卒 業 研 究 報 告 書

題　目　　識別子中の単語情報を用いた
Fault-proneモジュール予測手法の提案

指導教員　　水野　修　准教授

京都工芸繊維大学 工芸科学部 情報工学課程

学生番号　　10122012

氏　　名　　川島　尚己

平成26年2月14日提出

識別子中の単語情報を用いた Fault-prone モジュール予測手法の提案

平成 26 年 2 月 14 日　　10122012　川島　尚己

概　要

不具合を含んだモジュールを予測するという研究は，Fault-prone モジュール予測と呼ばれ，ソフトウェア工学上の研究テーマの一つとして，これまでにも数々の研究が行われてきた．本稿で取り扱うテーマは『ソースコード上の識別子に含まれる単語情報を用いた Fault-prone モジュール予測』である．ソースコード上の識別子とは変数名，クラス名，関数名などのことを指し，ソースコードを読み解く上での重要な情報源となっている．そして，識別子の多くは複数の単語を組み合わせて命名をされることが多い．そこで本稿では，識別子を単語単位に分割したのち，各単語のモジュールごとの出現回数を説明変数とした機械学習を行い Fault-prone モジュール予測モデルを構築した．そして，提案する Fault-prone モジュール予測モデルの性能の比較のため，既存の手法である CK メトリクスを説明変数として Fault-prone モジュール予測モデルとコード行数を説明変数とした Fault-prone モジュール予測モデルとの比較実験を行った．その結果，提案手法は，既存の手法に劣らない評価値を得ることが出来た．

目　次

1. 緒言

ソフトウェア開発の現場において，ソフトウェアのバグの約 60〜80 %は約 20 %のモジュールに集中して存在するといわれている (バグの偏在性)．つまり，ソフトウェア開発の早い段階で，不具合を含むモジュールを予測することができれば，不具合を含む可能性の高いと予測したモジュールに対するテストに工数をかけ，不具合を含む可能性の低いと予測したモジュールに対するテストの工数を減らすことで，コストの削減とソフトウェアの品質の向上が可能となると考えられる．参考として，Khoshgoftaar らは Fault-prone モジュールを予測することにより，テストにかかるコストが半減すると主張している．

これまでにも，欠陥を含む可能性の高いモジュール（Fault-prone モジュール）を推定するためのモデル（Fault-prone モジュール予測モデル）の研究が行われてきた [1]．Fault-prone モジュールを判別する尺度 (メトリクス) としては，ソースコード [2] や開発プロセス [3]，開発組織 [4]，地理的な位置関係 [5] を対象としたものなどを用いた様々な手法が提案されてきたが，現時点においては，デファクトスタンダード (事実上の標準) となっている手法は存在していない．

そこで，本研究では『ソースコード中の識別子名に含まれる単語情報に着目した Fault-prone モジュール予測』を新たに提案する．ソースコードにおいて，識別子名は，ソースコードのコメント文と同様に，ソースコードを理解するうえで重要な情報源としての役割を担っている．そして．識別子名は複数の単語の組み合わせで命名されていることが多いということが明らかにされているため [6],『ソースコード中の識別子名に含まれる単語情報に着目した Fault-prone モジュール予測』には妥当性があると考える．

そこで，本研究では，研究設問を設定し，ソースコード中の識別子名に含まれる単語情報についての分析を行った．研究設問については第 2.3 節に記述する．

提案手法の具体的な方法としてはまずソースコードに含まれる識別子名を単語単位に分割し，ファイル毎に単語の出現回数をカウントする．そして，各単語の出現回数を説明変数として，目的変数で不具合の有無を機械学習により予測する．なお，Fault-prone モジュール予測の性能を評価するための指標としては，正解率 (Accuracy)，再現率 (Recall)，適合率 (Precision)，F_1値 を用いた．

本論文における以降の構成を述べる．まず，第 2 章において，この研究の背景および意義について述べ，第 3 章で，私が提案する手法とその手法の実現方法を説明する．第 4 章では，実験上で利用する機械学習の理論について述べる．そして，第 5 章において，提案手法と既存の手法との比較実験の結果を記し，第 6 章で実験結果に対する考察を述べ，第 7 章に本研究の結言を記す.

