Vol.2011-SE-173 No.4
2011/7/21

ソフトウェアモジュールにおける識別子の語長と不具合出現に関する分析

川本公章[†1]　水野修[†1]

関数名や変数名などの識別子は，ソースコードを理解するための重要な情報源となっている．すなわち，識別子の命名方法はソフトウェアの理解に多大な影響を及ぼし，その結果として，ソースコードの品質を左右していると考えられる．本研究では，ソースコードにおける識別子と不具合出現に関する分析の一部として，識別子の語長に関する分析を実施した．分析の結果，識別子の語長を用いて不具合有無の予測がある程度可能であることがわかった．

An Investigation between Length of Identifiers and Existence of Faults in Software Modules

Kimiaki Kawamoto [†1] and Osamu Mizuno[†1]

Identifiers such as variable names and function names in source code are important information to understand code. The way of naming for identifiers has a lot of affects on code understanding, and thus, it is guessed that the naming of identifier affects software quality. In this study, we investigate the relationship between the length of identifiers and software faults in a software module. The results show that there is a certain relationship between the length of identifier and existence of software faults.

1. はじめに

プログラムで使われている関数名や変数名などの識別子は，識別子を構成する単語の意味を読み取ることで，ソースコード中のコメント文と同じようにソースコードを理解するために重要な情報源となっている[1)]．また，識別子の語長や単語の省略の方法などといった，識別子の命名がソースコードの品質や可読性に影響を与えている[2)3)]．識別子は，ソースコードを構成する要素の中でも重要なものであり，識別子を解析することで有益な情報が得られるのではと考えられる．

そこで，本研究では，識別子の要素の一つである語長とその識別子が含まれるモジュールの不具合出現に関する分析を行う．不具合を含むモジュールと含まないモジュールそれぞれに対して，モジュールに含まれる識別子の出現回数を語長毎に調べ，機械学習を用いて識別子の出現回数の分布から不具合有無を判別するモデルを構築する．新たなモジュールの不具合有無を，識別子の出現回数の分布にモデルを適用する事で判別し，その結果から，識別子の語長と不具合出現の関係について調べている．

ソフトウェア開発において不具合を含んでいそうなモジュール（fault-prone モジュール）を予測することは，効率的なソフトウェア開発を行うことを可能とし，開発コストの削減に役立てることができる．本研究の識別子の語長の分布を用いた不具合判別は，ソースコードを書いているときに，そのソースコードに含まれる不具合の予測を可能とするため，より効率的なソフトウェア開発が可能になると考えられる．

本論文の以降の構成を示す．第 2 節では，識別子の重要性について詳しく説明する．第 3 節では，識別子の語長を用いた不具合予測の実験について述べる．第 4 節では，実験の結果に対する考察を述べる．第 5 節では，本研究の妥当性について述べる．最後に，第 6 節では，本研究のまとめと今後の課題について述べる．

2. 識別子の重要性

ソースコードのおよそ 70%は識別子で構成されており[1)]，ソースコードを解析する時に識別子に注目する事は重要だと考えられる．識別子は予約語やリテラルでない関数名や変数名などの文字列の事であり，英単語を組み合わせた文字列になっていることが多い．

関数の処理を表す単語を組み合わせて関数名を作ることで，関数に対する注釈（コメント文）を読まなくても関数の処理内容を読み取ることができ，関数呼び出しで記述されている時も関数宣言を見なくても処理内容を知ることが可能となる．変数においても，格納する値の内容や型情報を表す単語を組み合わせて変数名を作ることで，ソースコード内で変数が使われている時に処理の内容を解りやすくできる．このように，識別子はソースコードの処理内容を理解するための情報を持たせることが可能であり，コメント文と同じようにソース

†1 京都工芸繊維大学 大学院工芸科学研究科
Graduate School of Science and Technology, Kyoto Institute of Technology

