卒 業 研 究 報 告 書

題　目　　ソフトウェアリリースを考慮した
　　変更レベルの不具合予測の検証

指導教員　　水野　修　教授

崔　恩瀞　助教

京都工芸繊維大学 工芸科学部 情報工学課程

学生番号　　16122020

氏　　名　　里形　洋道

令和2年2月12日提出

ソフトウェアリリースを考慮した変更レベルの不具合予測の検証

令和 2 年 2 月 12 日　　16122020　里形　洋道

概　要

ソフトウェアの品質を高く保つには，不具合の除去や可読性の向上が重要である．不具合を取り除く手段として不具合予測を用いることがある．また，不具合予測の研究では，Just-in-time defect prediction (JIT DP) という，変更に対して不具合予測を適用する研究が行われている．JIT DP における既存の手法では，全てのコミットの情報を混ぜた予測モデルを構築したり，時系列を考慮した予測モデルを構築したりしている．時系列を考慮した JIT DP の先行研究では，時系列を考慮することで，予測精度に変化があったという結果が上がっている．しかしながら，その変化の理由は解明されていない．

本研究では，リリース毎の不具合予測に使用するコミットの特徴の差異が予測結果に影響を与えるという仮説をたて，リリースタイミングを考慮した JIT DP を検証した．具体的には，(1) 目標リリースの一つ前のリリースのコミット情報のみを用いて構築した不具合予測モデルと，(2) 目標リリース以前のコミット全てを用いて構築した予測モデルで予測精度の比較検証を行った．また，不具合が修正された日付も考慮し，不具合が修正された期間が構築する予測モデルのリリース期間中であるコミットのみを用いて，リリース毎に予測モデルを構築する実験も行った．結果より，(2) で構築したモデルの方が，(1) のモデルよりも予測精度が良いことがわかった．一方で，今回の実験では，リリース毎に予測モデルを構築しようとすると訓練データが不足するという問題が発生することがわかった．この傾向は，不具合が修正された日付を考慮してデータ収集をした場合により顕著であり，約 64 % のケースで予測モデルを構築できなかった．

目　次

1. 緒言

ソフトウェアの品質を高く保つことは広く要求されており，複雑さを増すソフトウェア開発の現場では不具合を取り除くことに多くの時間や労力といったリソースが割かれている．しかし，予算や人員の関係上，ソフトウェア開発に投入可能なリソースは限られており，近年の大規模化したソフトウェアから全ての不具合を取り除くことは現実的でない．そこで，不具合がありそうなソフトウェア要素を予め発見することで，リソースを効率的に配分し，より不具合の少ないソフトウェアを目指す，不具合予測という手法が研究されている．

不具合予測には，不具合が含まれているファイルに対して予測するファイルレベルの不具合予測が存在する [1, 2]．しかし，この手法は予測対象の粒度が大きく，また，予測をするタイミングが遅いなどの問題が存在する．これらの問題を解決するために，コミットなど変更に対して不具合予測をする，変更レベルの不具合予測（Just-In-Time Defect Prediction，以下 JIT DP）が提案された [3]．

JIT DP は，不具合の特徴がソフトウェア開発中で変化しないということを仮定している．この仮定により，過去に行われた変更とそこで発生した不具合の特徴を学習データとして予測モデルを訓練することが可能となる．しかし，McIntosh らは，1 年以上前のデータを用いた予測モデルの訓練では精度の低下が見られ，不具合と関係する特徴が変化していることを指摘した．また，不具合を予測する上で重要な特徴が一定期間毎に異なることも指摘している [4]．しかし，その理由については未だ明らかではない．我々は，一定期間毎に行われるソフトウェア開発の活動の 1 つであるリリースがこの差の原因であると考え，リリースを考慮した JIT DP は，より正確に不具合の特徴を捉えることができるという仮説を立てた．

本研究では，リリースを考慮した JIT DP のモデルを構築し，その精度を調査した．具体的には，ソフトウェアリリース毎に不具合予測モデルを構築し，その次のリリースにおけるコミットに対して，不具合が含まれるか予測する．比較手法として，予測対象のコミットまでの全てのコミットを学習データとして利用する不具合予測モデルを用意し，予測精度を比較した．実験の結果は，リリースを考慮すると 1 リリースあたりのコミット量が少なく，学習データが不足したことにより既存のモデルと比較して予測精度が悪かった．また，実際の運用を想定し再現するために，

リリース毎に不具合予測モデルを構築するとき，後のリリースのコミットで修正された不具合を含むコミットを，不具合でないものとして扱った予測モデルも構築し，実験した．実運用を考慮した実験では，不具合データがさらに不足し，予測モデルの構築自体が困難であった．今回の実験では 253 個のリリースで不具合予測モデルを構築したが，約 64 %の 161 個のリリースは不具合が後のリリースで修正されており，約 81 %の不具合が後のリリースで修正されていた．その結果，不具合が含まれないリリースとなって，予測モデルの構築をできなかったリリースが多数あり，構築できたリリースでも顕著に精度が低かった．これらの結果より，リリースを考慮した JIT DP のモデル構築のためには，データ不足を解消する必要があるとわかった．

本紙の構成は次のようになっている．第 2 章で研究背景と意義を述べ，第 3 章で実験準備を行い，第 4 章で実験について説明し，第 5 章で結果を示し，第 6 章で考察を述べ，第 7 章をもって本紙の結言とする．

2. 研究背景・意義

2.1 関連研究

2.1.1 既存の不具合予測手法について

不具合予測についての概要を述べる．不具合予測とは，ソフトウェアの開発情報から不具合予測モデルを機械学習などで構築し，不具合を含むであろうファイルや変更を予測する手法である．不具合予測の粒度としては，ファイルレベル [1, 2] やモジュールレベル，変更レベル [3] などがあり，本稿の実験では変更レベルの不具合予測である JIT DP を扱う．

JIT DP とは，変更に焦点を当てた不具合予測モデルである．具体的に述べると，コミットしたときの変更内容に含まれる情報を用いて，不具合を予測するモデルである．他の不具合予測手法と比較すると，主に 2 点の利点がある．1 つは，コミットしたとき，その変更に不具合があるかどうかをすぐ予測でき，開発者はすぐに予測結果を知ることができる点である．もう 1 つは，変更レベルの予測はファイルレベルなどの予測よりも粒度が小さいため，開発者が欠陥を特定するために調べるコードの行数がはるかに少なくなる傾向にあるという点である [3]．ただし，既存の JIT DP では，データを 10-fold cross validation などによってデータを混ぜてしまうため，時系列の考慮などはされていない．

また，機械学習を用いる不具合予測モデルは，不具合データセットを収集して予測モデルを学習させる必要がある．不具合データセットは，さまざまなソフトウェアメトリクスとラベルで構成されている．具体的に述べると，追加された行数や修正されたファイルの数などといったメトリクスが用いられ，ラベルは，ソースコードが不具合であるかクリーンであるかを示す [5].