2. 研究背景・意義

2.1 Fault-prone モジュール予測の概要

Fault-prone モジュール予測についての概要を述べる．Fault-prone モジュール予測を主題とした研究としても，不具合の有無を議論するもの以外に，不具合の密度を扱うものや，不具合の深刻度を扱うものがある．本稿の実験では，モジュールの不具合の有無を予測の対象とする．

Fault-prone モジュール予測を行う粒度としては，メソッドレベルや，クラスレベル，ファイルレベルなどがあり，本稿の実験では，ファイルレベルでの Fault-prone モジュール予測を行う．

Fault-prone モジュール予測にのために測定する対象としてはコード，プロセス，開発組織，地理的位置関係という 4 つに分類され，測定に必要な履歴としては対象の版か，これまでの履歴の 2 種類に分類される．本稿の実験では，コードを対象とし，対象の版を用いて Fault-prone モジュール予測を行う．

2.2 識別子の重要性

プログラミング言語においては，識別子とは関数や変数などを識別するための，ソースコード中のトークンのことを指す．ソースコードを構成する要素のうち，識別子は約 70%を占めているといわれている．

ソースコード中のコメント文がソースコードの理解を助けるのと同様に，識別子名を適切に命名することができれば，ソースコードの可読性を向上させ，理解を容易にすることが可能となる．

例えば，関数名がその処理内容を表す単語の組み合わせで記述された場合には関数名だけで関数の処理内容を理解することが可能である．変数においても格納する値の内容に基づいた変数名をつけることでその変数がどのように使われているのかを理解しやすくなる．

ソースコードの可読性はソフトウェアの品質に影響を与えるので，識別子を用いた Fault-prone モジュール予測モデルの構築には妥当性があると考えられる．

本稿の研究の先行研究について言及すると，川本 [6] は識別子の語長および識別子

を構成する単語数を用いたFault-proneモジュール予測モデルの研究を行った.

山本[7]は識別子のうち変数名に着目し，モジュールに含まれる特定の変数名の個数を用いてバグ密度予測の研究を行った.

山本の研究では変数名の個数を用いてバグ密度予測を行ったが，識別子を単語単位への分割は行っていない．川本の研究により，識別子には複数の単語が含まれていることが多いということが判明しているので，識別子を単語単位に分割し，各単語の出現回数を用いて，Fault-proneモジュール予測モデルを構築することで，新たな知見を得られるのではないかと考える.

2.3 研究設問

本研究では次に示す研究設問(Reserch Question:RQ)について，検証を行う.

RQ1:ソースコードの識別子に含まれる各単語のモジュールごとの出現回数を用いたFault-proneモジュール予測は可能か？

RQ2:ソースコードの識別子に含まれる各単語のモジュールごとの出現回数を用いたFault-proneモジュール予測とその他の手法を用いたFault-proneモジュールではどちらが良いか？

3. 提案手法

提案する手法は，識別子に含まれる単語のモジュール内での出現回数を尺度とした Fault-prone モジュール予測モデルである．この手法により，識別子に含まれる単語の出現回数とモジュールの不具合との関連性を明らかにすると同時に，モジュールのバグの原因となりやすい識別子の命名の傾向や使用の傾向を明らかにすることが期待できる．

3.1 手順

提案する Fault-prone モジュール予測モデルを構築する手順を以下に示す．

1. LSCP を用いて，ソースコードから，識別子を抽出し単語単位に分割する．
2. モジュール毎に各単語の出現回数を計算しデータベースへまとめる．
3. 不具合情報と結び合わせた上で機械学習を行い Fault-prone モジュール予測モデルを構築する．

図 3.1 に Fault-prone モジュール予測モデルを構築する流れを図で示す．

3.2 実験データの収集

実験に用いるデータを収集するにはソフトウェア開発プロジェクトの共有サービスである GitHub (注 1)とソフトウェア工学の実験のためのデータセットを公開している Web サイト PROMISE (注 2)を利用する．

PROMISE で公開されているデータセットには不具合情報が含まれているものもあるため，Fault-prone モジュール予測モデルの実験に適している．PROMISE で不具合情報が公開されているオープンソースプロジェクトを GitHub より取得することで実験用のデータを収集することができる．

(注 1)：https://github.com/
(注 2)：http://promisedata.googlecode.com/

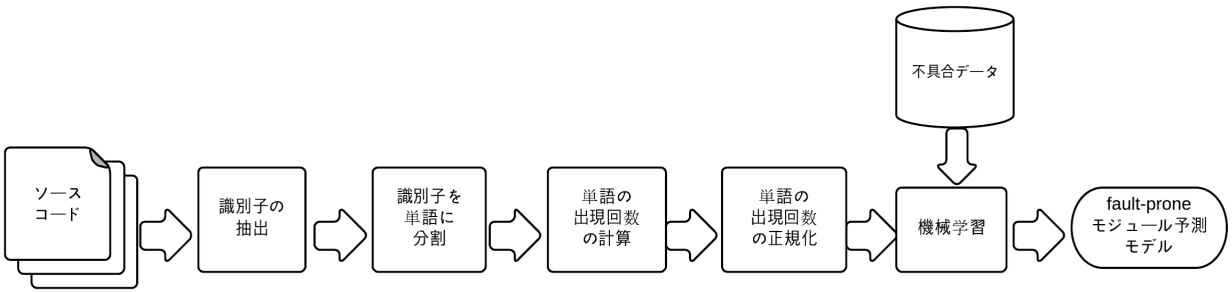

図 3.1　Fault-prone モジュール予測モデルを構築する流れ

3.2.1 PROMISE

ソフトウェア工学のためのデータセットを公開しているサイトである．PROMISE内で公開されているデータセットの中には，不具合情報や，提案手法との比較に利用することができる，CK メトリクスや LOC(ソースコードの行数) を含んだものもあるため，Fault-prone モジュール予測の実験を行うのに大変有用である．