コードを読むときの重要な情報源となり得る．

ソースコードの品質や可読性は，モジュールの不具合出現への影響があると考えられ，識別子がソースコードの品質や可読性に影響を与えるということは，識別子がモジュールの不具合出現に影響を与えることがあるのではないかと考えられる．

識別子を構成する文字列の長さ（語長），文字の並び，大文字と小文字，単語の省略の方法などといった要素は，プログラマー毎の識別子の命名における違いとなり，それらがソースコードの品質や可読性に影響を与えている[2)3)]．したがって，識別子の良い命名規則を作ることは，ソースコードの品質を高めることに繋がると考えられる．文献 4) では Ada と Java の為の識別子の命名規則というものも提案されている．

ソースコードの品質や可読性に影響を与える識別子命名の要素として以下に示すようなものがある．

- 識別子の語長
- 単語の数
- 単語の省略形や頭文字
- ハンガリー記法
- 大文字と小文字
- 数字や数の単語

識別子の語長と識別子を構成する単語の数は，識別子の可読性や情報量を左右し，単語の数が少なければ識別子の持つ情報は減るが，語長が短くなり読みやすくなる．単語の数が多ければ識別子の持つ情報は増えるが，語長が長くなり読みやすさは下がる．

単語の省略形や頭文字によって，よく使われている単語を省略して記述することで識別子の語長を短くすることが可能となるが，コードを読む者がプログラミングやソフトウェア開発に精通していない場合は可読性が下がることもある．省略の方法もプログラマによって異なることがあり，ソースコードの可読性を大きく左右する要素となる．

ハンガリー表記は，識別子に型情報やスコープ範囲を表す特別な接頭文字を付ける命名法である．これにより，識別子の情報を簡単に得ることが可能となるが，表記の統一を行わなければソースコードの可読性を下げることもある．

大文字と小文字は，識別子の見た目を左右する要素であり，ソースコードの可読性に影響を与える．

数字や数の単語は，同じような識別子を新たに作るときに，命名の負担を減らすために数字を付加して新しい識別子とすることがあるが，これは間違った識別子の指定の原因にもつながり，ソースコードの品質を下げることになる．

表 1 対象のプロジェクト
Table 1 Target Projects

	Eclipse	Netbeans
モジュールの数	308,966	393,531
faulty モジュールの数	159,818	165,560
non faulty モジュールの平均語長	8.98	8.55
faulty モジュールの平均語長	9.15	8.54
モジュール毎の平均識別子数	950	623
測定開始日	2001-05-03	1999-01-05
測定終了日	2010-05-22	2010-06-25

本研究では，識別子命名の要素の中で識別子の語長について注目している．識別子の語長は，言語に関係なく値を得ることが可能であり，ソースコードから手軽に測定することができる要素である．また，単語の数や省略形などの要素は，識別子の語長にも影響しており，語長の分析は他の命名要素が持つソースコードへの影響も取り入れた分析を可能にすると考えられる．

3. 適 用 実 験

2 つのオープンソースプロジェクトに対して，識別子の語長とその識別子が含まれるモジュールの不具合出現に関する分析を行う．不具合を含むモジュールと含まないモジュールそれぞれに対して，モジュールに含まれる識別子の出現回数を語長毎に調べ，機械学習を用いて識別子の出現回数の分布から不具合有無を判別するモデルを構築する．新たなモジュールの不具合有無を，識別子の出現回数の分布にモデルを適用する事で判別し，その結果から，識別子の語長と不具合出現の関係について調べる．

3.1 実 験 対 象

本実験では，オープンソースプロジェクトの Eclipse と Netbeans のソースコードを利用して実験を行なっている．Eclipse と Netbeans はともに Java 言語を使って開発されている統合開発環境のソフトウェアである．2 つのプロジェクトの比較を**表 1** に示す．