2.1.2 時系列を考慮した不具合予測に関して

JIT DP の先行研究では，不具合を誘発するコミットの特徴は大きく変化しないということを仮定している．しかし McIntosh らによると，システムは時間と共に複雑になるので，時間を経るほど専門知識がより重要になってくるなど変化が生じるはずであるという主張がなされている [4]．そして実際に，McIntosh らはシステムが進

化する過程で，修正を含む変更の重要な性質は変化をするのかという点を詳しく分析している．その結果，次の 2 点が示されている．1 つは，JIT DP は 1 年程度の時間経過とともに予測精度が悪くなること．もう 1 つは，不具合を予測する上で重要な特徴が一定期間毎に異なることである [4]．しかし，その差を生む直接的な要因については検証されておらず，未だ明らかではない．

McIntosh らによる研究は，不具合予測モデルに用いる訓練データを 3ヶ月，あるいは 6ヶ月という単位で区切って用いており [4]，一定期間で区切っているという特徴がある．ソフトウェア開発における一定期間毎に行われる活動には，コミットやリリースといったものがある．リリースとは，機能の追加や不具合の修正など改善したソフトウェアを公開することである．コミットとリリースを比較すると，時間的に十分な幅を持つリリースは先行研究における一定期間という条件を満たすと言える．例示すると，Apache Eclipse プロジェクト [6] においては 2007 年から 2018 年の間リリースが約 1 年おきに行われ，Apache hadoop プロジェクト（以下 Hadoop）[7] ではおよそ 1ヶ月から 2ヶ月の間隔でリリースが行われていることが多い．また，リリースにはいくつかの種類が存在し，例を挙げると不具合の修正が主であるリリースや，機能追加が主となるリリースなどが存在している．このように，リリースはリリース毎に時間的な特徴と性質的な特徴を備えている．そのため，不具合予測における重要な特徴が一定期間に異なる原因は，リリースが関係していると考えた．そこで，リリースを考慮した JIT DP は，より正確に不具合の特徴を捉えることができるという仮説を立てた．

2.2 研究の目的

本研究の目的は，リリースを考慮した JIT DP の予測モデル，およびリリースを考慮しない JIT DP の予測モデルを比較し，JIT DP においてリリースがどの程度影響を与えるかを示すことである．さらに，リリースを考慮した JIT DP を実運用と同等の環境設定で予測実験をし，ソフトウェア開発の現場で JIT DP を用いる手がかりを提供する．

2.3 研究設問

研究の目的を達成するため，本研究では次に示す研究設問（Research Question）を設け，検証する．

RQ1 リリース毎に構築した JIT DP モデルは既存の JIT DP モデルと比較して予測精度はどの程度か．

RQ2 不具合修正コミットのタイミングまで考慮したリリース毎の JIT DP モデルの構築は可能か．

RQ1 では，リリース毎に JIT DP モデルを構築し，既存手法と比較した場合どちらが優れているか検証する．

RQ2 では，リリース毎の JIT DP を実際に運用する環境を考慮して構築した場合に，十分な予測精度を出すことができるかを検証する．

3. 実験準備

3.1 実験に用いたデータ

3.1.1 GitHub

GitHub は，Git と呼ばれるバージョン管理システムのリポジトリを中心とした，バージョン管理と共同開発のためのプラットフォームである．個人的なソフトウェア開発から大規模なオープンソースソフトウェアの開発まで，世界中で様々なプロジェクトが Git 及び GitHub により管理されている．本実験では，実験対象となるプロジェクトデータの取得に利用した．

3.1.2 対象プロジェクト

実験の対象として，Hadoop を用いる．Hadoop とは，巨大なデータの取り扱いを目的とした，オープンソースの分散処理フレームワークであり，Java で記述されている．Hadoop はリリース数，コミット数が十分なデータ数であり，リリースがおおよそ 1，2ヶ月おきに行われているため，適していると判断した．Hadoop のリリース数，総コミット数，不具合コミット数，及び不具合混入率を表 3.1 に示す．

3.1.3 不具合データセット

不具合予測モデルの研究において実験データの公開性と透明性は重要であるとされている [8]．そのため，不具合データセットは，Rosen らが公開している Commit Guru[9, 10] を利用して収集した．Commit Guru は，指定した Git リポジトリから変更メトリクス [9]，及び，不具合混入コミットの情報を自動で計算する Web アプリケーションである．

本研究では，得られたメトリクスを不具合予測モデルの学習に使用し，不具合混入コミットの情報から不具合コミットと正常なコミットのラベル付けを行う．

3.1.4 学習に使用するメトリクス

本実験で用いたメトリクスは，Kamei らの不具合予測モデルで用いられている物と同様の，数値メトリクス 13 個，及び論理メトリクス 1 個からなる変更に関するメ

表 3.1　学習に用いたプロジェクト（Hadoop）の概要

項目名	数値
リリース数	253
総コミット数	21,657
不具合混入コミット数	5,407
不具合混入率	24.97 %

トリクスである [9]．各メトリクスの詳細を表 3.2 に示す．

3.2 不具合予測モデル

3.2.1 ロジスティック回帰モデル

本研究では，Kamei ら [9] にならい，不具合予測モデルにロジスティック回帰モデルを用いた．ロジスティック回帰による分類予測では，ロジスティック回帰モデルを使用することで与えられた学習データをロジスティック曲線に回帰させ，得られた回帰式に分類したいデータを与えることで分類できる．ロジスティック回帰モデルは一般化線形モデルの 1 種であり，通常の線形回帰モデルでは直線に回帰させるのに対して，ロジスティック回帰モデルでは 0 と 1 に漸近する，ロジスティック曲線に回帰させる．目的変数（正解ラベル）が 0 か 1 の 2 値を取る場合に有効な手法である．回帰式に新規データを入力した結果として，そのデータのラベルが 1 である確率を算出するため，通常は確率の閾値を 0.5 として分類を決定する．JIT DP においてはコミットのメトリクスから不具合の有無を TRUE（1）か FALSE（0）に分類するため，ロジスティック回帰が有効であると言える．