3.2.2 GitHub

GitHub は，Git と呼ばれるバージョン管理システムのリポジトリを中心にソフトウェア開発プロジェクトの共有サービスを展開している．個人的なソフトウェア開発から大規模なオープンソースソフトウェアの開発まで，様々なソフトウェア開発がGitHub を通して世界中で公開されており，利用者の多い開発コミュニティとなっている．本実験では．GitHub から実験対象となるプロジェクトのデータを収集する．

3.2.3 ファイル情報の結びつけ

PROMISE より取得したデータセットと GitHUb より取得したプロジェクトデータのファイル情報の結びつけの方法について記す．PROMISE のデータセットには，プロジェクト内のファイルに含まれる不具合の有無とともに，そのファイルへのパスが記されているため，それをもとにして，GitHub より取得したプロジェクトに含まれるファイルとの結びつけを行う．

なお，ファイル情報の結びつけを行えなかったファイルおよびデータは機械学習の訓練データおよびテストデータから除外する．

3.3 識別子の単語への分割

複数の単語からなる識別子を分割する規則について説明する．複数の単語からなる識別子の単語の連結方法は連結部分に数字や記号を用いている場合と，単語をそのまま連結している場合がある．以下に例を示しつつ，概要を示す.

1. 連結部分に記号を用いている場合:連結部分に記号を用いている場合は，連結部分の記号で識別子名を単語に分割することができる．

例. ”max_size”という識別子名は”max”と”size”という２つの単語に分割することができる.

2. キャメルケース:単語をそのまま連結している場合において単語の最初を大文字で記している場合をキャメルケースといい，キャメルケースでは大文字をもとにして識別子名を単語に分割を行うことができる.
   例. ”maxSize”という識別子名は”max”と”Size”という２つの単語に分けることができる.
3. 連結部分に数字を用いている場合:連結部分に数字を用いている場合，数字の前後で分割することができ，数字もひとつの単語とする.
   例. ”int2char”という識別子の場合は”int”, ”2”, ”char”という３つの単語に分けることができる.
4. 全てが小文字もしくは大文字の場合:識別子名に複数の単語が含まれている場合でも，識別子名が全て小文字もしくは大文字で記されている場合がある．この場合には辞書を用いた分割手法が提案されている.

本稿においては 1 と 2 の分割方法においては，識別子名を単語に分割することができるが，3 と 4 の場合については識別子名の単語への分割を行っていない.

本稿の実験では，ソースコードから識別子を抽出するには，ソフトウェア LSCP を使用する.

3.3.1 LSCP

LSCP はソースコードに対して，ヒューリスティック的手法により，ソースコードから特定の要素を抽出することができるソフトウェアである．更に LSCP は，オプションを指定することにより，抽出した要素を単語単位に分割することができる.

4. 機械学習

機械学習アルゴリズムにはRandom Forestを用いる．Random Forestは説明変数が多くても動作が上手くいくという特徴があり，ソースコード上の識別子に含まれる各単語の出現数を説明変数とした場合，説明変数は多くなるので，Random Forestはこの実験に適している．

4.1 Random Forest法

(1) 概要

Random Forest法とは，Leo Breimanが提案した機械学習アルゴリズムの一つである．Random Forest法は集団学習により高精度の識別，回帰，クラスタリングを行う．現在，Random Forest法がFault-proneモジュール予測において有効な手法であるということが実証されている．

(2) 決定木

決定木とは木構造の条件分岐を用いて入力パターンをクラス分類する分類器の一種である．データから決定木を作成する機械学習を決定木学習というが，これはデータマイニングの分野でも良く用いられる．その場合，決定木の葉が分類を表し，葉に至るまでの通り道となる枝がその分類の特徴を表す．決定木は目的変数に影響が大きい変数や境界値，順序を算出する．

