修　士　論　文

題　　目　　ソフトウェア不具合予測における
類似開発者分類の効果

主任指導教員　　水野　修　教授

指導教員　　崔　恩瀞　助教

京都工芸繊維大学 大学院工芸科学研究科

情報工学専攻

学生番号　　18622019

氏　　名　　北村　紗也加

令和2年2月10日提出

様式 2 号

学位論文内容の要旨（和文）

令和 2 年 2 月 10 日

京都工芸繊維大学
大学院工芸科学研究科長　殿

大学院工芸科学研究科 情報工学専攻
平成 30 年入学
学生番号　18622019
氏　名　北村　紗也加　㊞

（主任指導教員　水野　修　㊞ ）

本学学位規則第 4 条に基づき、下記のとおり学位論文内容の要旨を提出いたします。

1. 論文題目

ソフトウェア不具合予測における類似開発者分類の効果

2. 論文内容の要旨（400 字程度）

ソフトウェアの不具合予測の 1 つとして，開発者個人に対する予測モデルを構築する研究が行われている．開発者個人に対して予測モデルを構築するため，予測の根拠を示しやすいなど利点がある．しかし，既存の研究では変更数の多い開発者のみにしか予測モデルを構築しておらず，変更数の少ない開発者の多いオープンソースソフトウェア（OSS）への応用には課題が残っている．

本研究では，類似した開発者をまとめることで変更数が少ない開発者に対しても予測モデルを構築し，OSS へも応用が可能な手法を検討する．評価実験によって，提案手法は，既存の開発者個人に対する予測モデルよりも 3 つのプロジェクトで高い予測精度が出せること，および，全ての開発者に対して 1 つの予測モデルを構築するモデルと比較して，2 つのプロジェクトで高い予測精度が出せることがわかった．

また，変更数が少ない開発者は，変更が多いファイルを変更している開発者のクラスタに分類されていることがわかったため，変更しているファイルに着目することで，より良い分類が行える可能性を示した．

Effectiveness of Grouping Similar Developer in Software Defect Prediction

2020 18622019 *KITAMURA Sayaka*

Abstract

Studies to construct a software defect prediction model for individual developers have been conducted. Such studies have advantages, e.g., reveal the evidence of prediction easily, since they build prediction models for individual developers. However, in prior studies, prediction models are built only for developers with a large number of changes. Hence, there is still a challenge in application to open source software (OSS) with many developers with a small number of changes.

In this study, we propose a prediction model for developers with a small number of changes by grouping similar developers and discuss approaches applicable to OSS. In evaluation experiments, we found that the prediction accuracy of the proposed prediction model outperforms one of the existing prediction models for individual developers in three projects. In addition the proposed prediction model performs better than one of the existing prediction models using all developers in two projects. It also turned out that developers with a small number of changes are grouped into the cluster that includes developers who modify files with a large number of changes. Hence, it is possible to group better by focusing on the files being changed.

目　次

1. 緒言

我々の生活においてソフトウェアの重要性は日々増加している．それに伴って，ソフトウェアの品質を高い水準に保つことが求められているが，品質保証活動がその一助となる．そして，品質保証活動の一つとして，不具合予測手法の研究が広く行われている [1, 2, 3, 4, 5, 6, 7, 8]．

多くの既存研究では，機械学習などの手法を用いて，ソフトウェアの開発情報から不具合予測モデルを構築し，将来的に不具合になるであろうファイルや変更などを予測している [2, 3, 4, 5, 6, 7]．この時，多くの研究では，取得可能な全ての情報から 1 つの予測モデルを構築することが行われている．

一方，Jiang ら [4] は，開発者個人に対して，それぞれ不具合予測モデルを構築し．その結果，既存の予測モデルよりも予測精度が向上したことを示している．しかし，この研究では，コミット数が 100 回以上の上位 10 人の開発者に実験対象を絞っており，コミット数が少ない開発者は研究対象外となっている．

しかし，近年において広く使用されているオープンソースソフトウェア (OSS) では，プルリクエストなどを利用することで気軽に開発に参加することができるためコミット数が少ない開発者が多く存在する．そういった開発者に対しても予測モデルを構築することが求められるが，コミット数が少ない開発者は予測モデルを構築するために必要な訓練データが不足するため，予測モデルの構築は難しくなる．

そこで，本研究では，類似した開発者をまとめることで，コミット数が少ない開発者に対しても予測モデルを構築し，予測精度を向上させることを目指す．具体的には，教師なし学習のクラスタリング手法であるスペクトラルクラスタリングを用いて開発者を分類し，それぞれのクラスタに対して予測モデルを構築する．以降，本稿では，上記の提案手法を **LPDP** (Light-Personalized Defect Prediction)，既存の開発者個人に対してそれぞれ不具合予測モデルを構築する手法 [4] を **PDP** (Personalized Defect Prediction)，全ての開発者に対して 1 つの不具合予測モデルを構築する手法を **GDP** (Global Defect Prediction) と呼ぶ．

そして，LPDP を評価するため，次の 3 つの研究設問を設定した．

RQ1 既存の開発者個人に対する不具合予測の精度は全開発者に対する不具合予測よりも優れているか？

RQ2 類似した開発者の分類は不具合予測の予測精度向上に寄与するか？

RQ3 分類された開発者のクラスタにはどのような特徴があるか？

これらの研究設問に回答するため，6 つのオープンソースソフトウェアを用いて，LPDP，PDP，及び GDP 間の比較評価実験と，LPDP で作成された各クラスタにおいて分類された開発者の特徴の分析を行った．その結果，次のことが結果として得られた．

1. PDP を全ての開発者に対して適用すると，3 つのプロジェクトで GDP よりも予想精度が悪くなる
2. LPDP は既存の PDP よりも 3 つのプロジェクトで予測精度が良い
3. LPDP は通常の GDP よりも 2 つのプロジェクトで予測精度が良い
4. LPDP では，コミット数が少ない開発者は特定のファイルをよく編集している開発者が多いクラスタに分類される．

本研究の主な貢献を次にまとめる．

1. 開発者個人に着目した不具合予測モデルを，実際の開発に近い環境で再評価した．
2. 類似した開発者をまとめることで，既存の開発者個人に着目した不具合予測モデルよりも予測精度を向上させ，また通常の予測モデルに近い精度が出せることを示した．
3. 類似した開発者をまとめ，不具合予測モデルを構築し評価するための手順をまとめ，また，その結果どのような分類が行われているかを初めて分析した．

以降の本稿の構成を紹介する．第 2 節では，類似した開発者を分類して不具合予測モデルを構築した動機を述べる．第 3 節では，不具合予測手法，及び開発者個人に着目した予測モデルについて関連研究を紹介する．第 4 節では，提案手法である LPDP について説明する．第 5 節では，評価実験について説明する．第 6 節では，それぞれの研究設問に対する結果をまとめる．第 7 節では，分類されたクラスタ数について考察を行う．第 8 節では，本研究の妥当性の検証を行う．第 9 節では，本研究の結論と今後の拡張について述べる．

2. 研究動機

本研究で分析を行うOSSの詳細なデータを表2.1に示す．表中のNは変更(コミット)数を表しており，それぞれコミット数が100未満の開発者の割合，及びその開発者のコミット数の割合をそれぞれ示している．

既存研究[4]では，研究対象をコミット数が多い上位10人に絞り，その対象からそれぞれ100個のコミットを取得して実験に使用している．

しかし，表2.1より，多くのOSSではコミット数が100回未満の開発者が90%前後を占めることがわかる．また，そのような開発者のコミット数は全てのOSSで10%，場合によっては20%を超えることがわかる．

この結果より，コミット数が少ない開発者が多く存在し，そのような開発者に対しても個別の予測モデルを構築する必要があると考えられる．そこで，類似した開発者をまとめることで，コミット数が少ない開発者にも対応するよう，本研究を開始した．なお，開発者の同定方法については，第8節にて詳細を述べる．