3.2.2 Python

本実験では Python を使ってプログラムを作成した．Python は，オブジェクト指向のプログラミング言語の 1 種である．また，科学技術計算や機械学習に関するライブラリが充実しており，さまざまな分野で使われている．本実験で用いた機械学習のライブラリである scikit-learn もその 1 種である．

3.2.3 scikit-learn

scikit-learn は，Python のオープンソース機械学習ライブラリである [11]．scikit-learn はユーザーコミュニティの中で活発に開発，改良が加えられている．ドキュメントも整備されており，初学者でもスムーズに機械学習を始められるという利点がある．scikit-learn では，機械学習で用いられるクラス分類，回帰，クラスタリング，次元削減などができる．本研究で用いた scikit-learn のバージョンは 0.22.1 である．ロジスティック回帰モデルは sklearn.linear_model.LogisticRegression というクラスとし

表 3.2　不具合予測に用いたメトリクス一覧

メトリクス名	説明
ns	修正されたサブシステムの数
nd	修正されたディレクトリの数
nf	修正されたファイルの数
entropy	修正されたコードの各ファイル毎の分布
la	変更によって追加されたコードの行数
ld	変更によって削除されたコードの行数
lt	変更される前のファイルのコード行数
ndev	修正に関わった開発者の人数
age	最後の変更から最新の変更までの平均時間
nuc	修正されたファイルに対する変更の数
exp	開発者の経験，開発者の総コミット数
rexp	年齢によって重み付けされた exp
sexp	開発者のサブシステムに対する総コミット数
fix	不具合の修正であるか（TRUE/FALSE）

て実装されており [12]，本研究ではこれを利用した．ロジスティック回帰モデルでは 3.1.3 項で説明したパラメータを用い，各コミットが不具合を含むかどうかの 2 値に分類する．

4. 実験

4.1 実験方法

本実験では不具合データセットにて，各コミットの情報が何番目のリリースに含まれるコミットであるかをリリース番号でタグ付けしている．具体的には図 4.1 のように，リリース 1 以降からリリース 2 までのコミットは全て 1 とタグ付けされ，リリース 2 以降からリリース 3 までのコミットは全て 2 とタグ付けされる．リリース毎に分けるにあたって，リリース及びコミットがいつのものであるか判別するのに，不具合データセットにて author_date_unix_time と呼ばれるタイムスタンプで判別した．これは，コミットの著者がコミットを行った時間である author date の Unix time で記録されたタイムスタンプを指す．

また，実験方法の説明にあたって，不具合予測モデルの訓練に用いるリリースを Tr，不具合予測のテストを行うリリースを Te と呼ぶ．

4.1.1 実験方法 1

リリースごとの不具合予測モデルによる不具合予測の手順を以下に示す．

1. 不具合データセットの author_date_unix_time を用いて，各コミットをリリース区間毎にタグ分けを行う．
2. Tr のコミット情報のみを用いて不具合予測モデルを構築する．ただし，Tr のコミット数が 2 つ以下の場合データ数が不足しているとみなし，Tr の 1 つ前のリリースのコミットを足して不具合予測モデルを構築する．
3. Te のコミットを予測対象として不具合予測をする．
4. 2 と 3 の工程を，リリースタイミングを 1 つずつ未来にずらしながら繰り返し行う．