決定木の分岐基準としてはCART(Classification and Regression Trees)を用いる．CARTは分割統治アルゴリズムの一種である．CARTでは各ノードでの分岐基準となる説明変数は，木全体のGini係数を一番減少させる説明変数という基準で選択する．Gini係数とは，本来は社会における所得分配の均衡，不均衡を表す尺度である．Gini係数は0から1の値をとり，0に近いほど平等な社会であり，反対に1に近いほど格差が大きく，不平等 (高所得者への所得集中) な社会であるとされている．このGini係数はデータマイニングにおけるデータ分類にも利用できる．Gini係数が0の状態をこれ以上分割できないを意味する．逆に分類を行う前のデータのGini係数は1に近い．そのため，CARTでは少しでも多くのGini係数を減少させて0へ近づ

けるように説明変数を選択するのである．Gini 係数は式 (4.1) で表される．ここで $p(k|A)^2$ はノード A でクラス k をとる確率を表す.

$$Gini\ 係数 = 1 - \sum_k p(k|A)^2 \tag{4.1}$$

決定木には分類木と回帰木の 2 つに分けられる．分類木は目的変数が分類変数のものである．回帰木は目的変数が連続数値変数のものである．なお，本研究では不具合があるかないかの 2 値の分類を行うため，分類木として用いる．

(3) 集団学習

集団学習とはサンプルや重みの異なる複数の学習モデルを生成して，各モデルの評価結果を統合することで精度や汎用性をを向上させる機械学習のアルゴリズムである．結果の統合方法に関しては，分類の場合は多数決を用い，回帰の場合には平均値を用いる．

4.1.1 アルゴリズム

Random Forest は決定木の集団学習によって分類や予測を行うが，大きく分けて学習と評価の 2 つのステップに分けられる．

(1) 学習ステップ

まず，ブーストラップサンプリングを用いて，学習データから重複を許してランダムに B 組のサンプル集合を抽出する．そして各サンプルにおいて M 個の説明変数の中から m 個の変数をランダムに抽出して決定木を作成する．その結果 B 本の決定木が作成される．

(2) 評価ステップ

学習ステップで作成した B 本の全ての決定木で調査したいデータを分類，回帰予測を行う．そして得られた B 個の結果を分類なら多数決で，回帰予測なら平均で統合する．

4.1.2 特徴

Random Forest の長所としては次の 4 つなどが挙げられる．

- サポートベクターマシン (SVM) などの他の分類器と比較して分類，予測が高精度である．
- 説明変数が数百，数千など膨大な数になっても効率的に作動する．
- 目的変数に対する説明変数の重要度が推定できる．
- データに欠損値が存在しても動作可能である．

逆に短所は意味のある説明変数が意味のない説明変数よりも極端に少ない場合にうまく評価が行えないことなどである．

4.2 統計解析ソフト R

機械学習には統計解析ソフト R を利用する．R ではパッケージとして，Random forest 法を利用することができるライブラリが配布されている．そのため，R を用いることにより，Random forest 法を用いた Fault-prone モジュール予測モデルの構築を行うことができる．

5. 比較実験

5.1 実験対象となるプロジェクト

実験の対象として，apache のプロジェクトを使用する．実験の対象としたプロジェクトとプロジェクトの概要は以下の通りである．

Apache Ant: Java プログラムのビルドツールソフトウェアである．

Apache Xalan: XML 文書の XSLT 変換と XPath 検索を実装しているソフトウェアである．

Apache Lucene: 全文検索を行うソフトウェアである．

Apache POI: Microsoft Office 形式のファイルの読み込み・書き込みを行うことができるライブラリである．

実験対象としたプロジェクトは全て Java で書かれたものである．実験対象となる各プロジェクトの訓練データとテストデータのバージョンとファイル数を表 5.1 に示す．各訓練データおよびテストデータの不具合を含んだファイル数と不具合を含まないファイル数のデータを表 5.2 に示す．各プロジェクトの単語の種類数をまとめたデータを表 5.3 に記す.

5.2 その他の手法との比較

PROMISE で公開されているデータセットにはバグの有無以外にも，ファイルに対する様々な尺度 (メトリクス) が記されている．これらを利用して，提案手法との比較実験を行った．本稿の実験において提案手法との比較に用いたメトリクスは，コード行数 (LOC) と CK メトリクスである．以下にコード行数 (LOC) と CK メトリクスと説明を記す.

