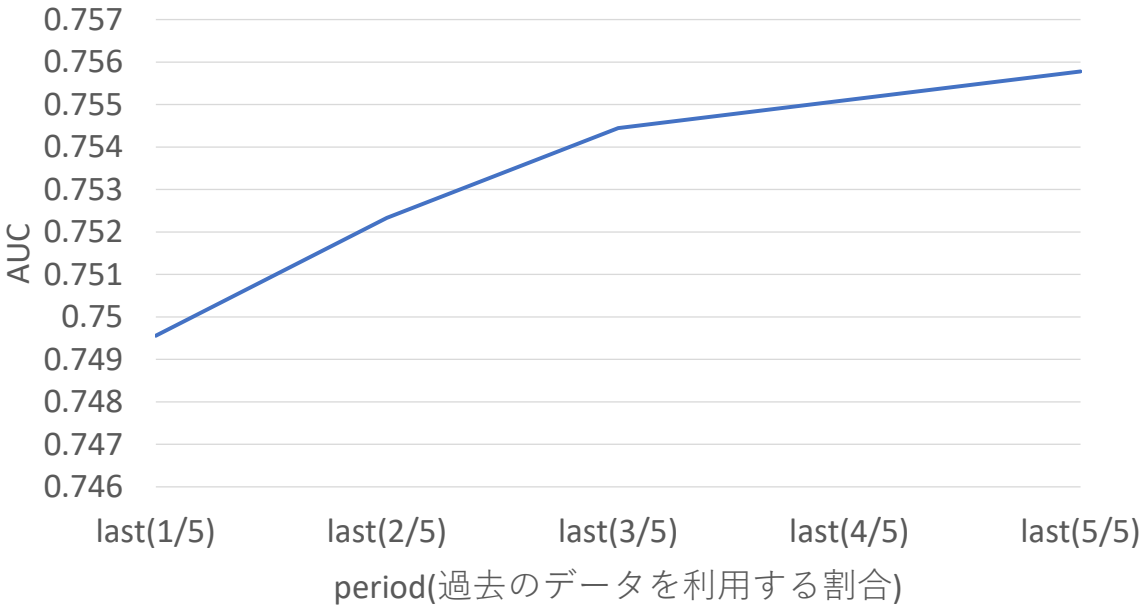

図 12　period に対する予測精度 (AUC)

7　RQ4

RQ4 では，説明変数としての時系列データの利用が精度改善に有効かを評価する．具体的には，説明変数にメトリクスを用いた予測と，それらのメトリクスの時系列表現 (時系列データ) を用いた予測について予測精度を比較する．RQ4 の実験工程を図 13 と以下に示す．

1. 比較対象である，メトリクスを説明変数とする場合・メトリクスの時系列表現を説明変数とする場合について，下記の操作を行う．

   (a) 実験用データセットを，リリースバイリリースを用いて対象プロジェクト・リリースごとに構築する．

   (b) 実験用データセットごとに，深層学習アルゴリズムを用いてモデルを構築・評価し，予測精度の平均値を算出する．

表 8　実行時間比較 (精度が最高のパターン vs リリースバイリリース)

設定パターン	データセット構築 (分)	パラメータチューニング (分)	モデル構築 (分)	合計 (分)
精度が最高のパターン	0.05	600	2.9	603
リリースバイリリース	0.14	600	10.7	611