2 つのプロジェクトはともに 10 年近く開発が続けられているプロジェクトで，モジュールの数も 30 万を超える大規模なものである．モジュールは，java ファイルの数としており，バージョン管理システムにファイルの変更がコミットされるごとに新たなモジュールとして扱っている．モジュールが不具合を含んでいるかの判別には，バージョン管理システムと

バグ管理システムの情報を組み合わせて不具合の特定を行うことのできる SZZ アルゴリズム[5] を用いている．

識別子の語長の分布と不具合モジュールに含まれる識別子の語長ごとの割合を，**図 1**（Eclipse）と**図 2**（Netbeans）に示す．図 1（a），(b）と図 2（a），(b）は，語長ごとの全モジュールにおける識別子の出現数を，不具合の有無に分けて示している．図 1（a），図 2（a）では，語長ごとに出現回数に大きく差があり，出現回数が少ない語長が見えないため，図 1（b），図 2（b）では，縦軸に対数を使っている．図 1（c），図 2（c）では，不具合有無の割合を示している．

図 1，図 2 から，識別子の語長の出現回数は，語長が 5〜10 の場合に多くなり，それより語長が長くなると急に出現回数が少なくなっていることがわかる．また，不具合有無の割合は，識別子の出現回数が多い語長では一定となっており，出現回数が少なくなると不具合有無の割合に変動が起きていることがわかる．表 1 に示したように，non faulty モジュールと faulty モジュールの平均語長はほとんど等しく，単純に語長を使って不具合を判別する事は難しいと考えられる．そこで，モジュールごとの識別子の分布を使うことで不具合の判別を提案する．

3.2 実験方法

実験の流れを**図 3** に示す．まず，モデルの構築を行い，その後構築したモデルを用いて新たなモジュールの判別を行う．

モデル構築では，non faulty モジュールと faulty モジュールを分けて，ソースコードのトークン化，識別子の抽出，語長の計算を行い，教師あり学習の Support Vector Machine（SVM）を用いてモデルの構築を行う．

予測では，新たなモジュールを，ソースコードのトークン化，識別子の抽出，語長の計算を行い，モデルを使って判別し，不具合有無の予測を行う．

SVM を用いた学習には，オープンソースのデータマイニングソフトウェア Weka の SVM を使った分類器である「SMO」を使用している．モデルの評価をする為に十重交差検証を行っている．

3.3 実験結果

実験結果として，予測結果の評価を**表 2** に，予測結果を**表 3** と**表 4** に示す．また，SVM で得られた各語長の係数（寄与度）を**図 4** と**図 5** に示す．

予測結果の評価である表 2 において，Accuracy（正答率）は，実測が faulty で予測も FP，実測が non faulty で予測も NFP と予測できたモジュールの割合を示す．Precision（適合