表 2.1　実験対象プロジェクトの詳細

プロジェクト	言語	変更数	不具合率	開発者数	$N < 100$ の開発者率	$N < 100$ の変更率
Bazel	Java	21,217	27.2%	567	89.6%	21.9%
Camel	Java	33,985	27.4%	622	95.7%	10.9%
Geode	Java	7,656	8.0%	158	87.3%	27.2%
Gerrit	Java	36,861	19.1%	399	92.2%	7.9%
Hadoop	Java	21,683	24.9%	299	78.3%	17.8%
OsmAnd	Java	54,386	12.9%	934	91.5%	14.3%

3. 関連研究

3.1 不具合予測手法

現在までに，様々な不具合予測手法が提案されている [1, 2, 9, 3, 4, 5, 6, 7, 10]．予測対象はファイルやパッケージ，各リリース，もしくは，各変更 (例えば，コミット) など様々な粒度で存在している．例えば，Kamei らの研究 [9] では，ソフトウェアの変更をメトリクス化して不具合予測を行なっている．一方で，Zimmermann らの研究 [7] では，ファイルもしくはパッケージのレベルで不具合予測を行なっている．

本研究では，コミットレベルでの不具合予測を行なった．この理由として，コミットレベルでの不具合予測は，次のような利点があるためである：

1. 変更直後に不具合が含まれていそうかがわかるため，素早いフィードバックが行える．
2. 変更直後であるため，どの開発者に修正を依頼すれば良いかがわかりやすい．
3. 変更を行なった開発者の記憶が新しく，書かれているコードを再度読む手間が最小限で済む．

また，予測を行う上でコミットを特徴量ベクトルとして表現しなければならない．コミットを特徴量ベクトルに変換するために，追加された行数などをメトリクスとして用いる手法と，ソースコードを用いる手法が存在する．ソースコードを用いる手法として，Mizuno らの研究 [1] では，ソースコードを学習データとし，スパムフィルタを用いて不具合予測を行なっている．本研究は，初めて開発者を分類して不具合予測を行うため，比較対象となり得る先行研究の数が少ない．そのため，より多くの研究で使用されている，変更メトリクス [9] を用いた手法を採用する．変更メトリクスとは，変更，つまりコミットの特徴をメトリクス化したものである．変更メトリクスの内容については，第 5.2 節にて詳細を述べる．

3.2 開発者個人に着目した不具合予測モデル

今までにも，開発者個人に着目することで，不具合予測モデルの精度を向上させようとする研究が行われている．例えば，Ostrand ら [8] はファイルやそのファイル

を変更している開発者に着目することで，不具合予測の精度を向上させようとしている．そして，何人かの異なる開発者が，ある 1 つのファイルを変更したかの情報が不具合予測に有用であることを発見している．Jiang [4] らは，開発者個人に対する不具合予測モデルを構築することで不具合予測の精度を向上させることができることを示している．ただし，Jiang らの研究はコミット数が多い上位 10 人の開発者に研究対象を絞っている．一方で，第 2 節で示しているように，OSS の開発においてはコミット数が少ない開発者が多く，これらの開発者に対しても不具合予測モデルを構築できるようになることが求められている．そこで，本研究では，類似した開発者を分類することで，コミット数が少ない開発者に対しても個別の不具合予測モデルを構築し，予測精度を向上させることを目指す.

3.3 スペクトラルクラスタリング

スペクトラルクラスタリングは，節点と枝を持つグラフ上で行われるクラスタリング手法である．各節点は実体 (分類したい要素)，各枝は実体間での類似度を表し，実体間の類似度に基づいて枝を削除しグラフを分割することで，指定した個数の非連結なグラフのクラスタを生成する．この手法は，不具合予測においても使用された実績がある [11]．そこで本研究では，類似度に基づいてデータを分割でき，実績もあるスペクトラルクラスタリングを開発者の分類に用いる．

4. 提案手法

本研究にて提案する，LPDP について説明する．LPDP は，類似した開発者をまとめ，それぞれのクラスタに対して予測モデルを構築する手法であり，以下に示す 3 つの手順に沿って進められる．

1. 変更メトリクスの前処理
2. 変更メトリクスを用いた開発者の分類
3. 予測モデルの生成

なお，ここで示す手順は，訓練データ，検証データ，テストデータから構成される 1 つの組に対して適用される．それらのデータの作成に関しては第 5.3 節にて述べる．

4.1 変更メトリクスの前処理

本研究では，表 4.1 に示す変更メトリクスを使用している．表中のサブシステムはルートディレクトリ，経験はそのコミット以前のコミット数を指している．これらのメトリクスは，コミットに対する不具合予測モデルにおいて頻繁に利用されている．

変更メトリクスには，Kamei ら [9] が行った前処理を行っている．具体的には，高い相関のあるメトリクスを除外，もしくは結合させることにより取り除く．ただし，得られたデータに負の値が含まれていたため，この研究では対数変換ではなく z-score を用いてメトリクスの標準化を行っている．

4.2 変更メトリクスを用いた開発者の分類

前処理を行った変更メトリクスを用いて，開発者の分類を行う．

本研究では，スペクトラルクラスタリング [12] を用いて開発者を分類している．スペクトラルクラスタリングにおける各節点の実体にそれぞれの開発者が該当し，各枝の類似度にはその開発者間の類似度となる．全ての開発者を含む類似度グラフを変更メトリクスより作成し，疎な枝を削除することで，指定した個数の非連結な

表 4.1　変更メトリクスの詳細

分類	名称	定義
Diffusion	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ごとの分布
Size	LA	追加されたコード行数
	LD	削除されたコード行数
	LT	変更される前のファイルのコード行数
Purpose	FIX	バグ修正の変更か否か
History	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	ユニークな変更の数
Experience	EXP	開発者の経験
	REXP	最近の開発者の経験
	SEXP	サブシステムについての開発者の経験

グラフのクラスタを作成する．そして，それぞれの連結グラフが類似した開発者のクラスタとなる．

スペクトラルクラスタリングを適用するために，開発者ごとに 1 つの特徴量ベクトルを作成する必要がある．特徴量ベクトルには変更メトリクスを用いるが，開発者ごとに複数のコミットが存在するため，その具体的な値も複数存在する．今回は，訓練データにおいて各開発者ごとにコミットをまとめ，変更メトリクスそれぞれの中央値を，その開発者の特徴量ベクトルとした．例えば，開発者 A が 10 回コミットを行っていた場合，それぞれのコミットにおいて前処理を行った変更メトリクスが計算される．この変更メトリクスそれぞれに対し，10 個全てのコミット間における中央値を計算し，それをこの開発者の特徴量ベクトルとする．

スペクトラルクラスタリングには，パラメータとして生成するクラスタ数が必要である．今回は，クラスタ数を 1 から 10 まで用意して，それぞれで予測モデルを構築し，検証データを用いて評価指標を計算し，最も精度が高かったものを選択した．具体的な値については，第 7 節にて結果を示す．

4.3 予測モデルの生成

分類されたそれぞれの開発者クラスタに対して，不具合予測モデルを作成する．ただし，予測モデルを学習するための訓練データおよび検証データには次の制約をつけることとする：

- 訓練データには 2 つ以上の不具合コミット，および 2 つ以上の正常なコミットのデータがそれぞれ含まれている．
- 検証データには 1 つ以上の不具合コミット，および 1 つ以上の正常なコミットのデータがそれぞれ含まれている．

検証データにおける制約は，不具合コミットもしくは正常なコミットのどちらかのデータが存在しない場合，モデルを評価する指標を計算できなくなるためである．訓練データにおける制約は，使用する予測モデルを構築するライブラリの制約により，2 つ以上としている．

訓練ができなかった開発者，およびテストデータにおいて初めて出てきた開発者に対しては，通常の不具合予測モデルである GDP を適用する．

予測モデルにはロジスティック回帰モデル [13] を使用する．しかし，Tantithamthavorn ら [14] によって，ロジスティック回帰モデルなどのモデルを実装するために使用可能な Python や R などのライブラリのデフォルトのパラメータを使用すると，最適なモデルを生成できず誤った結論を導く可能性が指摘されている．そこで，この研究ではロジスティック回帰モデルのパラメータである正規化項 C を検証データを用いて最適化している．方法はスペクトラルクラスタリングのクラスタ数を選択する方法と同様に，$C = \{0.001, 0.01, 0.1, 1, 10, 100\}$ を調査し，最も精度が高かったものを選択する．また，この値はクラスタ数とは独立したパラメータであるため，クラスタ数と正規化項の値の全ての組み合わせが調査される．なお，クラスタ数の考察（第 7 節）と異なり，正規化項の最適化は本研究の趣旨を違える予測モデルに関するパラメータであり，本論文の範囲を超えるため，具体的な数値に関する考察は行わない．

