変数名とスコープの長さ及びコメントに着目したフォールト潜在性に関する定量的調査

阿萬 裕久[1,a)]　天嵜 聡介[2]　佐々木 隆志[1]　川原 稔[1]

概要：ソフトウェア開発において，潜在フォールトの早期検出と除去は重要な課題であり，そのための支援技術の一つとしてプログラム中のコメントに着目する手法が研究されている．コメントはプログラムの理解容易性を高める上で有用な要素であるが，複雑で分かりにくい部分に対してその可読性の低さを補う目的で追記されることもある．これまでに Java プログラムを対象とした調査がいくつか行われ，メソッドの中にコメントが書かれている場合にそのメソッドのフォールト潜在性は他のメソッドに比べて高いという傾向が確認されている．本論文では，コメントに着目したフォールト潜在予測の精度向上に向け，新たな視点としてメソッドにおけるローカル変数の名前とスコープの長さに着目した調査（データ収集と分析）を行っている．三つの著名なオープンソースソフトウェアを対象とした調査の結果，コメントのみならず変数の名前の長さとスコープの長さも考慮した分類を行うことの有効性が確認されている．

1. はじめに

高品質なソフトウェアを効率的に開発していく上で，ソースコードの品質管理は欠かすことのできない重要な活動である．多くの場合は開発者，つまり人間がソースコードを書くことになるため，人為的な誤りの混入を完全に防止するというのは難しい．フォールトを作り込まないよう注意するのはもちろんであるが，出来上がったプログラムのレビューやテストを可能な限り網羅的に行い，早期にフォールトの検出と除去を行うことが望ましい．その際，フォールトの潜在が疑われる箇所を機械的に絞り込むこと（一般に fault-prone モジュール予測 [1] と呼ばれる）ができれば，レビューやテストの効率的な実行が可能になる．

これまで，フォールト潜在が疑われる箇所の絞り込みには，ソースコードの特徴をメトリクスによって数値化し，その上で統計モデルや機械学習モデルを使用する手法が広く研究されている [2]．従来より，規模や構造的な複雑さといった属性が多く使われているが，我々は近年，ソースコード中に記述されるコメントに着目した手法を提案している [3], [4], [5]．コメントはプログラムの実行には一切影響を及ぼさないため，多くの研究では無視されてきた要素であるが，実際のプログラミングでは開発者による内容説明や他者へのメッセージが書かれることが多く，品質に無関係であるとは考えにくい．オープンソースソフトウェアを対象とした実証実験では，コメントが多く書かれているプログラムの方がフォールト潜在の可能性が高いという傾向が確認されている．これは，コメントが品質に悪影響を及ぼしているのではなく，結果的にプログラムが複雑で分かりにくいものになってしまった場合に，その可読性の低さをコメントでカバーしようとしている表れではないかと考えられている．このように，プログラムの構造的な側面だけでなく，開発者の心理的な側面も考慮することでフォールト潜在予測の精度を高めることが期待される．

コメント以外で開発者の心理に関係する要素の一つとして“変数名”が挙げられる．特にローカル変数の名前は開発者の裁量で決定できる場合も多く，コメントと同様に個人差が出やすい要素であると考えられる．単純な繰り返し文での制御変数であれば，命名をさほど気にする必要はない（例えば `for` 文を使って配列内の要素を走査するために変数 `i` を用いる等）と言われている [6] が，スコープの長い変数となる場合には注意が必要なこともある．

これまで，文献 [3], [4], [5] ではプログラムにおけるコメントとフォールト潜在性の関係について分析してきているが，そこに登場する変数については考慮できていなかった．そこで本論文では，コメントのみならずローカル変数にも着目し，その名前の長さとスコープの長さを考慮した上でフォールト潜在性の調査・分析を行う．

以下，2 章ではコメントとローカル変数について調査を

[1] 愛媛大学
Ehime University, Matsuyama, Ehime 790–8577, Japan
[2] 岡山県立大学
Okayama Prefectural University, Soja, Okayama 719–1197, Japan
[a)] aman@ehime-u.ac.jp

