NAIST-IS-DD0861018

博士論文

プロダクトメトリクスと開発者メトリクスを用いたソフトウェア信頼性予測モデルの構築方法

柗本 真佑

2010年2月4日

奈良先端科学技術大学院大学
情報科学研究科 情報システム学専攻

本論文は奈良先端科学技術大学院大学情報科学研究科に
博士 (工学) 授与の要件として提出した博士論文である。

柗本 真佑

審査委員：

松本 健一 教授	（主指導教員）
関 浩之 教授	（指導教員）
飯田 元 教授	（指導教員）
門田 暁人 准教授	（指導教員）

プロダクトメトリクスと開発者メトリクスを用いたソフトウェア信頼性予測モデルの構築方法*

柗本 真佑

内容梗概

大規模化と複雑化が進むソフトウェア開発において，限られたリソースの中で高い信頼性を確保するためには効率的なテストの実施が必須である．その一つの手段は，信頼性の低い，すなわち fault（欠陥）を含むソフトウェアモジュールを推定し，テストに費やすリソースを適切に割り当てることである．これまで，過去の開発モジュールの fault 検出傾向に基づき，新規開発モジュールに含まれる fault の有無を判別する fault-prone モジュール判別モデルに関する研究が数多く実施されてきた．

本論文では，fault-prone モジュール判別の精度向上を目的として，モデル構築に用いられるフィットデータに関する 2 つの問題に取り組んだ．第 1 の問題は判別精度低下の原因となる外れ値の存在であり，第 2 の問題はモデル構築のための説明変数における人的要因の欠如である．本論文の具体的な成果は以下の通りである．

(1) 外れ値除去法適用の効果

一般に fault-prone モジュール判別モデル構築のためのフィットデータとしては，過去のプロジェクトで開発されたモジュールの全てが用いられる．しかし，フィットデータ中には他の標本の傾向と異なる外れ値が含まれており，この外れ値が判別モデルの精度を低下させる原因となることが指摘されている．このため，外れ値はあらかじめフィットデータから除去したうえで判別モデルを構築することが

* 奈良先端科学技術大学院大学 情報科学研究科 情報システム学専攻 博士論文, NAIST-IS-DD0861018, 2010 年 2 月 4 日.

望ましい．本論文では，3 つの外れ値除去法（MOA，LOFM，RBM）と新たに提案する外れ値除去法（CC-MOA）を代表的な 3 つの fault-prone モジュール判別モデルに適用し，その効果を実験的に比較した．実験の結果，LOFM を除く 3 つの外れ値除去法を用いた場合，いずれの判別モデル，データセットに対しても判別精度が改善され，F1 値の平均向上幅は MOA では 0.139，RBM では 0.137，CC-MOA では 0.123 であった．

(2) 開発者メトリクスの提案

従来，モデル構築の説明変数として用いられるメトリクスとしては，ソフトウェアプロダクトの特徴を表すメトリクスが広く用いられている．しかし，fault が混入される要因としてはプロダクトの特性のみならず，人的要因，すなわちプロダクトを作成した開発者の特性が強く影響すると考えられる．fault の有無を目的変数とする判別モデルとしては，fault 混入の原因となる要因は可能な限り説明変数として加えることが望ましい．そこで本論文では開発者に関するメトリクス（開発者メトリクス）を提案する．提案する開発者に関するメトリクスは，開発者個々の変更行数やコミット回数など，及びモジュール個々の開発に関わった開発者の数などである．実験では，まず fault の増加に対して開発者の特性がどの程度関連しているかについて分析を行い，次に fault-prone モジュール判別に対して開発者メトリクスを加えた際の効果を確かめた．Eclipse プロジェクトから収集されたモジュールデータを用いた分析の結果，開発者の fault 混入率には 5 倍以上の個人差があること，及び多くの開発者が変更を加えたモジュールほど fault が混入されやすいことが明らかとなった．また開発者に関する情報が判別モデルの精度向上に有効であることを確認し，その F1 値の平均向上幅は 0.093 であった．

キーワード

fault-prone モジュール判別，外れ値除去法，開発者メトリクス

Building a Software Reliability Prediction Model Using Product Metrics and Developer Metrics *

Shinsuke Matsumoto

Abstract

To meet today's growing demands for high quality, large scale but short life-cycle software, effective software testing is the fundamental requirement in software development. One of the promising strategies for effective testing is to optimally allocate testing resources to software modules likely to contain faults. So far, various studies have been conducted on fault-proneness models that classify software modules into either fault-prone or non-fault-prone based on module metrics.

The goal of this dissertation is to improve the prediction performance of fault-proneness models by resolving two problems related to a fit dataset used for model construction. The first problem is the existence of outliers that cause performance degradation of the model, and the second problem is the lack of human factors in independent variables in prediction models. Achievements of this dissertation are as follows.

(1) Applying outlier detection methods: In general, all modules developed in a past project are used as a fit dataset for constructing a fault-proneness model. However, the fit dataset usually includes outlier modules, which can decrease the prediction performance of the model. Therefore, outliers should be detected and

* Doctoral Dissertation, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0861018, February 4, 2010.

removed from the fit dataset before the model construction. This dissertation experimentally evaluated the existing three outlier methods as well as one new outlier method proposed in this study by applying them to three fault-proneness detection models. The result showed that three out of the four outlier detection methods (excluding LOFM) improved F1-values of all fault-proneness models for all datasets. The average improvements of F1-value by MOA, RBM and CC-MOA were 0.139, 0.137 and 0.123 respectively.

(2) Employing developer metrics in fault-proneness models: Software product metrics have been widely used as independent variables for the model construction. However, fault introduction is likely to be affected not only by software product features but also by software developers who created the software. From this perspective, the prediction model should include human factor metrics as well as conventional software product metrics. This dissertation proposes developer metrics such as the number of code churns made by each developer, the number of commits made by each developer and the number of developers for each module. By using the eclipse project dataset, we experimentally analyzed the relationship between the number of faults and developer metrics. Second, the effective of developer metrics for performance improvements of fault-proneness models were evaluated. The result revealed the wide variations (more than five times) in fault introduction rate of developers, and the modules touched by more developer contained more faults. Compared with conventional fault-proneness models, developer metrics improved the prediction performance and the average improvement of F1-value was 0.093.

Keywords:

fault-proneness detection, outlier detection, developer metrics

関連発表論文

第 2 章に関連する発表論文

学術論文誌

1. 柗本 真佑, 亀井 靖高, 門田 暁人, 松本 健一. Fault-prone モジュール判別における外れ値除去法の比較. 情報処理学会論文誌, Vol. 49, No. 3, pp. 1341-1351, 2008.

国際会議

1. Shinsuke Matsumoto, Yasutaka Kamei, Akito Monden, and Ken-ichi Matsumoto. Comparison of Outlier Detection Methods in Fault-Proneness Models. *Proc. International Symposium on Empirical Software Engineering and Measurement*, pp. 461-463, 2007.

国内会議（査読なし）

1. 柗本 真佑, 亀井 靖高, 門田 暁人, 松本 健一. Fault-prone モジュール判別モデルに対する外れ値除去法の適用効果. 情報処理学会研究報告 ソフトウェア工学, Vol. 2007-SE-155, No. 33, pp. 49-56, 2007.

第 3 章に関連する発表論文

学術論文誌

1. 柗本 真佑, 亀井 靖高, 門田 暁人, 松本 健一. 開発者メトリクスに基づくソフトウェア信頼性の分析. 電子情報通信学会論文誌, (投稿中).

国内会議（査読あり）

1. 柗本 真佑, 亀井 靖高, 門田 暁人, 松本 健一. 開発者メトリクスを用いたソフトウェア信頼性の分析. ソフトウェア工学の基礎 XVI, 日本ソフトウェア科学会, Vol. 35, pp. 207-214, 2009.

その他の発表論文

学術論文誌

1. 柗本 真佑, 上野 秀剛, 門田 暁人, 松本 健一, 片平 真史, 神武 直彦, 宮本 祐子, 氏原 頌悟, 吉川 茂雄. 不具合履歴に基づくソフトウェア IV&V 活動の定量的見える化手法. 電子情報通信学会論文誌, Vol. J92-D, No. 12, pp. 2195-2206, 2009.

2. 柿元 健, 門田 暁人, 亀井 靖高, 柗本 真佑, 松本 健一, 楠本 真二. Fault-Prone モジュール判別におけるテスト工数割り当てとソフトウェア信頼性のモデル化. 情報処理学会論文誌, Vol. 50, No. 7, pp. 1716-1724, 2009.

3. 柗本 真佑, 亀井 靖高, 大平 雅雄, 松本 健一. OSS コミュニティにおけるオープンコラボレーションの理解. 情報社会学会学会誌, Vol. 3, No. 2, pp. 29-42, 2009.

4. 亀井 靖高, 柗本 真佑, 柿元 健, 門田 暁人, 松本 健一. Fault-Prone モジュール判別におけるサンプリング法適用の効果. 情報処理学会論文誌, Vol. 48, No. 8, pp. 2651-2662, 2007.

国際会議

1. Masao Ohira, Kiwako Koyama, Akinori Ihara, Shinsuke Matsumoto, Yasutaka Kamei and Ken-ichi Matsumoto. A Time-Lag Analysis Toward Improving the Efficiency of Communications among OSS Developers. *Proc. International Workshop on Knowledge Collaboration in Software Development*, pp. 49-62, 2009.

2. Mizuki Yamamoto, Masao Ohira, Yasutaka Kamei, Shinsuke Matsumoto and Ken-ichi Matsumoto. Temporal Changes of the Openness of an Oss Community: A Case Study of the Apache Http Server Community. *International Conference on Collaboration Technologies*, pp. 64-65, 2009.

3. Masateru Tsunoda, Shinsuke Matsumoto, Akito Monden and Ken-ichi Matsumoto. A Proposal of Derivation Methods of Fp Transformation Formulas. *Workshop on Accountability and Traceability in Global Software Engineering*, pp. 25-26, 2008.

4. Tsutomu Matsumoto, Shinsuke Matsumoto, Hidetake Uwano, Akito Monden, Yuko Miyamoto, Shogo Ujihara, Naohiko Kohtake and Masafumi Katahira. Cost Effective IV&V Planning Activity Derived from Experiences on Jaxa's Spacecraft Projects. *ESA Workshop on Avionics Data, Control and Software Systems*, 2008.

5. Yasutaka Kamei, Shinsuke Matsumoto, Hirotaka Maeshima, Yoji Onishi, Masao Ohira and Ken-ichi Matsumoto. Analysis of Coordination between Developers and Users in the Apache Community. *Proc. International Conference on Open Source Systems*, pp. 81-92, 2008.

6. Shinsuke Matsumoto, Yasutaka Kamei, Masao Ohira and Ken-ichi Matsumoto. A Comparison Study on the Coordination between Developers and Users in Foss Communities. *Socio-Technical Congruence* , CD-ROM, No. 8, 2008.

7. Yasutaka Kamei, Akito Monden, Shinsuke Matsumoto, Takeshi Kakimoto and Ken-ichi Matsumoto. The Effects of Over and Under Sampling on Fault-Prone Module Detection. *Proc. International Symposium on Empirical Software Engineering and Measurement*, pp. 196-204, 2007.

国内会議（査読あり）

1. 小山 貴和子, 伊原 彰紀, 柗本 真佑, 亀井 靖高, 大平 雅雄, 松本 健一. OSS開発における情報交換の効率改善へ向けたタイムラグ分析手法の提案. 情報処理学会シンポジウム グループウェアとネットワークサービス研究会, Vol. 2009, No. 8, pp. 81-86, 2009.

2. 柿元 健, 亀井 靖高, 枩本 真佑, 門田 暁人, 松本 健一, 楠本 真二. バグ予測で信頼性はどれだけ向上するのか？ -テスト工数割り当ての観点からの従来研究の評価. ソフトウェア工学の基礎 XV, 日本ソフトウェア科学会, Vol. 34, pp. 63-68, 2008.

3. 大平 雅雄, 枩本 真佑, 伊原 彰紀, 松本 健一. オープンメディアを活用した知識コミュニティのデザインに関する一考察. 情報社会学会 知識共有コミュニティワークショップ —インターネット上の知識検索サービス研究—, pp. 1-10, 2008.

4. 伊原 彰紀, 大平 雅雄, 枩本 真佑, 亀井 靖高, 松本 健一. 複数のサブコミュニティを有する OSS コミュニティを対象としたネットワーク分析. マルチメディア, 分散, 協調とモバイル シンポジウム, pp. 297-303, 2008.

5. 柿元 健, 門田 暁人, 亀井 靖高, 枩本 真佑, 松本 健一. Fault-Prone モジュール判別における F1 値とソフトウェア信頼性の関係. ソフトウェア工学の基礎 XIV, 日本ソフトウェア科学会, Vol. 33, pp. 75-83, 2007.

6. 木浦 幹雄, 枩本 真佑, 亀井 靖高, 門田 暁人, 松本 健一. 異なるプロジェクト間における Fault-Prone モジュール判別の精度評価. ソフトウェア工学の基礎 XIV, 日本ソフトウェア科学会, Vol. 33, pp. 131-136, 2007.

7. 前島 弘敬, 枩本 真佑, 亀井 靖高, 柿元 健, 大西 洋司, 大平 雅雄, 松本 健一. コーディネータのコミュニティ媒介性の評価指標の提案. 情報処理学会シンポジウム グループウェアとネットワークサービス研究会, Vol. 2007, No. 11, pp. 71-76, 2007.

国内会議（査読なし）

1. 佐々木 辰也, 森崎 修司, 枩本 真佑, 松本 健一. レビュー指摘の記録における支援ツールの効果分析. 情報処理学会研究報告 ソフトウェア工学 (SIGSE), Vol. 2009-SE-166, No. 10, 2009.

2. 藏本 達也, 柗本 真佑, 亀井 靖高, 門田 暁人, 松本 健一. メトリクス値の標準化による Fault-Prone モジュール判別モデルの精度向上. 情報処理学会研究報告 ソフトウェア工学 (SIGSE), Vol. 2009-SE-166, No. 11, 2009.

3. 伊原 彰紀, 大平 雅雄, 柗本 真佑, 松本 健一. OSS の開発状況理解支援のための可視化手法の提案. 情報処理学会シンポジウム グループウェアとネットワークサービス研究会, pp. 63-64, 2009.

4. 大平 雅雄, 柗本 真佑, 松本 健一. コミュニティ媒介性: OSS 開発における協調作業への媒介度を評価する指標. 情報処理学会 第 5 回ネットワーク生態学シンポジウム, pp. 260-261, 2009.

5. 伊原 彰紀, 亀井 靖高, 大平 雅雄, 柗本 真佑, 松本 健一. OSS プロジェクトにおける障害に関する情報共有の分析. 平成 20 年度 情報処理学会関西支部支部大会 講演論文集, Vol. 2008, pp. 69-72, 2008.

6. 亀井 靖高, 大平 雅雄, 柗本 真佑, 松本 健一. Apache コミュニティにおける開発者とユーザとのコーディネーションの分析. ソフトウェアエンジニアリングシンポジウム, pp. 183-184, 2008.

7. 伊原 彰紀, 前島 弘敬, 柗本 真佑, 亀井 靖高, 大平 雅雄, 松本 健一. 複数のサブコミュニティを有する OSS コミュニティにおけるコーディネータの分析. 情報処理学会シンポジウム グループウェアとネットワークサービス研究会, Vol. 2007, No. 11, pp. 13-18, 2007.

8. 大平 雅雄, 柗本 真佑, 前島 弘敬, 亀井 靖高, 松本 健一. OSS コミュニティにおける共同作業プロセス理解のための中心性分析. 情報処理学会シンポジウム グループウェアとネットワークサービス研究会, Vol. 2007, No. 11, pp. 7-12, 2007.

9. 木浦 幹雄, 柗本 真佑, 亀井 靖高, 門田 暁人, 松本 健一. 他プロジェクトの実績データに基づく Fault-Prone モジュール判別の試み. ソフトウェア信頼性研究会 第 4 回ワークショップ, pp. 45-53, 2007.