最後に作成された予測モデルをテストデータに対して適用し，評価値を計算して，予測モデルの精度を確かめる．

5. 評価実験

提案手法である LPDP 評価するために，既存の PDP および GDP との比較を行う．また，分類されたクラスタの分析も行う．実験の主要なステップを次にまとめる．

1. リポジトリからコミットを取得し，Commit Guru から得られるコミットが不具合を混入したかの情報を基に，コミットをラベル付けする．
2. Commit Guru から表 4.1 に示した 14 個の変更メトリクスを取得する．
3. 検証手法 (第 5.3 節) を用いて，データセットを複数の訓練データ，検証データ，テストデータから成る組に分割し，GDP モデルを作成する．
4. それぞれの訓練データ，検証データ，およびテストデータにおいて，開発者ごとにデータを分割し，PDP モデルを作成する．
5. スペクトラルクラスタリングを用いて開発者を分類し，提案手法である LPDP モデルを作成する．
6. 評価指標に基づいてそれぞれのモデルの予測精度を評価する．
7. スペクトラルクラスタリングによって分類されたクラスタ特徴を分析する．

以降，それぞれのステップについて詳細を説明する．

5.1 Commit Guru による分析データの準備

不具合予測手法において，実験データの公開性と透明性は重要である [15]．そのため，本研究では Rosen らによって公開されている Commit Guru [16, 9] から得られるデータを実験に利用する．Commit Guru は，Git のリポジトリを指定すると変更メトリクス [9]，および不具合混入コミットの情報を自動で計算可能な Web アプリケーションである．

本研究では，不具合混入コミットの情報より，不具合混入コミットと正常なコミットのラベル付けを行う．また，計算された変更メトリクスをコミットの特徴量ベクトル化に利用する．

Commit Guru の主な機能である，(1) 不具合混入コミットの特定およびラベル付けと (2) 変更メトリクスの収集について，次に述べる．

5.1.1 不具合混入コミットのラベル付け

Commit Guru は各コミットに対し，不具合を混入したコミット，およびそれ以外の不具合を混入していないコミットに分けてラベル付けをする．不具合混入コミットは，不具合を修正したコミットから推定される．まずコミットメッセージを解析し，Hindle らの研究 [17] によるコミットを分類するためのキーワードのうち，表 5.1 中の Corrective に該当するものがあれば，そのコミットを不具合を修正しているコミットと推定する．次に，その修正を行ったコミットの修正行を特定し，その行が追加されたコミットを不具合混入コミットとしてラベル付けを行う．

5.1.2 メトリクスの収集

収集されるメトリクスは 14 個の変更メトリクスである（表 4.1） [9]．これらのメトリクスを，コミットの特徴量ベクトルとして利用する．

5.2 対象とするプロジェクトデータ

本研究では，オープンソースソフトウェア (OSS) である Bazel プロジェクト, Camel プロジェクト, Geode プロジェクト, Gerrit プロジェクト, Hadoop プロジェクト, および OsmAnd プロジェクトの 6 つのリポジトリを利用する．プロジェクトのデータの詳細を表 2.1 に示す．比較的規模が大きく，時系列の解析がしやすいプロジェクトを利用している．

Bazel プロジェクト

ソフトウェアのビルドおよびテストの自動化ツール

Camel プロジェクト

エンタープライズ統合パターンに基づく統合フレームワーク

Geode プロジェクト

クラウドからアプリケーションへのアクセス管理プラットフォーム

Gerrit プロジェクト

Web ベースのチームコードコラボレーションツール

表 5.1　コミットを分類するためのキーワード

分類	キーワード	定義
Corrective	bug, fix, wrong, error, fail, problem, patch	不具合対応
Feature Addition	new, add, requirement, initial, create	新機能の実装
Merge	merge	コミットのマージ
Non Functional	doc	実装ではない要件
Perfective	clean, better	性能向上
Preventive	test, junit, coverage, asset	不具合のテスト

Hadoop プロジェクト

大規模データの分散処理フレームワーク

OsmAnd プロジェクト

Android および iOS 対象の地図ナビケーションアプリ

5.3 予測精度検証手法

予測精度検証手法として，訓練データとテストデータの選択によるバイアスがかからないように，交差検証やブートストラップ法などが適用されることが多い．一方で，コミットに対する不具合予測モデルを評価する際にこれらを利用すると，未来のコミットを訓練データ，過去のコミットをテストデータとして使うなど現実的でない組み合わせが発生する場合がある．また，不具合の発見・修正には時間がかかるため，プロジェクトの後半のデータは不具合があっても発見されていないために不具合であると判断されにくいなど，不具合予測固有の問題がある．そのため，不具合予測においては，時系列を考慮しつつ，不具合予測固有の問題に対応した検証手法を使用する必要がある [18].

そこで，時系列を考慮した稼働検証法 (online change classification) が Tan らによって提案されている [18]．これは，次の 3 点の問題を解決する検証手法である：

1. 不具合混入コミットは発見され，修正されるまで通常 100 日から 300 日程度かかるため，実際には不具合混入コミットであるにも関わらず正常であると判断されるコミットがデータに混入する．
2. 時系列の全データを 1 回だけ分割し，分割時以前のデータを訓練データおよび検証データ，分割時以降のデータをテストデータとして使用すると，分割の仕方によってデータ選択のバイアスがかかる恐れがある．
3. 訓練データおよび検証データとテストデータ間に大きな時間差があると，プログラマが変わるなどして，同じ分布から得られたデータであると仮定することが難しくなる．

本研究では，この検証手法に改良を加えたものを利用する．

1 の問題への対応としては，**間 g** を訓練データ・検証データとテストデータとの間に取ることで解決する．間は訓練データおよび検証データの不具合を発見するた

めの十分な時間を与え，より正確なラベル付けを可能にする．間には不具合混入コミットを修正するために要した時間の平均もしくは中央値を利用する．準備実験より，表 5.2 に示す値とした．

2 および 3 の問題への対応としては，訓練データ・検証データおよびテストデータを複数回更新しながらサンプリングすることで解決する．具体的には，訓練データ・検証データに用いるコミットの時間間隔 (**訓練間隔 Tr**)，およびテストデータに用いるコミットの時間間隔 (**テスト間隔 Te**)，そして上記で説明した間を過去から未来にずらしながら，データをサンプリングする．これにより，特定の訓練データ・検証データやテストデータから来るバイアスを回避する．また，複数回のサンプリングを行うことで，テスト間隔を短く取ることができ，訓練データ・検証データとテストデータの母集団の差異をできる限り小さくする．

それぞれの間隔はある一定のオフセット (**単位 u**) の日数分，未来に動くことを t 回繰り返すこととなる．今回は単位として 30 日を利用し，テスト間隔もこの単位と同じ 30 日とした．

この繰り返し回数 t，および訓練間隔は，分析対象のコミットデータ期間 p，最も新しいコミット日 d_l，分析を開始するコミット日 d_s，最低限利用する終端間の期間である安全間隔 m を用いて，次のような式で定義される：

$$p = (d_l - d_s) - m \tag{5.1}$$

$$t = \frac{\frac{p}{2} - g}{u} \tag{5.2}$$

$$Tr = t \cdot u \tag{5.3}$$

本実験では，安全間隔として 365 日を採用した．なお，単位はパラメータであるが，準備実験より，この単位の変化は不具合予測に大きな影響を与えないことを確認している．

さらに，本研究ではこれに加え，始端間および終端間を導入した．これはプロジェクトの始端と終端のコミットからどれだけのコミットをデータに利用しないかを表している．これらを導入した理由としては，プロジェクトの始端ではプロジェクト初期固有の変更が多くなると考えられ，プロジェクトの終端では不具合混入コミットとして同定されていないコミットが多いためである．