行うことの意義を関連研究とともに述べる．そして，3 章で調査の目的，対象，結果，考察並びに妥当性への脅威について述べる．最後に 4 章において本調査で得られた知見についてまとめ，今後の課題について述べる．

2. コメントとローカル変数

コメントがプログラムの理解容易性を高める上で有用な存在であることは広く知られている [7], [8], [9]．しかしながら，コメント無しでは内容を理解できないようなプログラムになってしまうことは避けなければならない．Kernighan らは，プログラムに対して丁寧なコメント追記が必要になった場合は，コメントを書くよりもむしろそのプログラムの方を書き直すべきであると主張している [6]．リファクタリングの分野でもコメントは “不吉な臭い（匂い）[*1] の予兆” と言われ，コメントが書かれている部分を見直しの対象とすることが推奨されている [10]．ただし，コメント自体が悪い存在というわけではなく “消臭剤” の働きをしているという指摘である．つまり，結果的にコメントの存在が低品質なソースコードの目印になってしまっていることがあるというものである．

我々はこれまで，オープンソースソフトウェアを対象とした分析を通じて，コメントの書かれているプログラムの方がフォールト潜在性は高いという傾向を確認している [3], [4], [5]．コメントの記述は開発者の裁量によるものではあるが，プログラミングの際にコメントの必要性を感じた場合，コメント無しでも理解できるようにその部分を改善できないか考える価値はあると思われる．また，コードレビューの際にコメントが書いてある箇所の改善を検討してみるのも有効であると考えられる．これに関連して，メソッドの途中に 1 行だけコメントが書かれていた場合は，そのコメントの内容をメソッド名としたメソッド抽出リファクタリングを検討するとよいという提案もある [11]．

前述したように，ローカル変数の名前もコメント同様に開発者が自分の裁量で決定でき（その可能性が高く），個人差が出やすい要素である．一般には，どういった情報を保持しているのかが分かるように変数名を付けるべきであるが，スコープが短い場合は省略形や頭文字のみでも問題ないと言われている [6], [12]．Lawrie らは変数名を英単語のフルスペル，省略形（例えば `count` を `cnt`，`length` を `len` と省略）及び頭文字のみ（1 文字）の三つの種類に分け，100 人を超えるプログラマに対してアンケート調査を行って変数名の違いがプログラムの理解度に及ぼす影響を調査している [13]．その結果，フルスペルの変数名が最も分かりやすいが，略語であってもさほど問題にならないこと，そして 1 文字変数が最も内容を伝えにくいという傾向が確認されている．この結果から見ても，1 文字変数（通常，意図したい単語の頭文字を使用）それ自身で伝えることのできる情報は限定的であり，他の開発者へ適切に意図が伝わる可能性は低いと考えられる．つまり，1 文字変数のように短い名前で意図が伝わりにくいローカル変数を使用しており，しかもそのスコープが長い場合，そのような変数が登場するメソッドでは内容の誤解や誤った記述を引き起こしてしまうリスクが高いと懸念される．それゆえ，そのような変数が登場するメソッドではフォールト潜在性が他のメソッドよりも高くなってしまう恐れもある．

一方，変数名が長い方が単純に望ましいとは言い切れない点にも注意が必要である．経験的には，ローカル変数に付ける名前は長すぎない方がよいとも言われている [14]．Kawamoto らはオープンソースソフトウェアを対象とした分析を通じて，長い名前の変数が登場しているプログラムの方がフォールト潜在性は高いという傾向を報告している [15]．つまり変数名は，長い方がその意味するところを伝えやすいが，フォールト混入の可能性を考えると必ずしも長い方が望ましいとは言えず，スコープ長といった他の側面も考慮する必要があると思われる．

我々はこれまで，メソッドにおけるコメントの存在とフォールト潜在性の関係について実証実験を行ってきた [3], [4], [5] が，ローカル変数との関係については考慮してこなかった．ローカル変数の名前の長さとスコープの長さの組合せによってフォールト潜在性に差が生じることも考えられるが，そこにコメントが添えられている場合とそうでない場合にそれぞれどのような傾向が見られるかについては明らかにできていない．そこで本論文では，次章において複数のソフトウェアに対する調査と分析を行い，コメントのみならずローカル変数の名前とスコープの長さも考慮した上でフォールト潜在性の傾向について考察する．