また，比較対象となる，Tr 以前のコミット全てを学習に用いた不具合予測モデルによる不具合予測の手順を以下に示す．

1. 不具合データセットの author_date_unix_time を用いて，各コミットをリリース区間毎にタグ分けを行う．

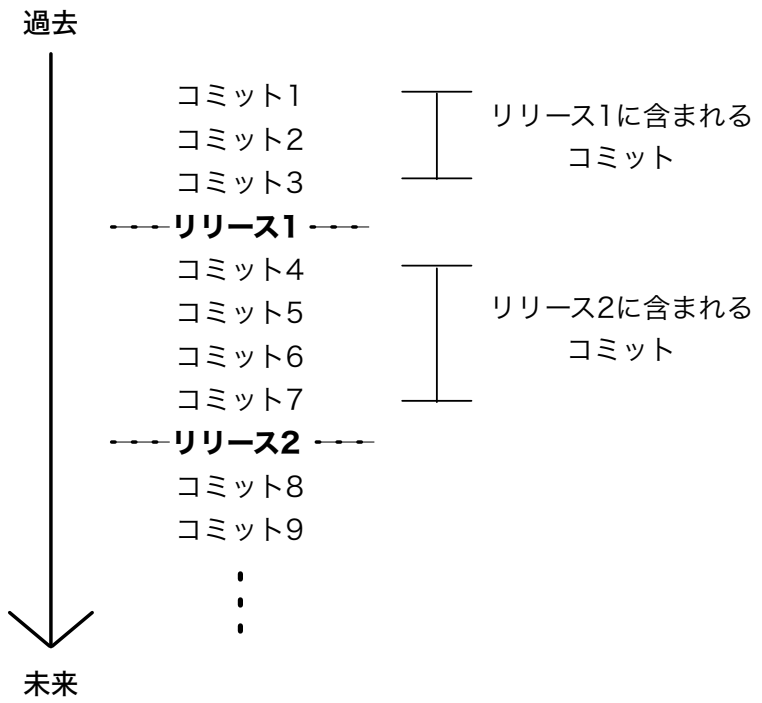

図 **4.1**　リリースとコミットの関係

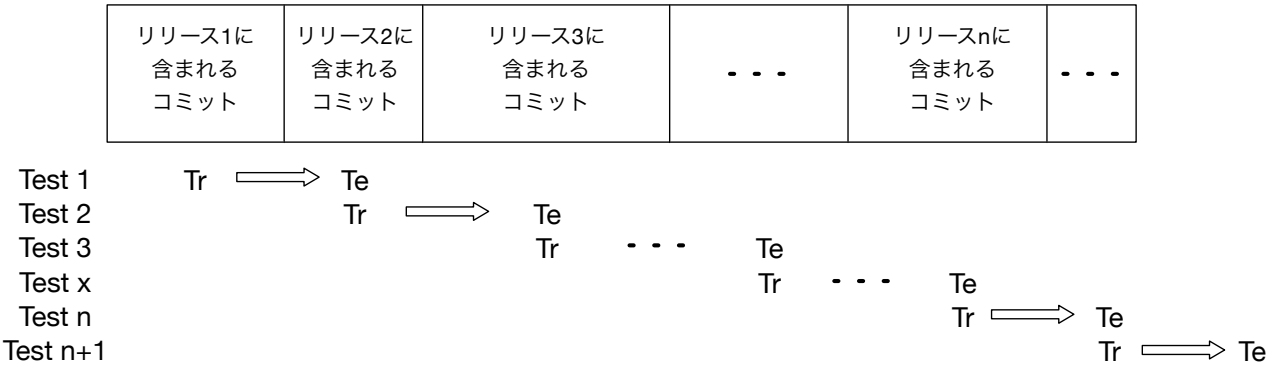

図 **4.2**　訓練データとテストデータの関係

2. Tr 以前のコミット情報を全て用いて不具合予測モデルを構築する．
3. Te のコミットに対して不具合予測をする．
4. 2 と 3 の工程を，リリースタイミングを 1 つずつ未来にずらしながら繰り返し行う．

4.1.2 実験方法 2

不具合が修正されたタイミングを考慮した不具合予測モデルによる不具合予測の手順を以下に示す．

1. Tr のコミット情報のみを取得する．ただし，リリース内でのコミット数が 2 つ以下の場合，1 つ前のリリースのコミット情報を足す．
2. Tr 内の不具合コミットで．対応する不具合修正コミットが Tr に含まれているか確認する．含まれていない場合，そのコミットは不具合コミットとして扱わない．これは，実運用を考慮した際に，このコミットが不具合コミットであることが，Tr の時点ではわかっていないためである．
3. 残ったコミットの情報で不具合予測モデルを構築する．なお，残ったコミットの情報が不具合を含まないコミットのみである場合，不具合予測モデルは構築せず 1 の工程にもどる．
4. Te のコミットを予測対象として不具合予測をする．
5. 1 から 4 の工程を，リリースタイミングを 1 つずつ未来にずらしながら繰り返し行う．

4.2 評価尺度

本稿の実験結果の評価値としては以下に示す AUC（Area Under the Curve），再現率（Recall），適合率（Precision），F_1 値の 4 つを用いる．各評価値の概要，及び計算式を以下に示す．

4.2.1 AUC（Area Under the Curve）

この論文における AUC とは，ROC 曲線（Receiver Operating Characteristic curve）以下の面積のことを指す．図 4.3 にて黒く塗り潰されている面積が AUC にあたる．

ROC 曲線とは，閾値を変化させたときの偽陽性率と真陽性率による各点を結んだものである．偽陽性率が大きくなるように閾値を設定するほど真陽性率も大きくなるため，ROC 曲線は常に右肩上がりのグラフとなる．そのため，偽陽性率が低い段階から真陽性率が高い不具合予測モデルほど良いモデルであるといえる．そのことから，AUC は面積が大きいほど良いモデルであることを示すといえる．

4.2.2 再現率（Recall）

再現率（Recall）とは，真値が正であるもののうち，実際に正であると予測できたものの割合のことである．本実験では，不具合を含むコミットを，不具合ありと予測できた割合を指す．表 4.1 の凡例にのっとると，式 4.1 のように定義される．

$$Recall = \frac{TP}{TP + FN} \tag{4.1}$$

不具合予測において，再現率は不具合を未然に検出するという点から，重要な指標であるといえる．

4.2.3 適合率（Precision）

適合率（Precision）とは，正と予測したもののうち，実際に真値が正であった割合を指す．本稿の実験では，不具合を含むと予測したコミットが実際に不具合を含んでいた割合を指す．表 4.1 の凡例にのっとると，式 4.2 のように定義される．

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

適合率が低いということは，不具合を含まないコミットを不具合ありと予測する割合が高いということを示す．つまり，不具合を含まないコミットに対してのテスト工数を増やすことになりうるということを表す．適合率とは，不具合を発見するためのコストを示すための指標といえる．