目 次

図 目 次

表 目 次

第1章　序論

1. 研究の背景と目的

大規模化と複雑化が進むソフトウェア開発において，限られたリソースの中で高い信頼性を確保するためには効率的なテストの実施が必須である [34]．その一つの手段は，信頼性の低い，すなわち fault（欠陥）を含むソフトウェアモジュールを推定し，テストに費やすリソースを適切に割り当てることである [30, 31, 38, 50]．これまで，過去の開発モジュールの fault 検出傾向に基づき，新規開発モジュールに含まれる fault の有無を判別する fault-prone モジュール判別モデルに関する研究が数多く実施されてきた [19, 25, 31, 32, 35, 40, 50]．fault-prone モジュール判別モデルは fault の有無を目的変数とし，モジュールの特徴を表すプロダクトメトリクスを説明変数とする数学的モデルであり，線形判別分析やロジスティック回帰分析などの手法を用いて構築される．

fault-prone モジュール判別の流れを図 1.1 に示す．まず，過去のプロジェクトで開発されたモジュールのメトリクス値と fault の有無が記録されたフィットデータを用意し，線形判別分析などの手法を用いて判別モデルを構築する．次に，この判別モデルに対して新規開発モジュールのメトリクス値を入力することで，新規モジュールの fault の有無を推定する．テストの計画者は，fault ありと推定されたモジュールに対して fault なしのモジュールよりも多くのテスト工数を割り当てることにより，効率的な fault 検出が期待できる．

本論文では，fault-prone モジュールの判別精度向上を目的として，fault-prone モジュール判別問題に対する 2 つの問題に取り組む．2 つの問題を図 1.2 に示す．

第 1 の問題は，判別モデル構築用のフィットデータ中にノイズとなるモジュールが多数存在することである．これまでの fault-prone モジュール判別に関する研

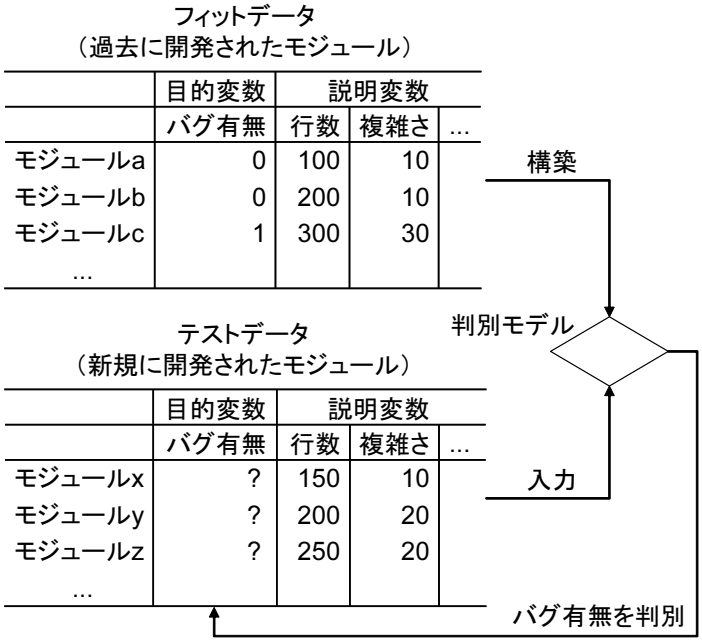


図 1.1 fault-prone モジュール判別の流れ

究の多くは，過去のプロジェクトで開発された全てのモジュールをフィットデータとして用いている [19, 25, 31, 32, 35]．しかし，フィットデータの中には他のモジュール群の特徴と異なった傾向を持つモジュール（外れ値）が存在する．この外れ値は，(1) 群の傾向から極端に外れた特徴を持つモジュールと，(2) 群の傾向と相反して fault を含む／含まないモジュールの 2 つに分類できる．具体的な例としては，前者は他のモジュールに比べて規模や複雑さなどのメトリクス値が極端に大きい（もしくは小さい）モジュールなどであり図 1.2 に示すモジュール f が該当する．後者は規模が小さく平易であるにも関わらず fault を含んでいるモジュールなどであり，図 1.2 のモジュール c が該当する．このような外れ値をフィットデータに含めた場合，構築される判別モデルは正常集合のみならず外れ値も正しく判別しようとするため，全体的な判別精度が低下することが指摘されている [24]．

第 2 の問題はモデル構築のための説明変数に人的な要因が含まれていないことである．一般に説明変数として用いられるメトリクスとしては，ソースコードの規模や複雑さといった静的メトリクスが広く用いられている．他にもオブジェクト指向言語の特徴に基づくメトリクスのほか，近年では変更履歴から算出された

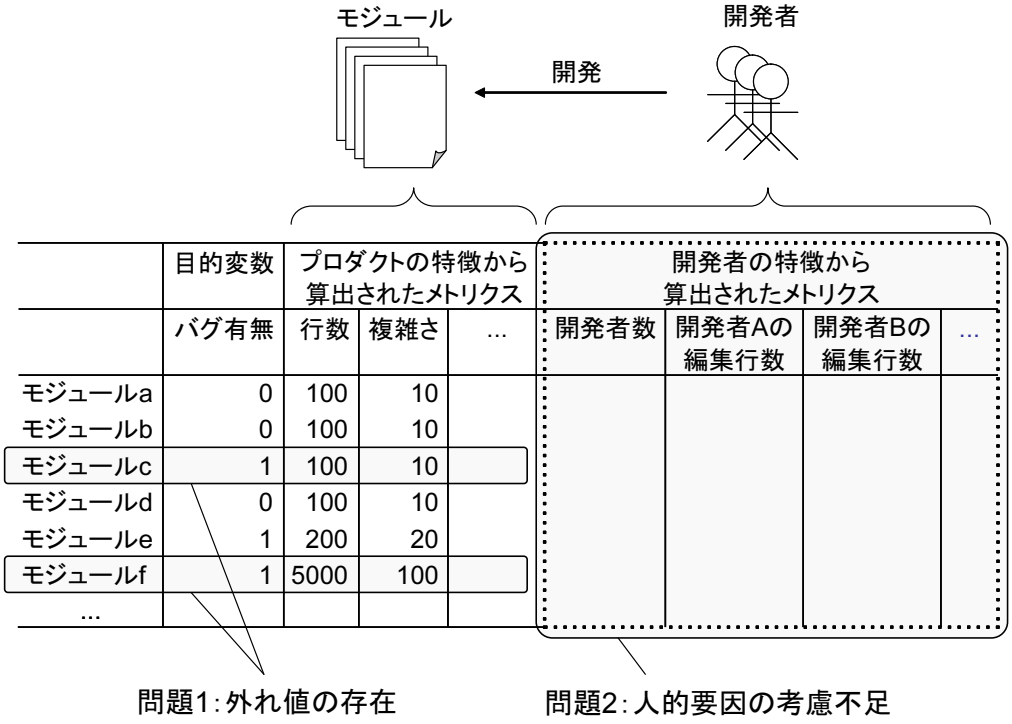


図 1.2 fault-prone モジュール判別に関する 2 つの問題

変更メトリクスなど多数のメトリクスが提案されている．これらはプロダクトの特徴から算出されたメトリクスであるが，欠陥混入の原因としては人的要因，すなわちプロダクトを作成した開発者の要因が無視できないと考えられる．例えば，規模や複雑さが同じモジュールであっても，経験豊かな開発者ほど欠陥を混入させにくいといえる．また複数の開発者によって変更されたモジュールほど，思考過程の違いや機能の実装方法の混在により意図の読み違いが発生しやすく，fault を埋め込む可能性が高くなると考えられる．fault の有無を判別するモデルに対して高い判別精度を確保するためには，fault 混入の原因となる事象は可能な限り説明変数として加えることが望ましい．

本論文ではこれら 2 つの問題を解決し，fault-prone モジュール判別の判別精度向上を目的とする．まず，問題 1 に対する解決策としてデータ中の外れ値を除去する手法に着目する．外れ値除去法は多変量データの中から特異な個体を検出し除去する方法であり，これまで様々な手法が提案されている [4, 26]．前述の外れ

値 (1) と (2) のうち，どちらを外れ値と見なし除去するかは各手法により異なっており，fault-prone モジュール判別に対してどの手法が高い効果を示すかは明らかにされていない．

問題 2 に対する解決策として，本論文では開発者に着目したメトリクス（開発者メトリクス）を提案する．提案する開発者メトリクスは開発者個々に付随するメトリクス（開発者 A の編集行数や，開発者 A のコミット回数など）と，モジュール個々に付随するメトリクス（モジュール X の開発者数や，モジュール X に対する開発者 A の編集行数など）の 2 種類である．従来の説明変数として用いられていたプロダクトメトリクスに加え，人的要因を表現する開発者メトリクスを用いることで，fault 増減に対して高い表現能力を持つ判別モデルの構築が可能である．

2. 論文構成

本論文の主要部分は大きく 2 つの章から構成される．続く第 2 章では問題 1 に対して，外れ値除去法を適用した際の効果を比較実験により確かめた．実験では，前述の (1) 群の傾向から極端に外れた特徴を持つモジュールを優先的に除去する 2 つの外れ値除去法（MOA と LOFM）と，(2) 群の傾向と相反して fault を含む／含まないモジュールを優先的に除去する 2 つの外れ値除去法（RBM と CC-MOA）の計 4 手法を用い，代表的な 3 つの判別モデル（線形判別分析，ロジスティック回帰分析，分類木）に対して交差検証法により評価を行った．実験の結果，LOFM を除く 3 つの外れ値除去法を用いた場合，いずれの判別モデル，データセットに対しても判別精度が改善され，F1 値の平均向上幅は MOA では 0.139，RBM では 0.137，CC-MOA では 0.123 であった．

第 3 章では問題 2 に対して，開発者メトリクスに基づく以下の 4 つの仮説について検証を行った．仮説 1a：fault の混入のさせやすさに個人差がある．仮説 1b：fault の混入のさせやすさは開発者の特性（変更行数やコミット回数など）から判断できる．仮説 2：多くの開発者が変更したモジュールには fault が含まれやすい．仮説 3：開発者メトリクスが fault を含んでいる可能性の高いモジュー

ル（fault-prone モジュール）の判別に有効である．これら 4 つの仮説の検証により，fault 増減に対する人的要因の影響を明らかにし，さらに開発者メトリクスが fault-prone モジュール判別に対してどの程度効果的かを実験により確かめた．分析の結果，開発者個々の fault 混入率には 5 倍以上の個人差があること，及び多くの開発者が変更を加えたモジュールほど fault が混入されやすいことが明らかとなった．また開発者メトリクスが判別モデルの精度向上に有効であることを確認し，その F1 値の平均向上幅は 0.093 であった．

第2章　外れ値除去法による fault-prone モジュール判別精度向上の試み

1. はじめに

一般に，fault-prone モジュール判別モデル構築に用いられるフィットデータには，多の群の傾向と異なる特徴を持つモジュール（外れ値）が多数含まれており，この外れ値が判別モデルの精度を低下させる原因となることが指摘されている [24]．このため，外れ値はあらかじめフィットデータから除去した上で，判別モデルを構築することが望ましい．

従来，多変量データの標本から外れ値（特異な個体）を検出し，除去する手法（外れ値除去法）が多数提案されている [4, 26] が，fault-prone モジュール判別モデルの精度を向上するという目的には必ずしも利用できない．fault-prone モジュール判別モデルにおけるフィットデータでは，fault を含むか否かによって「特異である」と判断すべき特徴が異なると考えられるためである．例えば，ソースコード行数が短く分岐やループも少ないモジュールは，一般に fault が混入しにくい．そのため，fault を含まない場合には特異とはいえない．ところが逆に fault を含むモジュールとしては特異であるといえる．

そこで，本章では，fault を含む標本と含まない標本のそれぞれに対し，外れ値除去法を独立に適用することを考える．これにより，fault を含む／含まない場合のそれぞれにおいて特異とみなせるモジュールの除去が期待できる．本章では，代表的な外れ値除去法である Mahalanobis Outlier Analysis (MOA)[23] と Local Outlier Factor Method (LOFM)[4] を適用した場合の効果を評価する．

さらに，本章では，標本の中に2つのクラス（本章の場合は，fault あり／なしの2クラス）があることを前提とした，新たな外れ値除去法 Cross-Class Mahalanobis Outlier Analysis (CC-MOA) を提案する．この手法では，一方のクラスに含まれるモジュールが，他方のクラスの重心に近い場合に外れ値であるとみなす（3.4節）．このような2クラスの判別問題を対象とした外れ値除去法としては，他に，Rule-Based Modeling (RBM) が提案されている [24]（3.3 節）．RBM は，CBR (Case-Based Reasoning) モデルに対する効果は実験的に示されている [24] が，他の判別モデルに対する効果は不明である．

これら4つの手法のうち MOA と LOFM は fault を含む標本（もしくは含まない標本）の中から，群の傾向と極端に外れた特徴を持つものを優先的に外れ値と見なす手法であり，RBM と CC-MOA は群の傾向と相反する目的変数をもつものを優先的に外れ値と見なす手法であるといえる．

本章では，これら4つの外れ値除去法（MOA，LOFM，RBM，CC-MOA）を，3つの代表的な fault-prone モジュール判別モデル（線形判別分析 [12]，ロジスティック回帰分析 [22]，分類木 [3]）に適用した場合の計 12 通りの判別精度を実験的に比較し，いずれの外れ値除去法が fault-prone モジュール判別において最も効果があるかを明らかにする．実験には NASA/WVU が公開しているモジュールの特性値と fault の有無を記したデータセット [36] のうち，3つ（モジュール数の多いものから3プロジェクト) のデータセットを用いた．

2章の構成は以下の通りである．まず，2節で実験に用いる fault-prone モジュール判別モデルについて説明し，3節で比較対象とする外れ値除去法について説明する．4節で実験の方法と手順について述べ，5節でその実験結果について述べる．6節で実験結果についての考察を行い，7節で関連研究について説明し，最後に8節で本章のまとめを述べる．

2. fault-prone モジュール判別モデル

fault-prone モジュール判別モデルは，モジュールが fault を含んでいるか否かを判別することを目的とし，過去に開発されたモジュールの特性値と fault の有

無を記したデータセットを用いて構築される．これまでに多数の判別モデルが提案されているが，本論文では fault-prone モジュール判別モデルとして広く用いられている 3 つのモデリング手法を採用する．

2.1 線形判別分析

線形判別分析は 2 クラスの標本を判別する境界を直線として表現する [12]．個体 x の説明変数を x_i，説明変数の総数を p とした場合，判別値 Z を算出する判別関数は以下のように表される．

$$Z = \alpha + \beta_1 x_1 + \cdots + \beta_p x_p \tag{2.1}$$

ここで α は定数，β_i は判別係数であり，判別モデルの構築時に α と β_i が決定される．判別対象の個体は，判別値 Z が正であるか負であるかによって 2 値のいずれに属するか判別される．

2.2 ロジスティック回帰分析

ロジスティック回帰分析では，2 クラスの標本を判別する判別関数にロジスティック関数を用いて表現する [22]．

$$P(y = 1|x) = \frac{1}{1 + e^{-(\alpha+\beta_1 x_1+\cdots+\beta_p x_p)}} \tag{2.2}$$

ここで，$y \in \{0, 1\}$ は 2 値のクラスを表す目的変数であり，$P(y = 1|x)$ は個体 x に対して y が 1 のクラスに属する確率である．本章では，$P(y = 1|x)$ の値が 0.5 を超える場合に，個体 x が 1 のクラスに属すると判別する．

2.3 分類木

分類木は説明変数と目的変数の関係を木構造で表現する非線形モデルである [3]．木の各ノードは 2 個以上の子ノードを持ち，説明変数の値によっていずれかの子ノードへ分岐し，リーフノードには 2 値のいずれかが割り当てられる．判

別対象の個体は説明変数の値によってノードをたどり，リーフノードによって 2 値のいずれに属するか判別される．本論文では，分類木の構築アルゴリズムには Classification And Regression Trees (CART)[3] を用い，ノードの分岐の基準には Gini 係数 [3] を用いた．

3. 外れ値除去法

外れ値除去法とは，標本の中から他の個体と比べて特異な傾向を持った個体を，外れ値として検出し除去する手法である．以降，実験に用いた 3 つの既存の外れ値除去法と本章で提案する新たな外れ値除去法について説明する．

3.1 Mahalanobis Outlier Analysis

Mahalanobis Outlier Analysis (MOA) とは，多変量データを対象に，マハラノビス距離が閾値 θ_{MOA} を超える個体を外れ値と見なし除去する手法である [23]．マハラノビス距離とは，標本の重心から計算対象とする個体までの距離を，標本のばらつき度合いに基づいて算出される距離尺度である．

図 2.1 に，ある標本に対するマハラノビス距離の例を示す．図 2.1 の点は個体を表し，楕円はマハラノビス距離の等高線を表す．このように個体が楕円状に分布している場合，マハラノビス距離の等高線は楕円を描くように算出される．

$\overline{x}$ を各変数の平均値のベクトル，x_i を個体 i の持つ変数のベクトル，n を個体の総数，T を転置ベクトルとしたとき，共分散行列 S は

$$S = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{x})(x_i - \overline{x})^T \tag{2.3}$$

と表され，個体 i のマハラノビス距離 md_i は

$$md_i = \sqrt{(x_i - \overline{x})^T S^{-1} (x_i - \overline{x})} \tag{2.4}$$

と表される．md_i は平均が 1 になるように正規化され，個体 i が重心と同じ場所に位置する場合に md_i は最小（$md_i = 0$）となり，重心から離れた個体ほど大き

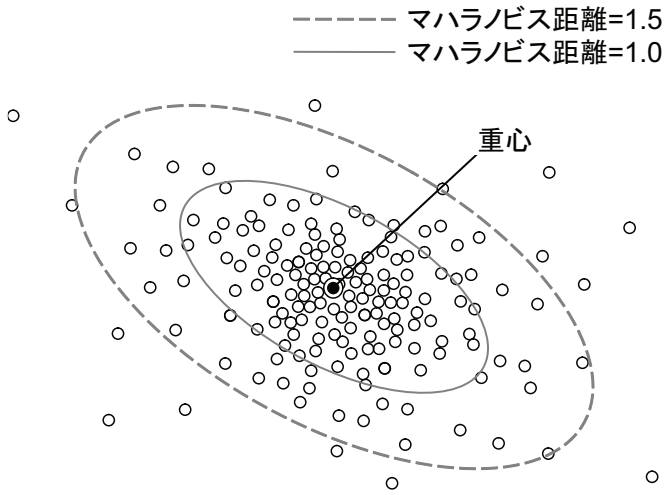


図 2.1 マハラノビス距離の例

な値をとる．これらのことから，MOA とは標本全体の重心から離れた個体を外れ値と見なし除去する手法であるといえる．

本章では，fault を含むクラスと含まないクラスのそれぞれに対し，MOA を独立に適用することで，各クラスの外れ値の除去を行う．

3.2 Local Outlier Factor Method

Local Outlier Factor (LOF) とは，個体ごとの最も距離の近い k 個の固体の集合（k-nearest neighbor）との分布密度に基づいて算出される値であり，各個体が標本の中からどの程度外れているかを表す尺度である [4]．本章では，LOF が閾値 θ_{LOFM} を超える個体を外れ値と見なし除去する手法を LOF Method (LOFM) と呼ぶ．