5.2.1 コード行数 (LOC:line of code)

ファイルごとのソースコードのコード行数を指す．コード行数が多ければ多いほどプログラムの複雑性は増し．不具合を含みやすいといえる．

表 5.1　各プロジェクトの訓練データとテストデータ

	訓練データ		テストデータ	
プロジェクト	バージョン	ファイル数	バージョン	ファイル数
Ant	1.4, 1.5, 1.6	819	1.7	741
Lucene	2.0, 2.2	420	2.4	330
Xalan	2.4	682	2.5	762
POI	1.5, 2.0, 2.5	924	3.0	438

表 5.2　各プロジェクトの不具合データ

	訓練データ		テストデータ	
プロジェクト	不具合有り	不具合無し	不具合有り	不具合無し
Ant	164	655	166	575
Lucene	234	186	203	127
Xalan	110	572	387	375
POI	426	498	281	157

表 5.3　各プロジェクトの単語の種類数

プロジェクト	単語の種類数
Ant	2726
Lucene	1079
Xalan	4295
POI	1983

5.2.2 CK メトリクス

CK メトリクスとはオブジェクト指向設計に対する複雑度のメトリクス群である．CK メトリクスは提案者である Chaidamber と Kemerer の頭文字をとって命名された．CK メトリクスは 6 つのメトリクスで構成され，クラスの規模と複雑性，継承の使用，クラス間の結合度，クラスの凝集度，クラス間の協調を測定する．6 つのメトリクスの概要を以下に示す.

- WMC(Weighted Methods for Class):各クラスの全メソッド数の複雑度の合計．このメトリクスは，各クラスの実装とテストに必要な工数の量を示すものとして使われてきた．このメトリクスの値が高ければ，クラスが大きすぎるとされる．
- DIT(Depth of Inheritance Tree):継承ツリーの深さを指す．このメトリクスは，どれだけ多くの継承の層が所定のクラス階層を構成しているかを測定する．このメトリクスの値が高ければ，設計の複雑性が高いが，再利用性も高いことを示す.
- NOC(Number of Children): 継承上の子の数．このメトリクスは，クラスの直接の直接のサブクラス数を測定する．ここでも，このメトリクスの値が高ければ，親クラスの抽象化が弱まっているとされ，テストの必要性が高まるが，再利用性も高いことを示す.
- CBO(Coupling Between Objects):オブジェクトクラス間の結合度．このメトリクスは，他の多くのクラスが当該クラスに依存する (およびその逆の) 度合いを測定する．このメトリクスの値は，そのクラスが結合しているクラスの数である．1 つのクラスで宣言されているメソッドが他のクラスのメソッドが他のメソッドやインスタンス変数を使っているとき，2 つのクラスは結合しているとみなされる．このメトリクスが高ければ，複雑性が高く，保守性と再利用性が低いことを示す.
- RFC(Response for Class): クラスの応答．このメトリクスは，オブジェクトが受け取ったメッセージへの応答として実行される可能性のあるメソッド数を測定する．このメトリクスの値は，メソッド数の総計に，当該クラスのメソッドが直接に呼び出す他クラスのメソッドの数を加えたものである．RFC が増加す

ると複雑性も増加し，テストの必要性も高まるとされる．

- LCOM(Lack of Cohesion of Methods):メソッドの凝集性欠如．このメトリクスは，所定のインスタンス変数を参照するクラス内のさまざまなメソッドの数を数える．LCOM が増加すると，複雑度と設計の困難さが増加する．

また，CK メトリクスの高い値は，以下のことと相関することも示されている．

- 生産性の低さ
- クラスを再利用するのにかかる工数
- クラスを設計するのにかかる工数
- クラスの実装の難しさ
- 保守における変更回数
- 欠陥のあるクラスの数
- 欠陥数
- ユーザが報告してきた問題数

5.3 結果の凡例

以下に Fault-prone モジュール予測を行う際の，結果の凡例を示す.

- True Positive(TP):True Positive とは正と予測したもので，結果が正であったものである．本稿の実験では不具合有りと予測したもので，実際に不具合が有ったものを指す.
- False Positive(FP):False Positive とは正と予測したもので，結果が負であったものである．本稿の実験では不具合有りと予測したもので，実際は不具合が無かったものを指す.
- True Negative(TN):True Negative と負と予測したもので，結果が負であったものである．本稿の実験では不具合無しと予測したもので，実際に不具合が無かったものを指す.
- False Negative(FN):False Positive とは負と予測したもので，結果が正であったものである．本稿の実験では不具合なしと予測したもので，実際は不具合が有ったものを指す.

これらの凡例を表5.4にまとめる．

5.4 評価値

本稿の実験結果の評価値としては以下に示す正解率 (Accuracy)，再現率 (Recall)，適合率 (Precision)，F_1値 の 4 つを用いる．各評価値の概要及び計算式を以下に記す．

(1) 正解率 (Accuracy)

正解率 (Accuracy) とは，予測結果全体に対して正解している割合のことである．全モジュールに対して，実際に不具合を含むモジュールを不具合有り，不具合を含まないモジュールを不具合を含まないと予測できた割合を指す．これを表5.4の凡例に則ると，式5.1のように定義される．

$$Accuracy = \frac{TN + TP}{TN + TP + FN + FP} \tag{5.1}$$

正解率は予測全体の傾向をつかむ上で便利な指標であるが，対象データが，正のデータに対し，負のデータが極端に多い場合など，データの偏りの影響を受けやすいため，正解率以外にも，以下の再現率，適合率，F_1値 という指標を用いる．