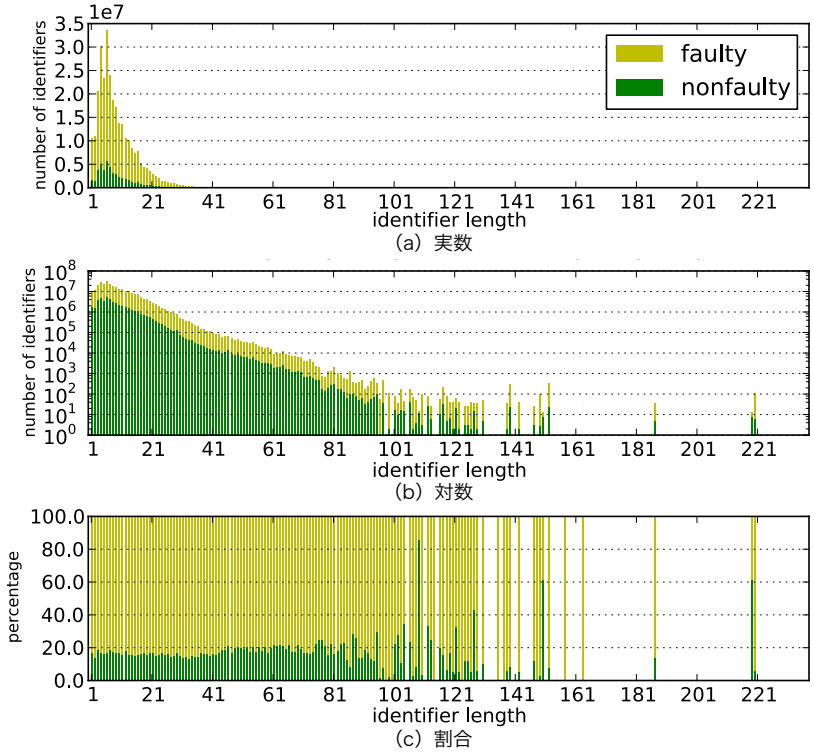

図 1 語長の分布（Eclipse）
Fig. 1 Distribution of identifier length（Eclipse）

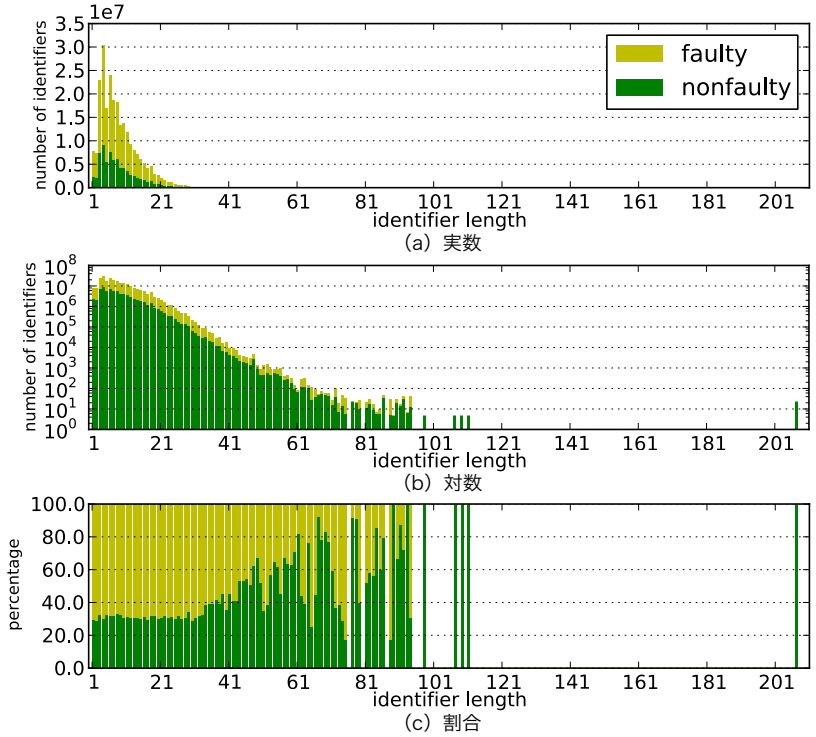

図 2 語長の分布（Netbeans）
Fig. 2 Distribution of identifier length（Netbeans）

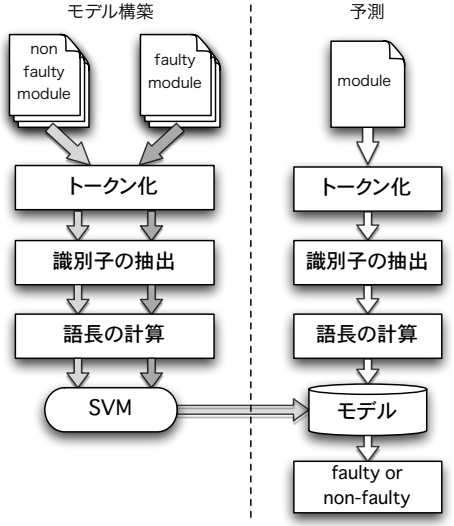

図 3 実験の流れ
Fig. 3 A flow chart of experiments

率）は，予測で FP と判断して実測でも faulty であったモジュールの割合を示し，正確性の指標となる．Recall（再現率）は，実測が faulty の中で予測でも FP と判断できたモジュールの割合を示し，網羅率の指標となる．F-Measure（F_1 値）は，適合率と再現率の調和平均であり，値が高いほど予測精度が高いといえる．