3. 調査

3.1 目的

本調査では，オープンソースソフトウェアを対象として，ローカル変数の名前とスコープの長さ並びにコメント記述について次の二つの視点から調査・分析を行う．

調査（1）：ローカル変数の名前とスコープの長さはそれぞれどのような分布になっているか．さらには変数名の長さとスコープの長さの間には何か関係性が見られるか．

調査（2）：メソッドをそこに登場するローカル変数の名前の長さ，スコープの長さ及びコメント記述の有無で分類した場合にフォールト潜在性に違いが見られるか．

□

まず，調査（1）を行うことで，実際のソフトウェアにおけるローカル変数の変数名とスコープの長さの状況を把握する．これにより，変数名の長さとスコープの長さについ

[*1] 文献 [10] の翻訳として 2000 年に出版された書籍では code smell の訳語として “不吉な 匂い” が使われており，2014 年に新装版として出版されたものでは “不吉な 臭い” が使われていることから，本論文では括弧書きで併記することにする．

て，それぞれどの程度の長さが一般的であり，どの程度のものが “長い” といえるのかを定量的に判定できるようになる．続いて調査（2）では，ローカル変数の名前の長さ，スコープの長さ及びコメント記述の有無という三つの特徴の組合せに従ってメソッドを分類し，スコープの長い変数や短い変数が登場している場合にコメント記述の有無でフォールト潜在性に差があるかを確認する．これにより，従来から行われているコメントとフォールト潜在性との関係に関する研究に対して，ローカル変数の名前とスコープの長さという特徴も考慮すべきかどうかを考察する．

3.2 対象

本調査では，三つのオープンソースソフトウェア Eclipse Checkstyle Plug-in[*2], PMD[*3] 及び SQuirreL SQL Client[*4] を対象としてデータ収集と分析を行った（**表 1**）．なお，表 1 に示したソースファイル数は 2015 年 4 月 28 日現在のものであり，テスト用若しくはドキュメント用と思われるソースファイル [*5] は除外してある．

3.3 調査（1）：変数名とスコープの長さの分布

3.3.1 変数名の長さとスコープの長さ

まず，対象ソフトウェアに登場するローカル変数について，それぞれの名前の長さを調べた．変数名の長さの分布を **図 1** 及び**表 2**[*6] に示す．その結果，1 文字の変数が最も多く，変数名が長くなるにつれてその数は概ね指数関数的に減少する傾向にあった．そして，大半のローカル変数は，その長さが数文字未満となっていた（表 2）．つまり，ローカル変数の名前としては，その長さが 5 文字を超えるものは “どちらかといえば長い方” に分類されるといえる．

続いて，各ローカル変数のスコープの長さを調べた．ここでいう “スコープの長さ” とは，調査対象のソースプログラムにおいて，当該変数が利用可能となっている範囲を行単位で表したものとなっている．結果として得られたスコープの長さの分布を**図 2** 及び**表 3** に示す．変数のスコープ長は 5 行以下と短いものが最も多く，変数名の長さの場合と同様に，スコープ長が長くなるほど該当するローカル変数の数は概ね指数関数的に減少する傾向にあった．大半のスコープ長は 10 行以下となっていたが，50 行や 100 行を超えるものも少なからず存在しており，スコープ長については個体差が大きいことがうかがえる．

表 1 調査対象
Table 1 Open source software products surveyed in this work.