4.2.4 F_1 値

F_1 値とは，適合率と再現率の調和平均である．再現率と適合率はトレードオフの関係であるため，両指標の総合的な評価するために F_1 値を用いる．この値が高けれ

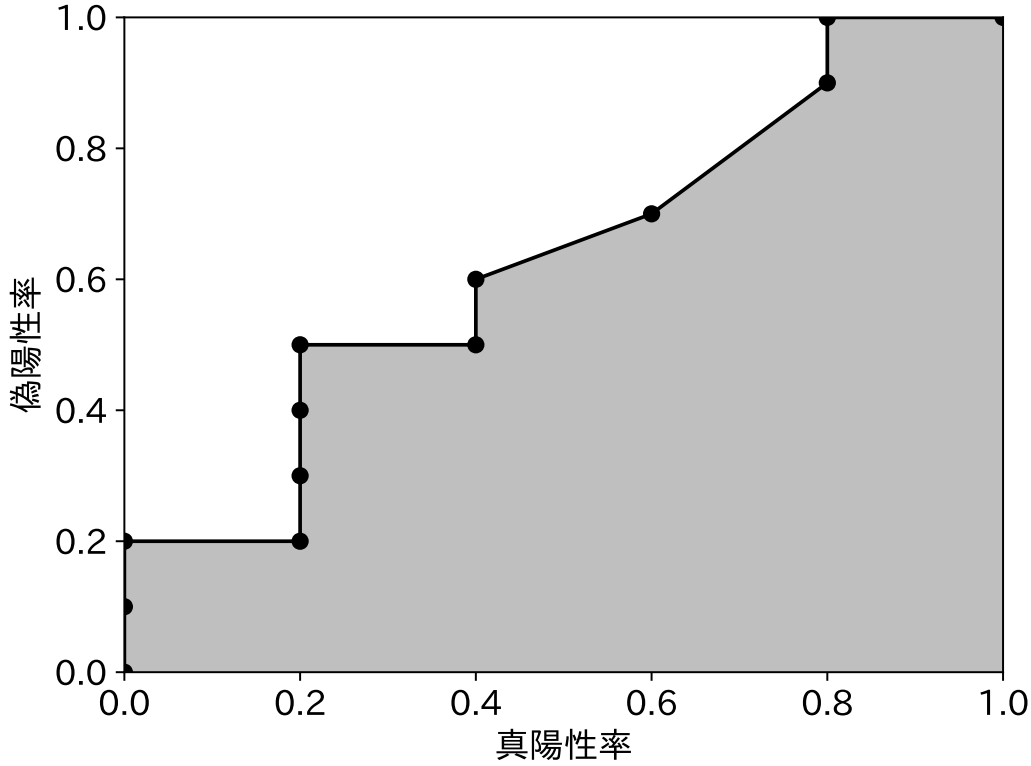

図 4.3　ROC 曲線と AUC の関係

表 4.1　評価の凡例

		実測	
		不具合を含む	不具合を含まない
予測	不具合を含む	True Positive（TP）	False Positive（FP）
	不具合を含まない	False Negative（FN）	True Negative（TN）

ば高いほど，予測精度が高いと言える．再現率と適合率を用いて式 4.3 のように定義される．

$$F_1 = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{4.3}$$

5. 実験結果

5.1 RQ1:リリース毎に構築した JIT DP モデルは既存の JIT DP モデルと比較して予測精度はどの程度か.

5.1.1 動機

時系列を考慮した不具合予測モデルは存在しているが，これまで述べたとおり，リリースを考慮した不具合予測モデルは存在していない．第 2.1 節の関連研究でも述べたが，Kamei ら [9] の研究では 3ヶ月，あるいは 6ヶ月でコミットを区切って不具合予測をしている．その結果一部のメトリクスの不具合に関する重要度が期間にしたがって変動したという結果が出ている．我々は，この変化をもたらした原因がリリースであり，リリース毎に異なる不具合コミットの特徴があると考え，リリース毎に学習データを分ける事で高い精度の不具合予測ができると仮定した．この仮定より，リリース毎に JIT DP を構築した場合と，Tr 以前のコミット全てを用いて JIT DP を構築した場合で，予測精度を比較調査した．

5.1.2 結果

実験は実験方法 1 を用いて行った．Hadoop を用いて比較実験をした結果を図 5.1 ～図 5.4，および表 5.1 に示す.

リリース毎に構築した JIT DP モデル（以降，JIT DP R）は既存の JIT DP モデル（以降，JIT DP）と比較すると，AUC においては第 1 四分位数以外の精度が劣っていた．表 5.1 を見ると，中央値では JIT DP R が約 51 %であるのに対し，JIT DP は約 63 %であった．また，第 3 四分位数は JIT DP R が約 54 %であったのに対し，JIT DP では約 67 %であった．Recall は，最小値と第 1 四分位数では劣っているものの，中央値と第 3 四分位数，最大値で JIT DP の精度を上回っていた．JIT DP R の第 1 四分位数は約 6 %であるのに対し，JIT DP では 25 %であった．Precision に関しては中央値と第 1 四分位数以外で精度が JIT DP を上回っていた．JIT DP R の中央値と第 1 四分位数は約 58 %と約 36 %であったが，JIT DP は約 54 %と 40 %であった．F_1 値は，JIT DP よりも最小値，中央値，および第 1 四分位数が劣っており，第 3 四分位数と最大値は優れていた．JIT DP R の中央値と第 1 四分位数は約 43 %と約 11 %