(2) 再現率 (Recall)

再現率 (Recall) とは，結果が正である中で，予測できた割合のことである．本稿の実験では，実際に不具合を含むモジュールを，不具合有りと予測できた割合を指す．これを表5.4の凡例に則ると，式5.2のように定義される．

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

Fault-prone モジュール予測において，再現率は不具合を未然に検出するという視点から，重要な指標といえる．

表 5.4　実験結果の凡例

		実測	
		Fault-prone	Not Fault-prone
予測	不具合を含む	True positive(TP)	False positive(FP)
	不具合を含まない	False negative(FN)	True negative(TN)

(3) 適合率 (Precision)

適合率 (Precision) とは，正と予測した中で，実際に結果が正であった割合を示す．本稿の実験では，不具合を含むと予測したモジュールが，実際に不具合有りであった割合を指す．これを表 5.4 の凡例に則ると，式 5.3 のように定義される．

$$Precision = \frac{TP}{TP + FP} \tag{5.3}$$

適合率が低いということは，不具合を含まないモジュールを不具合有りと予測する割合が高いということを示し，バグを含まないモジュールに対してのテスト工数を増やすことになりうる．適合率とは，バグを発見するためのコストを示すための指標といえる．

(4) F_1値

F_1値 とは，適合率と再現率の調和平均である．再現率 (Recall) と適合率 (Precision) はトレードオフの関係のため，両指標の総合的な評価を行うために F_1値 を用いる．この値が高ければ予測精度が高いと評価できる．

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{5.4}$$

ただし，F_1値 においても実験対象となるデータの偏りの影響を受けることがある．

5.5 実験結果

プロジェクト Ant を用いて，実験を行った結果を表 5.5〜表 5.7 に，プロジェクト Lucene を用いて，実験を行った結果を表 5.8〜表 5.10 に，プロジェクト Xalan を用いて，実験を行った結果を表 5.11〜表 5.13 にそれぞれ記載する．プロジェクト POI を用いて，実験を行った結果を表 5.14〜表 5.16 にそれぞれ記載する．それぞれの実験結果は表 5.4 の凡例に則って記述している．

表 5.5　実験結果 Ant(提案手法)

		実測	
		不具合を含む	不具合を含まない
予測	不具合を含む	57	28
	不具合を含まない	109	547

表 5.6　実験結果 Ant(CK メトリクス)

		実測	
		不具合を含む	不具合を含まない
予測	不具合を含む	61	29
	不具合を含まない	105	546

表 5.7　実験結果 Ant(LOC)

		実測	
		不具合を含む	不具合を含まない
予測	不具合を含む	49	58
	不具合を含まない	117	517

表 5.8　実験結果 Lucene(提案手法)

		実測	
		不具合を含む	不具合を含まない
予測	不具合を含む	123	61
	不具合を含まない	80	66

表 5.9　実験結果 Lucene(CK メトリクス)

		実測	
		不具合を含む	不具合を含まない
予測	不具合を含む	128	67
	不具合を含まない	75	60

表 5.10　実験結果 Lucene(LOC)

		実測	
		不具合を含む	不具合を含まない
予測	不具合を含む	106	57
	不具合を含まない	97	70

表 5.11　実験結果 Xalan(提案手法)

		実測	
		不具合を含む	不具合を含まない
予測	不具合を含む	55	33
	不具合を含まない	332	342

表 5.12　実験結果 Xalan(CK メトリクス)

		実測	
		不具合を含む	不具合を含まない
予測	不具合を含む	49	21
	不具合を含まない	338	354

表 5.13　実験結果 Xalan(コード行数)

		実測	
		不具合を含む	不具合を含まない
予測	不具合を含む	49	40
	不具合を含まない	338	335

表 5.14　実験結果 POI(提案手法)

		実測	
		不具合を含む	不具合を含まない
予測	不具合を含む	157	54
	不具合を含まない	124	103

表 5.15　実験結果 POI(CK メトリクス)

		実測	
		不具合を含む	不具合を含まない
予測	不具合を含む	166	51
	不具合を含まない	115	106

表 5.16　実験結果 POI(コード行数)

		実測	
		不具合を含む	不具合を含まない
予測	不具合を含む	136	46
	不具合を含まない	145	111

5.6 評価値

5.4 節で説明した評価値を実験結果に適用した結果を記載する．プロジェクト Ant の評価値は表 5.17，プロジェクト Lucene の評価値は表 5.18，プロジェクト Xalan の評価値は表 5.19 にそれぞれ記載する．プロジェクト POI の評価値は表 5.20 にそれぞれ記載する．