ソフトウェア名	概要	ソースファイル数
Eclipse Checkstyle Plugin	静的コード解析用 Eclipse プラグイン	220
PMD	静的コード解析ツール	938
SQuirreL SQL Client	データベース管理ツール	3,898

[*2] `http://eclipse-cs.sourceforge.net/`
[*3] `http://pmd.sourceforge.net/`
[*4] `http://squirrel-sql.sourceforge.net/`
[*5] ファイルパスに `test` または `documentation` という文字列が含まれているものを指す．
[*6] “75% 点（第 3 四分位数）+ 1.5 × 四分位範囲” よりも大きな値は外れ値として扱われる．今回の分布では第 3 四分位数が 9，四分位範囲（第 3 四分位数 − 第 1 四分位数）が 7 であることから，9 + 1.5 × 7 = 19.5 が外れ値の閾値となる．

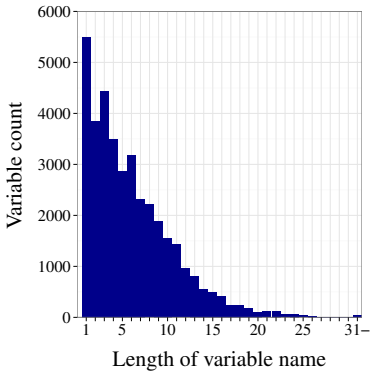

図 1 ローカル変数名の長さのヒストグラム
Fig. 1 Histogram of the length of local variable names.

表 2 ローカル変数名の長さの分布
Table 2 Distribution of the length of local variable names.

最小値	25% 点	中央値	75% 点	最大値	外れ値を除いた最大値
1	2	5	9	41	19

3.3.2 変数名の長さとスコープの長さの関係

次に，変数名の長さとスコープの長さの関係を分析した．**図 3** にローカル変数名の長さとスコープの長さの散布図を示す．散布図から分かるように，変数名の長さ（横軸）が長くなってもスコープ長（縦軸）も長くなるという傾向にはなく，短い名前のローカル変数であってもスコープの

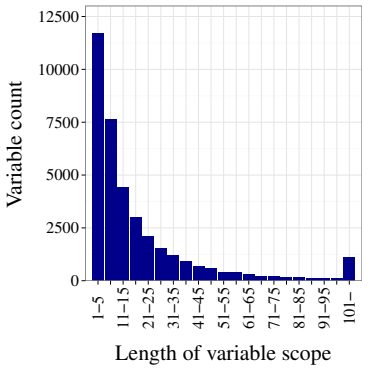

図 2 ローカル変数のスコープ長のヒストグラム
Fig. 2 Histogram of the scope length of local variables.

表 3 ローカル変数のスコープ長の分布
Table 3 Distribution of the scope length of local variables.

最小値	25% 点	中央値	75% 点	最大値	外れ値を除いた最大値
1	4	10	23	1313	51

長いものが比較的多く存在していることが分かった．スピアマンの順位相関係数を計算したところ 0.237 となり，やはり変数名の長さとスコープの長さの間には強い相関関係は見られなかった．ただし，無相関というわけではなかったこと，さらには散布図から分かるようにばらつきが大きいことから，スコープ長の中央値と変数名の長さの関係についても確認した．結果を **図 4** に示す．図 4 から分かるように，緩やかではあるが変数名が長くなることでスコープ長も長くなる傾向は見られた．

3.4 調査（2）：ローカル変数の名前の長さ，スコープの長さ及びコメント記述の有無によるメソッドの分類とフォールト潜在性の比較

調査（2）では，各メソッドで使われているローカル変数の名前の長さ，スコープの長さ及びコメント記述の有無という特徴に従ってメソッドの分類を行い，分類間でのフォールト潜在性を比較していく．

3.4.1 フォールト潜在性に関するデータ収集

まずは，各メソッドにおけるフォールト潜在性の評価データを収集する．本調査では，対象ソフトウェアの各ソースファイルにおける各メソッドの変更履歴を追跡し，各メソッドの初期バージョンを分析の対象とする．そして，その後にフォールト（バグ）修正が行われていたメソッドを“フォールトあり”，そうでないものを“フォールト無し”と見なす．なお，バグ修正が行われていたかどうかは，コミットログに “bug”, “fix” または “defect” という単語が登場しているかどうかで判定する．今回の対象ソフトウェアでは 11,137 個のメソッドについてデータを得ることができ，その中の 543 個（約 5%）のメソッドがフォールトありと判断された．

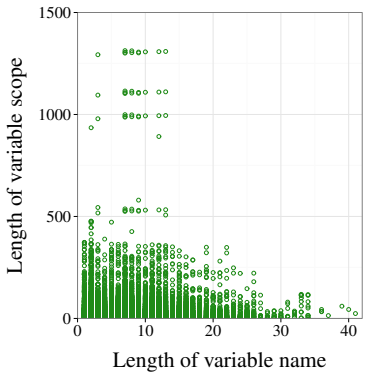

図 3 ローカル変数名の長さとスコープの長さの散布図
Fig. 3 Length of variable names vs length of scopes.