始端間は，そのプロジェクトにおけるコミット数が増加し，減少した後の日までとした．つまり，最初のコミットが多い時期が続く期間であり，この間でプロジェクトのソースコードが安定すると仮定している．そして，減少した後の日を d_s とし，これ以降のコミットを分析対象として用いる．最後に終端間 g_e は，次のような式で定義される：

$$d_e = d_s + (2 \cdot Tr + g) \tag{5.4}$$

$$g_e = d_l - d_e \tag{5.5}$$

ここでの訓練期間に入ったコミットが訓練データ・検証データとして利用される．これらのコミットは，訓練期間および間に含まれるデータを用いてラベル付けがされる．テスト期間に入っているデータを含めて，より未来にあるコミットはラベル付けには利用されない．

本実験において使用した稼働検証法を図 5.1，各パラメータの具体的な値を表 5.2 に示す．

5.4 予測モデルとパラメータ

第 4.3 節で示した通り，LPDP にはロジスティック回帰モデルを利用し，正規化項 C のパラメータを調整する．PDP および GDP においても同様の方法を取りロジスティック回帰モデルを利用するが，これらの場合にはスペクトラルクラスタリングを適用する必要がないため，最適化するパラメータはロジスティック回帰モデルの正規化項 C のみである．また，ここで構築された GDP に則った予測モデルは，提案手法の LPDP において，訓練できなかった開発者のグループおよびテストデータで初めて出てきた開発者に対して使用する予測モデルとしても利用される．

PDP の手順は提案手法である LPDP のクラスタ数が開発者数と同じである場合と等価である．ゆえに，コミット数が上位の 10 人に対して適用していた Jiang ら [4] の方法とは異なり，全ての開発者に対してモデルを作成することを試みる．PDP も LPDP と同様に，モデルを構築できなかった開発者に対しては GDP に則って構築されたモデルを適用する．

GDP の手順は，第 4.1 節にて述べた，提案手法の手順 1 の変更メトリクスの前処

表 **5.2**　実験に使用した稼働検証法におけるパラメータ値（日数）

プロジェクト	始端間	終端間	間	単位・テスト間隔	訓練間隔	繰り返し回数	安全間隔
Bazel	328	506	95	30	300	10	365
Camel	743	466	45	30	1,500	50	365
Geode	216	431	50	30	360	12	365
Gerrit	375	499	129	30	1,410	47	365
Hadoop	925	524	125	30	1,020	34	365
OsmAnd	1,011	395	5	30	930	31	365

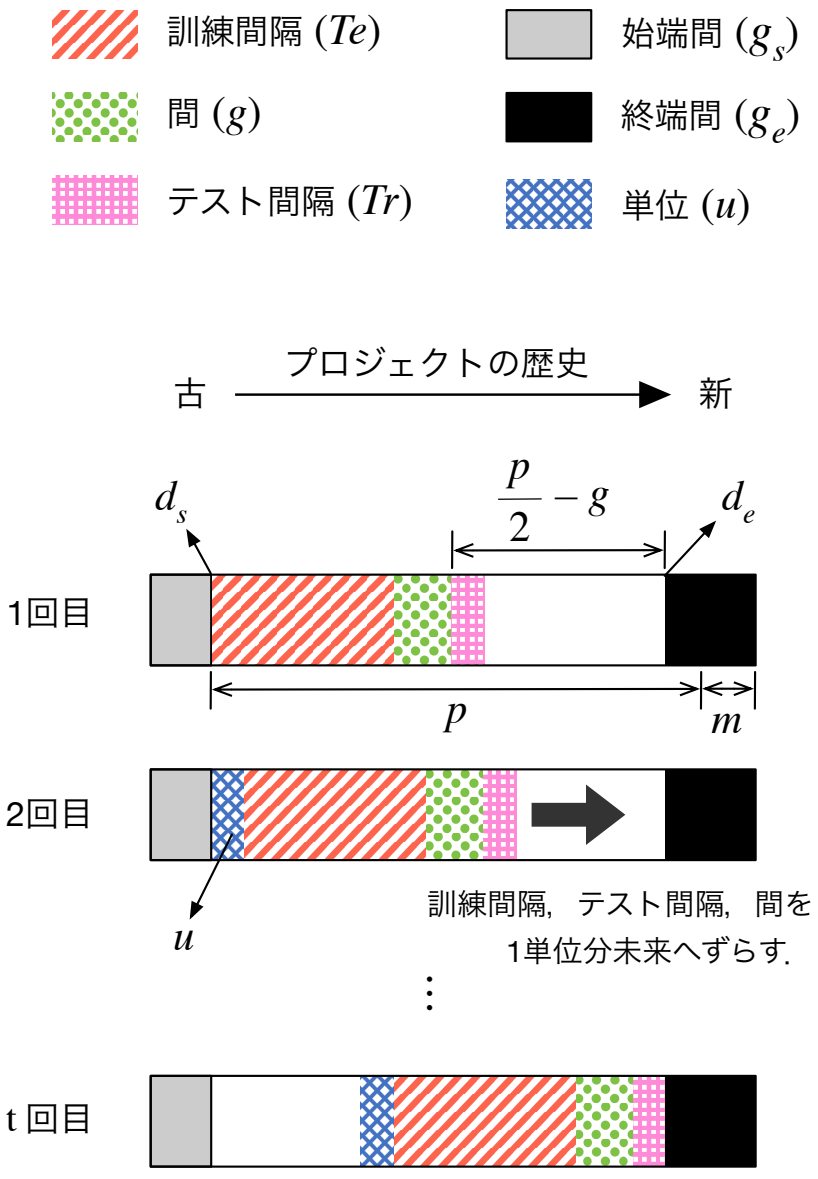
訓練間隔 (Te)
始端間 (g_s)
間 (g)
終端間 (g_e)
テスト間隔 (Tr)
単位 (u)
プロジェクトの歴史
古
新
d_s
$\frac{p}{2} - g$
d_e
1回目
p
m
2回目
u
訓練間隔，テスト間隔，間を
1単位分未来へずらす.
t 回目

図 5.1　稼働検証法

理までは同じである．その後，訓練データ全てを使ってロジスティック回帰モデルを作成し，検証データを用いて正規化項Cを評価する．最後に，最適な正規化項Cを使用して訓練したロジスティック回帰モデルをテストデータ上に適用し，その精度を評価する．

5.5 評価指標

評価指標には，次の6つの評価指標を用いる．これらのメトリクスを全てのテストデータに対して計算し，それぞれのプロジェクトで中央値を取ったものを最終的な予測精度とする．

AUC (Area Under the Curve)

ROC (Receiver Operating Characteristic) 曲線を作図した際の，グラフより下の面積部分．値は0から1までをとり，ランダムな予測では0.5，予測と実測が完全一致ならば1, 不完全一致ならば0となる．

適合率 (Precision)

陽性であると予測したもののうち，実際に陽性であるものの割合．値は0から1までをとり，1に近いほど性能が良いことを示す．真陽性をもつものをTP，偽陽性をもつものをFPとすると，適合率 Precision は次の式で求められる：

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.6}$$

再現率 (Recall)

実際に陽性であるもののうち，陽性であると予測したものの割合．適合率とはトレードオフの関係にある．値は0から1までをとり，1に近いほど性能が良いことを示す．真陽性をもつものをTP，偽陰性をもつものをFNとすると，再現率 Recall は次の式で求められる：

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.7}$$

F1 値

適合率と再現率の調和平均．値は0から1までをとり，1に近いほど性能が良

いことを示す．5.6 式における Precision と 5.7 式における Recall を用いて，F1 値 F-measure は次の式で求められる：

$$\text{F-measure} = \frac{2\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.8}$$

MCC (Matthews correlation coefficient; マシューズ相関関数)

2 値分類モデルにおいてその精度を測る尺度であり，不均衡データに耐性があるとの報告がある．値は-1 から 1 までをとり，ランダムな予測では 0，予測と実測が完全一致ならば 1，完全不一致であれば-1 となる．真陽性をもつものを TP，偽陽性をもつものを FP，真陰性をもつものを TN，偽陰性をもつものを FN とすると，MCC は次の式で求められる：