図 2.2 は $k = 3$ のときの LOF の例である．個体 x に対する k-nearest neighbor を $nn_k(x)$ と表し，個体 x と個体群 $nn_k(x)$ の最大の距離を $k\text{-}distance(x)$ と表す（図 2.2 の場合，$k\text{-}distance(x)$ は個体 x と個体 $nn_3(x)$ の距離を指す）．ここで，個体 x の LOF の値 $LOF_k(x)$ は，$k\text{-}distance(x)$ が $k\text{-}distance(nn_k(x))$ に比べて大きい場合に 1 を超える値をとるように算出される．他の個体から孤立して位置する個

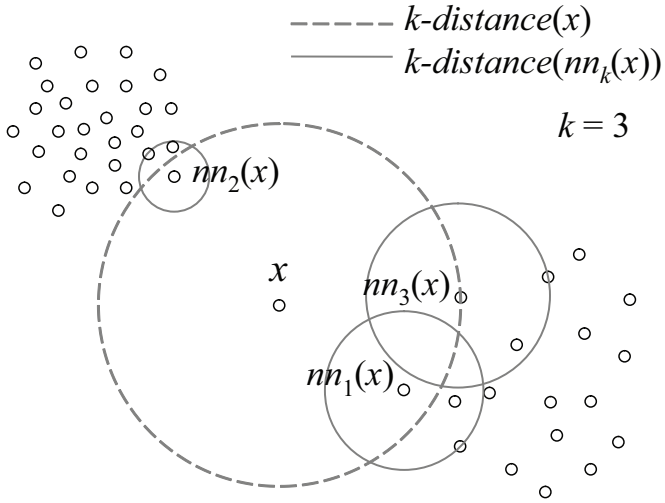


図 2.2 Local Outlier Factor の例

体は $k\text{-}distance(x)$ が $k\text{-}distance(nn_k(x))$ に比べて大きく，$LOF_k(x) \gg 1$ となる．逆に一定の密度でばらついている個体は $k\text{-}distance(x)$ と $k\text{-}distance(nn_k(x))$ が近い値となるため，$LOF_k(x) \fallingdotseq 1$ となる．図 2.2 の個体 x の場合，$k\text{-}distance(x)$ が $k\text{-}distance(nn_k(x))$ に比べ非常に大きいため，$LOF_k(x) \gg 1$ となる．これらのことから，LOFM は疎な空間に存在する個体を外れ値とみなす手法であるといえる．

本章では，fault を含むクラスと含まないクラスのそれぞれに対し，LOFM を独立に適用する．

3.3 Rule-Based Modeling

Khoshgoftaar らは 2 クラス判別問題のための外れ値除去法として，Rule-Based Modeling (RBM) を提案している [24]．ここで 2 つのクラスとは fault を含む (*fp*) か，含まない (*nfp*) かを指す．従来，RBM の CBR (Case-Based Reasoning) モデルに対する効果は実験的に示されている [24] が，他の判別モデルに対する効果は不明である．

RBM では，まず，全ての変数に対して 2 つのクラスを最も良く分割する閾値

表 2.1 Rule-Based Modeling で用いる Boolean ルールの例

#	ルールの内容
1	$(x_1 \leq c_1) \wedge (x_2 \leq c_2) \Rightarrow \mathit{nfp}$
2	$(x_1 \leq c_1) \wedge (x_2 > c_2) \Rightarrow \mathit{nfp}$
3	$(x_1 > c_1) \wedge (x_2 \leq c_2) \Rightarrow \mathit{nfp}$
4	$(x_1 > c_1) \wedge (x_2 > c_2) \Rightarrow \mathit{fp}$

を，fault ありモジュールの総数と fault なしモジュールの総数の割合に基づき 2 標本 Kolmogorov Smirnov 検定 [6] により計算する．次に，計算された閾値で区切られた空間ごとに，fault あり／なしのどちらに属するかを判別する Boolean ルールを構築する．2 つの変数を持つ標本の場合の Boolean ルールの例を，表 2.1 に示す．変数の数が 2 つの場合，2 つの閾値（c_1, c_2）を用いた 4 つの Boolean ルールが構築される．

最後に，標本の中から $(x_1 \leq c_1) \wedge (x_2 \leq c_2) \wedge \mathit{fp}$ のような，ルールに反する（この場合ルール 1 に反する）個体を外れ値と見なし除去する．

つまり RBM は，自身の属するクラスと異なったクラスの特徴を持つと判定された個体の全てを，外れ値とみなす手法である．また，変数ごとの閾値は Kolmogorov Smirnov 検定を用いて自動的に決定されるため，MOA や LOFM と異なり閾値決定のための事前実験が不要である．

3.4 Cross-Class Mahalanobis Outlier Analysis

本章ではこれまでに提案されている上記の 3 つの手法に加え，Cross-Class Mahalanobis Outlier Analysis（CC-MOA）を提案する．CC-MOA は，RBM と同様，2 つのクラス（fault あり／なし）が存在することを前提とした手法である．図 2.3 に CC-MOA の例を示す．CC-MOA では，あるクラスの個体（例えば，fault を含むモジュール）のうち，もう一方のクラスの標本（fault を含まないモジュールの集合）の重心とのマハラノビス距離が近い個体を外れ値であるとみなす．つまり，他のクラスからのマハラノビス距離が閾値 $\theta_{\mathrm{CC-MOA}}$ を下回る個体を外れ値とみ

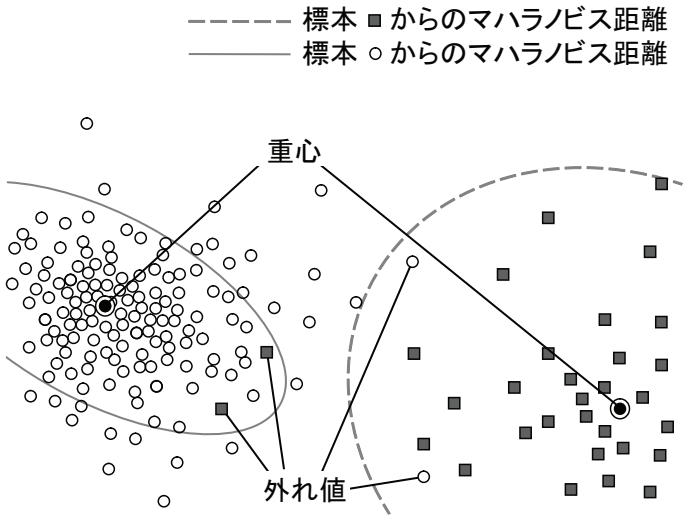


図 2.3 Cross-Class Mahalanobis Outlier Analysis の例

なす手法である．

CC-MOA の手順を以下に示す．

Step 1. クラス α に属する個体 x_α に対して，クラス β からのマハラノビス距離 md_{x_α} を計算する．

Step 2. md_{x_α} が $\theta_{\mathrm{CC-MOA}}$ を下回る場合，x_α を外れ値とみなす．

Step 3. 全ての x_α に対して Step 1 から Step 2 を繰り返す．

Step 4. α と β を入れ替えて Step 1 から Step 3 を実行する．

MOA（3.1 節）は自身の属するクラスからのマハラノビス距離を用いるのに対して，CC-MOA では他のクラスからのマハラノビス距離を用いる点が異なる．これにより，例えば fault を含むにも関わらず，fault を含まないモジュールと似た特徴を持つような個体を外れ値として検出することが可能となる．

4. 実験

4.1 実験概要

実験の目的は，代表的な fault-prone モジュール判別モデルに対して，最も判別精度の向上につながる外れ値除去法を明らかにすることである．そのために，4 つの外れ値除去法（MOA，LOFM，RBM，CC-MOA）と，3 つの fault-prone モジュール判別モデル（線形判別分析，ロジスティック回帰分析，分類木）の組合わせの合計 12 通りについて判別精度の比較を行った．

まず，事前実験として RBM を除く閾値を要する外れ値除去法（MOA，LOFM，CC-MOA）に対して閾値（θ_{MOA}，θ_{LOFM}，$\theta_{\mathrm{CC-MOA}}$）を決定した．次に，本実験としてモデル構築用のフィットデータに対して，事前実験で決定した閾値を用いて外れ値除去法を適用した．最後に，外れ値除去済みのフィットデータを用いて判別モデルを構築し，モデル評価用のテストデータを用いてその判別精度を算出した．実験の信頼性を確保するために，交差検証法により事前実験と本実験をそれぞれ 10 回ずつ繰り返した．

LOF を算出する際に用いる k-nearest neighbor の k については，Breunig らの決定方法 [4] に従ってあらかじめ実験を行い，$k = 30 \sim 50$ とした．

4.2 データセット

実験には NASA/WVU IV&V Facility Metrics Data Program (MDP) が公開しているデータセット [36] のうち，モジュール数の多いものから順に 3 つのデータセット（プロジェクト KC1，JM1，PC5）を用いた[1]．各データセットの概要を表 2.2 に示す．また，説明変数として用いたプロダクトメトリクスの一覧を表 2.3 と表 2.4 に示す．表中の「x」は各データセット中において記録されているプロダクトメトリクスを表す．

[1] MDP における公開データセットでは，プロジェクト PC2 と MC1 もモジュール数が多いが，fault を含むモジュールがほとんど含まれていなかった（1%未満）ため，本章では用いなかった．

表 2.2 データセット概要

	データセット名		
	KC1	JM1	PC5
モジュール総数	2,107	10,878	17,186
fault なしモジュール数	1,782	8,776	16,670
fault ありモジュール数	325	2,102	516
プロダクトメトリクス数	21	21	40
記述言語	C++	C++	C++
SLOC	43K	315K	164K

実験では，fault の有無を目的変数，プロダクトメトリクスを説明変数として判別モデルを構築する．判別モデルを構築する際には，データセットをランダムに二等分し，一方をフィットデータ，もう一方をテストデータとする．

4.3 評価基準

fault-prone モジュール判別モデルの評価基準として，再現率，適合率，F1 値 [20] を用いる．再現率（Recall）とは fault を含んでいるモジュールのうち，正しく fault-prone と判別したモジュールの割合であり，表 2.5 に示す記号を用いると以下のように定義される．

$$Recall = \frac{n_{22}}{n_{21} + n_{22}} \tag{2.5}$$

適合率（Precision）とは fault-prone と判別されたモジュールのうち，実際に fault を含んでいるモジュールの割合であり，以下のように定義される．

$$Precision = \frac{n_{22}}{n_{12} + n_{22}} \tag{2.6}$$

これら再現率と適合率は一方が高くなると，もう一方が低くなりやすいという関係にある．そこで本章では，再現率と適合率の調和平均である F1 値を評価基準として用いる．F1 値は以下のように定義される．

$$F1\text{-}measure = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{2.7}$$

表 2.3 各データセットのメトリクス（1/2）

メトリクス名	データセット名		
	KC1	JM1	PC5
Branch count	x	x	x
Call pairs			x
Condition count			x
Cyclomatic complexity	x	x	x
Cyclomatic density			x
Decision count			x
Decision density			x
Design complexity	x	x	x
Design density			x
Edge count			x
Essential complexity	x	x	x
Essential density			x
Global data complexity			x
Global data density			x
Halstead content	x	x	x
Halstead difficulty	x	x	x
Halstead effort	x	x	x
Halstead error estimation	x	x	x
Halstead length	x	x	x
Halstead level	x	x	x
Halstead programming time	x	x	x
Halstead volume	x	x	x
LOC blank	x	x	x
LOC code and comment	x	x	x
LOC comment	x	x	x

表 2.4 各データセットのメトリクス（2/2）

メトリクス名	データセット名 KC1	JM1	PC5
LOC executable	x	x	x
LOC total	x	x	x
Maintenance severity			x
Modified condition count			x
Multiple condition count			x
Node count			x
Normalized cyclomatic count			x
Number of operands	x	x	x
Number of operators	x	x	x
Number of unique operands	x	x	x
Number of unique operators	x	x	x
Number of lines			x
Parameter count			x
Pathological complexity			x
Percent comment			x

表 2.5 判別結果の分類表

		判別値 fault なし	fault あり
実測値	fault なし	n_{11}	n_{12}
	fault あり	n_{21}	n_{22}

3 つの評価基準は値域 $[0, 1]$ を取り，値が高いほど判別精度が高いことを表す.

4.4 実験手順

閾値の決定手順

外れ値除去法の閾値（θ_{MOA}，θ_{LOFM}，$\theta_{\mathrm{CC-MOA}}$）を決定するために事前実験を行う．事前実験では本実験で用いるフィットデータのみを用いて，判別モデルの構築と評価を行い，最も F1 値が高くなる閾値を求める．この閾値は，fault-prone モジュール判別モデルと外れ値除去法の全ての組合わせについて決定される．閾値の決定手順を以下に示す．

Step 1. フィットデータ fit を，fit_A と fit_B の 2 つにランダムに分割する．

Step 2. fit_A に対して外れ値除去法を適用し，fit'_A を作成する．

Step 3. fit'_A を用いて，判別モデルを構築する．

Step 4. fit_B を用いて，構築した判別モデルの判別精度（F1 値）を求める．

Step 5. 閾値を変化させて，Step 2 から Step 4 を繰り返す．

Step 6. Step 1 から Step 5 を 10 回繰り返し，F1 値の平均値が最も高かった閾値を本実験に用いる閾値とする．

本実験の手順

4.4 項で決定した閾値を用いて，外れ値除去法と fault-prone モジュール判別モデルの全ての組合わせ 12 通りについて判別精度を求める．本実験の手順を以下に示す．

Step 1. データセットを，フィットデータ fit とテストデータ $test$ の 2 つにランダムに分割する．

Step 2. fit に対して事前実験で決定した閾値を用いて外れ値除去法を適用し，fit' を作成する．

Step 3. fit' を用いて，判別モデルを構築する．

Step 4. *test* を用いて，構築した判別モデルの判別精度を求める．

Step 5. Step 1 から Step 4 を 10 回繰り返し，判別精度（再現率，適合率，F1 値）の平均値をそれぞれ求める．

5. 実験結果

5.1 事前実験の結果

4.4 項で述べた事前実験により決定された各外れ値除去法ごとの閾値を表 2.6 に示す．PC5 に対して LOFM を適用した場合については，閾値の計算が終了しなかったため記載していない[2] ．この原因は，LOFM が他の手法より大きな計算量を要することと，PC5 のモジュール数とプロダクトメトリクスの数が他のデータセットより大きかったためと考えられる．また，RBM については，閾値決定のための事前実験が不要であるため記載していない．

表 2.6 より，KC1 と JM1 に対しての各外れ値除去法の最適な閾値は，判別モデル間で大きな違いがなかったが，PC5 については判別モデル間でばらつきがあった．また，データセット間でも，各外れ値除去法の閾値にはばらつきがあった．このことから，いずれの外れ値除去法においても，最適な閾値はデータセットと判別モデルに依存するため，事前実験により閾値を決定することが必須といえる．

5.2 本実験の結果

4.4 項で述べた本実験の結果をデータセット KC1，JM1，PC5 のそれぞれについて表 2.7，表 2.8，表 2.9 に示す．表中の未適用とは，外れ値を除去せずに判別モデルを構築した場合の判別精度を表す．また表中の「$^{+}$」は有意水準 5%で平均値の差を検定した結果，未適用と比べて判別精度が有意に向上したものを表し，「$^{-}$」は有意に低下したものを表す．以降，判別精度の変化について，外れ値除去法ごとと判別モデルごとに述べる．

[2] Pentium 4，3GHz の計算機を用いて 100 時間以内に結果が得られなかった．

表 2.6 事前実験で決定した閾値

データセット	外れ値除去法の閾値	判別モデル LDA	LRA	CT
KC1	θ_{MOA}	0.3	0.3	0.4
	θ_{LOFM}	1.2	1.2	1.2
	$\theta_{\mathrm{CC-MOA}}$	1.3	1.2	1.2
JM1	θ_{MOA}	0.3	0.3	0.4
	θ_{LOFM}	1.0	1.0	1.0
	$\theta_{\mathrm{CC-MOA}}$	0.4	0.8	0.8
PC5	θ_{MOA}	9.8	5.7	2.3
	θ_{LOFM}	—	—	—
	$\theta_{\mathrm{CC-MOA}}$	0.7	0.9	0.5

LDA：線形判別分析，LRA：ロジスティック回帰分析，CT：分類木

外れ値除去法間の比較

表 2.7 に示すとおり KC1 では，MOA，RBM，CC-MOA はいずれの判別モデルに対しても適合率が若干減少した（−0.043 〜 −0.333）ものの，再現率がそれ以上に向上し（+0.099 〜 +0.538），これらの調和平均である F1 値も向上した（+0.042 〜 +0.188）．このことは，JM1 (表 2.8)，PC5 (表 2.9) においても同様であった．一方で，LOFM はいずれのデータセット，判別モデルにおいても判別精度の有意な変化が適合率，再現率，F1 値の全てにおいて見られなかった．以上のことから，MOA，RBM，CC-MOA はデータセット，及び判別モデルに関わらず判別精度向上の効果があり，逆に，LOFM は，fault-prone モジュール判別問題に対しては効果が期待できないといえる．

LOFM 以外の外れ値除去手法（MOA，RBM，CC-MOA）による判別精度向上の効果は，ほぼ同程度であった．4 つの外れ値除去法それぞれの F1 値平均を，（MOA，LOFM，RBM，CC-MOA）の順でデータセットごとに述べると，KC1 では（0.409，0.322，0.406，0.416）であり，JM1 では（0.417，0.164，0.412，0.361），PC5 では（0.460，—，0.458，0.447）であった．