表 5.17　評価値 Ant

	評価値			
メトリクス	Accuracy	Recall	Precision	F_1
提案手法	0.815	0.343	0.671	0.454
CK メトリクス	0.819	0.367	0.678	0.477
コード行数	0.764	0.295	0.458	0.359

表 5.18　評価値 Lucene

	評価値			
メトリクス	Accuracy	Recall	Precision	F_1
提案手法	0.573	0.606	0.668	0.636
CK メトリクス	0.570	0.631	0.656	0.643
コード行数	0.533	0.522	0.650	0.579

表 5.19　評価値 Xalan

	評価値			
メトリクス	Accuracy	Recall	Precision	F_1
提案手法	0.521	0.142	0.625	0.232
CK メトリクス	0.529	0.127	0.7	0.214
コード行数	0.504	0.127	0.551	0.206

表 5.20　評価値 POI

	評価値			
メトリクス	Accuracy	Recall	Precision	F_1
提案手法	0.594	0.559	0.744	0.638
CK メトリクス	0.621	0.591	0.765	0.667
コード行数	0.564	0.484	0.747	0.587

6. 考察

6.1 妥当性の検証

プログラムの不備 実験の過程で，識別子の抽出および単語への分割には，フリーソフトである LSCP を用いて行い，単語の出現回数のカウントや不具合情報の結びつけは著者が作成したプログラムを用いて行ったが，プログラムの不備により，正しくない結果を得た可能性が存在する．

データセットの不備 本研究の実験対象のプロジェクトには，PROMISE において公開されているデータセットを基にして不具合情報を結びつけ実験を行ったが，PROMISE で公開されている不具合情報に誤りがある可能性があるため，それが原因で実験結果の妥当性が損なわれてしまった可能性が存在する．

ソースコードの自動生成 ソースコードの中にはツールを用いることにより，自動生成されたソースコードも存在する．そのような場合の識別子の命名には可読性などが考慮されていないので，実験結果の妥当性に影響を与えてしまっている可能性が存在する．

6.2 説明変数の個数の比較

CK メトリクスを用いた Fault-prone モジュール予測では説明変数の個数は 6 個である．コード行数を用いる場合の説明変数の個数は 1 個である．それに対して，提案手法では各プロジェクトに含まれる単語の種類の数が説明変数の数となる．表 5.3 に示すようにプロジェクトのソースコードに含まれる単語の種類は不定であり，今回の実験対象のプロジェクトのように数千種類以上の単語を含む場合もある．つまり，提案手法ではその他の手法に比べると説明変数の個数が多くなる．

6.3 評価値

各評価値に対する考察を述べる．

6.3.1 正解率

表 5.17〜表 5.20 のデータを見ると，コード行数を指標とした Fault-prone モジュール予測と提案手法を比較すると，プロジェクト Ant, Xalan, Lucene, POI において提案手法の正解率はコード行数を指標とした場合よりも正解率が高いという結果が出た．

CK メトリクスを指標とした Fault-prone モジュール予測と提案手法を比較すると，プロジェクト Lucene においては提案手法の方がわずかに正解率が高いという結果がでたが，プロジェクト Ant, Xalan, POI においては提案手法は CK メトリクスを用いた場合に比べて，正解率が劣るという結果になった．

6.3.2 再現率

表 5.17〜表 5.20 のデータを見ると，コード行数を指標とした Fault-prone モジュール予測と提案手法を比較すると，プロジェクト Ant, Xalan, Lucene, POI において提案手法の再現率はコード行数を指標とした場合よりも精度が高いという結果が出た．

CK メトリクスを指標とした Fault-prone モジュール予測と提案手法を比較すると，プロジェクト Xalan においては提案手法の方が再現率が高いという結果がでたが，プロジェクト Ant, Lucene, POI においては提案手法は CK メトリクスを用いた場合に比べて，再現率が劣るという結果になった．

6.3.3 適合率

表 5.17〜表 5.20 のデータを見ると，コード行数を指標とした Fault-prone モジュール予測と提案手法を比較すると，プロジェクト Ant, Xalan, Lucene において提案手法はコード行数を指標とした場合よりも適合率が高いという結果が出た．プロジェクト POI では提案手法はコード行数を指標とした場合より適合率が低いという結果が出た．

CK メトリクスを指標とした Fault-prone モジュール予測と提案手法を比較すると，プロジェクト Lucene においては提案手法の方がわずかに適合率が高いという結果がでたが，プロジェクト Ant, Xalan, POI においては提案手法は CK メトリクスを用いた場合に比べて，適合率が劣るという結果になった．