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{5.9}$$

Brier Score

ある事柄の予測確率とその結果がどれだけ乖離しているかを示す指標である．値は 0 から 1 までをとり，0 に近いほど精度が良いことを示す．データ数 N のデータにおける t 番目のデータに対し，予測した事柄の発生確率を f_t，実際の結果を o_t とすると，Brier Score は次の式で求められる：

$$\text{BrierScore} = \frac{1}{N}\sum_{t=1}^{N}(f_t - o_t)^2 \tag{5.10}$$

$$\text{ただし}, o_t = \begin{cases} 0(\text{発生しなかった場合}) \\ 1(\text{発生した場合}) \end{cases}$$

6. 結果

6.1 RQ1: 既存の開発者個人に対する不具合予測の精度は全開発者に対する不具合予測よりも優れているか？

6.1.1 動機

開発者個人に対する不具合予測モデルを評価した既存研究は存在するが，ここまでで述べてきたように，実験対象をコミット数が上位の開発者のみに絞るなど制約が多い．その一方で，OSS においてはコミット数が少ない開発者が 90%前後を占めており，そのような開発者に対して既存研究をそのまま適用すると，実験対象外のために効果が得られない可能性が存在する（第 2 節）．そこで，開発者個人に対する不具合予測モデルを全ての開発者に適用し，開発者個人に対する不具合予測モデルがコミット数が少ない開発者にとっても効果があるのかを分析する．

6.1.2 アプローチ

PDP および GDP を比較することによって，開発者個人に対する不具合予測モデルが通常の不具合予測モデルよりも優れているのかどうか分析する．PDP および GDP の手順については，第 5.4 節にて説明している．

6.1.3 結果

PDP，GDP の予測精度として得られた AUC の中央値を表 6.1 に示す．太文字は PDP，および GDP において最も精度の高いものを表す．

表 6.1 から，全体的に見ると，PDP は Geode プロジェクト，および OsmAnd プロジェクトの 2 つにおいて，GDP より高い精度を達成した．その一方で，半数以上である 4 つのプロジェクトにおいては精度が下がった．

この結果より，既存の開発者個人に着目した不具合予測が全開発者に対する不具合予測より優れているのは一部の場合に留まっており，改善の余地が存在する．

表 6.1　各評価指標の中央値 (PDP, GDP)

	評価指標	Bazel	Camel	Geode	Gerrit	Hadoop	OsmAnd
PDP	AUC	0.713	0.750	**0.755**	0.836	0.759	**0.890**
	F1 値	0.138	**0.158**	**0.101**	0.271	0.240	**0.240**
	適合率	0.492	**0.577**	**0.072**	**0.615**	0.616	**0.526**
	再現率	0.080	0.091	**0.099**	0.165	0.156	0.155
	MCC	0.106	**0.176**	**0.002**	0.260	0.263	**0.226**
	Brier Score	0.195	0.139	**0.068**	0.123	0.143	**0.078**
GDP	AUC	**0.725**	**0.760**	0.675	**0.851**	**0.778**	0.887
	F1 値	**0.156**	0.156	0.000	**0.353**	**0.275**	0.231
	適合率	**0.512**	0.575	0.000	0.579	**0.667**	0.433
	再現率	**0.091**	**0.091**	0.000	**0.242**	**0.179**	**0.164**
	MCC	**0.123**	0.167	0.000	**0.313**	**0.305**	0.204
	Brier Score	**0.187**	**0.136**	0.070	**0.117**	**0.140**	0.081

6.2 RQ2: 類似した開発者の分類は不具合予測の予測精度向上に寄与するか？

6.2.1 動機

RQ1 より，開発者個人に着目した不具合予測をコミット数が少ない開発者に対して適用すると，半数以上のプロジェクトにおいて，予測精度が悪化することがわかった．以上より，いくつかのプロジェクトにおいては，既存の研究とは異なった方法で，コミット数が少ない開発者に対する不具合予測モデルの構築を考慮することが必要であると考える．そこで，類似した開発者を分類することにより，不具合予測の精度を向上させることができるのかを検証する．

6.2.2 アプローチ

提案手法である LPDP と，PDP，および GDP の予測精度を比較する．これにより，類似した開発者をまとめることで，不具合予測の精度を向上させることができるかを検証する．LPDP の手順については，第 4 節にて説明している．

6.2.3 結果

LPDP，PDP，および GDP の予測精度として得られた AUC の中央値を表 6.2 に示す．下線部は LPDP，および PDP において最も精度が高いもの，太文字は各予測モデルにおいて最も精度の高いものを表す．

表 6.2 より，LPDP と PDP のみを比較すると，Bazel プロジェクト，Gerrit プロジェクト，Hadoop プロジェクト，および OsmAnd プロジェクトの 4 つにおいて，LPDP は PDP よりも高い精度を達成したと言える．また，LPDP，PDP，GDP の 3 つを比較すると，Hadoop プロジェクト，および OsmAnd プロジェクトの 2 つにおいて，LPDP は最も高い精度を達成したと言える．

この結果より，類似した開発者を分類することは，開発者個人に着目した不具合予測において，予測精度の向上に寄与する可能性がある．分類手法などにはまだ改善の余地があり，それらをより詳細に詰めていくことで，より良い LPDP が構築できる可能性がある．

表 6.2　各評価指標の中央値 (PDP, GDP)

	評価指標	Bazel	Camel	Geode	Gerrit	Hadoop	OsmAnd
LPDP	AUC	**0.725**	0.749	0.671	**0.851**	**0.781**	**0.892**
	F1 値	0.109	0.143	0.000	0.352	**0.293**	**0.262**
	適合率	**0.546**	0.525	0.000	0.579	**0.669**	0.486
	再現率	0.060	0.082	0.000	**0.242**	**0.190**	**0.190**
	MCC	0.102	0.165	0.000	**0.313**	0.269	**0.250**
	Brier Score	0.192	0.138	0.071	**0.117**	**0.140**	**0.077**
PDP	AUC	0.713	0.750	**0.755**	0.836	0.759	0.890
	F1 値	0.138	**0.158**	**0.101**	0.271	0.240	0.240
	適合率	0.492	**0.577**	**0.072**	**0.615**	0.616	**0.526**
	再現率	0.080	0.091	**0.099**	0.165	0.156	0.155
	MCC	0.106	**0.176**	**0.002**	0.260	0.263	0.226
	Brier Score	0.195	0.139	**0.068**	0.123	0.143	0.078
GDP	AUC	**0.725**	**0.760**	0.675	**0.851**	0.778	0.887
	F1 値	**0.156**	0.156	0.000	**0.353**	0.275	0.231
	適合率	0.512	0.575	0.000	0.579	0.667	0.433
	再現率	**0.091**	**0.091**	0.000	**0.242**	0.179	0.164
	MCC	**0.123**	0.167	0.000	**0.313**	**0.305**	0.204
	Brier Score	**0.187**	**0.136**	0.070	**0.117**	**0.140**	0.081

6.3 RQ3: 分類された開発者のクラスタにはどのような特徴があるか?

6.3.1 動機

RQ2 より，開発者を分類することで，開発者に着目した不具合予測の予測精度を向上させることができる可能性を示した．一方で，どのように分類しているのかについては分析していない．同じクラスタに分類されている開発者の特徴を分析することで，どの特徴が分類する際に注目されているのか，そして，なぜ分類することで予測精度が向上するのかについて調査を行う．

6.3.2 アプローチ

ここでは，2 つの点を調査する：

1. コミット数が少ない開発者 (以降，**一時開発者**と呼称) は同じクラスタに分類されるのか
2. 一時開発者が多いクラスタの特徴は何か

なお，本実験において，一時開発者のコミット数の閾値は 100 未満とした．この理由として，第 2 節において述べたように，コミット数が 100 未満の開発者は既存研究の対象外の開発者であり，本研究の目的に最も沿った開発者であるためである．

1 を調査するために，稼働検証法におけるそれぞれの繰り返しにおいて，各クラスタにおいて，どれだけの一時開発者が存在するのかを分析する．それぞれのクラスタにおいてこの割合をまとめ，一時開発者が同じクラスタに分類されることが多いのか，それとも別のクラスタに分類されることが多いのかを調べる．