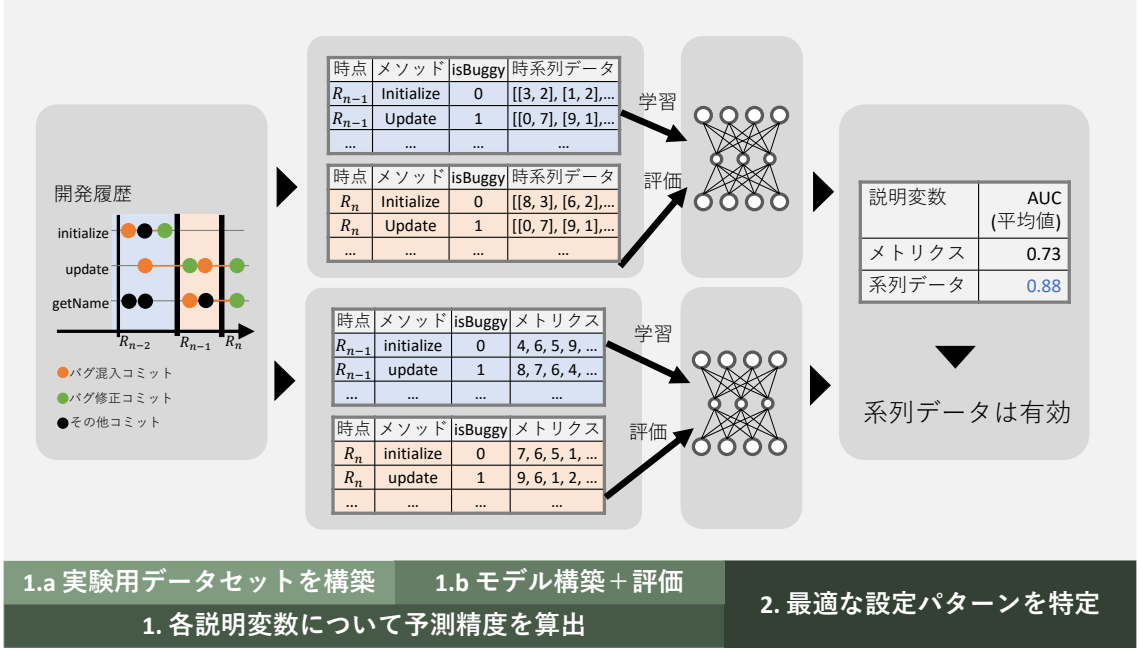

図 13　実験の概要 (RQ4)

2. それぞれの場合について，予測精度を比較する．時系列データを説明変数とする場合の予測精度が優れていれば，当該手法は精度改善に有効と言える．

7.1　実験設定

7.1.1　目的変数

本調査における目的変数は isBuggy である．説明は 2.1 章を参照のこと．

7.1.2　説明変数

本調査では説明変数にメトリクスを用いた場合・説明変数に時系列データを用いた場合の予測精度を比較する．つまり，下記の 2 種類の説明変数を算出する．

- $Giger_m$: Giger らが提案したメトリクスセットである [3]．具体的には，表 9 に記載されたメトリクスからなる 1 次元ベクトルである．
- $Giger_{seq}$: $Giger_m$ の時系列表現である．具体的には，メソッドに対するコミットごとに算出される 1 次元ベクトル（コミットベクトル）を時系列順に並べた 2 次元ベクトルである．コミットベクトルは表 10 に記載されたメトリクスからなる 1 次元ベクトルである．

7.1.3 実験用データセット

RQ2 と同様に，リリースバイリリースを用いて下記のように実験用データセットを構築する．

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．
    - 説明変数を，R_{n-1} から R_n までの開発期間を参照して算出する．
    - 目的変数を，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．
    - 説明変数を，R_{n-2} から R_{n-1} までの開発期間を参照して算出する．
    - 目的変数を，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) [13]．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の件数が同じになるまで，

メトリクス名	概要
MethodHistories	コミット回数
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 9　$Giger_m$ の要素

要素名	概要
StmtAdded	追加されたステートメントの総数
StmtDeleted	削除されたステートメントの総数
Churn	StmtAdded - StmtDeleted
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 10　$Giger_{seq}$ のコミットベクトルの要素