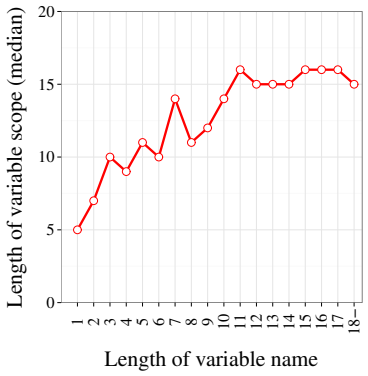

図 4 ローカル変数名の長さとスコープ長の中央値
Fig. 4 Length of variable names vs median length of scopes.

3.4.2 変数名の長さに基づいた分類

次に，ローカル変数の名前の長さに従って，メソッドを分類（カテゴリ分け）していく．一つのメソッドには任意個のローカル変数を宣言可能であるため，ここではそのメソッドの中で最も長いスコープを持つ変数に着目することとする．例えば，あるメソッドに二つのローカル変数 `x` と `msg` が宣言されており，前者のスコープ長が 10 行，後者のスコープ長が 8 行であったとする．その場合，スコープ長の長い方の変数（`x`，つまり 1 文字変数）に着目し，そのメソッドは 1 文字変数が使われているものとして分類する．

調査（1）の結果（表 2）を考慮して，メソッドを次の三つのカテゴリ A, B, C へ分類する：

カテゴリ A：ローカル変数が 1 文字または 2 文字．
カテゴリ B：ローカル変数が 3 文字以上 5 文字以下．
カテゴリ C：ローカル変数が 6 文字以上 19 文字以下．

□

カテゴリ A は変数名の長さの分布でいうところの 25% 点（第 1 四分位数）以下に相当し（表 2 参照），特に短い名前のローカル変数が使われているメソッドが属することになる．同様に，カテゴリ B は 25% 点よりも大きく中央値以下に相当し，どちらかといえば短い方の名前のローカル変数が使われているメソッドが属することになる．最後にカテゴリ C は中央値よりも大きく（外れ値を除いた）最大値以下に相当し，どちらかといえば長い方の名前のローカル変数が使われているメソッドが属することになる．なお，外れ値に該当する変数名は極めて長い名前という扱いになるが，データ数の少なさ故に分析結果の一般性を保証できないため，本論文では分析の対象外とした．

カテゴリ A にて 1 文字変数と 2 文字変数がまとめて扱われている点が懸念されるが，実際には**図 5** に示す変数が 2 文字変数として登場しており，1 文字変数と同様にその名前だけでは何の略語なのか，何を意味しているのかが分かりにくいものが大半であった．一方，調査対象に登場した 3 文字変数は **図 6** に示した通りであり，何の略語なのかが推察可能なものの割合は比較的高くなっているように思われ（例えば，`pos` は `position`，`str` は `string` の略語であると推察される），一般に使われている単語も登場するようになっている（例えば，`dir`，`doc`，`job` 等）．図 5 と 図 6 を見比べる限りでは，2 文字変数と 3 文字変数では変数名そのものが提供できる情報量に比較的大きな差が

```
aa ad be bg bm bp br bs bw c1 ci cl cp cr db e1 er
ex f1 fc fd fi fm fn fw h2 in is it ix l1 m1 md mh
nd nl nn oi pe pf pp pt rc rs rv s2 sb sc si sn so
sp sr st sw tf th ti tm tn tt y1 zf
```