2 を調査するために，各クラスタの特徴をさらに詳しく分析する．具体的には，一時開発者が一番多く分類されたクラスタ（**一時開発者クラスタ**）と，そうではないクラスタ間で，変更メトリクスのどのメトリクスに差があるのかを調査する．一時開発者クラスタとそれ以外のクラスタの全ての変更メトリクスをそれぞれまとめて，箱ひげ図で表示し，その差を調べる．

6.3.3 結果

(1) クラスタにおける一時開発者の割合

表 6.3 は，各繰り返しにおける一時開発者が最も多いクラスタ（一時開発者クラスタ）とそれ以外のクラスタにおける，クラスタ内の一時開発者，および一時開発者の人数の中央値である．なお，クラスタ数が 1 の場合における繰り返し，つまり GDP と等価であるモデルを構築するデータは省いているため，そのデータの割合も示している．なお，GDP と等価であるモデルを構築するデータに対する議論については，第 7.1 節にて行う．

全てのプロジェクトにおいて，一時開発者は，約 70〜90%が同じクラスタに分類され，それ以外の一時開発者は 0〜5 人程度の規模で別々のクラスタに分類されることが多い傾向にある．それに対し，一時開発者ではない開発者は（コミット数が 100 以上の開発者）は，一時開発者クラスタに分類されることが多い．一時開発者が等分割されるような分類が行われている繰り返しも見られたが，その頻度はごくわずかであった．

また，小規模なクラスタへの分類に際し，そのクラスタ数はプロジェクトによって大きく異なる．Camel プロジェクトはクラスタ数が多いが，OsmAnd プロジェクトや Bazel プロジェクトは少ないクラスタ数となっている．

この結果より，全プロジェクトにおけるほとんどの繰り返しで，一時開発者の大半は大規模な同じクラスタに分類され，残りの開発者は小規模なクラスタに分類されることがわかった．しかし，小規模なクラスタの数は，プロジェクトごとに大きく異なる．

(2) クラスタの変更メトリクスの特徴

図 6.1 は，Camel プロジェクトにおいて，一時開発者と他の開発者で差が存在した変更メトリクスの 1 つである EXP（開発者の経験，表 4.1）の箱ひげ図である．“target” とされている箱ひげ図が，一時開発者クラスタの EXP をまとめたものであり，“other” とされている箱ひげ図がそれ以外のクラスタの EXP をまとめたものである．一時発者のクラスタの方が高い値を示しており，他の全てのプロジェクトでも同様の結果が得られた．これは SEXP（サブシステムについての開発者の経験，

表 6.3　各クラスタにおける一時開発者数の中央値およびその割合

プロジェクト	一時開発者クラスタ			それ以外のクラスタ				除外データ
	開発者数	一時開発者数		開発者数	一時開発者数		クラスタ数	
Bazel	130.00	99.00	(76.15%)	2.00	2.00	(100.00%)	2.0	40.00%
Camel	112.00	95.00	(84.82%)	2.00	2.00	(100.00%)	8.0	26.00%
Geode	42.50	32.00	(75.29%)	3.25	2.75	(84.62%)	3.5	8.33%
Gerrit	120.00	103.50	(86.25%)	3.00	3.00	(100.00%)	4.0	17.02%
Hadoop	94.00	53.00	(56.38%)	3.00	2.00	(66.67%)	4.0	41.18%
OsmAnd	350.50	287.00	(81.88%)	1.00	1.00	(100.00%)	1.0	64.52%

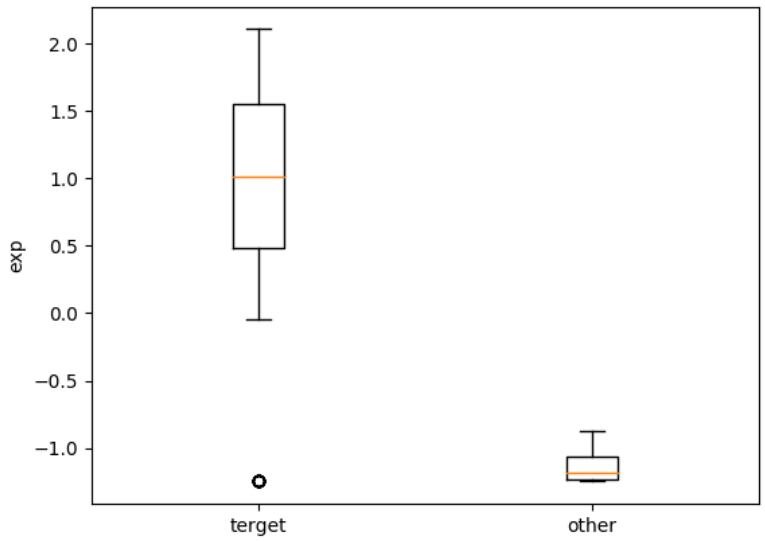

図 6.1　ある繰り返し実行における EXP の値 (Camel プロジェクト)

表 4.1）でも同様の結果であった．全プロジェクトにおいて，一時開発者クラスタはそれ以外のクラスタに比べ，EXP，および SEXP の値が比較的高くなった．EXP と SEXP はそれぞれ，そのコミットでその開発者が変更をしたファイルを，以前にその開発者が変更した回数，もしくは，そのサブシステムを変更した回数より計算される．このことから，大半の一時開発者は，頻繁に修正が行われているファイルを変更している開発者が多いクラスタに分類されていると考えられる．これは OSS においては，変更数が少なく情報が少ない箇所よりも，頻繁に更新され情報が手に入りやすいファイルの方が情報が多く，プルリクエストを作成しやすいためであると考えられる．

しかしながら，この一時開発者クラスタには一時開発者ではない開発者，つまりコミット数が多い開発者も同様に多数分類されている．また，一時開発者クラスタ以外のクラスタに分類されている開発者は，そのほとんどが一時開発者であることがわかる．よって，EXP，および SEXP の値が多少高い，もしくは他のメトリクスに特徴がない場合，一時開発者クラスタに分類されることが考えられ，開発者を十分に分類できていない，という可能性が存在する．

以上より，大半の一時開発者の分類では，どの開発者がどのファイルをどれだけ変更してきたかの情報が影響を与えていることがわかった．よって，変更したファイルに着目することで，より良い分類が行える可能性がある．一方で，それは同時に開発者を十分に分類できていない可能性も示すものであり，分類を再検討する必要性を示すものである．

7. 考察

7.1 クラスタ数の最適化

本研究において，スペクトラルクラスタリングによって開発者を分類することは重要な要素である．今回の実験では，開発者を最大で 10 個の異なるクラスタに分類する．その際に，それぞれの繰り返し実行において検証データを用いて予測を行い，最も良い精度を達成したクラスタ数が最終的に選択される．そこで，最も良い精度を達成したものとして選択されたクラスタ数を具体的に分析する．

表 7.1 に，最も良い精度を達成したとして実際に選択されたクラスタ数を示す．それぞれの列がクラスタ数に対応しており，それぞれの行がプロジェクトにおいてそれぞれのクラスタ数を最適値として実行した繰り返し実行数を示している．例えば Bazel プロジェクトでは，全繰り返し実行 10 回のうち，クラスタ数 1 を最適値として実行した繰り返し実行は 4 回である．

全プロジェクトを通して見ると，3 分の 1 程度の繰り返し実行において，全開発者を 1 つのクラスタに分類しており，プロジェクト単位で見ても，クラスタ数を 1 とした繰り返し実行が最も多かったプロジェクトが半数以上を占める．しかし，複数のクラスタへと分類する傾向はプロジェクトごとに大きく異なることが分かる．

各プロジェクトにおいて，各繰り返しにおけるクラスタ数の傾向は大きく 2 つに分かれる：

1. **最適なクラスタ数が.**

   Bazel プロジェクト，Hadoop プロジェクト，OsmAnd プロジェクトでは，最適なクラスタ数として小さい値が選択された繰り返し実行数が多い傾向が見られた．実際に，最適なクラスタ数として 5 以下が選ばれた繰り返し実行では，Bazel プロジェクト，Hadoop プロジェクトでは 8 割程度，OsmAnd プロジェクトでは 9 割以上となっている．