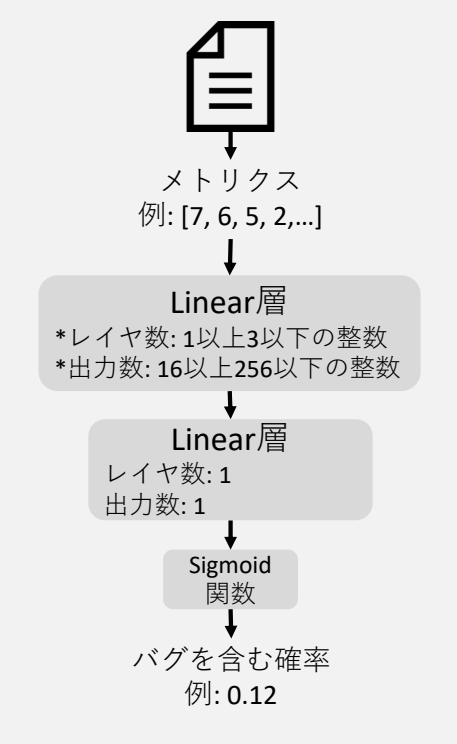

図 14　$Model(Giger_m)$ の構造

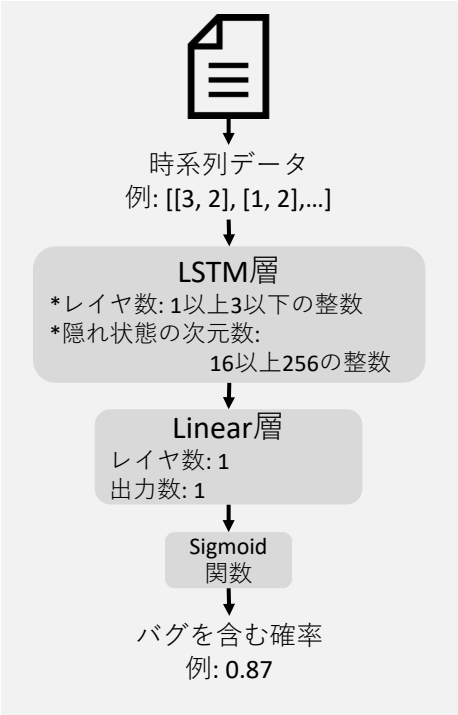

図 15　$Model(Giger_{seq})$ の構造

isBuggy について真であるレコードを無作為に復元抽出する [14].

7.1.4　モデル構造

それぞれの説明変数に一対一対応するモデル構造について以下に説明する.

- $Model(Giger_m)$: $Giger_m$ を入力とするモデルである．モデルのネットワーク構造は図 14 の示す通り，メソッドから算出されたメトリクスセット（$Giger_m$）を連結された Linear 層へ入力し，Linear 層からの出力値を Sigmoid 関数へ入力することで，最終出力としてメソッドがバグを含む確率を得る．なお，*印が付いているパラメータはハイパーパラメータチューニングにより決定する.
- $Model(Giger_{seq})$: $Giger_{seq}$ を入力とするモデルである.
  モデルのネットワーク構造は図 15 の示す通り, メソッドから算出された時系列データ（$Giger_{seq}$）を連結された LSTM 層へ入力し，LSTM 層からの出力値を Linear 層へ入力し，Linear 層からの出力値を Sigmoid 関数へ入力することで最終出力としてメソッドがバグを含む確率を得る．なお，*印が付いているパラメータはハイパーパラメータチューニングにより決定する.

7.1.5　学習アルゴリズム

本調査においては，下記のようにモデルの学習が進行する．なお，学習過程におけるハイパーパラメータの一覧を表 11 に示す．

1. 学習用データの説明変数をモデルに入力して目的変数を予測させ，予測値と真の値との誤差を算出する．ここで，予測値と真の値との誤差を測る関数（損失関数）としてバイナリ交差エントロピー [21] を用いる．
2. 予測値と真の値との誤差が小さくなるようにモデルのパラメータを更新する．ここで，更新方法（オプティマイザ）としてバグ予測の先行研究においても採用例がある Adam [22] を採用する．
3. 適切な回数，1-2 を繰り返す．本調査においては，繰り返す回数 (エポック数) をハイパーパラメータチューニングにより決定する．

7.1.6　ハイパーパラメータチューニング

本調査のモデル構築におけるハイパーパラメータとその探索範囲は，7.1.4 項および 7.1.5 項で述べた．それらのハイパーパラメータについて，下記のようにチューニングを行った．

1. 学習用データセットを構成するレコードの 2 割をハイパーパラメータチューニングにおける検証用データセットとし，残りの 8 割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とする．
2. ハイパーパラメータチューニング手法，ベイズ最適化 [19] の実装である optuna [20] 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．

表 11　学習過程におけるハイパーパラメータ

パラメータ名	値 (または探索候補値)
*lerningRate	float 型で $1.0 * 10^{-4}$ 以上 $1.0 * 10^{-6}$ 以下の数
betas[0]	0.9
betas[1]	0.999
eps	$1.0 * 10^{-8}$
weightDecay	0
*numOfEpochs	int 型で 1 以上 10000 以下の数
sizeOfBatch	学習用レコード数/100

*ハイパーパラメータチューニングの対象

3. H を用いて学習用データセットに基づいてモデルを構築し，検証用データセットを用いてモデルの予測精度を評価する．
4. 2. および 3. を AMD Ryzen 9 3950X を搭載したコンピュータを利用して 24 時間が経過するまで繰り返し，最も高い予測精度が得られたハイパーパラメータを採用する．

7.1.7 評価指標

本調査における評価指標は再現率, 適合率, F 値，ROC-AUC である．説明は 2.4 節を参照のこと．

7.1.8 精度目標

本調査における精度目標は ROC-AUC=0.78 である．説明は 2.5 節を参照のこと．

7.1.9 対象プロジェクト・リリース

本調査においては，オープンソースの大規模 Java プロジェクトを実験対象とした．具体的には表 12 の通り，3 つのプロジェクト・リリースの組を対象とした．

7.2 実験結果

表 12 は，メトリクス・時系列データのそれぞれを説明変数として利用した場合の予測精度を示す．時系列データを用いることで，F 値（平均値）は 4.2% 低下し，AUC（平均値）は 1.4% 向上している．AUC の差が統計的に有意かどうか，ウィルコクソンの符号順位検定を用いて検定したところ，p 値 $= 0.5 > 0.05$ であり，有意とは判定されなかった．よって，メトリクスに対応する時系列データを説明変数として利用するだけでは，必ずしも精度向上にはつながらないと判明した．