図 5 調査対象で登場している 2 文字変数
Fig. 5 Two-character variables in surveyed methods.

```
adm ape api app ast bos box brd btn buf cal cca
cdl cfg cid cls clz cmd cmp col con cpd csb ctx
DB2 dbf dim dir doc doi dtm eci end env esc fds
fmt g2d gbc hql iae img ioe job key lbl len lhs
lis loc log lsm map max min mnd msg nfe now obj
occ opt out pcr pmd pnl pos psz rad ref res ret
rhs row rpt sel sep sql str tab tbl tcc tci tcm
tfc tmp tok tpa uee url val var vid vmc wss xml
```

図 6 調査対象で登場している 3 文字変数
Fig. 6 Three-character variables in surveyed methods.

あり，2 文字と 3 文字の間でカテゴリ分けの境界を設けることに大きな矛盾はないものと筆者らは考える．さらに文字数を増やせば英単語がフルスペルの形で変数名として使われている場合もあり，文字数が多い（変数名が長い）ほど文字数で細かくカテゴリ分けする意味は薄いと思われることから，カテゴリ C では中央値よりも長い名前の変数でひとまとめに扱うこととした．

カテゴリ分けを行った結果，スコープ長の分布は **表 4** に示す通りとなった．

3.4.3 メソッドのフォールト潜在性分析

（i）スコープ長が特に長いローカル変数を持ったメソッドのフォールト潜在性

表 4 をもとに，各カテゴリにおいてスコープ長が 75% 点を超えるものを “特にスコープの長い変数を持ったメソッド” と考える．さらに，そのようなメソッドを “コメント記述あり” と “コメント記述無し” の 2 種類に分類し，各分類におけるフォールト潜在率を算出する．なお，ここでいうフォールト潜在率とは，当該集合（分類）に属するメソッドの中でフォールトありメソッドが占める割合であり，フォールト潜在性の高さを定量的に表すものである．コメント記述の有無については，先行研究においてフォールト潜在性との関係性が確認されているという理由から，メソッドの中に書かれているコメント（Javadoc のようにメソッドの前に書かれているものを除く）のみを対象とする．コメントアウトはこれに含めない．

表 4 各カテゴリでのローカル変数のスコープ長の分布
Table 4 Distribution of the scope lengths for each category.

カテゴリ	最小値	25% 点	中央値	75% 点	最大値*
A	1	3	5	11	23
B	2	5	11	20	42
C	2	6	12	23	48

(*外れ値を除く)

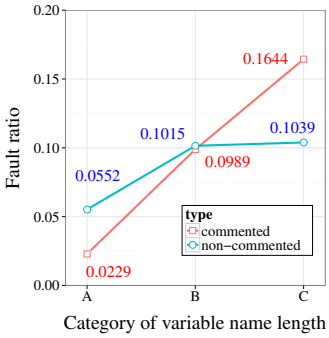

図 7 スコープの長い変数を含むメソッドでのフォールト潜在率
Fig. 7 Fault ratios of methods having longer scope variables.

今回の調査では当初，ソフトウェアごとにフォールト潜在率を算出することを想定していたが，実際に変数名の長さ，スコープの長さ及びコメント記述の有無という 3 段階でデータを分類していくと，分類（部分集合）によってはデータ数が少なくなり，結果の一般性が損なわれることが懸念された．そこで今回は，特定のソフトウェアに偏ることなく三つのソフトウェアそれぞれから同数の標本を無作為抽出してフォールト潜在率を算出するという作業を 10,000 回繰り返し，その平均を利用することにした．

結果として得られたフォールト潜在率を **図 7** に示す．カテゴリ A ではコメント無しのメソッドの方がコメントありのメソッドよりもフォールト潜在率が高く（コメント無しではコメントありの約 2.4 倍），カテゴリ B ではほぼ同じになり，カテゴリ C では逆にコメントありのメソッドの方がフォールト潜在率が高い（コメントありではコメント無しの約 1.6 倍）という結果になった．