表 2.7 外れ値除去法適用の効果（データセット：KC1）

		判別モデル		
評価尺度	外れ値除去法	LDA	LRA	CT
再現率	未適用	0.277	0.154	0.241
	MOA	0.376^{+}	0.389^{+}	0.378^{+}
	LOFM	0.276	0.160	0.301
	RBM	0.622^{+}	0.692^{+}	0.711^{+}
	CC-MOA	0.743^{+}	0.479^{+}	0.640^{+}
適合率	未適用	0.557	0.652	0.483
	MOA	0.452^{-}	0.452^{-}	0.440^{-}
	LOFM	0.515	0.625	0.460
	RBM	0.342^{-}	0.389^{-}	0.314^{-}
	CC-MOA	0.344^{-}	0.381^{-}	0.301^{-}
F1 値	未適用	0.367	0.247	0.317
	MOA	0.409^{+}	0.416^{+}	0.402^{+}
	LOFM	0.356	0.252	0.357
	RBM	0.440^{+}	0.435^{+}	0.434^{+}
	CC-MOA	0.437^{+}	0.405^{+}	0.407^{+}

LDA：線形判別分析，LRA：ロジスティック回帰分析，CT：分類木

判別モデル間の比較

外れ値除去法を適用しない場合，判別モデル間の判別精度には大きなばらつきが見られた．特に，ロジスティック回帰分析は，いずれのデータセットにおいても線形判別分析と比べて F1 値が有意に小さかった．ところが，このモデル間の差は，外れ値除去法を用いることで小さくなった．例えば，KC1（表 2.7）に対して外れ値除去法を適用しない場合の判別モデルごとの精度を（線形判別分析，ロジスティック回帰分析，分類木）の順で並べると（0.367，0.247，0.317）であったが，MOA を用いた場合には（0.409，0.416，0.402）となり，0.4 付近に収束した．同様に，RBM では（0.440，0.435，0.434），CC-MOA では（0.437，0.405，

表 2.8 外れ値除去法適用の効果（データセット：JM1）

		判別モデル		
評価尺度	外れ値除去法	LDA	LRA	CT
再現率	未適用	0.139	0.072	0.089
	MOA	0.483^{+}	0.474^{+}	0.402^{+}
	LOFM	0.146	0.067	0.100
	RBM	0.401^{+}	0.547^{+}	0.625^{+}
	CC-MOA	0.832^{+}	0.691^{+}	0.779^{+}
適合率	未適用	0.561	0.648	0.559
	MOA	0.375^{-}	0.381^{-}	0.413^{-}
	LOFM	0.508^{-}	0.555^{-}	0.361^{-}
	RBM	0.398^{-}	0.358^{-}	0.323^{-}
	CC-MOA	0.228^{-}	0.226^{-}	0.251^{-}
F1 値	未適用	0.222	0.129	0.158
	MOA	0.422^{+}	0.422^{+}	0.406^{+}
	LOFM	0.223	0.117	0.153
	RBM	0.399^{+}	0.432^{+}	0.406^{+}
	CC-MOA	0.357^{+}	0.351^{+}	0.374^{+}

LDA：線形判別分析，LRA：ロジスティック回帰分析，CT：分類木

0.407）となった．この傾向は，他のデータセット JM1，PC5 でも同様であった．このことから，外れ値による影響（判別精度の低下の度合い）は判別モデルによって異なっていたが，外れ値を除去することで，いずれの判別モデルも性能を発揮できるようになり，結果として性能の違いがなくなったといえる．

5.3 本実験でのモジュール除去率

各外れ値除去法によって除去されたモジュールの割合（除去率）を表 2.10 に示す．モジュールの総数を N_{total}，除去されたモジュールの数を $N_{removed}$ としたと

表 2.9 外れ値除去法適用の効果（データセット：PC5）

		判別モデル		
評価尺度	外れ値除去法	LDA	LRA	CT
再現率	未適用	0.538	0.182	0.369
	MOA	0.538	0.352^{+}	0.618^{+}
	LOFM	—	—	—
	RBM	0.747^{+}	0.824^{+}	0.820^{+}
	CC-MOA	0.743^{+}	0.742^{+}	0.602^{+}
適合率	未適用	0.393	0.615	0.542
	MOA	0.413^{+}	0.523^{-}	0.415^{-}
	LOFM	—	—	—
	RBM	0.361^{-}	0.309^{-}	0.298^{-}
	CC-MOA	0.344^{-}	0.298^{-}	0.415^{-}
F1 値	未適用	0.454	0.279	0.437
	MOA	0.467^{+}	0.420^{+}	0.493^{+}
	LOFM	—	—	—
	RBM	0.487^{+}	0.449^{+}	0.437
	CC-MOA	0.470	0.424^{+}	0.490^{+}

LDA：線形判別分析，LRA：ロジスティック回帰分析，CT：分類木

き，除去率 $P_{removed}$ は以下のように表される．

$$P_{removed} = \frac{N_{removed}}{N_{total}} \tag{2.8}$$

除去率はデータセット間で大きな差があった．例えば，KC1 では除去率が 20.3% 〜 74.0%であったのに対し，PC5 では 0.7% 〜 21.7%と小さかった．データセット間の fault を含むモジュールの割合の違いが影響している可能性がある．

外れ値除去法の違いによっても，除去率には大きな差があった．例えば，MOA は除去率が 0.7% 〜 37.5%であったのに対し，CC-MOA では 80%を超える場合もあった．ただし，前節で述べたように MOA と CC-MOA では判別精度への効果に大きな差がなかったことから，除去率が判別精度へ与える影響は小さいと考え

表 2.10 外れ値として除去されたモジュールの割合（%）

データセット	外れ値除去法の除去割合	判別モデル LDA	LRA	CT
KC1	MOA	20.3	37.5	27.7
	LOFM	55.5	55.5	55.5
	RBM	30.3	30.3	30.3
	CC-MOA	74.0	71.6	71.6
JM1	MOA	32.6	32.6	23.6
	LOFM	37.9	37.9	37.9
	RBM	33.1	33.1	33.1
	CC-MOA	69.5	80.5	80.5
PC5	MOA	0.7	1.4	5.5
	LOFM	—	—	—
	RBM	6.9	6.9	6.9
	CC-MOA	18.7	21.7	11.2

LDA：線形判別分析，LRA：ロジスティック回帰分析，CT：分類木

られる．

6. 考察

実験結果より，MOA，RBM，CC-MOA はいずれのデータセット，及び，判別モデルに対しても，適合率が若干減少したものの，再現率がそれ以上に向上し，これらの調和平均である F1 値も向上した．一般に，fault-prone モジュールの判別問題においては，再現率の低下と比べると，適合率の低下は許容される [32] ため，適合率の少しの低下と引き換えに再現率をより大きく向上できたことは，テスト効率の改善に有効であると考える．

さらに，MOA，RBM，CC-MOA は，モデル間の判別精度の差を小さくする効果があった．いずれのデータセット，判別モデルにおいても，MOA，RBM，CC-MOA により外れ値を除去した場合，F1 値は向上し，0.4 付近に収束した．従来，

最適な判別モデルはデータセットによって異なることが指摘されており [18]，データセットごとに適した判別モデルを見つけることが一つの課題であったが，外れ値除去によって判別モデル間の性能差がなくなり，モデル利用者にとって利便性が増すことが期待される．特に，RBM は閾値決定のための事前実験が不要であるという特徴を持つため，最初に試す手法として有力と考えられる．

外れ値除去によってモデル間の性能差が小さくなったことに対する一つの解釈としては，外れ値のないデータセットはそもそも 2 クラスの判別が容易であるため，いずれの種類のモデルにおいても，性能の良い（2 クラスをうまく判別する）モデルが得られた可能性がある．一方で，判別対象のテストデータには依然として外れ値が含まれており，たとえ性能の良いモデルであっても達成可能な判別精度には限界があったために，得られた F1 値は 0.4 付近にとどまった可能性がある．

次に，LOFM では判別精度向上の効果がほとんど見られなかった原因について，外れ値除去による各クラスのモジュールの分布の変化に着目して考察を行う．判別モデルと外れ値除去法の全ての組合せについて確認することは困難であるため，ここではデータセット KC1 に対して線形判別分析を用いた場合を取り上げる．

各外れ値除去法によって除去されたモジュールのヒストグラムを図 2.4 に，除去法適用後の残ったモジュールのヒストグラムを図 2.5 に示す．白色の棒は fault ありモジュール，灰色の棒は fault なしモジュールを表す．横軸には，議論の簡単化のために最も直感的な特性値である SLOC（ソースコード行数）を取った．図中の未適用とは外れ値除去法を適用しなかった場合のことであり，図 2.5 の未適用はデータセット KC1 に含まれる全てのモジュールの分布を表す．

まず，MOA による各モジュールの分布の変化に着目する．図 2.5 の未適用より，fault なしモジュール群のマハラノビス距離の重心は SLOC $= 0$ 付近にあり，fault ありモジュール群のマハラノビス距離の重心は SLOC $= 10 \sim 20$ 付近であった．図 2.4 より，MOA 適用後は，各群において重心付近のモジュールは除去されず，重心から離れたモジュールが除去されたことが窺える．SLOC 以外の特性値においても，fault あり／なしによって重心位置に差があり，各クラスの重心付近のモジュールが除去されず残された結果，2 クラスの判別精度が向上した可能性がある．

次に，LOFM に着目すると，fault あり／なしのいずれのモジュール群についても，外れ値除去前のモジュール群（図 2.5：未適用），除去後に残ったモジュール群（図 2.5：LOFM），除去されたモジュール群（図 2.4：LOFM）のヒストグラムはほぼ同じ形であった．このことから，LOFM では，ソースコード行数や fault あり／なしに関わらずほぼランダムに除去が行われた結果，判別精度に変化が見られなかったことが窺える．LOFM は,「近隣の個体と比べて疎な空間にある個体を除去する」方式であり，クラスタから外れた個体を外れ値として特定し，除去する手法である．ところが，今回のデータセットでは，サイズの小さなモジュールから大きなモジュールまで途切れなく存在していたため，大きなクラスタがそもそも存在せず，巨視的にはランダムに近い除去が行われた可能性がある．

次に，RBM では SLOC = 0 〜 30 の辺りを閾値として，サイズの大きな fault なしモジュール，及び，サイズの小さな fault ありモジュールの全てが除去されている．これは，RBM における Boolean ルールの一つに，SLOC を含むものがあったためと考えられる．一般に，サイズの小さいモジュールほど fault を含む割合が低く，また，サイズが大きくなるほど fault を含む割合が高くなることから,「サイズの大きな fault なしモジュール」と「サイズの小さな fault ありモジュール」は，ともに外れ値とみなすことが妥当であり，RBM は，SLOC に関しては，外れ値の除去が効果的に行えたことが示唆される．

CC-MOA に関しては，fault なしモジュールのうち，サイズの非常に小さなモジュール以外を大量に除去している．また，fault ありモジュールについては，サイズの小さなモジュールを全般的に除去している．その結果，RBM と同様のモジュールが除去され，判別精度の向上につながったことが窺える．なお，表 2.10 に示すとおり，CC-MOA のモジュール除去率は最大で約 8 割と大きく，外れ値を除去しているというよりはむしろ，判別に有用なモジュールのみを残してそれ以外を削除していると考えられる．

なお，本論文では fault 有無の判別精度向上を目的として外れ値適用の効果を確かめたが，fault 有無の確率を推定するという目的に対しては，これら外れ値除去は精度の低下に繋がる可能性がある点には注意されたい．特に RBM や CC-MOA では多の群に入り込んでいるモジュール（例えばサイズが小さいにも関わらず

fault を含むモジュールなど）を優先的に除去するが，これらの外れ値は fault の確率推定に対しては有用な情報であり，削除されるべきではないといえる．

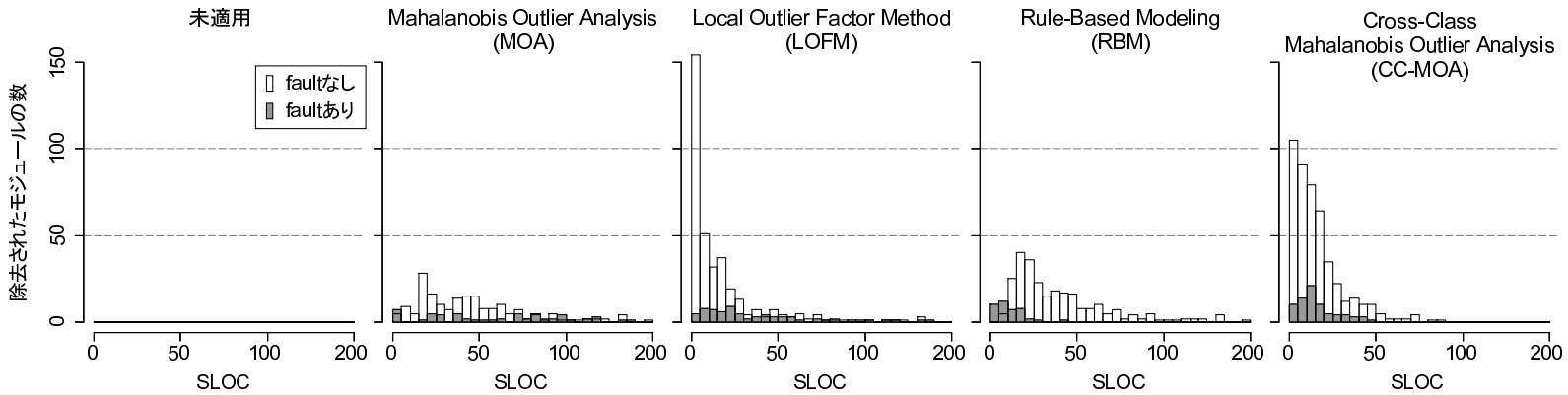


図 2.4 外れ値除去法によって除去されたモジュールのヒストグラム

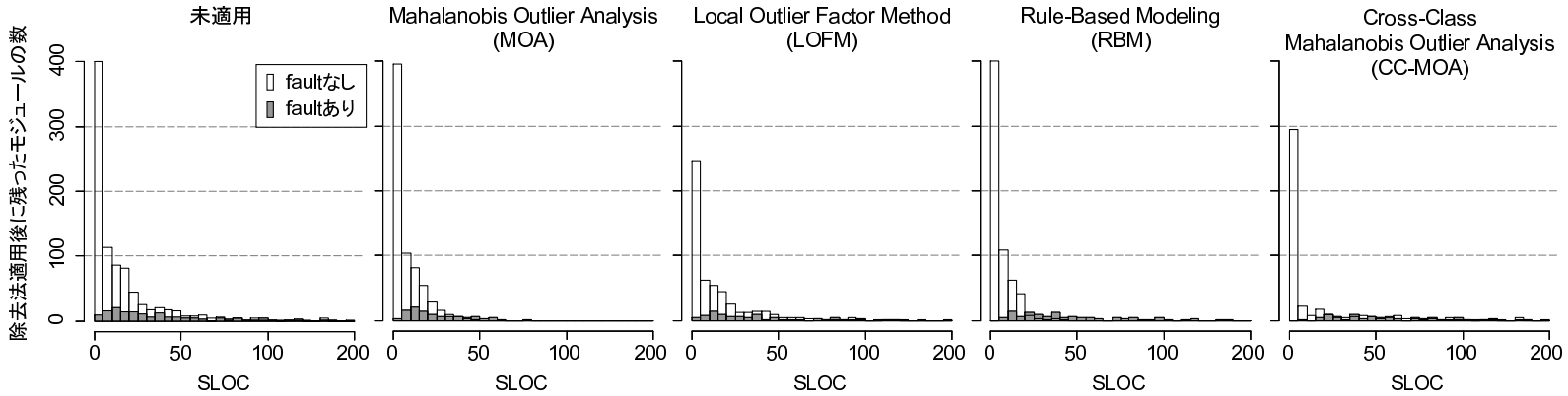


図 2.5 外れ値除去法適用後に残ったモジュールのヒストグラム

7. 関連研究

従来，様々なデータマイニングの分野で，多数の外れ値除去法が提案されている．Hodge らは既存の外れ値検出法を大きく 3 つの種類（統計モデル，ニューラルネット，機械学習）に分類した上で，それぞれの手法について調査を行った [21]．ただし，Victoria らの研究は各手法の特徴や特性についての調査であり，各手法の性能の良し悪しを実験的に比較するものではなく，また，線形判別分析などの判別モデルに対する各外れ値除去法に対しての効果についても論及していない．本研究は，fault-prone モジュール判別モデルに対しての各外れ値除去法の適用の効果を実験的に比較し，いずれの外れ値除去法が判別モデルに対して適しているか評価したという点で異なる．

Otey らは大量の IP パケットデータの中から攻撃目的のパケットやエラーパケットなどの異常パケットを検知するために，3 つの外れ値除去法の適用の効果を比較している [39]．比較する外れ値除去法としては，距離ベース手法の ORCA[2]，リンクベース手法の LOADED[16]，LOADED の拡張である RELOADED[39] の 3 つを用いている．各手法の比較項目としては異常パケットの検出率に加え，リアルタイムでの利用を目的とすることから実行速度と主記憶領域の使用量に着目している．これらの手法は IP パケットデータのうち異常パケットであるものを外れ値と見なすものであり，データセットの中からいずれが外れ値であるか明確な指標を与えることの可能な教師あり学習である．しかし，本章における fault-prone モジュール判別問題の場合，いずれのモジュールが外れ値であるかをあらかじめ明示することは不可能であり，これらの外れ値除去法を fault-prone モジュール判別モデルに対して適用することは難しい．

外れ値などのノイズを多く含んだデータセットに対して判別精度を改善する方法として，フィットデータから外れ値を除去する方法以外に，構築する判別モデルそのものを改善する方法も提案されている．Hubert らは外れ値を含むデータセットからガウス分布の平均ベクトルと共分散行列を頑健に推定する手法である Minimum Covariance Determinant (MCD) 基準を用いた 2 つの判別モデルを提案している [29]．Hubert らの研究は外れ値の影響を受けにくい判別モデルの提案である一方で，本研究は判別モデル構築に用いるデータセットに対する前処理で