表 12　説明変数に対する予測精度

		説明変数：メトリクス				説明変数：時系列データ			
プロジェクト名	学習→評価	適合率	再現率	F 値	AUC	適合率	再現率	F 値	AUC
cassandra	R1 → R2	0.200	0.575	0.297	0.728	0.204	0.626	0.307	0.728
egit	R1 → R2	0.574	0.133	0.216	0.614	0.477	0.149	0.227	0.543
poi	R3 → R4	0.108	0.627	0.184	0.585	0.098	0.675	0.171	0.585
平均値		0.135	0.508	0.213	0.567	0.125	0.551	0.204	0.575

8 妥当性の脅威

8.1 内的妥当性

8.1.1 データ算出アルゴリズムの妥当性

説明変数であるメトリクス・時系列データ及び目的変数である *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 らによる説明 [3]，Ming らによる説明 [9] 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 つの問題が有る．

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 [23, 24]．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 [23]．
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み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．

8.1.2 ハイパーパラメータチューニングの不足

ハイパー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 [18]．本調査では各モデルの構築時に 10 時間または 24 時間を費やしてハイパー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．

8.2 外的妥当性

8.2.1 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当てはまらない可能性がある．

8.2.2 対象プロジェクト

本調査では比較的大規模な OSS プロジェクトのみを実験対象としており，その他のプロジェクトに関しては本調査で得られた知見が当てはまらない可能性がある．

9 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，下記の 4 つの調査を行った．

1 つ目に，*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のかを調査した．その結果，*isBuggy* の分布と *hasBeenFixed* の分布とが大きく異なるため，*hasBeenFixed* を目的変数として予測するモデルは *isBuggy* を正確に予測できないと判明した．

2 つ目に，ユースケースにおけるメソッド粒度バグ予測の予測精度を評価した．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，メソッド粒度バグ予測の実用化には課題が残されていると判明した．

3 つ目に，ユースケースにおけるメソッド粒度バグ予測の予測精度改善を目的として，説明変数算出時に参照する変更履歴の期間・学習に用いるメソッドの存在期間の変更について，有効性を評価した．その結果，予測精度が F 値の観点で約 15.2%，AUC の観点で約 2.9% 向上し，有効性が確認できた．

4 つ目に，ユースケースにおけるメソッド粒度バグ予測の予測精度改善を目的として，深層学習による時系列データを入力とする予測の有効性を評価した．その結果，F 値は 4.2% 低下し，AUC は 1.4% 向上し，有効性は確認できなかった．

バグ予測の実用化には多くの課題が存在する [25]．例えば，予測精度の改善や，予測した根拠の明示，エディターへの統合等が挙げられる．中でも最も重要な課題は予測精度の改善だと考える．なぜなら，予測精度が低ければ他の問題に取り組む意義がないためである．よって，メソッド粒度バグ予測において，当面の課題は予測精度の改善である．まずは，ファイル粒度バグ予測で有効性が確認されている手法をメソッド粒度バグ予測に応用してみるべきだと考える．

謝辞

本研究に対し数々のコメントをいただきました，楠本 真二 教授に感謝いたします．特に発表練習の場では，発表の質を大きく向上させることができました，ありがとうございました．

本研究の全過程で熱心に指導していただきました，肥後 芳樹 准教授に感謝いたします．数多くのアドバイスや励ましの言葉をいただきました，ありがとうございました．また，何度も論文・発表をチェックしていただき，的確な改善案を示していただきました，ありがとうございました．

輪講や発表練習の場で数多くの指導をいただきました，柗本 真佑 助教に感謝いたします．私のメタ能力（プレゼンテーション能力等）の向上につながりました，ありがとうございました．

様々な事務手続きの際にお世話になりました，事務補佐員の神谷智子氏に感謝いたします．学会参加費の建て替えの際には大変お世話になりました．

同じく，様々な手続きの際にお世話になり，また研究室の利便性を向上していただきました，事務補佐員の橋本美砂子氏に感謝いたします．ティーチングアシスタントにまつわる手続きの際には大変お世話になりました．

普段の生活のなかでお世話になりました，学生の皆様に感謝いたします．特に，先輩及び同輩の皆様には，修士課程から研究室に配属され慣れない状況にある私に優しく接してくださったり，研究や就職の相談に乗っていただきました，ありがとうございました．

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. https://www.ipa.go.jp/files/000085879.pdf(accessed at 2022/02/09).