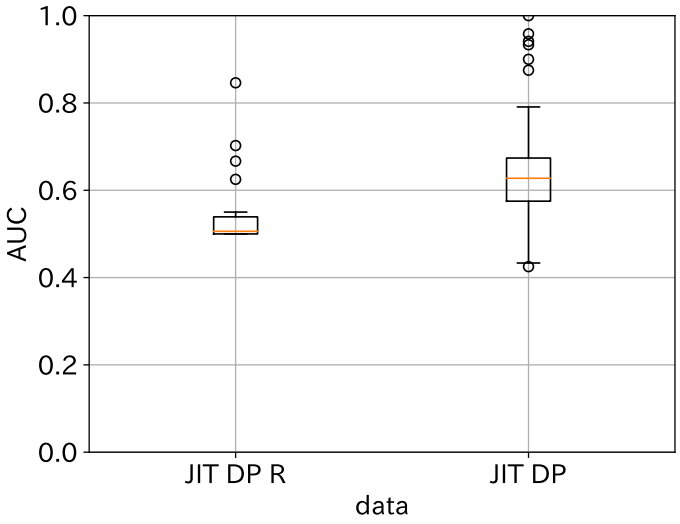

図 5.1　AUC の比較

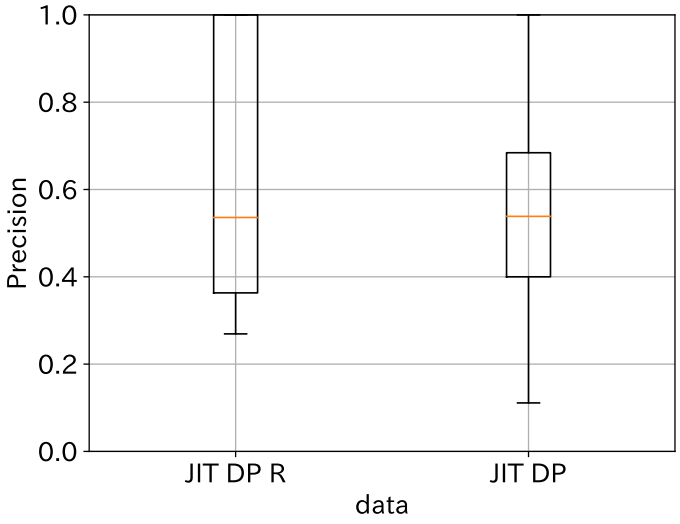

図 5.2　Precision の比較

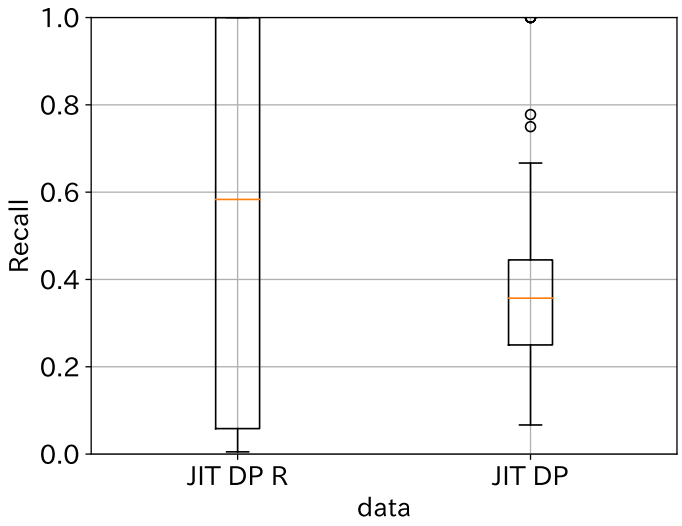

図 **5.3**　**Recall** の比較

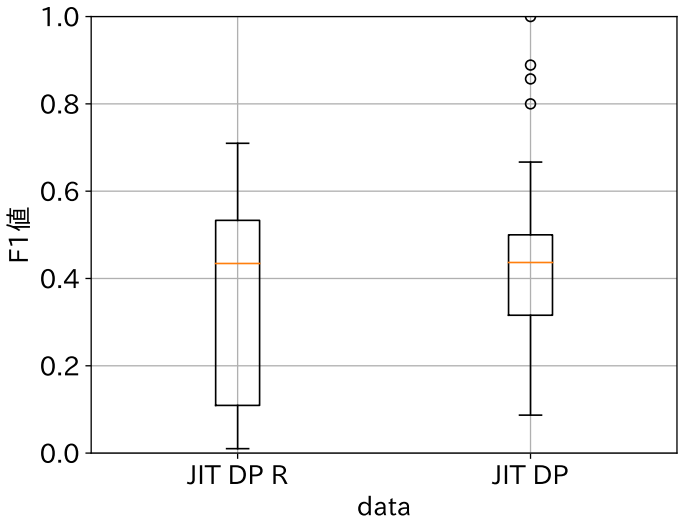

図 **5.4**　F_1 の比較

表 **5.1**　各評価値の詳細な数値

		JIT DP R	JIT DP
AUC	第 1 四分位数	0.50	0.58
	中央値	0.50	0.63
	第 3 四分位数	0.54	0.67
	四分位範囲	0.04	0.10
Precision	第 1 四分位数	0.36	0.40
	中央値	0.54	0.54
	第 3 四分位数	1.00	0.68
	四分位範囲	0.64	0.28
Recall	第 1 四分位数	0.06	0.25
	中央値	0.58	0.36
	第 3 四分位数	1.00	0.44
	四分位範囲	0.58	0.19
F1 値	第 1 四分位数	0.11	0.32
	中央値	0.43	0.44
	第 3 四分位数	0.53	0.50
	四分位範囲	0.42	0.18

表 **5.2**　不具合予測モデルの構築成功率

JIT DP R	JIT DP
9.95 %	85.52 %

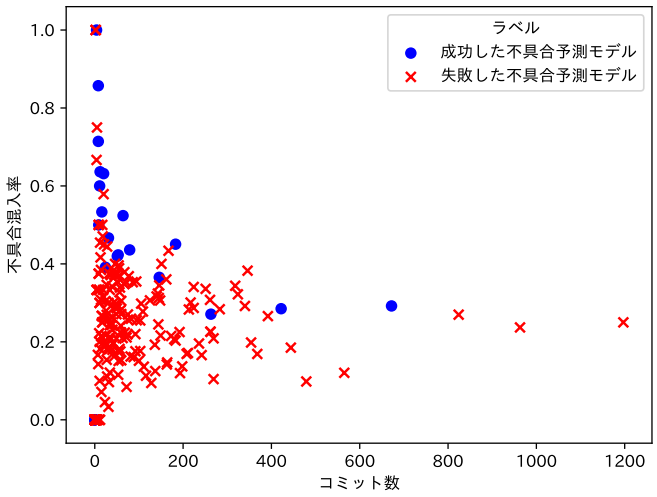

図 5.5　JIT DP R 構築の成否の散布図

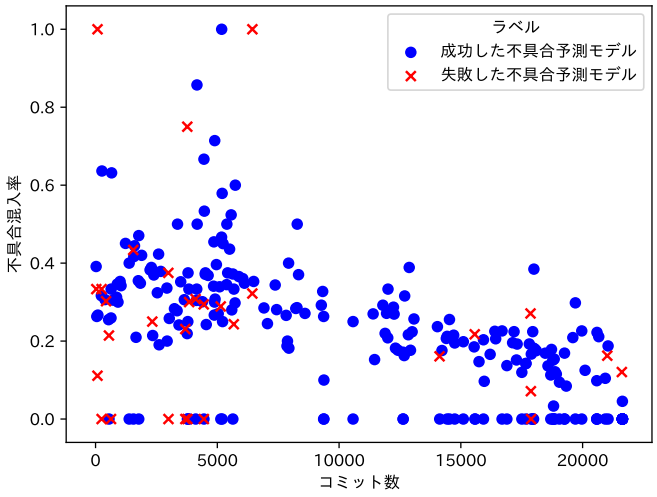

図 5.6　JIT DP 構築の成否の散布図

であったが，JIT DP は約 44 %と 32 %であった．

実験中に不具合予測モデルを構築できない場合が発生した．これは，不具合混入率が低い事が原因であると考えられる．それぞれの不具合予測手法において予測モデルの構築に成功した Tr と失敗した Tr を，リリース数と不具合混入率でプロットした散布図として図 5.5 と図 5.6 に示す．そして，それぞれ手法で予測モデルの構築に成功した割合を表 5.2 に示す．