表 2，表 3，表 4 より，Eclipse と Netbeans ともに，正答率が約 7 割で F_1 値も約 7 割と不具合の有無に関わらずバランスよく予測ができていることがわかる．

SVM で得られた各語長の係数を示した図 4 と図 5 では，係数の値を棒グラフで示し，±2 の語長で単純移動平均をとった値を折れ線グラフで示している．単純移動平均に注目することで，変動の激しい値に対する大まかな傾向をつかむことができると考えられる．係数の値が大きくなるほど（正の数）その語長は faulty モジュールに判断されることに寄与し，係数の値が小さくなるほど（負の数）その語長は non faulty モジュールに判断されることに寄与している．

図 4 と図 5 より，係数の変動はとても大きくなっている．しかし，語長が短いときに係数が大きく，語長が長くなるほど係数が小さくなる傾向が見られる．

表 2 予測の評価
Table 2 Evaluation of prediction

	Accuracy	Precision	Recall	F-Measure
Eclipse	0.713	0.737	0.713	0.708
Netbeans	0.699	0.711	0.699	0.680

表 3 予測結果（Eclipse）
Table 3 Prediction result: Eclipse

		予測	
		NFP	FP
実測	faulty	129,025	21,264
	non-faulty	68,110	92,849

表 4 予測結果（Netbeans）
Table 4 Prediction result: Netbeans

		予測	
		NFP	FP
実測	faulty	202,215	24,406
	non-faulty	93,906	72,997

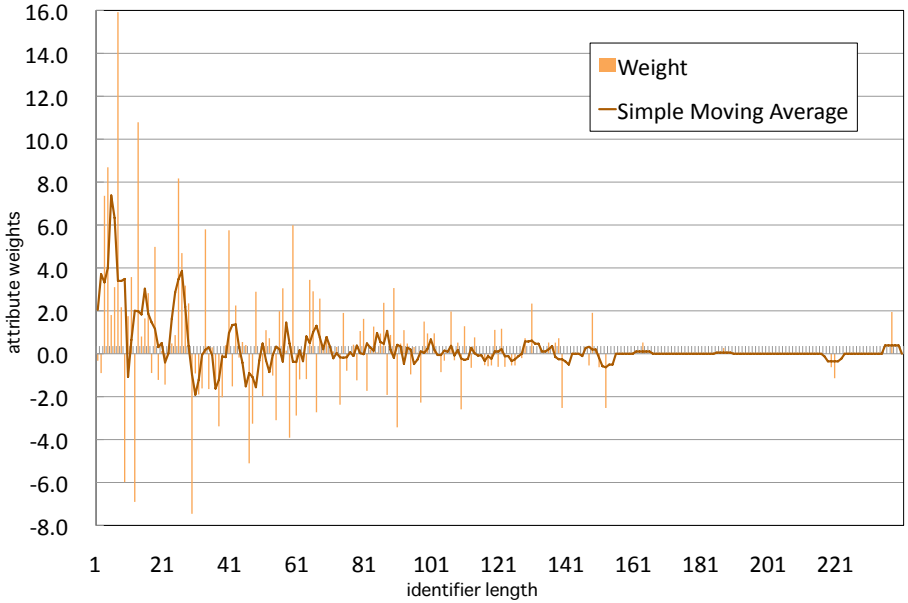

図 4 各語長の係数（Eclipse）
Fig. 4 Weights of attributes in SVM (Eclipse)