特徴的な点として，カテゴリ A とカテゴリ C の間で，コメントの存在とフォールト潜在性との関係性（傾向）が逆転していることが挙げられる．一般に，プログラムに対してコメントを記述することは，そのプログラムの可読性や理解容易性を向上させる働きがあると考えられる．しかしながら，これまでに行われてきた実証研究 [3], [4], [5] では，結果的にコメントが書かれているプログラムの方がフォールト潜在性が高く，品質の低いものであったと報告されている．これは，コメントが品質に悪影響を及ぼしているのではなく，結果的にプログラムが複雑で分かりにくいものになってしまった場合に，その可読性の低さをコメントでカバーしようとしている表れではないかと考えられている．カテゴリ C での結果がそういったこれまでの研究結果に符合している一方，カテゴリ A での結果は逆の傾向を示している．

（ii）スコープの長さが中程度のローカル変数を持ったメソッドのフォールト潜在性

次に，(i) と同様の分析を “スコープの長さが中程度の変数を持ったメソッド” に対して行った．ただし，スコープの長さが “中程度” というのは 表 4 での 25% 点より大き

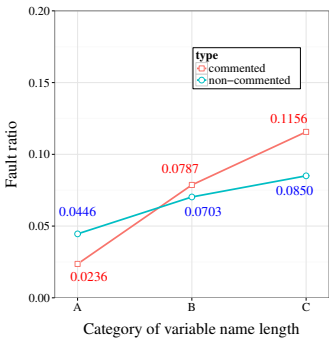

図 8 スコープ長が中程度の変数を含むメソッドでのフォールト潜在率

Fig. 8 Fault ratios of methods having moderate scope variables.

く 75% 点以下に相当している．分析結果を **図 8** に示す．

カテゴリ A ではコメント無しのメソッドの方がコメントありのメソッドよりもフォールト潜在率が高く（コメント無しではコメントありの約 1.9 倍），カテゴリ B ではほぼ同じになり，カテゴリ C では逆にコメントありのメソッドの方がフォールト潜在率が高い（コメントありではコメント無しの約 1.4 倍）という結果になった．即ち，スコープ長が特に長い場合の結果（図 7）と概ね同じ傾向となった．ただし，全体的なフォールト潜在率はスコープが特に長い場合に比べて低くなっていた．

（iii）スコープが特に短いローカル変数を持ったメソッドのフォールト潜在性

前述の（i）及び（ii）と同様の分析を“スコープが特に短いローカル変数を持ったメソッド”に対しても行った．ただし，ここでいう“特に短い”とは 表 4 での 25% 点以下に対応している．分析結果を **図 9** に示す．

コメント無しの場合（図 9 中の青線）は（i）及び（ii）の場合と同様の傾向であったが，コメントありの場合は（図 9 中の赤線）は先の二つとは逆の傾向となった．実際，カテゴリ A ではコメントありのメソッドの方がコメント無しのメソッドよりもフォールト潜在率が高く（コメントありではコメント無しの約 3.2 倍），カテゴリ B ではほぼ同じになり，カテゴリ C では逆にコメント無しのメソッドの方がフォールト潜在率が高い（コメント無しではコメントありの約 1.8 倍）という結果になった．つまり，スコープ長が 1 ～ 3 行と特に短く（表 4 参照），なおかつ変数名も 1 ～ 2 文字と短い（カテゴリ A）場合，そこにコメントが書かれているとフォールト潜在性は高いという結果になった．現時点では詳細な解析には至っていないが，変数名とスコープの両方が特に短い変数しか登場しないにもかかわらずコメントが書かれているということは，その部分に着目すべき特別な理由が存在していた可能性が考えられる．

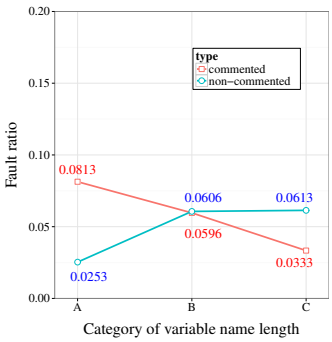

図 9 スコープの短い変数を含むメソッドでのフォールト潜在率

Fig. 9 Fault ratios ofmethods having shorter scope variables.