予測モデルの構築成功率自体は JIT DP に比べると遥かに劣っており，JIT DP では約 85.52 %の Tr で不具合予測モデルの構築に成功していたにもかかわらず，JIT DP R では約 9.95 %の Tr でしか不具合予測モデルを構築できなかった．

図 5.5 より，JIT DP R は約 40 %以上の不具合混入率であるか，約 250 以上のコミット数があると，不具合予測モデルを構築できる可能性が高くなると考えられる．現実的には，コミット数の多いプロジェクトを用いることが，JIT DP R の構築には必要であると考えられる．また，図 5.6 より，構築に失敗している Tr はコミット数が約 5,000 以下の部分に多くが集まっている事が読み取れる．

5.2 RQ2:不具合修正コミットのタイミングまで考慮したリリース毎の JIT DP モデルの構築は可能か．

5.2.1 動機

RQ1 における実験では，不具合が修正されたタイミングを考慮していないため，Tr よりも未来のリリースに含まれているコミットで修正された不具合が Tr に含まれていた．これは，実環境での不具合予測をすることを考慮すると，不具合予測モデルを構築する段階で知る事ができない未来の情報を利用しており，不適切な訓練データで学習していることになる．そのため，実験 2 では不具合を修正したコミットが Tr 外である不具合を，不具合を含まないコミットとして扱い，不具合予測をした．

5.2.2 結果

実験は実験方法 2 を用いて行った．この実験においては，予測モデルの構築に成功した場合でも，全て不具合がないと予測したため，AUC， Precision，Recall，F_1 値は全て 0 になった．不具合が修正されたタイミングを考慮した不具合予測モデル

の詳細な情報を表 5.3 に示す．また，不具合修正コミットのタイミングまで考慮すると，約 80 %の不具合が Tr より未来のコミットで修正されていたため，不具合予測モデルの構築が難しいという結果となった．

表 5.3　不具合が修正されたタイミングを考慮した JIT DP R のデータ

項目	数値
全 Tr 数	253 個
JIT DP R が構築できなかった Tr 数	161 個
JIT DP R が構築できた Tr 数	92 個
不具合を含むコミット数	5,407 個
不具合を含まないとみなされた不具合を含むコミット数	4354 個
不具合を含むとみなしたコミット数	1,053 個
不具合とみなされたコミットの割合	19.47 %

6. 考察

6.1 RQ1：リリース毎に構築したJIT DPモデルは既存のJIT DPモデルと比較して予測精度はどの程度か.

リリースを考慮した不具合予測モデルはPrecisionとRecallにおいて第3四分位数以降で100 %の精度を出しており，既存の手法を一部で上回る精度を出している．しかし，F_1 値に関しては大きな差はなく，さらにAUCの最大値が既存手法と比べ下回っていることから精度に関しては既存手法の方が良いと考えられる．

また，図5.5と図5.6より，安定した不具合予測モデルの構築には約5,000以上のコミット数があれば良いと考えられる．しかし，図5.6をみると，不具合予測モデルの構築に失敗しているTrが15,000〜20,000個程度のコミット数付近にて確認できる．これらのTrは不具合の混入率がおおよそ2割以下あたりであることも読み取れる．つまり，一概にコミット数が5,000以上あれば不具合予測モデルが構築できるとは言い難い．

以上を踏まえ，RQ1に対して解答をする．

リリース毎に構築したJIT DPは一部分に関しては既存手法を超える予測精度を出す事ができるが，予測モデル自体の構築が難しいため，一概に優れているとは言い難い．

6.2 RQ2：不具合修正コミットのタイミングまで考慮したリリース毎のJIT DPモデルの構築は可能か.

不具合予測モデルの構築ができた場合でも，予測結果は全て不具合を含まないコミットと予測していた．これは，不具合予測に用いた不具合のデータが極端に少なくなったため，不具合を含むコミットの特徴を十分に抽出できなくなった事が原因と考えられる．RQ1の考察でも述べたが，不具合予測をするには十分なデータ量が必要と考えられる，

以上を踏まえ，RQ2に対して解答する．

不具合修正コミットのタイミングまで考慮したリリース毎のJIT DPモデルの構

築は，本実験で用いたデータでは一部で可能であったが，十分な予測をできなかった．不具合修正コミットのタイミングを考慮するのであれば，よりデータ数の多いプロジェクトを用いると，不具合予測ができると考えられる．

6.3 妥当性への脅威

本研究に存在する潜在的な妥当性への脅威に関して以下に示す．

6.3.1 構成概念妥当性

RQ1 で用いた評価値の内 Precision，Recall, および F_1 値は，実験対象となるデータの偏りの影響を受ける事がある．そのため，実験結果がこれに左右される可能性があることを考慮する必要がある [13]．Tantihamthavorn ら [13] によると閾値を使用しない評価指標を使うべきであり，AUC はこの基準を満たしている．

本実験では，不具合予測にはロジスティック回帰モデルを用いている．ロジスティック回帰モデルを選択した理由としては，既存手法でよく使用されているからである [9]．また，その他の手法を用いなかった理由としては，今回は予測モデルを使うことではなく，リリース毎に予測モデルを構築することにより予測精度が向上するかを確かめることを優先したためである．

検証手法に関して，交差検証などが一般的に用いられているが，時系列データに対して使用する事が難しく，また，ソフトウェアプロジェクト特有の性質を考慮する必要があったため，本実験では交差検証などを用いなかった．本実験においては時系列とリリースに重点を置いているため，我々独自の手法が妥当であると考えた．

本実験におけるリリースの分割は git の author date を用いて行った．具体的には，リリースとコミットの author date を比較し，リリースの author date よりもコミットの author date が過去のもので尚且つ前のリリースよりも後のものであれば，そのリリースに属するコミットとして分類した．しかし，この方法ではブランチを考慮せずに分割していることになり，厳密にはリリース毎に分けられていない可能性がある．ブランチとは，履歴の流れを分岐して記録していくためのものであり，本流（以下，master）とは別のコミットやリリースが記録される．つまり，master やブランチ毎に時系列やコミット，リリースが存在していることになり，それらを 1 つにま