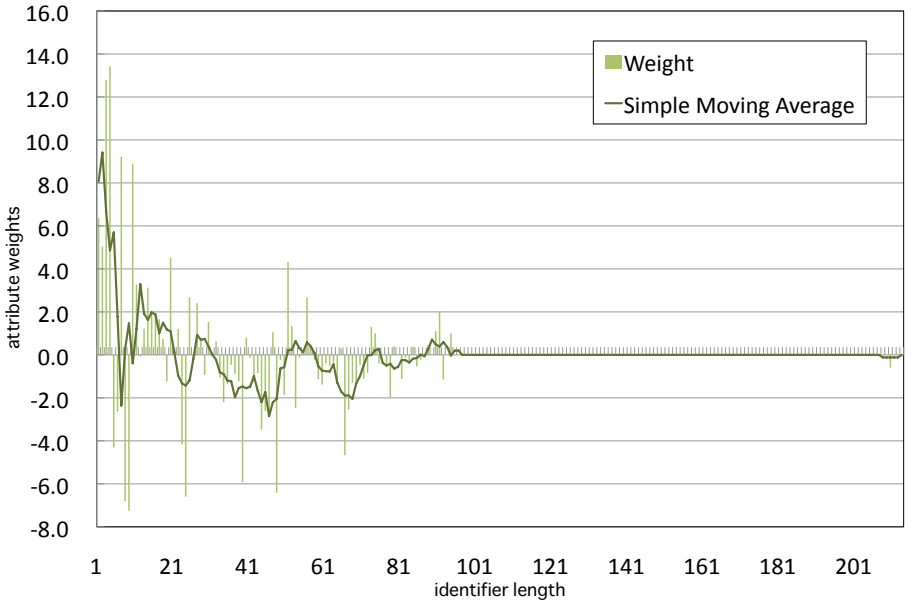

図 5 各語長の係数（Netbeans）
Fig. 5 Weights of attributes in SVM (Netbeans)

4. 考　　察

予測結果の評価である表 2 より，正答率，F_1 値ともに約 7 割の値となっており，モジュールごとの識別子の語長の分布を用いた不具合出現の予測が出来ていると考えられる．識別子の語長の分布を用いた予測のためのモデル構築には，ソフトウェア開発のリポジトリと不具合有無の情報が必要となるが，新しいモジュールの判定には，そのモジュールに含まれる識別子の語長を調べてモデルを使った判定を行うので，時間を掛けずに不具合の予測が可能である．

SVM で得られた各語長の係数を示した図 4 と図 5 より，各語長の係数の値は大きく変動しており傾向がつかめないため，単純移動平均の値に注目してみる．また，図 1，図 2 の語長ごとの識別子の出現回数を見てみると，語長が長くなると出現回数が小さくなっているので，出現回数の多い語長 1 から 21 の部分に注目し，SVM で得られた各語長の係数のグラ