あるため，既存の判別モデルのいずれに対しても適用可能であるという利点を持つ．fault-prone モジュール判別モデルに対して，これらノイズに対する頑健な判別モデルを用いる方法と，外れ値除去法を適用する提案手法との比較を行うことは残された課題の一つである．

8. まとめ

本章では，fault-prone モジュール判別モデル構築の前処理として，モデル構築用のフィットデータに外れ値除去法を適用した効果を，交差検証法により実験的に比較した．実験には 4 つの外れ値除去法（MOA，LOFM，RBM，CC-MOA）と，fault-prone モジュール判別モデルとして一般的に用いられる 3 つのモデル（線形判別分析，ロジスティック回帰分析，分類木）を用いた．得られた主な結果は次のとおりである．

- LOFM を除く MOA，RBM，CC-MOA は，いずれのデータセット，及び，判別モデルにおいても，適合率が若干減少するものの，再現率が大きく向上し，これらの調和平均である F1 値も向上した．F1 値の平均向上幅は MOA では 0.139，RBM では 0.137，CC-MOA では 0.123 であった．一般に，fault-prone モジュールの判別問題では，適合率よりも再現率の方が重要である [32] ことから，テスト効率の改善に有効であると考える．

- MOA，RBM，CC-MOA では，判別モデル間の性能差を小さくする効果があった．いずれのデータセット，判別モデルにおいても，MOA，RBM，CC-MOA により外れ値を除去した場合，F1 値は向上し，0.4 付近に収束した．外れ値除去によって，データセットごとに最適なモデルが異なるという問題が緩和され，モデル利用者にとって利便性が増すことが期待される．

- LOFM は，いずれのデータセット及び判別モデルに対しても，ほとんど効果がなかった．LOFM は，fault-prone モジュール判別問題に不向きであることが示唆された．

第3章　開発者メトリクスを用いたfault-proneモジュール判別方法

1. はじめに

ソフトウェアの品質管理や品質向上，fault-prone モジュール判別の精度改善などを目的として，ソフトウェアプロダクトの特徴を表す尺度（メトリクス）の算出方法が数多く提案されている．一般的にはプロダクトの特徴を表すメトリクスとして，ソースコード行数や Cyclomatic の複雑度，オブジェクト指向メトリクスなどの静的メトリクス [7] が広く用いられている．規模が大きく複雑なモジュールほど理解が難しく品質が低下しやすいことが指摘されており [28, 33]，このような特徴に基づいてテスト計画立案などの品質改善活動が実施される [17, 48]．静的メトリクスの他にもバージョンアップ時の差分情報から算出された変更メトリクス [31] や，モジュールの依存関係ネットワークから算出された構造メトリクス [51] などが提案されている．

しかし fault の混入といった信頼性を低下させる要因は，ソフトウェアプロダクトそのものの特徴のみならずソフトウェアを作成した開発者の特性に依存する部分にも少なからず存在する．例えば規模や複雑さが同じモジュールでも，経験の浅い開発者が作ったモジュールほど fault が混入される可能性が高いといえる．また多くの開発者によって変更が加えられたモジュールほど，個々の思考過程の違いや機能の実装手段の混在により意図の読み違いを誘発しやすく，結果として fault が混入されやすくなると考えられる．

従来，開発者個々の特性（経験年数やプログラミングスキルなど）によるコー

ディング速度やデバッグ効率の違いについては数多くの指摘があり [9, 42]，プログラマの行動を計測するシステムも提案されている [47]．その一方で，開発者個々の活動の計測と計測結果に基づく管理に対する批判も存在する [1, 8]．個人の活動の計測という行為そのものが意図せずとも個人の評価につながるという指摘もあり [1]，これらの計測，及び計測結果に基づく分析は避けられてきた．しかしながら，肥大化の一途を辿りつつも短期間かつ限られたコストの中で高い信頼性が求められる近年のソフトウェア開発においては，効率的な信頼性の確保が求められていることも事実である．そのため，fault 混入の原因となる要因については定量的な分析に基づく正しい理解を得ることが必要であると考える．

本章では fault 増加に対する人的要因の理解と fault-prone モジュール判別の精度向上を目的として，開発者に着目したメトリクスに基づいた 4 つの仮説について分析を行う．用いるメトリクスは個々の開発者に関する変更行数やコミット回数などと，個々のモジュールに付随する開発者数やある開発者 X の編集行数など（開発者メトリクス）である．具体的な仮説としては，仮説 1a：fault の混入のさせやすさに個人差がある，に関してまず分析を行い，この仮説が支持された場合に，仮説 1b：fault の混入のさせやすさは開発者の特性（変更行数やコミット回数など）から判断できる，について分析する．さらに仮説 2：多くの開発者が変更したモジュールには fault が含まれやすい，及び仮説 3：開発者に関する情報が fault を含んでいる可能性の高いモジュール（fault-prone モジュール）の判別に有効である，を実験的に確かめる．分析及び実験の対象としては開発プロダクトの規模が大きく，数多くの開発者が参加している Eclipse の開発プロジェクトから収集したメトリクスデータを用いる．

仮説 1a や仮説 1b に関しては経験的には正しいと理解されている部分も一部あり，それを前提とした研究もある [37, 52] が，実際のソフトウェア開発データに対する分析の報告事例は少ない．特に Eclipse のような高い能力を持った開発者が集う巨大なソフトウェア開発プロジェクトで，開発者の fault の埋め込みやすさにどの程度の差があるかについては調査されていない．また仮説 2 に関しても開発者の増加により信頼性が低下することは経験的には知られているが，具体的にどの程度の強さの相関があるかは明らかにされていない．本章の貢献は，これ

らの仮説を一般公開されたソフトウェア開発プロジェクトデータから調査し，定量的な分析に基づいた知見を得たことにある．

3 章の構成は以下の通りである．まず，2 節で本章における開発者に関するメトリクスの説明と，4 つの仮説について述べ，3 節では分析対象のデータと仮説の分析手段について説明する．4 節で分析と実験の結果について述べ，5 節で研究の制約と貢献について議論を行い，6 節で関連研究について述べ，7 節で本章のまとめについて述べる．

2. 開発者メトリクスと信頼性に関する仮説

本節では本章で用いる開発者に関するメトリクスについてと，4 つの仮説について説明する．

2.1 開発者に関するメトリクス

本論文で用いる開発者に関する情報は以下の 2 種類である．

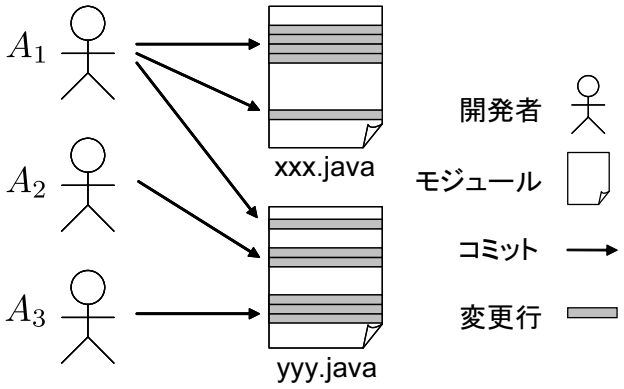
- 個々の開発者に関する特性（開発者 A の編集行数やコミット回数など）
- 個々のモジュールに付随し，かつ，そのモジュールの開発者に関する情報（モジュール X の開発者数や，モジュール X に対する開発者 A の編集行数など）

なお本論文では前者を「開発者に関する特性」と呼び，後者を「開発者メトリクス」と明記する．

2 種類のメトリクスの例を図 3.1 に示す．図 3.1 の上側の図 (a) が開発者と変更されたソースコードの関係を表す．表 (b) が図 (a) を基に算出された開発者に関する特性を表し，表 (c) が図 (a) を基に算出された開発者メトリクスを表す．

開発者個人に着目する仮説 1a と仮説 1b に関しては，個々の開発者に関する特性を用い，モジュールごとの分析を行う仮説 2 と仮説 3 に関しては，モジュール個々に付随するメトリクス（開発者メトリクス，及び従来の静的メトリクスと変更メトリクス）を用いる

(a) 開発者とモジュールの変更関係



(b) 算出された開発者個々の特性

	コミット数	変更行数	変更ソース数	…
A_1	3	6	2	
A_2	1	2	1	
A_3	1	3	1	

(c) 算出された個々のモジュールに関する開発者メトリクス

	開発者数	A_iが変更を加えた行数		
	NAuth	LOC(A_1)	LOC(A_2)	LOC(A_3)
xxx.java	1	5	0	0
yyy.java	3	1	2	3

図 3.1 開発者に関するメトリクスの例

2.2 開発者メトリクスに基づく仮説

仮説 1a：fault の混入のさせやすさには個人差がある．

開発者個々の fault の混入のさせやすさに個人差があるかを定量的に明らかにする．従来，コーディング速度やデバッグ効率の個人差については様々な指摘がされており [9, 42]，fault の埋め込みやすさについても個人差が少なからず存在すると考えられる．本章では開発者の fault の埋め込みやすさを表す指標として，1 コミットあたりいくつの fault を埋め込んでいたか（以降，fault 混入率）の値

を用いる．ソフトウェア進化の分析事例 [14] や可視化研究 [15] などでも，変更行為の 1 単位としてコミットが用いられており，本論文でもこれらを踏襲し単位コミットに基づく分析を行う．

仮説 1b：fault の混入のさせやすさは開発者の特性から判断できる．

仮説 1a の結果より fault の混入率に個人差があると確認された場合に限り，開発者個々の fault 混入率を個々の特性により判断可能かどうかを分析する．本章では開発者に関する特性として開発者個々の，1．コミット数，2．変更行数，3．変更したモジュールの数，4．変更したパッケージの数，5．前バージョンでの fault の埋め込み数，6. 新人であるかどうかの 6 つのメトリクスを用いる．各メトリクスの算出例は図 3.1 の図 (a) と表 (b) に示すとおりである．ここでの新人とは前のバージョンで一度も編集を行わなかった開発者のことを指す．これらの特性により fault の混入のさせやすさを判断できれば，fault が混入されていやすい部分をモジュールの開発担当からある程度推定できる可能性がある．

ここで開発者の特徴を表す指標として，リポジトリから収集可能なもののみを扱った理由は以下の 2 点が挙げられる．一つは，リポジトリから収集できる指標は自動で収集できるため，現場で役立てる際に収集コストの面で利点があるということである．もう一つは，個人の能力やスキルを表す指標の基準は主観に依存する部分が多く，その基準について曖昧さが含まれるためである．一方，上述した 6 つの指標はリポジトリからの自動収集が可能であり，また主観に依存しない客観的な定量的指標である．

仮説 2：多くの開発者が変更したモジュールには fault が混入されやすい．

この仮説の根拠としては，多くの開発者によって変更されたモジュールには，個々の思考過程の違いや複数の機能の実装手段が混在しやすく，理解容易性の低下や意図の読み違いにより fault を埋め込んでしまう確率が高くなると考えられるためである．また逆に一人の開発者により作成されたモジュールの場合，理解がしやすく fault 混入率が下がると考えられる．モジュールの規模や変更行数が

混入 fault 数と高い正の相関があることは数多くの指摘があり [27, 35]，変更を加えた開発者の数と混入 fault 数の関係についても分析を行う．

仮説 3：開発者に関する情報は fault-prone モジュールの判別に役立つ．

テストの効率化を目的として，fault が含まれている可能性の高いモジュール（fault-prone モジュール）の判別問題に関する研究が行われている [19, 25, 35]．fault-prone モジュール判別は，統計的な判別モデルを用いて fault が混入されているモジュールをテストの前に特定し，テスト工数を適切に割り振ることでテストの効率化や信頼性の向上を図る手段である．一般に fault-prone モジュールの判別に用いるモデルへの入力としては，ソフトウェアプロダクトから算出されたソースコードメトリクス（静的メトリクス [7] や変更メトリクス [31] など）が用いられる．本章ではこれら従来のソースコードメトリクスに加え，開発者の特徴から算出されたメトリクス（開発者メトリクス）を加えた実験を行い，開発者の情報が fault-prone モジュールの判別精度の向上に寄与するかを確かめる．

具体的な開発者メトリクスとしては，モジュールごとの変更を加えた開発者の数：NAuth と，ある開発者 A_i の変更を加えた行数：LOC(A_i) の 2 種類を用いる．このうち，LOC(A_i) は個々のモジュールについて，変更を加えた開発者の数（NAuth）だけ存在するメトリクスとなる（$i = 1..\text{NAuth}$）．図 3.1 の図 (a)，及び表 (c) に 2 つのメトリクスの算出例を示す．

3. 分析方法

3.1 分析対象のデータセット

実験には統合開発環境 Eclipse のバージョン 3.x 系列の 3 つのマイナーバージョン（ver.3.00，ver.3.10，ver.3.20）から収集したメトリクスデータを用いる．Eclipse を分析対象とする理由は，Eclipse は数多くの開発者が参加する大規模なプロジェクトであり，複数の開発者を対象とした分析が可能であるためである．表 3.1 に

表 3.1 データセット概要

	ver.3.00	ver.3.10	ver.3.20
開発者数	69	66	72
モジュール数	8,313	9,663	11,525
変更ありモジュール数	7,080	9,428	8,950
fault ありモジュール数	2,986	3,302	2,506
コミット回数	61,366	53,302	45,441
混入 fault 数	6,351	7,667	4,772
1 コミットあたりの fault 混入率	10.7%	17.4%	14.5%

収集したデータの概要を示す．なお，本章では 1 ソースコードファイルを 1 つの意味を持った機能の固まりである 1 モジュールと見なす．

収集したモジュール単位のメトリクスは提案する開発者メトリクス，及び静的メトリクス，変更メトリクスの 3 つである．各メトリクスの名称，及びその概要を表 3.2 に示す．開発者メトリクスとしては，変更を行った開発者の数：NAuth と，ある開発者 A_i の変更した行数：LOC(A_i) の 2 種類を収集した．静的メトリクスは Eclipse Metrics Plugin[1] を用いてソースコードファイルから 15 種類のメトリクスを収集した．変更メトリクスはバージョン管理ツールのリポジトリにおけるコミットログから，Moser らの提案する変更メトリクス [31] のうち代表的な（最大値や平均値を除く）メトリクス 7 種類を収集した．

モジュールごとの fault の有無と開発者ごとの混入 fault 数に関しては，Śliwerski らの提案する SZZ アルゴリズム [45] を用いて収集した．SZZ アルゴリズムはバージョン管理システムに記録されたソースコードのコミット日時，コミット者，コメントなどの情報と，障害管理システムに記録された報告 fault の ID，登録日時，コメントなどの情報を比較することにより，いつ，どのファイルに，誰が fault を混入させたかを特定する方法である．

[1] http://eclipse-metrics.sourceforge.net

表 3.2 モジュール単位のメトリクス

	名称	概要
開発者	NAuth	変更を加えた開発者の数
メトリクス	LOC(A_i)	開発者 A_i が変更した行数
静的	TLOC	総行数
メトリクス	MLOC	実行行数
	PAR	パラメータ数
	NOF	フィールド数
	NOM	メソッド数
	NORM	オーバライドしたメソッド数
	NSC	サブクラス数
	NSF	静的フィールドの数
	NSM	静的メソッドの数
	NBD	最大ネスト数
	VG	Cyclomatic の複雑度
	DIT	継承の深さ
	LCOM	凝集性欠如の度合い
	WMC	VG の総和
	SIX	特殊化指標の平均 (NORM+DIT)/NOM
変更	Codechurn	変更行数 （追加行数＋削除行数）
メトリクス	LOCAdded	追加行数
	LOCDeleted	削除行数
	Revisions	改訂回数
	Age	経過時間
	BugFixes	fault 修正が行われた回数
	Refactorings	リファクタリングが行われた回数

3.2 各仮説の分析方法

仮説 1a と仮説 1b の分析方法

仮説 1a の分析手段としては，まずバージョン全体での全開発者の fault 混入率 $P(A_i)$ の度数の分布を確認する．さらに，どのバージョンでも同程度の fault 混入率を持つ開発者に着目し，その fault 混入率を比較することにより個人差の有無を確認する．これは度数分布の確認のみでは，fault 混入率の偶然のばらつきによって個人差があると見なしてしまう可能性があるためである．例えばコミット数の少ない開発者の場合，わずかな混入 fault 数の増減によって fault 混入率の値が大きくばらつきやすい．このような偶然のばらつきを個人差と見なしてしまうことを避けるために，バージョンごとに開発者個々の fault 混入率のばらつきを考慮に入れて分析を行う．

さらに仮説 1a が支持，すなわち fault 混入率に個人差があると確認できた場合のみ，fault 混入率が開発者の特性と相関があるかを調べる．開発者個々の特性を表す指標として 2 章の仮説 1b で述べた 5 つのメトリクスと開発者の fault 混入率それぞれの単相関係数を算出し，無相関検定により相関の有無を確かめる．

仮説 2 の分析方法

仮説 2 を確かめるために，1. 偏相関係数と 2. 重回帰分析を用いた分析を行う．偏相関係数を用いることにより fault 数と開発者数の間の相関の有無を確かめ，重回帰分析を用いることにより開発者数が他のメトリクスと比較してどの程度強い関係を持つかを調べる．

偏相関係数を用いる理由としては，fault 数と開発者数の単相関係数のみを調べた場合，他の変数の寄与による擬相関が検出される可能性があるためである．例えばモジュールの規模（行数）は，fault 数と正の相関を持っており [27]，また開発者数とも正の相関があると考えられる．このような交絡変数が存在する場合，fault 数と開発者数の間が実際には無関係であっても見かけ上の相関（擬似相関）が得られてしまう．そこで本章では単相関係数による分析に加え，偏相関係数 [11] を用いた分析を行う．偏相関係数は他の変数による寄与を排除した相関のことで