とめて扱うことは時系列や不具合の混同などに繋がる．

本研究の実験データへのラベル付け，および変更メトリクスの計算にはCommit Guruを用いているが，ラベル付けの精度は完全ではない．具体的には特定のキーワードを含まない不具合修正コミットは見逃してしまう[10]．一方で，ソースコードが公開されており，Webアプリケーションが公開されていることから，実験の再現性が高く，Commit Guruの採用は妥当であると考える．

6.3.2 外的妥当性

本実験で用いたプロジェクトがHadoopのみである現状では，検証したプロジェクト数が不足おり，結果を全てのプロジェクトに一般化できない．

本実験で用いたデータセットはオープンソースソフトウェアから収集した．オープンソースソフトウェアにおけるデータの性質は，商用に開発されたプロジェクトによるものとは異なると考えられる．よって，今回実験に用いた手法が商用プロジェクトにおいても同様の結果を示すか実験をする必要がある．これは今後の課題となる．

6.3.3 内的妥当性

本実験で用いたスクリプトにも不具合が含まれている可能性が存在している．スクリプトの検証はしたが，実験結果の妥当性への脅威となりうる．

6.4 今後の課題

以上の考察を踏まえ，今後の課題について述べる．

- リリースの間隔が6ヶ月以上あるデータの規模がより大きなプロジェクトでの，リリース毎のJIT DPの検証．例を挙げると，eclipseプロジェクトはリリースが約1年おきに行われており，リリース毎での不具合予測に適しているのではないかと予想している．
- 構成概念妥当性でも触れたが，今回の実験ではブランチを考慮せずに時系列のみ考慮してリリース毎に分けている．しかし，実際にはブランチが異なれば，masterやその他のブランチと異なるリリースに属するコミットになっているはずである．ブランチ毎にリリースがなされているので，全てのブランチを1つ

にまとめてしまうことは，別のブランチに存在していない不具合を不具合として混同する事に繋がる．その点を考慮してコミットやリリースをブランチ毎に分けるべきと考えている．

7. 結言

本研究では，リリースを考慮した JIT DP のモデルを構築し，既存手法と精度を比較調査した．また，不具合が修正されたタイミングも考慮した不具合予測モデルを構築し，精度を検証した．実験の結果，リリースを考慮した JIT DP は既存手法に比べ予測精度が劣っており，その原因はデータ数が少ないことである可能性を示した．不具合が修正されたタイミングを考慮した不具合予測モデルでは，データ数の不足により予測モデルの構築自体が困難であった．

今後の課題は，本実験において予測精度低下の原因が，真にデータ数の不足であったのか詳細に検証する事である．具体的には，本実験で用いたプロジェクトよりもリリースの間隔が長く，よりデータ数があるプロジェクトを用いて同様の実験をする事が考えられる．また，本実験においてはブランチを考慮していなかったため，リリース毎に分割する際にブランチを考慮する事でコミットを正確に分割し，不具合予測の結果を検証することを今後行う．

謝辞

本研究を行うにあたり，研究課題の設定や研究に対する姿勢，本報告書の作成に至るまで，全ての面で丁寧なご指導を頂きました，本学情報工学・人間科学系水野修教授，崔恩瀞助教に厚く御礼申し上げます．本報告書執筆にあたり貴重な助言を多数頂きました，本学設計工学専攻 近藤将成先輩，西浦生成先輩，情報工学専攻 國領正真先輩，情報工学課程 山本凱君，杉浦智基君をはじめとする，ソフトウェア工学研究室の皆さん，学生生活を通じて著者の支えとなった家族や友人に深く感謝致します．

参考文献

[1] Q. Li, B. Li, and J. Liu, “Predicting developers’ contribution with developer networks and social network analysis,” Journal of Computational Information Systems, vol.7, no.15, pp.5553–5560, 2011.

[2] N. Bettenburg, M. Nagappan, and A.E. Hassan, “Think locally, act globally: Improving defect and effort prediction models,” Proceeding of the 2012 IEEE International Working Conference on Mining Software Repositories (MSR), pp.60–69, IEEE, 2012.

[3] M. Tan, L. Tan, S. Dara, and C. Mayeux, “Online Defect Prediction for Imbalanced Data,” Proceedings of the 2015 International Conference on Software Engineering (ICSE), pp.99–108, IEEE, 2015.

[4] S. McIntosh and Y. Kamei, “Are Fix-Inducing Changes a Moving Target? A Longitudinal Case Study of Just-In-Time Defect Prediction,” IEEE Transactions on Software Engineering, vol.44, no.5, pp.412–428, 2018.

[5] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, “Heterogeneous Defect Prediction,” IEEE Transactions on Software Engineering, vol.44, no.9, pp.874–896, 2018.

[6] Eclipse Foundation, Eclipse, （オンライン）, 入手先〈https://www.eclipse.org〉（参照 2020-02-11）.

[7] The Apache Software Foundation, Hadoop, （オンライン）, 入手先〈https://github.com/apache/hadoop〉（参照 2020-02-05）.

[8] Y. Kamei, T. Fukushima, S. McIntosh, K. Yamashita, N. Ubayashi, and A.E. Hassan, “Studying just-in-time defect prediction using cross-project models,” Empirical Software Engineering, vol.21, no.5, pp.2072–2106, 2016.

[9] 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 Transactions on Software Engineering, vol.39, no.6, pp.757–773, 2013.

[10] C. Rosen, B. Grawi, and E. Shihab, “Commit guru: Analytics and risk prediction of software commits,” Proceedingsof the 2015 10th Joint Meeting of the European

Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), pp.966–969, ESEC/FSE, 2015.

[11] scikit-learndevelopers, scikit-learn, （オンライン）, 入手先〈https://scikit-learn.org/stable/index.html〉（参照 2020-02-05）.

[12] scikit-learndevelopers, sklearn.linear_model.LogisticRegression, （オンライン）, 入手先〈https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html〉（参照 2020-02-05）.

[13] C. Tantithamthavorn and A.E. Hassan, "An experience report on defect modelling in practice: Pitfalls and challenges," Proceedings of International Conference on Software Engineering (ICSE), pp.286–295, ICSE, 2018.