2. **最適なクラスタ数がデータによって大きく異なる傾向.**

   Camel プロジェクト，Geode プロジェクト，Gerrit プロジェクトでは，最適なクラスタ数が繰り返し実行によって大きく異なる傾向である．例として，

1 に当てはまる Bazel プロジェクト，Hadoop プロジェクト，OsmAnd プロジェクト

表**7.1**　各繰り返し実行において，最も良い精度を達成したとして選択されたクラスタ数

プロジェクト	クラスタ数										合計
	1	2	3	4	5	6	7	8	9	10	
Bazel	4	1	2	1	0	2	0	0	0	0	10
Camel	13	4	5	3	1	1	1	3	8	11	50
Geode	1	1	0	5	1	1	1	2	0	0	12
Gerrit	8	0	13	5	6	0	2	5	4	4	47
Hadoop	14	6	2	1	4	5	1	1	0	0	34
OsmAnd	20	7	0	3	0	0	1	0	0	0	31
合計	60	19	22	18	12	9	6	11	12	15	184

の 3 つは，稼働検証法における訓練期間に含まれる開発者の変更メトリクスが近い値を取るために，多くの開発者が同じクラスタに分類されたと考えられる．

2 に当てはまる Camel プロジェクト，Geode プロジェクト，Gerrit プロジェクトの 3 つは，稼働検証法における訓練期間に含まれる開発者の変更メトリクスが繰り返し実行ごとにばらつきがあるため，分類されるクラスタ数にも差が生じた可能性が考えられる．

以上の結果より，開発者が複数のクラスタに分類されるケースは半分以上となった．また，最適値として選択されたクラスタ数の範囲は，プロジェクトによって大きく異なった．この原因として，各繰り返し実行において用いられる訓練データにおける開発者の分布が関係していると考えられる．

7.2 実際に構築されたモデル

第 7.1 節より，各プロジェクトにおけるクラスタ数の傾向は異なることがわかった．しかしながら，十分に分類されたと考えられていても，LPDP が PDP，および GDP よりも劣る結果になるプロジェクトが存在する．一方で，十分に分類されていないと考えられていても，LPDP が最高精度を達成するプロジェクトも存在する．LPDP は，訓練できなかったクラスタや，データの制約条件に反するクラスタなどは GDP で予測を行う．そこで，本節では，実際に分類されたクラスタに対しどれだけのモデルが構築されたのかを調査する．

表 7.2 に，全クラスタ数のうち，モデルが構築できなかったクラスタの割合と平均コミット数，およびその原因を示す．表中の原因において,「不足」はデータが存在しなかったことを,「制約」は制約条件 (第 4.3 節) に反したことを意味する．本研究は，訓練データを用いてクラスタを作成し，その時の訓練間隔，およびテスト間隔に含まれるコミットのうち，そのクラスタに存在する開発者のものを取得して，検証データ，およびテストデータとしている．そのため，訓練データにおいて存在していても，検証データ，テストデータには存在しない開発者のみでクラスタが構成されている場合，検証データ，テストデータが不足する．訓練データの不足は，分類するデータにおける変更メトリクスの値が全て一致するもの（重複データ）が存在しており，非重複データ数が指定したクラスタ数より少なくなった場合に起きる

表 **7.2**　モデルを構築できなかったクラスタ

プロジェクト	該当クラスタ			訓練データ		検証データ		テストデータ	
	クラスタ数		平均コミット数	不足	制約	不足	制約	不足	制約
Bazel	15	(53.6%)	1.53	0	15	14	0	12	3
Camel	207	(76.1%)	6.56	0	154	194	4	176	13
Geode	38	(66.7%)	26.29	0	24	20	11	19	6
Gerrit	177	(78.0%)	3.08	0	154	106	9	168	7
Hadoop	57	(56.4%)	29.96	0	27	40	1	40	8
OsmAnd	21	(39.6%)	147.23	1	17	13	2	15	6

ものである．なお，モデルが構築できなかった原因は，訓練データ，検証データ，およびテストデータにおいてそれぞれ独立に計測しているため，全てを足すとモデルを構築できなかったクラスタ数を超える場合がある．

表 7.2 より，モデルを構築できなかったクラスタ数はプロジェクトによって大きく異なる．OsmAnd プロジェクトは全体の 4 割程度で一番少なく，Camel プロジェクトと Gerrit プロジェクトは全体の 8 割程度であり，他の 3 つのプロジェクトでは 5 割から 6 割程度となった．また，モデルが構築できなかったクラスタ数の平均コミット数は Bazel プロジェクト，Camel プロジェクト，Gerrit プロジェクトは 5 程度の値となり，Geode プロジェクト，Hadoop プロジェクトも 30 程度の値となった．よって，モデルが構築できなかったクラスタは訓練データのコミット数が小さく，100 程度のコミット数でもモデルが構築できない場合がある．これより，研究目的であるコミット数が少ない開発者に対してもモデルを構築できるような分類ができていない可能性が存在する．また，実際に構築できなかった原因としては，検証データおよびテストデータの不足，訓練データの制約に依るものが多く，これは全プロジェクトに等しく見られるものである．

以上より，プロジェクトによって数値は異なるものの，LPDP で分類したクラスタの半数程度ではモデルが構築できていないことがわかった．モデルが構築できないクラスタのコミット数にも違いが見られるが，その値は決して大きいものではなく，研究目的に沿うような分類ができていない可能性が存在する．また，モデルが構築できない原因は主に検証データとテストデータの不足，および訓練データの制約に依るものである．

7.3 GDP との比較

LPDP は，2 つのプロジェクトで最も良い予測精度を達成したものの，他のプロジェクトでは予測精度は GDP よりも劣っている．本節では，その原因を考察する．

LPDP は GDP と比較すると，その予測精度は Bazel プロジェクト，Camel プロジェクト，Geode プロジェクトで多少劣っており，Gerrit プロジェクトではほぼ同等である．そして，この精度の差は LPDP において構築したモデルに起因するものである．LPDP は，分類されたクラスタに対してモデルを構築するが，モデルが構築できな

かったクラスタに対してはGDPを用いて予測を行う．よって，LPDPとGDPとの予測精度の差は，クラスタに対して構築されたモデルにおいて生じたものである．複数のクラスタが生成され，訓練データを分割して複数のモデルが構築される場合，LPDPは必然的にGDPよりも訓練データが少なくなる．これにより十分にモデルが訓練できず，LPDPがGDPよりも予測精度が低下した可能性がある．本手法では，そのような場合においてもLPDPの予測モデルが構築に成功すればそれが採用されるため，結果としてLPDPがGDPよりも低い精度を示すプロジェクトが見られたと考えられる．ただし，常にLPDPの予測モデルがGDPより低い制度になるとは限らない．実際に第7.2節より，Hadoopプロジェクトでは5割程度，OsmAndプロジェクトでは6割程度でLPDPの予測モデルの構築に成功し，GDPよりも評価指標の中央値が高い．一方，Bazelプロジェクトでは，5割程度でLPDPの予測モデルの構築に成功したが，GDPの方が評価指標の中央値が高い．よって，LPDPは全てのプロジェクトに対して有効ではない可能性があると考えられる．

7.4 GDPモデルの予測制度について

本研究では，提案手法との比較対象としてGDPモデルを用いている．しかし，そのGDPのモデルは先行研究 [9] と比較すると予測モデルの精度が大きく下がっている．先行研究における予測モデルは，GDPと同じ前処理を行なった変更メトリクスを用いて，ロジスティック回帰モデルを用いて予測モデルを構築しているため，本研究においてGDPモデルと等価である．よって，予測モデルの予測精度の差は検証手法の違いに起因するものである．

本研究では稼働検証法を用いて評価しているが，先行手法では10–分割交差検証を用いている．稼働検証法では訓練期間を定式化したため，2分割したデータから間を引いたコミットが訓練データとして用いられる．その一方，10–分割交差検証ではデータを10分割し，そのうちの1つをテストデータ，残りを訓練データとしている．そのため，訓練データに含まれるコミット数が大きく異なり，先行研究では18,000から54,000程度のコミットを用いてモデルを訓練しているが，本研究では100程度から12,000程度のコミットを用いてモデルを構築している．この訓練データ数の差が，モデルの予測精度に差が生じた原因であることが考えられる．ただし，こ