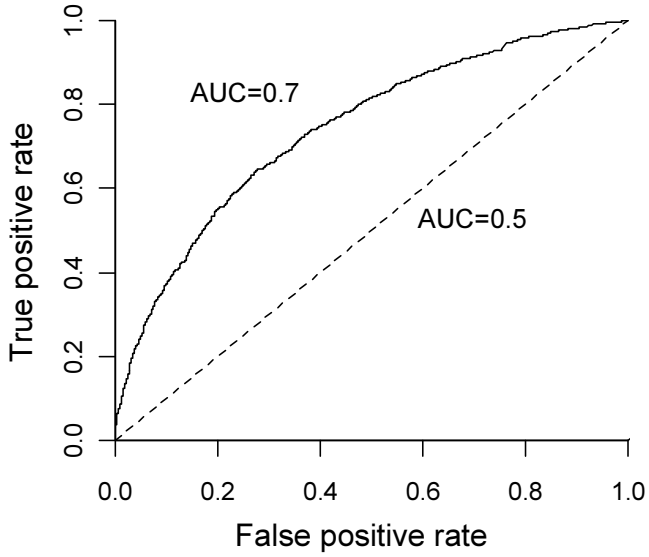


図 3.2 ROC 曲線と対応する AUC 値の例

あり，他変数の値が同一という条件の下でのある 2 つの変数間の相関と言い換えることが可能である．偏相関係数を用いることで，上記の擬似相関の問題を避けることが可能である．寄与を取り除く変数としては，静的メトリクスと変更メトリクスの代表値として，それぞれ直感的かつ fault 数と強い相関を持つとされる行数に関するメトリクス（総行数と変更行数）を用いる．

重回帰分析とは，目的変数と目的変数に関連する複数の説明変数の関係を一次式で表現する手法の一つである．fault 数を目的変数とする重回帰モデルを構築し，推定された各説明変数の標準偏回帰係数を比較することで，fault 数の増加に対する寄与の強さをメトリクス間で比較することが可能である．重回帰モデルの構築に用いるメトリクスとしては，開発者数に加え，偏相関係数の分析と同様に総行数と変更行数の 2 つを比較対象として用いる．

仮説 3 の fault-prone モジュールの判別実験

仮説 3 を確かめるために，開発者メトリクス，静的メトリクス，変更メトリクスのそれぞれを利用する/しない場合の組合せ計 7 通りの fault-prone モジュール判

別実験を行い，各メトリクスの有無による判別精度の変化を比較する．判別モデルとしては，2 章と同様に線形判別分析 [12]，ロジスティック回帰分析 [22]，分類木 [3] の 3 つを用いる．各判別モデルの詳細について 2 章 2 節に示すとおりである．実験では，過去のバージョンデータでモデルを構築し，次期バージョンのモジュールの fault の有無を推定するという利用方法を想定し，ver.3.00 から ver.3.10，及び ver.3.10 から ver.3.20 の 2 通りについて判別実験を行う．

判別結果の評価指標としては 2 章 4.3 節で述べた 3 つの尺度（再現率，適合率，F1 値）に加え，ROC 曲線（Receiver Operating Characteristics Curve）の曲線下面積（AUC: Area Under the Curve）を用いる．ROC 曲線は横軸に False Positive Rate，縦軸に True Positive Rate を取ったときの判別結果の変化を表す曲線である．AUC 値は曲線下の面積のことであり，ROC 曲線に対する AUC 値とは判別結果がどの程度擬陽性率を抑えつつ陽性率を確保できるかを表す指標となる．ROC 曲線とそれに対応する AUC 値の例を図 3.2 に示す．ROC 曲線に対する AUC 値（以降単に AUC 値と略す）の値域は [0-1] を取り，値が 1 に近いほど判別精度が高くランダムに判別を行った場合におよそ 0.5 の値を取る．

2 章 4.3 節で述べた 3 つの尺度（再現率，適合率，F1 値）は fault 有無の二値判別結果の正しさを表し，ROC 曲線に対する AUC 値は fault を含んでいる可能性の高いモジュール順に並べたときの順序の正しさを表す．

4. 実験結果

4.1 仮説 1a：fault の混入のさせやすさには個人差がある．

全バージョンにおける全開発者（108 人）の平均 fault 混入率のヒストグラムを図 3.3 に示す．モジュール全体での平均 fault 混入率は 17.4%（1 コミットあたり混入される fault 数が約 0.17 個）であり，開発者の全体の約半数は fault 混入率 0%〜20%の区間に分布している．一方で fault 混入率が 30%以上の開発者は 17%（18 人）存在しているほか，fault 混入率が 50%を超える開発者も存在しており，fault 混入率にある程度の個人差があることが読み取れる．

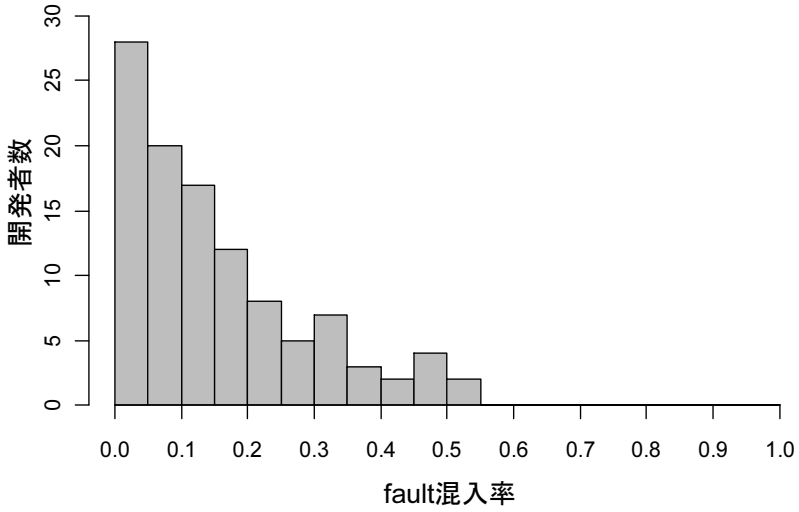


図 3.3 平均 fault 混入率のヒストグラム

次に開発者個々のバージョンごとの fault 混入率を分析する．開発者及びバージョンごとの fault 混入率を図 3.4 に示す．ここでは全てのバージョンで 1 回以上の変更を加えた開発者 39 人のみを抽出している．破線は全モジュールの平均 fault 混入率（17.4%）を表す．開発者のソート基準は 3 バージョンの fault 混入率の変動係数（相対的なばらつき度合い）の昇順であり，左側の開発者ほどバージョンごとの fault 混入率のばらつきが小さい．なお，開発者のラベルについては個人名を特定できないようにソート順にインデックスを割り振られている．

図より，全てのバージョンで fault 混入率が低い開発者（A_1，A_3，A_7，A_{10}，A_{12}）や，高い開発者（A_{13-14}，A_{17}）が確認できる．このうち A_{12} と A_{14} は 3 つのバージョン全てで 1,000 回以上のコミットを行っていた開発者であるが，それぞれの平均 fault 混入率は 0.07 と 0.38 であり，5 倍以上の個人差があることが分かる．また，Friedman 検定を用いて開発者ごとの fault 混入率の差を検定したところ有意な差が確認できた（p 値=1.29E-05）．これらのことから，仮説 1a は支持されたといえる．

なお上記の結果は，1 コミットあたりの fault 数を fault 混入率として用いた結果であるが，単位行数あたりの fault 数を用いても同様の傾向が得られた．1,000 行あたりの混入 fault 数に基づく結果を付録の図 A.1 と図 A.2 に示す．

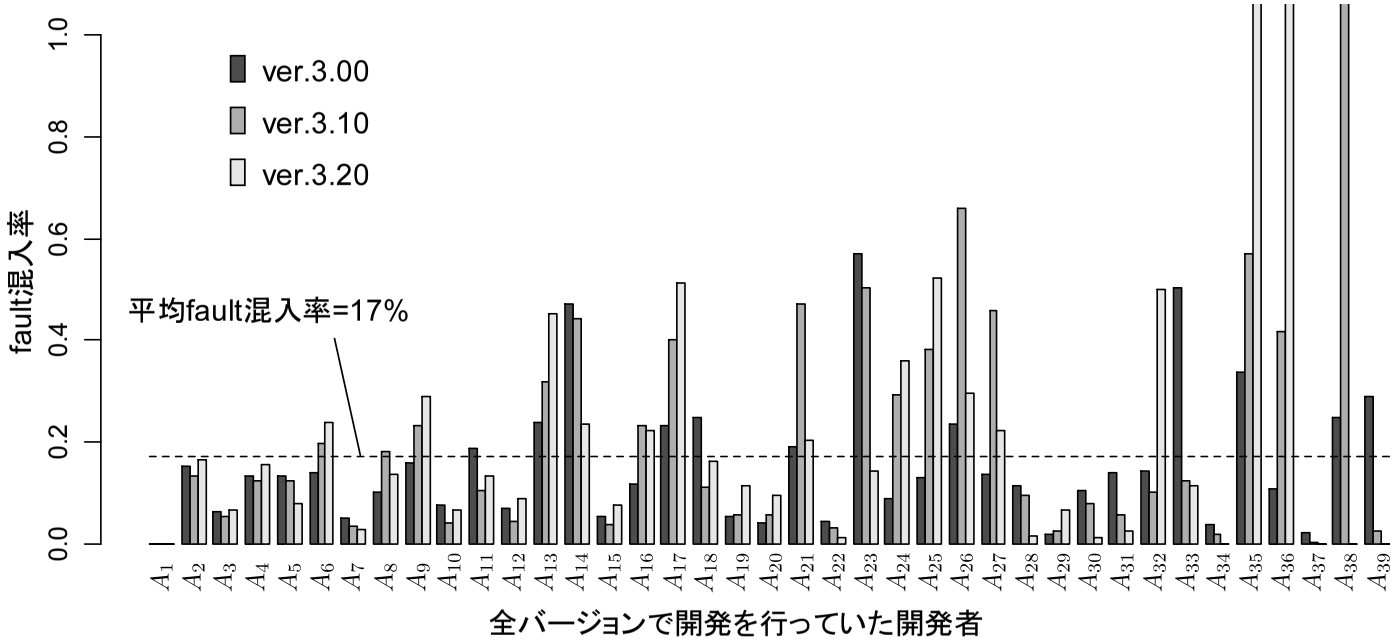


図 3.4 各開発者のバージョンごとの fault 混入率

4.2 仮説 1b：fault の混入のさせやすさは開発者の特性から判断できる．

開発者個々の 6 つの特性それぞれと fault 混入率との相関係数を表 3.3 に示す．表中の「*」は有意水準 5%の無相関検定により相関が有意にあったケースを指す．なお，ver.3.00 の前バージョンでの fault 混入率と新人であるか否かは，ver.3.00 がバージョン 3 系列の第一リリースバージョンであるため算出不可である．

表より，前バージョンでの fault 混入率は現バージョンでの fault 混入率と正の相関を持っていることが分かる．これは前バージョンで多くの fault を混入させていた開発者は，次のバージョンでも多くの fault を混入させる可能性が高いことを意味する．一方，変更行数，変更ソースコード数，コミット数に関してはほぼ無相関となっており，変更パッケージの数に関しても ver.3.00 でのみ正の相関を持っているが，他のバージョンでは有意な相関はみられない．すなわち，fault 混入率はコミット数や変更行数といったアクティビティの高さや，変更ソースコード数や変更パッケージ数といったどの程度幅広くプロジェクトに関与しているかといった値では一概に判断できないといえる．また新規参入開発者とバグ混入率の間にも一貫した関係は見られない．ただし本論文で対象とした Eclipse のような OSS 開発では，新規参入者であっても高い技術を持っている可能性が高い．そのため一般的な企業では異なる結果が得られる可能性がある．

これらの結果から fault の混入のさせやすさは前バージョンでの fault 混入率からのみある程度推測可能であり，仮説 1b は支持されたといえる．

なお，単位行数あたりの fault 数を fault 混入率として用いた結果も同様に，前バージョンでの fault 混入率のみ有意な相関が見られた．結果を付録の表 A.1 に示す．

表 3.3 開発者ごとの特性と fault 混入率の相関

	ver.3.00	ver.3.10	ver.3.20
コミット数	-0.07	-0.19	-0.18
変更行数	-0.05	0.09	-0.14
変更ソースコード数	0.00	-0.14	-0.22
変更パッケージ数	0.35*	-0.06	-0.08
前バージョンでの fault 混入率	算出不可	0.47*	0.32*
新人であるかどうか	算出不可	-0.04	0.03

*：有意水準 5%の無相関検定により有意に相関あり

表 3.4 混入 fault 数と開発者数の単/偏相関係数

	ver.3.00	ver.3.10	ver.3.20
単相関係数	0.303*	0.389*	0.303*
偏相関係数	0.092*	0.166*	0.104*

*：有意水準 5%の無相関検定により有意に相関あり

4.3 仮説 2：多くの開発者が変更したモジュールには fault が混入されやすい．

偏相関係数による分析結果

fault 数と変更した開発者の数（NAuth）の単相関係数及び，総行数（SLOC）と変更行数（Codechurn）の影響を取り除いたときの偏相関係数を算出し，その関係の強さを確かめる．fault 数と開発者数の単相関係数と偏相関係数を表 3.4 に示す．表中の「*」は，有意水準 5%の無相関検定により相関が有意にあったケースを指す．

有意水準 5%の無相関検定により，全てのバージョンについて fault 数と開発者数は有意に正の相関を持っていた．具体的な値としては，その相関の強さはどのバージョンでも 0.1 程度と弱く，開発者数の増加は fault 数の多さに対して弱く寄与しているといえる．

表 3.5 fault 数を目的変数とする重回帰モデルの標準偏回帰係数

	ver.3.00	ver.3.10	ver.3.20
開発者数: NAuth	0.101	0.164	0.124
総行数: TLOC	0.279	0.091	0.141
変更行数: Codechurn	0.276	0.537	0.383

重回帰分析による分析結果

混入 fault 数を目的変数とした重回帰モデルを構築し，3 つのメトリクス（開発者数，総行数，変更行数）の標準偏回帰係数を比較する．表 3.5 に 3 つのメトリクスの標準偏回帰係数をバージョンごとに示す．

最も fault 数に対して最も強く寄与しているのはどのバージョンにおいても変更行数であった．開発者数の標準偏回帰係数はどのバージョンでも 0.10〜0.16 であり，開発者数は fault 数を目的変数とする重回帰モデルの説明変数として寄与していることが分かる．特に ver.3.10 では開発者数の標準偏回帰係数は総行数よりも大きく，これは総行数よりも開発者数の方が fault の増加に強く寄与していると見なすことができる．

これらの結果から，同規模・同変更量であれば開発者の数が多いほどわずかではあるが fault が含まれやすくなり，またバージョンによっては開発者数は行数よりも強く寄与するといえる．つまり仮説 2 は支持されたと考えられる．

4.4 仮説 3：開発者に関する情報は fault-prone モジュールの判別に役立つ．

F1 値による判別精度の評価

3 種類のメトリクスの組合せ 7 通りを用いて，次バージョンの fault-prone モジュールの判別実験を行った際の結果（二値判別結果の評価尺度：再現率，適合率，F1 値）を表 3.6 に示す．メトリクスの列の「o」は該当するメトリクスを判別に用いた場合を指している．表中の平均ランクとは各モデルごとに判別精度の

表 3.6 fault-prone モジュール判別の結果（評価尺度：再現率，適合率，F1 値）

尺度	#	メトリクス 開発者	静的	変更	ver.3.00 から ver.3.10 を判別 LDA	LRA	CT	ver.3.10 から ver.3.20 を判別 LDA	LRA	CT	平均 rank
再現率	1	o			0.303	0.413	0.239	0.147	0.204	0.145	4.8
	2		o		0.217	0.119	0.164	0.148	0.078	0.121	6.3
	3	o	o		0.343^{+}	0.443^{+}	0.354^{+}	0.182^{+}	0.216^{+}	0.263^{+}	3.7^{+}
	4			o	0.146	0.199	0.441	0.223	0.230	0.397	4.2
	5	o		o	0.215^{+}	0.343^{+}	0.541^{+}	0.237^{+}	0.279^{+}	0.320^{-}	3.0^{+}
	6		o	o	0.193	0.209	0.441	0.247	0.231	0.396	3.3
	7	o	o	o	0.241^{+}	0.347^{+}	0.541^{+}	0.241^{-}	0.287^{+}	0.320^{-}	2.2^{+}
適合率	8	o			0.717	0.669	0.626	0.711	0.799	0.649	3.7
	9		o		0.539	0.599	0.574	0.428	0.478	0.431	6.8
	10	o	o		0.683^{+}	0.670^{+}	0.518^{-}	0.652^{+}	0.796^{+}	0.522^{+}	5.3^{+}
	11			o	0.780	0.791	0.656	0.715	0.705	0.632	3.2
	12	o		o	0.841^{+}	0.763^{-}	0.595^{-}	0.719^{+}	0.748^{+}	0.736^{+}	2.3^{+}
	13		o	o	0.776	0.772	0.656	0.709	0.719	0.634	3.5
	14	o	o	o	0.817^{+}	0.763^{-}	0.595^{-}	0.717^{+}	0.757^{+}	0.736^{+}	2.7^{+}
F1 値	15	o			0.426	0.511	0.346	0.243	0.325	0.237	4.7
	16		o		0.310	0.198	0.255	0.220	0.135	0.189	6.7
	17	o	o		0.457^{+}	0.534^{+}	0.420^{+}	0.285^{+}	0.340^{+}	0.349^{+}	3.7^{+}
	18			o	0.245	0.319	0.528	0.339	0.347	0.488	4.2
	19	o		o	0.342^{+}	0.474^{+}	0.566^{+}	0.356^{+}	0.406^{+}	0.446^{-}	2.8^{+}
	20		o	o	0.309	0.329	0.528	0.367	0.350	0.487	3.3
	21	o	o	o	0.372^{+}	0.477^{+}	0.566^{+}	0.361^{-}	0.416^{+}	0.446^{-}	2.2^{+}