6.3.4 F_1値

表5.17〜表5.20のデータを見ると，コード行数を指標としたFault-proneモジュール予測と提案手法を比較すると，プロジェクトAnt, Xalan, Lucene, POIにおいて提案手法はコード行数を指標とした場合よりもF_1値が高いという結果が出た．

CKメトリクスを指標としたFault-proneモジュール予測と提案手法を比較すると，プロジェクトXalanにおいては提案手法の方がわずかにF_1値が高いという結果がでたが，プロジェクトAnt, Lucene, POIにおいては提案手法はCKメトリクスを用いた場合に比べて，F_1値が劣るという結果になった．

6.4 研究設問への解答

RQ1:ソースコードの識別子に含まれる各単語のモジュールごとの出現回数を用いたFault-proneモジュール予測は可能か？

実験の結果である表5.5，表5.8，表5.11，表5.14および，評価値をまとめた表5.17〜表5.20の提案手法の項目を見ると，プロジェクトにより振れ幅があるものの，ある程度の予測ができているということが分かる．つまり，ソースコードの識別子に含まれる各単語のモジュールごとの出現回数を用いたFault-proneモジュール予測は可能であると考えられる．

RQ2:ソースコードの識別子に含まれる各単語のモジュールごとの出現回数を用いたFault-proneモジュール予測とその他の手法を用いたFault-proneモジュールではどちらが良いか？

実験結果である表5.5〜5.16および，評価値をまとめた表5.17〜表5.20より，提案手法は，既存の手法であるコード行数および，CKメトリクスをもとにしたFault-proneモジュール予測に比べても，実験結果および評価値は大きく劣るわけではないということが分かる．

しかしながら，識別子に含まれる単語の出現回数を用いる提案手法では，プロジェクトに含まれる単語の種類数が説明変数の数となり，表5.3に記されているように，プロジェクトに含まれる単語の種類はとても多い．コード行数を用いる場合の説明変数が1個であり，CKメトリクスを用いる場合の説明変数が6個であることと比べ

ても，提案手法の説明変数の数が多いということが分かる．説明変数が多ければ，Fault-prone モジュール予測モデルの構築にかかる時間は長くなるので，提案手法はその点ではその他の手法に比べて劣っているといえる．

7. 結言

本稿では,『識別子中の単語情報を用いた Fault-prone モジュール予測』についての研究背景，そして，提案手法の実現方法，理論および比較実験における結果，考察を述べた.

実験結果をもとに考察を行い，提案手法である識別子中の単語情報を用いた Fault-prone モジュール予測は可能であるが，その他の Fault-prone モジュール予測手法と比較すると，予測精度という点ではその他の手法に劣らないが，Fault-prone モジュール予測モデルの構築にかかる時間という観点では，提案手法では説明変数の多さが原因となって時間がかかってしまい，その他の手法に劣るという結論に至った,

今後の課題としては，不具合を含むモジュールにおける単語の傾向を明らかにすることで，ソースコードの品質を向上させることができる識別子名の命名規則を明らかにすることができるのではないかと考える．他にも，説明変数として用いる単語の種類を選別することで，説明変数の種類を絞ることにより，Fault-prone モジュール予測の精度の向上および，予測モデルの構築の高速化が期待できる.

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，本学情報工学部門水野 修准教授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，本学情報工学専攻修士二回生の椋代 凜先輩，大西 哲朗先輩，修士一回生の岡嶋 秀記先輩，胡 軼凡先輩，情報工学課程 学部四回生の河端 駿也君，山田 晃久君，および，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] 畑　秀明，水野　修，菊野　亨，“不具合予測に関するメトリクスについての研究論文の系統的レビュー,” コンピュータソフトウェア，vol.29，no.1，pp.106–117，Feb. 2012.

[2] Briand L.C., Melo W.L., and W. J, “Assessing the applicability of fault-proneness models across object-oriented software projects,” Software Engineering, IEEE Transactions, vol.28, no.7, pp.706–720, July 2002.

[3] T.J. Ostrand, E.J. Weyuker, and R.M. Bell, “Predicting the location and number of faults in large software systems,” Software Engineering, IEEE Transactions, vol.31, no.4, pp.340–355, April 2005.

[4] L. Graves, A.F. Karr, J.S. Marron, and H. Siy, “Predicting fault incidence using software change history,” IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, vol.26, no.7, pp.653–661, July 2000.

[5] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy, “Does distributed development affect software quality? an empirical case study of windows vista,” Communications of the ACM, vol.52, no.8, pp.85–03, Aug. 2009.

[6] 川本公章，“ソースコード中の識別子の語長や単語数の傾向が品質に与える影響の分析,” 卒業研究報告, 京都工芸繊維大学，2013.

[7] 山本秀之，“変数名に基づくソフトウェアバグ密度の予測,” 修士論文，奈良先端科学技術大学院大学，2010.