の稼働検証法は第 ??節でも述べたように，実運用を考慮した検証手法であり，テストデータに未来のコミットを含むことは無い．訓練期間の増加を考慮する必要はあるが，そして，今回の LPDP はモデルの訓練できない等，GDP を使う場合が存在する．上記のように，GDP は妥当であるが，その評価精度の低さが，LPDP の予測制度にも影響を与えていることが考えられる．

8. 妥当性の検証

8.1 構成概念妥当性

8.1.1 予測精度の評価指標

本研究では，予測精度の評価指標として AUC の他，再現率，適合率，F1 値，MCC，Brier Score を用いている．不具合予測でよく指標される他の評価指標として，適合率や再現率などが挙げられるが，これらの指標は結果にバイアスを混入させる恐れがあると報告されている [19, 20, 21]．Tantithamthavorn ら [20] によって閾値を使用しない評価指標を用いるべきであると指摘されており，AUC はこの基準を満たすものである．一方で，不具合データセットのような不均衡データにおける場合，AUC のみで予測精度の評価を判断することは結果にバイアスを混入させる恐れがあるため，他の評価指標も用いている．

なお，本実験において，閾値として 0.5 を採用した．これは先行研究 [9] によって採用されており，0 から 1 の範囲で値とる予測値としての閾値としては妥当であると考えられる．

8.1.2 検証手法

本実験では，時系列データであるコミットを扱うため，Tan ら [18] によって提案されている手法を改良して，提案手法を評価している．交差検証などが一般的に使用される手法であるが，時系列データに対して使用することは難しい．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，妥当な検証手法であると考える．

8.1.3 開発者の同定

本研究において，開発者の同定に利用したのは Commit Guru が取得した各コミットにおける開発者名である．これは，Git によってコマンドである `git log --pretty=format:"%an"` によって得られる名前であるため，例えば，同じ開発者が異なるアカウントでコミットをした場合が存在した場合など，同一の開発者であっても同定することに失敗する場合が存在する．今後，追加の分析を行う必要がある．

8.2 外的妥当性

本実験では 6 つの OSS を選択肢，それらから収集したコミットを実験データとして利用している．選択したプロジェクトは複数の分野を考慮しているが完全ではなく，規模も比較的大きいものが多いため，今後対象のプロジェクトを増やす必要がある．また，選択されたプロジェクトは全て Java によって書かれた OSS である．これは，今後クラス間の依存性解析などを行う際にツールが豊富であり，分析も容易である，などの理由から妥当な制約であると考える．

8.3 内的妥当性

本研究では，コミットのラベル付けに際して不具合混入コミット推定手法を用いており，その中でも Commit Guru [16] を使用している．他の不具合混入コミットを特定する一般的な方法として SZZ アルゴリズム [22] があるが，SZZ アルゴリズムは公開されている信頼性の高いライブラリなどがなく，それぞれの研究で独自の実装が試みられている．Commit Guru はソースコードが公開されており，また Web アプリケーションが利用可能であるという点から，実験の再現性が高く，今後の応用も容易であるため，本研究では SZZ アルゴリズムではなく Commit Guru を利用した．

9. 結言

本研究では，類似した開発者を分類することによって，開発者個人に対する不具合予測の精度を高めることができるかを検証した．その結果，既存の開発者個人に対する不具合予測モデルと比較し，3つのプロジェクトで良好な予測精度を達成した．また，従来の全てのデータを使った予測モデルと比較すると，2つのプロジェクトで予測精度が改善した．

本研究には，まだ多くの改善の余地がある．具体的には，次の4つが挙げられる．

1. 変更メトリクスのみならず，変更されたソースコードをベクトル化したものをコミットの特徴量として追加する．
2. 開発者の特徴量に，今までにその開発者がどのファイルを主に変更してきたのか，また，どれだけの期間このプロジェクトに関わっているのかの情報を追加する．
3. 開発者の特徴量を中央値を用いて表現しているが，代表値を使わない手法，例えば制約付きボルツマンマシンなどの内部状態を学習する手法などを使う．
4. 開発者の分類にスペクトラルクラスタリングのみならず，他の分類手法を用いる．

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，並びに崔 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，本学情報工学専攻近藤 将成先輩，西浦 生成先輩，情報工学専攻 國領 正真君，塩津 拓真君をはじめとする，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] O. Mizuno and T. Kikuno, "Training on errors experiment to detect fault-prone software modules by spam filter," Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE), pp.405–414, ACM, 2007.

[2] S. Kim, E.J. Whitehead Jr, and Y. Zhang, "Classifying software changes: Clean or buggy?," IEEE Transactions on Software Engineering, vol.34, no.2, pp.181–196, 2008.

[3] L. Aversano, L. Cerulo, and C. Del Grosso, "Learning from bug-introducing changes to prevent fault prone code," Proceedings of the 9th International Workshop on Principles of Software Evolution (IWPSE), pp.19–26, ACM, 2007.

[4] T. Jiang, L. Tan, and S. Kim, "Personalized defect prediction," Proceeding of the 28th International Conference on Automated Software Engineering (ASE), pp.279–289, IEEE, 2013.

[5] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, "Deep learning for just-in-time defect prediction," Proceedings of the 2015 Software Quality, Reliability and Security (QRS), pp.17–26, IEEE, 2015.

[6] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," Proceedings of the 38th International Conference on Software Engineering (ICSE), pp.297–308, ACM, 2016.

[7] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," Proceedings of the 3th International Workshop on Predictor Models in Software Engineering (PROMISE), p.9, IEEE, 2007.

[8] T.J. Ostrand, E.J. Weyuker, and R.M. Bell, "Programmer-based fault prediction," Proceedings of the 6th International Conference on Predictive Models in Software Engineering (PROMISE), p.19, ACM, 2010.

[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, June 2013.

[10] J. Li, P. He, J. Zhu, and M.R. Lyu, "Software defect prediction via convolutional neural network," Proceedings of the 2017 Software Quality, Reliability and Security (QRS), pp.318–328, IEEE, 2017.

[11] F. Zhang, Q. Zheng, Y. Zou, and A.E. Hassan, "Cross-project defect prediction using a connectivity-based unsupervised classifier," Proceedings of the 38th International Conference on Software Engineering (ICSE), pp.309–320, ACM, 2016.

[12] U. Von Luxburg, "A tutorial on spectral clustering," Statistics and computing, vol.17, no.4, pp.395–416, 2007.

[13] J.H. McDonald, Handbook of Biological Statistics (3rd ed.)., Sparky House Publishing, Baltimore, Maryland., 2014.

[14] C. Tantithamthavorn, S. McIntosh, A.E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," Proceedings of the 38th International Conference on Software Engineering (ICSE), pp.321–332, ACM, 2016.

[15] Y. Kamei and E. Shihab, "Defect prediction: Accomplishments and future challenges," in proceeding of the 23th International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol.5, pp.33–45, IEEE, 2016.

[16] C. Rosen, B. Grawi, and E. Shihab, "Commit guru: Analytics and risk prediction of software commits," Proceedings of the 10th Joint Meeting on Foundations of Software Engineering and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), pp.966–969, ACM, 2015.

[17] 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 (MSR), pp.99–108, ACM, 2008.

[18] M. Tan, L. Tan, S. Dara, and C. Mayeux, "Online defect prediction for imbalanced data," Proceedings of the 37th International Conference on Software Engineering

(ICSE), pp.99–108, IEEE, 2015.

[19] D. Bowes, T. Hall, and D. Gray, "Comparing the performance of fault prediction models which report multiple performance measures: recomputing the confusion matrix," Proceedings of the 8th International Conference on Predictive Models in Software Engineering (PROMISE), pp.109–118, ACM, 2012.

[20] C. Tantithamthavorn and A.E. Hassan, "An experience report on defect modelling in practice: Pitfalls and challenges," Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP ' 18), p.To Appear, 2018.

[21] D. Chicco, "Ten quick tips for machine learning in computational biology," BioData mining, vol.10, no.1, p.35, 2017.

[22] J. Śliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?," Sigsoft Software Engineering Notes, vol.30, no.4, pp.1–5, ACM, 2005.