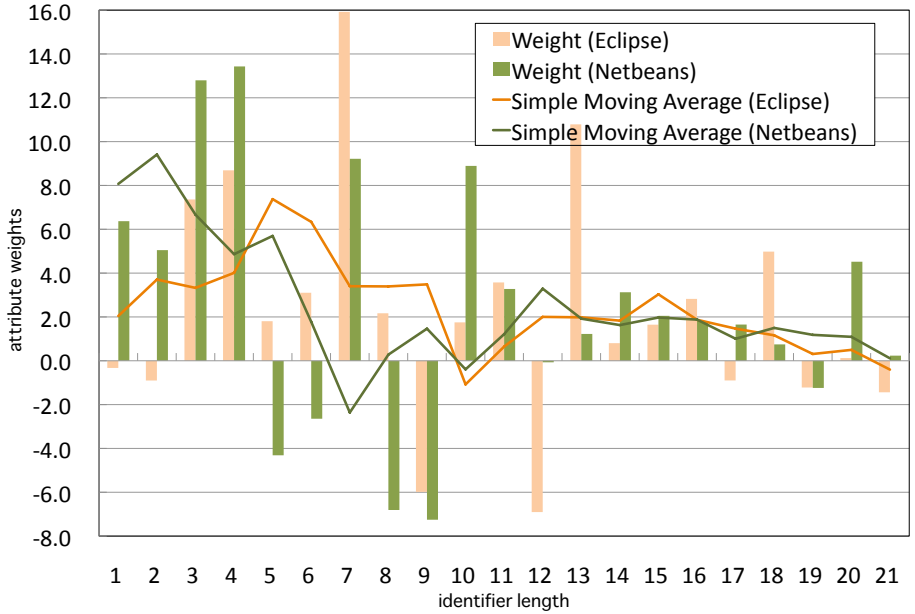

図 6 各語長の係数
Fig. 6 Weights of attributes in SVM

フを**図 6** に示す．

図 6 より，語長 3, 4, 7 で Eclipse と Netbeans ともに係数の値が大きくなっており，語長 3, 4, 7 の識別子は不具合を含むモジュールによく現れていると考えられる．しかし，語長 8 から 12 では係数の値が小さい傾向にあり，語長 8 から 12 の識別子は不具合を含まないモジュールによく現れていると考えられる．従って，語長が 8 より短い識別子が多用されるモジュールには不具合が入り込みやすくなると類推される．

5. 妥当性の検証

- プログラムの不備
  実験の過程で，ソースコードのトークン化，識別子の抽出，語長の計算を行っているが，この部分の計算に不備があった場合は，正しい識別子の語長を計算出来ていないため，不具合の予測が正しくないことが考えられる．
- 学習データの不備
  モデル構築の学習データに用いた不具合有無の情報は，SZZ アルゴリズム[5] によって計算されているが，この時に用いたバージョン管理システムとバグ管理システムの情報の不備やアルゴリズム自体の不備がある場合，不具合の情報に間違いが含まれ，正しくモデル構築のための学習が行えていないことが考えられる．
- 学習アルゴリズムの不備
  本実験では，オープンソースのデータマイニングソフトウェア Weka の SVM を使った分類器の SMO を使用しているが，このアルゴリズムに不備があった場合正しくモデルの構築が行えないため，予測結果が正しくないことが考えられる．

6. ま と め

本研究では，ソースコード内で識別子が重要と捉え，識別子のもつ要素の中で識別子の語長に注目し，モジュール内の識別子の語長の分布を使って不具合予測のモデルを構築し，識別子の語長を使った不具合予測を行った．その結果，識別子の語長を用いて不具合の予測が可能なことがわかった．

今後の課題として，語長ごとの不具合出現との関連をより深く調査することや，2 節で挙げた識別子の要素の中で語長以外の要素を特徴量に追加し，不具合予測のモデル構築を行うことが考えられる．

参 考 文 献

1) Deissenboeck, F. and Pizka, M.: Concise and consistent naming, *Software Quality Control*, Vol.14, pp.261–282 (2006).
2) Butler, S., Wermelinger, M., Yu, Y. and Sharp, H.: Relating Identifier Naming Flaws and Code Quality: an empirical study, *Proc. of the Working Conf. on Reverse Engineering*, IEEE Computer Society, pp.31–35 (2009).
3) Lawrie, D., Morrell, C., Feild, H. and Binkley, D.: What's in a Name? A Study of Identifiers, *Proceedings of the 14th IEEE International Conference on Program Comprehension*, Washington, DC, USA, IEEE Computer Society, pp.3–12 (2006).
4) Relf, P.A.: Achieving Software Quality Through Identifier Names, *Qualcon 2004* (2004).
5) Śliwerski, J., Zimmermann, T. and Zeller, A.: When do changes induce fixes?, *Proceedings of the 2005 international workshop on Mining software repositories*, New York, NY, USA, ACM, pp.1–5 (2005). St. Louis, Missouri.