LDA：線形判別分析　　$^{+}$：開発者メトリクスとの組合せにより精度向上
LRA：ロジスティック回帰分析　　$^{-}$：開発者メトリクスとの組合せにより精度低下
CT　：分類木

順位付けを行った際の平均の順位を表しており，値が小さいほど精度が高いメトリクスの組合せであることを意味する．各評価尺度の横の「$^{+}$」と「$^{-}$」は，あるメトリクスを用いた際の判別結果に対して，開発者メトリクスを加えたことで精度が向上／低下した場合を指す．

まず再現率と適合率の調和平均である F1 値の平均ランクに着目すると，どのメトリクスに対しても開発者メトリクスを加えることで精度が改善されており，その平均改善幅は 0.093 である．また最も精度が高いメトリクスの組合せは全メトリクスを加えた場合（21 行目，以降では#21 と表記）であり，従来よく用いられていた静的メトリクス単体（#16）は最も精度が低く，次いで開発者メトリクス単体（#15）の精度が低い．

再現率と適合率それぞれに着目すると，開発者メトリクスを加えたことによる具体的な精度の改善幅は，再現率は平均+0.084 であり，適合率は+0.057 であった．再現率，適合率の両方の精度が改善されたことから，開発者メトリクスを加えることで fault ありの見逃しと fault ありの誤検出の両方を減らすことができたといえる．

ROC 曲線の AUC 値による判別精度の評価

順序の考えに基づく指標（ROC 曲線の AUC 値）を判別結果の評価尺度として用いた際の実験結果を表 3.7 に示す．まず，各メトリクス単体で判別を試みた場合に着目すると，変更メトリクス（#2）が最も精度が高く，次いで開発者メトリクス（#3）であり，従来用いられていた静的メトリクス（#1）は最も精度が低かった．また開発者メトリクスを加える場合と加えない場合で平均ランクを比較すると，静的メトリクスは 6.7 から 3.3，変更メトリクスは 3.5 から 2.0，静的と変更メトリクスの組合せは 4.8 から 2.0 と，開発者メトリクスを加えることによる精度の改善が確認できる．

開発者メトリクスとの組合せにより判別精度の向上が確認できたため，仮説 3 は支持されたといえる．

表 3.7 fault-prone モジュール判別の結果（評価尺度：ROC 曲線の AUC 値）

#	メトリクス 開発者	静的	変更	ver.3.00 から ver.3.10 を判別 LDA	LRA	CT	ver.3.10 から ver.3.20 を判別 LDA	LRA	CT	平均 rank
1	o			0.832	0.846	0.657	0.815	0.833	0.660	5.2
2		o		0.732	0.739	0.697	0.720	0.722	0.690	6.7
3	o	o		0.825^{+}	0.849^{+}	0.653^{-}	0.818^{+}	0.842^{+}	0.713^{+}	4.8^{+}
4			o	0.818	0.838	0.738	0.893	0.894	0.771	3.5
5	o		o	0.861^{+}	0.876^{+}	0.767^{+}	0.883^{-}	0.887^{-}	0.774^{+}	2.0^{+}
6		o	o	0.820	0.834	0.738	0.888	0.894	0.771	3.3
7	o	o	o	0.853^{+}	0.872^{+}	0.767^{+}	0.887^{-}	0.890^{-}	0.774^{+}	2.0^{+}

LDA：線形判別分析　　$^{+}$：開発者メトリクスとの組合せにより精度向上
LRA：ロジスティック回帰分析　　$^{-}$：開発者メトリクスとの組合せにより精度低下
CT　：分類木

5. 研究の制約と貢献

本章では開発者の特性と信頼性の関係の理解を目的として，開発者個々の特性に基づいた分析を行ったが，これらの計測手段及び分析方法を用いて開発者個人の評価を行うことは適切ではない．ソフトウェア開発において個人の活動を計測することに対する批判は数多く存在する [1, 8]．ソフトウェア開発のような知識生産では，生産性や fault の混入率のような指標のみでは個人の能力を判断できない [1] ことがその理由の一つである．

例えば，仮説 1b で示した A_{14} の場合，その fault 混入率は全てのバージョンで平均よりも高かった（図 3.4 参照）．しかし，これは A_{14} の開発担当パッケージの難易度の高さに起因している可能性もあり，A_{14} が fault を埋め込みやすい開発者と考えるべきではない．特に仮説 1b での fault 混入率は前バージョンと次バージョンで正の相関を持つという結果については，開発担当の難易度が原因である可能性もあり，また開発者の学習効果も存在するため，多くの fault を混入させた開発者の全てが次バージョンでも多くの fault を混入させるとは断言できない．

さらに A_{14} は全バージョンで 1,000 回以上のコミットを行っていた．分析対象とした Eclipse のようなオープンソースソフトウェアの開発では，ある程度 fault が混入していても頻繁にリリースするべきである [41] という指摘もあり，この観点では A_{14} は Eclipse 開発プロジェクトに大きく貢献していた開発者であるともいえる．

このように開発者の能力は表層的な指標のみで判断することは難しく，また数値化しにくい部分も多いため，これらの指標に基づいた開発者の評価を行うことは不適当である．

本章の貢献は開発者個々の評価ではなく，定量的な分析に基いたソフトウェア開発に対する以下の理解を得たことにある．

- fault 混入率はプログラミングスキルやコーディング速度と同様に個人差がある．
- fault 混入率は前バージョンと現バージョンで正の相関を持つ．
- 規模や複雑さと同様に，変更を加えた開発者数は fault 数と正の相関を持つ．
- 開発者メトリクスは fault-prone モジュールの判別精度向上に寄与する．

これらの知見に基づいた信頼性確保のための活動としては以下のような点が考えられる．

- 前バージョンで fault 混入率が高かった開発者の担当部分には重点的にテストを行う．ただし，この開発者個々の fault 混入率を利用する際には慎重な判断が必要である．この指標値は品質保証部門などの開発チームから独立した部門が品質改善や品質管理のためのみに役立てる．
- 多くの開発者が関与しているモジュールに対しては，開発担当者割り当ての再検討を行う．もしくは重点的にレビューやテストを実施する．
- fault-prone モジュールの判別を行う際には，一般に用いられている静的メトリクスや変更メトリクスのみならず，開発者メトリクスを加えることで判別精度の向上が見込まれる．

また本章で示した，リポジトリデータから抽出される開発者に関するメトリクスは，プロダクトに対する品質改善活動のみならず，開発者の教育という目的にも役立てられる可能性がある．例えば他の開発者と比べて fault 混入率が恒常的に高い開発者を本章でのリポジトリマイニング方法から特定し，経験豊富な開発者とのペアプログラミングなどを通すことにより，当該開発者の技術の向上を図ることも可能である．さらにマイニングされた開発者の活動履歴を開発者自身に提示することで，開発者が自分自身の活動結果に対して正しい認識を得るという PSP（Personal Software Process）にも役立てることも可能である．

さらに人的要因を考慮に入れることによる効果として，fault 混入という現象をより幅広い視点から捉えられるという点が挙げられる．fault の混入という現象には人的要因が絡んでいることは直感的にも経験的にも明らかであり，信頼性の改善という観点をプロダクトの特徴のみから論じるのは十分であるとはいえない．ソフトウェアの信頼性に対して人的要因を考慮に入れた定量的な理解を得たことは本研究の貢献の一つである．

6. 関連研究

ソースコードなどのソフトウェアプロダクトの特徴から計測されたメトリクスに基づいた信頼性に関する研究が多数行われている [5, 10, 13, 27, 35, 43, 46, 50]．特にモジュールの規模に基づいた信頼性の分析は，これまで数多くの報告がなされている [13, 27, 35]．Gaffney[13] は規模の増大により fault 数が増加することを指摘しており，Koru ら [27] は規模が大きいモジュールほど混入 fault 数は多いが，単位規模あたりの fault 数（fault 密度）は規模の小さいモジュールほど多いことを指摘している．規模の他にもモジュールの複雑さに基づいた分析 [43, 10] や，オブジェクト指向言語特有のメトリクスに基づいた分析 [5, 46, 50] などが行われている．これらの研究は，規模や複雑さといったプロダクトの特徴と信頼性の関係の分析であるが，プロダクトの開発プロセス，特に開発者に着目した分析事例は少ない．

開発者の特性に基づく分析結果の報告事例として Schröter ら [44] の研究がある．

Schröter らは障害管理システムからのデータマイニング方法を示し，その分析結果の一部として本章の仮説 1a に該当する fault 混入率には個人差があること，及び仮説 1b の一部に該当する fault 混入率と変更ファイル数の間には関連がないことを確かめている．この結果は本章の結果とも一致する．Schröter らの分析結果は単一バージョンの分析結果である一方，本章では複数のバージョンで仮説の検証を行っており，また変更ファイル数のみならずコミット数や変更行数，変更モジュール数などの複数の指標と混入 fault 数の関連を調べている点で異なる．

過去のバージョンのメトリクスデータを用いて，次期バージョンに fault が残存するかの判別を試みる fault-prone モジュール判別に関する研究も数多く実施されている [19, 25, 31, 32, 35, 40, 50]．Moser ら [31] はバージョンアップ時の差分情報を用いた変更メトリクスを提案し，判別精度の向上を試みている．本章の仮説 2 と仮説 3 で用いた変更メトリクスはこの研究の提案に基づく．Moser らの提案する変更メトリクスには変更を加えた開発者の数というメトリクスが含まれているが，実験では 17 種の変更メトリクス（変更行数やコミット数など）全てを加えたときの判別精度の改善を示しており，開発者数そのものが混入 fault 数の増加に寄与しているか（本章の仮説 2 に該当），またどの程度 fault-prone 判別の精度向上に寄与しているか（仮説 3 に該当）については明らかにされていない．

fault-prone モジュール判別に開発者の情報を用いた事例として，Weyuker ら [49] による研究がある．この研究では開発者の増加が fault の増加と関連を持つかという本章の仮説 2 に関連する分析を行っている．分析の結果として Weyuker らは，fault の増加は様々な要因によるものあり，開発者数の増加は主たる原因にはならないと結論付けている．しかし，Weyuker らの分析方法は従来の静的メトリクスを用いた fault-prone モジュール判別モデルに対して，開発者に関するメトリクスを加えた際の判別精度の変化が小さいことから得たものである．この分析手段は判別モデルの精度改善幅の確認による間接的な方法であり，開発者数と混入 fault 数が直接的にどのような関係を持つか（持たないか）については言及されていない．一方，本章では開発者数と fault 数の偏相関係数を調べており（仮説 2），fault 数と強い相関を持つ [13, 31] とされる行数と変更行数の影響を取り除いている点，及びその結果として開発者数と fault 数には少なからず正の相関

があることを確かめている点で異なる．さらに fault-prone モジュール判別に対する開発者メトリクスの寄与の分析（仮説 3）においては，複数の判別モデル，及び複数のメトリクスの組合せについて詳細に実験している点で異なる．

7. まとめ

本章では fault-prone モジュール判別モデルの精度向上を目的として，開発者の特性を表す開発者メトリクスの提案を行い開発者メトリクスに基づいた以下の 4 つの仮説について分析を行った．

- 仮説 1a：fault の混入のさせやすさには個人差がある．
- 仮説 1b：fault の混入のさせやすさは開発者の特性から判断できる．
- 仮説 2：多くの開発者が変更したモジュールには fault が混入されやすい．
- 仮説 3：開発者に関する情報は fault-prone モジュールの判別に役立つ．

Eclipse プロジェクトから収集した 3 つのバージョンのモジュールデータを用いた分析と実験の結果得られた知見は以下の通りである．

- fault 混入率には少なくとも 5 倍以上の個人差がある．
- 前バージョンで多くの fault を混入させていた開発者は，次期バージョンでも多くの fault を混入させる可能性が高い．
- 多くの開発者が変更を加えたモジュールほど fault が混入されやすい．変更を加えた開発者の数と fault 数の具体的な偏相関係数の値は 0.104 であった．
- 開発者メトリクスが判別モデルの精度向上に有効であり，他のメトリクスに対して開発者メトリクスを加えた際の F1 値の平均向上幅は 0.093 であった．また開発者メトリクスを加えることで再現率，適合率の両方の精度が改善されたことから，開発者メトリクスにより fault ありの見逃しと fault ありの誤検出の両方を減らすことができたといえる．

第4章　結論

本論文では，fault-prone モジュール判別の判別精度向上を目的として，フィットデータ中の特異なケースである外れ値を除去する手法の適用，及び fault 有無に対する人的要因を考慮するための開発者メトリクスの提案を行った．

まず，特異なケースを削除する外れ値除去法をフィットデータに適用した際の効果を実験的に確かめた．実験では，NASA/WVU IV&V Facility が公開している 3 つのプロジェクトのデータセットを題材として，群の傾向から極端に外れた傾向を持つモジュールを優先的に除去する手法 2 種（MOA，LOFM）と，群の傾向と相反して fault を含む／含まないモジュールを優先的に除去する手法 2 種（RBM，CC-MOA）の 4 つの外れ値除去法の適用の効果を，3 つの代表的な判別モデルについて交差検証法により評価した．実験の結果，LOFM を除く 3 つの外れ値除去法を用いた場合，いずれの判別モデル，データセットに対しても判別精度が改善され，F1 値の平均向上幅は MOA では 0.139，RBM では 0.137，CC-MOA では 0.123 であった．また精度が改善した 3 つの外れ値除去法については，判別モデル間の性能差を小さくする効果が見られた．外れ値除去を適用することで，データセットごとに最適なモデルが異なるという問題が緩和され，モデル利用者にとって利便性が増すことが期待される．

次に，ソフトウェアプロダクトを作成した開発者の特性を表す開発者メトリクスの提案を行い，4 つの仮説に基づいた分析と実験を行った．分析の結果，開発者個々の fault 混入率には 5 倍以上の個人差があること，及び多くの開発者が変更を加えたモジュールほど fault が混入されやすいことが分かった．また開発者メトリクスを用いた fault-prone モジュールの判別実験により，開発者メトリクスが判別精度の向上に寄与することが確認し，その F1 値の平均向上幅は 0.093 であった．開発者メトリクスの追加により再現率，適合率の両方の精度が改善され

たことから，fault を含むモジュールの見逃しと誤検出の両方を減らすことができたといえる．

これら本論文の成果により，ノイズが少なく，かつ人的要因を考慮に入れたフィットデータに基づいて判別モデルを作成することが可能となり，fault の有無を正確に推定することが可能となる．これにより fault 検出を目的とするテスト計画者は，fault が含まれているモジュールをテスト前に正確に特定することが可能となり，テストの効率化とソフトウェア信頼性の向上につながると考えられる．

謝辞

本研究を進めるに当たり多くの方々に，御指導，御協力，御支援頂きました．ここに感謝の意と共に御名前を記させて頂きます．

はじめに，主指導教官であり，私が博士前期課程及び博士後期課程に至るまで御指導を頂いた奈良先端科学技術大学院大学 情報科学研究科 松本 健一 教授に心より感謝致します．研究に対する直接的な指導に限らず，研究に取り組む上での心構えから研究者としての在り方など，ありとあらゆる面で熱心な御指導を頂きました．私が本研究を成し遂げることができたのは，先生の研究に対する深い御理解と的確かつ暖かい御助言に励まされたおかげです．先生の御尽力に敬意を表し，最初に深く感謝致します．

奈良先端科学技術大学院大学 情報科学研究科 関 浩之 教授には，本研究の発表において的確・適切な御意見，御指摘を頂きました．頂いた御助言や研究に対する鋭い御指摘は常に研究の展望を見据えたものであり，本研究の発展に欠かせないものばかりでした．心より厚く感謝致します．

奈良先端科学技術大学院大学 情報科学研究科 飯田 元 教授には，本研究の成果発表にあたり数多くの的確な御意見・御指摘を頂きました．頂いた御指摘により本研究をよりよい方向へ進歩させることができました．心より厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 門田 暁人 准教授には，本研究に対して的確なアドバイスを賜り，研究の着想から論文執筆，プレゼンテーションの作成に至るまで，常に数限りない適切な御助言と懇篤なる御指導を頂きました．また，先生の研究に対する真摯で熱意ある姿勢から，研究に対し大きな志を抱くことができました．本学で 4 年間，先生の元で研究を進めることができたことは大変幸せなことでした．心より厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 大平 雅雄 助教には，研究に対する姿勢，研究の難しさ，楽しさ，様々なことを学ばせて頂きました．また，研究以外でも大変親しく接して頂き，公私にわたり多大な御尽力を賜りました．心より厚く感謝致します．

奈良先端科学技術大学院大学 情報科学研究科 森崎 修司 助教には，研究に対

する姿勢や社会人としての厳しさなど大変勉強させて頂きました．厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 角田 雅照 特任助教には，本研究の遂行に必須となる統計学に関し，数多くの御指導を頂きました．心より厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 松村 知子 特任助教には，研究生活に関する御助言を賜りました．厚く御礼申し上げます．

奈良先端科学技術大学院大学 情報科学研究科 川口 真司 助教には，研究を進めるにあたり的確な御意見，御指導を頂きました．深く感謝致します．

神戸大学大学院 工学研究科 中村 匡秀 准教授には，論文執筆に関する御指摘や研究の楽しさ，学生指導に対する姿勢など，数多くの事柄を学ばせて頂きました．心より厚く御礼申し上げます．

神戸大学大学院 工学研究科 井垣 宏 特命助教，京都産業大学 コンピュータ理工学部 玉田 春昭 助教，大阪大学大学院 情報科学研究科 柿元 健 氏，ソニー株式会社 岡本 圭司 氏には研究生活のみならず，私生活の場においても非常に楽しい時間を共有させて頂きました．心より厚く御礼申し上げます．