[2] Tom Britton, Lisa Jeng, Graham Carver, Paul Cheak, and Tomer KKatzenellenbogen. Increasing software development productivity with reversible debugging. https://undo.io/media/uploads/files/Undo_ReversibleDebugging_Whitepaper.pdf(accessed at 2022/02/09).

[3] Emanuel Giger, Marco D'Ambros, Martin Pinzger, and Harald Gall. Method-level bug prediction. In *International Symposium on Empirical Software Engineering and Measurement*, pp. 171–180, 2012.

[4] Hideaki Hata, Osamu Mizuno, and Tohru Kikuno. Bug prediction based on fine-grained module histories. In *International Conference on Software Engineering*, pp. 200–210, 2012.

[5] Luca Pascarella, Fabio Palomba, and Alberto Bacchelli. Re-evaluating method-level bug prediction. In *International Conference on Software Analysis, Evolution and Reengineering*, pp. 592–601, 2018.

[6] Jacek Sliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? *SIGSOFT Software Engineering Notes*, Vol. 30, No. 4, pp. 1–5, 2005.

[7] Markus Borg, Oscar Svensson, Kristian Berg, and Daniel Hansson. 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. In *ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*, pp. 7–12, 2019.

[8] Takafumi Fukushima, Yasutaka Kamei, Shane McIntosh, Kazuhiro Yamashita, and Naoyasu Ubayashi. An empirical study of just-in-time defect prediction using cross-project models. In *Working Conference on Mining Software Repositories*, pp. 172–181, 2014.

[9] Ming Wen, Rongxin Wu, and Shing-Chi Cheung. How well do change sequences predict defects? sequence learning from software changes. *IEEE Transactions on Software Engineering*, Vol. 46, No. 11, pp. 1155–1175, 2020.

[10] Hongliang Liang, Yue Yu, Lin Jiang, and Zhuosi Xie. Seml: A semantic lstm model for software defect prediction. *IEEE Access*, Vol. 7, pp. 83812–83824, 2019.

[11] Cong Pan, Minyan Lu, and Biao Xu. An empirical study on software defect prediction using codebert model. *Applied Sciences*, Vol. 11, No. 11, pp. 200–210, 2021.

[12] Carsten F. Dormann, Jane Elith, Sven Bacher, Carsten Buchmann, Gudrun Carl, Gabriel

Carré, Jaime R. García Marquéz, Bernd Gruber, Bruno Lafourcade, Pedro J. Leitão, Tamara Münkemüller, Colin McClean, Patrick E. Osborne, Björn Reineking, Boris Schröder, Andrew K. Skidmore, Damaris Zurell, and Sven Lautenbach. 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.

[13] Xinjian Guo, Yilong Yin, Cailing Dong, Gongping Yang, and Guangtong Zhou. On the class imbalance problem. In *International Conference on Natural Computation*, Vol. 4, pp. 192–201, 2008.

[14] Nitesh V. Chawla. *Data Mining and Knowledge Discovery Handbook*, pp. 875–886. Springer, 2010.

[15] Stefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. 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.

[16] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. In *R news*, pp. 18–22, 2002.

[17] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, Vol. 12, pp. 2825–2830, 2011.

[18] 尾崎嘉彦, 野村将寛, 大西正輝. 機械学習におけるハイパ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.

[19] Jonas Mockus. On bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference Novosibirsk*, pp. 400–404, 1975.

[20] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631, 2019.

[21] Torch Contributors. pytorch document. `https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html`.

[22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] Chadd Williams and Jaime Spacco. Szz revisited: Verifying when changes induce fixes. In

Workshop on Defects in Large Software Systems, pp. 32–36, 2008.

[24] Daniel Costa, Shane McIntosh, Weiyi Shang, Uirá Kulesza, Roberta Coelho, and Ahmed E. Hassan. A framework for evaluating the results of the szz approach for identifying bug-introducing changes. *IEEE Transactions on Software Engineering*, Vol. 43, No. 7, pp. 641–657, 2017.

[25] Zhiyuan Wan, Xin Xia, Ahmed E. Hassan, David Lo, Jianwei Yin, and Xiaohu Yang. Perceptions, expectations, and challenges in defect prediction. *IEEE Transactions on Software Engineering*, Vol. 46, No. 11, pp. 1241–1266, 2020.