宇宙航空研究開発機構 情報・計算工学センター 片平 真史 氏には，ご多忙の中，氏の御好意により宇宙航空研究開発機構でのインターンシップの機会を賜りました．本インターンシップは新たな発見に満ち溢れており，私の見識を広めさせて頂くとともに，学生生活では得難い貴重な経験となりました．氏をはじめとする同センターの皆様に，心より深く感謝申し上げます．

慶應義塾大学 先導研究センター 神武 直彦 准教授には，宇宙航空研究開発機構との共同研究に際し，ご多忙の中，多大な御助力を頂きました．心より厚く御礼申し上げます．

京都産業大学 コンピュータ理工学部 平井 重行 准教授には，大学院への進学の際に親身に相談に乗って頂き，また先生の御助言，御鞭撻により充実した大学生活を過ごすことができました．心より厚く御礼申し上げます．

阿迦井塾 阿迦井 和嘉 先生には，学ぶことの楽しさ，考えることの楽しさを教えて頂きました．先生との出会いなくして現在の私はないものと確信しておりま

す．心より厚く御礼申し上げます．

奈良先端科学技術大学院大学 ソフトウェア工学講座 乾 純子 氏には，物品の発注や出張，雇用に係る書類など様々な事務処理に対し多大な御助力を頂きました．心より感謝致します．

奈良工業専門高等学校 上野 秀剛 氏には，論文執筆に対するアドバイス，プレゼンテーションの作法等，数多くの有益な御意見を頂きました．氏の深い洞察や幅広い見識は私にとって新たな発見に満ちたものでした．心より厚く御礼申し上げます．

奈良先端科学技術大学院大学 ソフトウェア工学講座 山内 寛己 氏，戸田 航史 氏，並びにソフトウェア設計学講座 大蔵 君治 氏，伏田 享平 氏には研究に関する議論のみならず，私生活の場においても楽しい時間を共有させて頂きました．心より感謝致します．

奈良先端科学技術大学院大学 ソフトウェア工学講座 亀井 靖高 氏には，本研究を進めるにあたり，熱心な御助力，御指導を頂きました．氏との忌憚のない議論は研究の進展に欠かせないものばかりであり，本研究は氏の存在無くしては完遂することはできませんでした．また研究に対する真摯な姿勢や，絶えず努力し続ける姿勢など見習うべきことばかりです．心より厚く御礼申し上げます．

奈良先端科学技術大学院大学 ソフトウェア工学講座の皆様には，日頃より多大な御協力と御助言を賜りました．研究発表の準備や研究に対する御助力，励ましの言葉により，私は研究を完遂することができました．皆様の明るさ，おおらかさにより楽しく実り多い学生生活を過ごすことができました．本当にありがとう．心より感謝致します．

最後に，大学において研究するにあたり並々ならぬ支援を頂き，また温かく見守って下さった私の家族に深く感謝致します．

参考文献

[1] Robert D. Austin. *Measuring and Managing Performance in Organizations*. Dorset House Publishing Company, New York, 1996.

[2] Stephen D. Bay and Mark Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. *Proc. ACM International Conference on Knowledge Discovery and Data Mining*, pp. 29–38, 2003.

[3] Leo Breiman, Jerome H. Friedman, Richard A. Olshen and Charles J. Stone. *Classification and Regression Trees*. Wadsworth, California, 1984.

[4] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng and Jörg Sander. LOF: Identifying density-based local outliers. *Proc. ACM International Conference on Management of Data*, pp. 93–104, 2000.

[5] Lionel C. Briand, Jürgen Wüst, John W. Daly and D. Victor Porter. Exploring the relationships between design measures and software quality in object-oriented systems. *Journal of Systems and Software*, Vol. 51, No. 2, pp. 245–273, 2000.

[6] William J. Conover. *Practical Nonparametric Statistics*. Wiley, New York, 1971.

[7] Stewart G. Crawford, Allen A. McIntosh and Daryl Pregibon. An analysis of static metrics and faults in C software. *Journal of Systems and Software*, Vol. 5, No. 1, pp. 37–48, 1985.

[8] Tom Demarco and Timothy Lister. *Peopleware: Productive Projects and Teams, 2nd Ed.* Dorset House Publishing Company, New York, 1999.

[9] Dennis E. Egan. Individual differences in human-computer interaction. *Handbook of Human-Computer Interaction*, pp. 543–568, Amsterdam, 1988. Elsevier Science Publishing Company.

[10] Norman E. Fenton and Niclas Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Transactions on Software Engineering*, Vol. 26, No. 8, pp. 797–814, 2000.

[11] Ronald A. Fisher. The distribution of the partial correlation coefficient. *Metron*, Vol. 3, pp. 329–332, 1924.

[12] Ronald A. Fisher. The use of multiple measurements in taxonomic problems. *Annals Eugenics*, Vol. 7, Part II, pp. 179–188, 1936.

[13] John E. Gaffney. Estimating the number of faults in code. *IEEE Transactions on Software Engineering*, Vol. 10, No. 3, pp. 584–585, 1984.

[14] Daniel M. German. Using software trails to reconstruct the evolution of software: Research articles. *Journal of Software Maintenance and Evolution*, Vol. 16, No. 6, pp. 367–384, 2004.

[15] Daniel M. German and Abram Hindle. Visualizing the evolution of software using softchange. *International Journal of Software Engineering and Knowledge Engineering*, Vol. 16, No. 1, pp. 5–22, 2006.

[16] Amol Ghoting, Matthew E. Otey and Srinivasan Parthasarathy. LOADED: Link-based outlier and anomaly detection in evolving data sets. *Proc. IEEE International Conference on Data Mining*, pp. 387–390, 2004.

[17] Robert B. Grady. Successfully applying software metrics. *IEEE Computer*, Vol. 27, No. 9, pp. 18–25, 1994.

[18] Andrew R. Gray and Stephen G. MacDonell. Software metrics data analysis—exploring the relative performance of some commonly used modeling techniques. *Empirical Software Engineering*, Vol. 4, No. 4, pp. 297–316, 1999.

[19] Ahmed E. Hassan. Predicting faults using the complexity of code changes. *Proc. International Conference on Software Engineering*, pp. 16–24, 2009.

[20] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems*, Vol. 22, No. 1, pp. 5–53, 2004.

[21] Victoria J. Hodge and Jim Austin. A survey of outlier detection methodologies. *Artifical Intelligence Review*, Vol. 22, No. 2, pp. 85–126, 2004.

[22] David W. Hosmer and Stanley Lemeshow. *Applied Logistic Regression*. Wiley, New York, 1989.

[23] Sergio A. Jimenez-Marquez, Christophe Lacroix and Jules Thibault. Statistical data validation methods for large cheese plant database. *Journal of Dairy Science*, Vol. 85, No. 9, pp. 2081–2097, 2002.

[24] Taghi M. Khoshgoftaar, Lofton A. Bullard and Kehan Gao. Detecting outliers using rule-based modeling for improving cbr-based software quality classification models. *Proc. International Conference on Case-Based Reasoning*, pp. 216–230, 2003.

[25] Sunghun Kim, E. James Whitehead Jr. and Yi Zhang. Classifying software changes: Clean or buggy? *IEEE Transactions on Software Engineering*, Vol. 34, No. 2, pp. 181–196, 2008.

[26] Edwin M. Knorr and Raymond T. Ng. Algorithms for mining distance-based outliers in large datasets. *Proc. International Conference on Very Large Data Bases*, pp. 392–403, 1998.

[27] Akif G. Koru, Dongsong Zhang, Khaled E. Emam and Hongfang Liu. An investigation into the functional form of the size-defect relationship for software modules. *IEEE Transactions on Software Engineering*, Vol. 35, No. 2, pp. 293–304, 2009.

[28] Ken S. Lew, Tharam S. Dillon and Kevin E. Forward. Software complexity and its impact on software reliability. *IEEE Transactions on Software*

Engineering, Vol. 14, No. 11, pp. 1645–1655, 1988.

[29] Hubert Mia and van Driessen Katrien. Fast and robust discriminant analysis. *Computational Statistics & Data Analysis*, Vol. 45, No. 2, pp. 301–320, 2004.

[30] Osamu Mizuno, Shiro Ikami, Shuya Nakaichi and Tohru Kikuno. Fault-prone filtering: Detection of fault-prone modules using spam filtering technique. *Proc. International Symposium on Empirical Software Engineering and Measurement*, pp. 374–383, 2007.

[31] Raimund Moser, Witold Pedrycz and Giancarlo Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. *Proc. International Conference on Software Engineering*, pp. 181–190, 2008.

[32] John C. Munson and Taghi M. Khoshgoftaar. The detection of fault-prone programs. *IEEE Transactions on Software Engineering*, Vol. 18, No. 5, pp. 423–433, 1992.

[33] John C. Munson and Taghi M. Khoshgoftaar. *Software Metrics for Reliability Assessment*. McGraw-Hill, 1996.

[34] Glenford J. Myers. *The Art of Software Testing*. Wiley, New York, 1979.

[35] Nachiappan Nagappan, Laurie Williams, John Hudepohl, Will Snipes and Mladen Vouk. Preliminary results on using static analysis tools for software inspection. *Proc. International Symposium on Software Reliability Engineering*, pp. 429–439. IEEE Computer Society, 2004.

[36] NASA/WVU. IV&V facility, metrics data program. `http://mdp.ivv.nasa.gov`, (available online. last accessed 2009-12-01).

[37] Martin E. Nordberg III. Managing code ownership. *IEEE Software*, Vol. 20, No. 2, pp. 26–33, 2003.

[38] Thomas J. Ostrand, Elaine J. Weyuker and Robert M. Bell. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, Vol. 31, No. 4, pp. 340–355, 2005.

[39] Matthew E. Otey, Srinivasan Parthasarathy and Amol Ghoting. Fast lightweight outlier detection in mixed-attribute data. Technical report, Department of Computer Science and Engineering, The Ohio State University, 2005.

[40] Maurizio Pighin and Roberto Zamolo. A predictive metric based on discriminant statistical analysis. *Proc. International Conference on Software Engineering*, pp. 262–270, 1997.

[41] Eric S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* O'Reilly and Associates, California, 1999.

[42] Harold Sackman, Warren J. Erikson and Eugene E. Grant. Exploratory experimental studies comparing online and offline programming performance. *Communications of the ACM*, Vol. 11, No. 1, pp. 3–11, 1968.

[43] Norman F. Schneidewind and Heinz-Michael Hoffmann. An experiment in software error data collection and analysis. *IEEE Transactions on Software Engineering*, Vol. 5, No. 3, pp. 276–286, 1979.

[44] Adrian Schröter, Thomas Zimmermann, Rahul Premraj and Andreas Zeller. If your bug database could talk... *Proc. International Symposium on Empirical Software Engineering*, pp. 18–20, 2006.

[45] Jacek Śliwerski, Thomas Zimmermann and Andreas Zeller. When do changes induce fixes? *Proc. International Conference on Mining Software Repositories*, pp. 1–5, 2005.

[46] Ramanath Subramanyam and M. S. Krishnan. Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. *IEEE Transactions on Software Engineering*, Vol. 29, No. 4, pp. 297–310, 2003.

[47] Yoshihiro Takada, Ken-ichi Matsumoto and Koji Torii. A programmer performance measure based on programmer state transitions in testing and debugging process. *Proc. International Conference on Software Engineering*, pp. 123–132, 1994.

[48] Arthur H. Watson and Thomas J. McCabe. Structured testing: A testing methodology using the cyclomatic complexity metric. Technical report, National Institute of Standards and Technology, 1996.

[49] Elaine J. Weyuker, Thomas J. Ostrand and Robert M. Bell. Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models. *Empirical Software Engineering*, Vol. 13, No. 5, pp. 539–559, 2008.

[50] Ping Yu, Tarja Systa and Hausi Muller. Predicting fault-proneness using OO metrics: An industrial case study. *Proc. European Conference on Software Maintenance and Reengineering*, pp. 99–107, 2002.

[51] Thomas Zimmermann and Nachiappan Nagappan. Predicting defects using network analysis on dependency graphs. *Proc. International Conference on Software Engineering*, pp. 531–540, 2008.

[52] 岩間太, 中村大賀. コーディングスタイルに基づくメトリクスを用いたソースコードからの属人性検出. ソフトウェア工学の基礎XV, 日本ソフトウェア科学会, pp. 129–134, 2008.

付録

A. 単位行数あたりの fault 混入率の分析結果

A.1 仮説 1a：fault の混入のさせやすさには個人差がある．

単位コミットではなく，単位行数あたりの埋め込み数を fault 混入率とした際の分析結果を示す．全バージョン・全開発者（108 人）の平均 fault 混入率のヒストグラムを図 A.1 に，全バージョンで出現していた開発者 39 人のバージョンごとの fault 混入率を図 A.2 に示す．

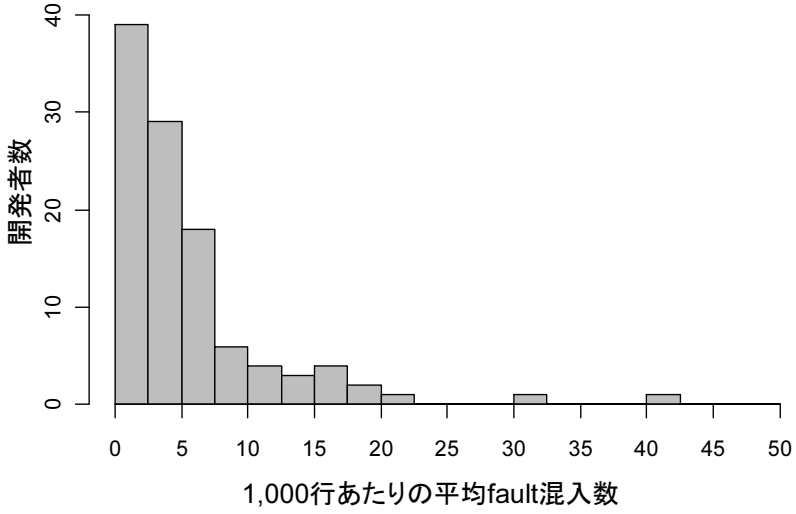


図 A.1 平均 fault 混入率のヒストグラム（単位行数）

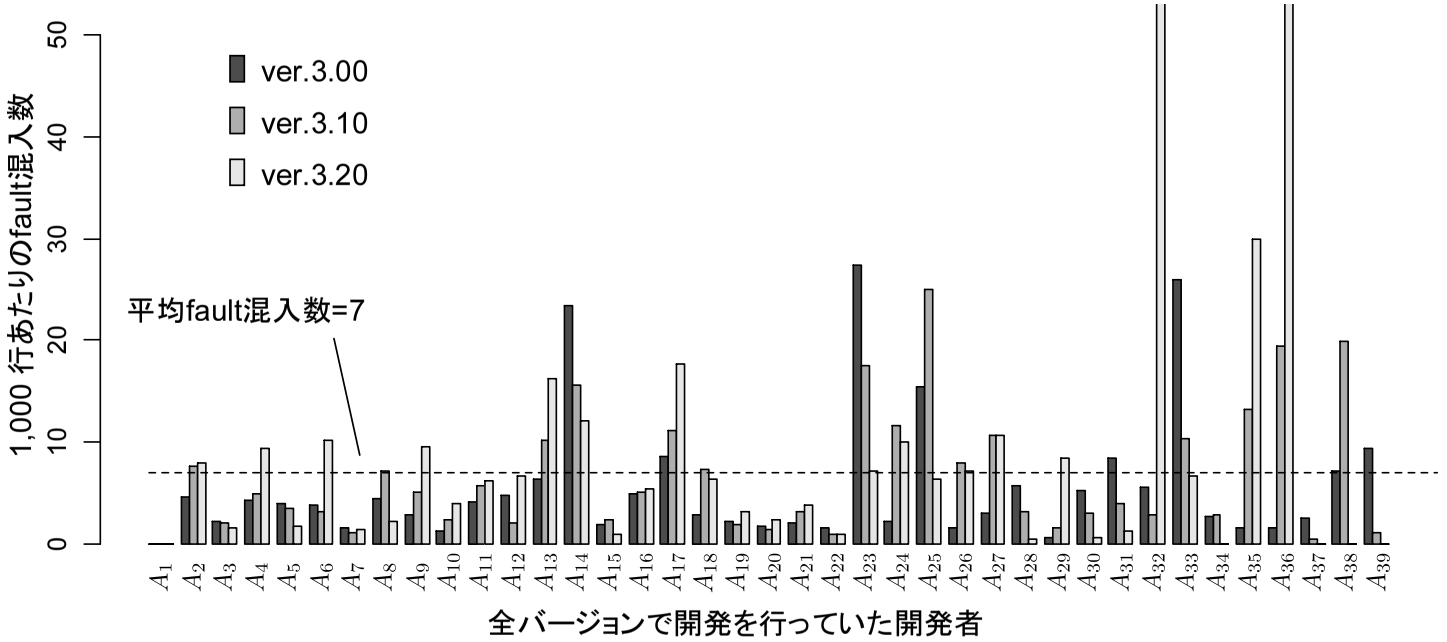


図 A.2 各開発者のバージョンごとの fault 混入率（単位行数）

A.2 仮説1b：faultの混入のさせやすさは開発者の特性から判断できる．

開発者個々の6つの特性それぞれとfault混入率（単位行数）との相関係数を表A.1に示す．

表 A.1 開発者ごとの特性とfault混入率（単位行数）の相関

	ver.3.00	ver.3.10	ver.3.20
コミット数	-0.17	-0.14	-0.21
変更行数	0.13	-0.09	-0.21
変更ソースコード数	0.44*	0.08	0.01
変更パッケージ数	0.05	-0.17	-0.17
前バージョンでのfault混入率	算出不可	0.53*	0.37*
新人であるかどうか	算出不可	-0.09	0.11

*：有意水準5%の無相関検定により有意に相関あり