3.5 考察

一般にコメントの存在が全てのプログラムに対して品質的に負の意味を持つとは考えにくく，これまでの研究では，コメントが品質向上に役立っている場合と品質的に問題のあるプログラムを見つけ出すための目印になっている場合とを区別できずにデータ処理を行っていた可能性もある．今回の調査では，新たにローカル変数の名前とスコープの長さについても考慮してデータ収集と分析を行っており，それによってコメントの存在がフォールト潜在の“目印となっている場合”と“そうでない場合”とを切り分けることができているのではないかと考えられる．

以上の結果から，コメントの存在や行数，ローカル変数名の長さ及び変数のスコープ長はそれぞれフォールト潜在予測モデルを構築する上で有用な説明変数になる可能性を見てとれた．ただし，図 7 ～ 9 から分かるように，(1) コメントの有無，(2) 変数名の長さ，(3) スコープの長さ，という三つの要素は独立したかたちでフォールト潜在性予測に寄与するものではない．即ち，変数名の長さとスコープの長さの組合せによって場合分けを行わなければ，コメントの存在がフォールト潜在の疑わしさを強める場合と弱める場合とを分けることができず，結果として適切なフォールト潜在予測モデルを構築できなくなってしまう恐れがある（**表 5**）．例えば，代表的な予測モデルとしてロジステック回帰モデルが挙げられるが，三つの要素をそれぞれ説明変数とした場合，表 5 に見られるように変数名の長さとスコープの長さの組合せによってコメント有無の効果が異なっているため，事前に場合分けを行うか，あるいは変数間での相互作用を考慮したモデルにする必要がある．今回の調査結果を踏まえ，適切なフォールト潜在予測モデルについて検討していくことが今後の重要な課題である．

表 5 分析結果として得られた傾向

フォールト潜在性の高い方		変数名の長さ		
		特に短い	中程度	特に長い
スコープの長さ	特に短い	コメントあり	–	コメント無し
	中程度	コメント無し	–	コメントあり
	特に長い	コメント無し	–	コメントあり

3.6 妥当性への脅威

3.6.1 開発言語の違いが及ぼす影響

本論文での調査は，測定ツールの都合により Java で書かれたプログラムしか対象にできていない．他言語で書かれたプログラムでは異なった傾向が結果として得られる可能性は否定できないが，C や C++ といった主要な手続き型言語及びオブジェクト指向言語であれば，基本的なプログラムの書き方や変数の宣言，スコープの概念は Java と共通であり，大きな差異はないものと考えられる．

3.6.2 スコープ長の測定方法に関する懸念

本調査のために筆者らが開発したスコープ測定ツール [*7] は，実装の都合上，検出可能なスコープの情報がソースプログラム上での位置に限られており，スコープ範囲として空行やコメント行も計上されている場合がある．そのため，測定されるスコープの長さが実質よりも長いものとなってしまうことがあるが，メソッド内に多くの空行やコメント行が含まれているようなプログラムが多数登場しない限り，今回の調査結果に大きな影響はないものと考えられる．

3.6.3 変数名の構造や使われ方の違いが及ぼす影響

今回の分析ではローカル変数の名前の長さとして文字数にのみ着目しており，Lawrie ら [13] が行った分析のように変数名の構造，即ち単語の形式（省略形なのか非省略形なのか）は考慮できていない．分析におけるカテゴリ分け（A, B 及び C）では，実際の変数名を観察し，省略形または比較的短い単語がカテゴリ B に，それよりも長いものがカテゴリ C に分かれるよう工夫はしてあるが，それでも省略形・非省略形が多く混在していた可能性は否定できない．今後，キャメルケースやスネークケースといった命名規則，さらには自然言語処理技術も利用することで変数名が提供できる情報量に基づいたより精緻な分類を行っていく必要があると思われる．あわせて，変数の使われ方についても今回の分析では十分に考慮できておらず，スコープの長さによる分類に留まってしまっている．例えば，for 文の制御変数として使われているローカル変数ならば，多少スコープが長くなっていてもその用途は明確であると考えられ，そういった場合でのデータが他の場合のデータと混在して分析されてしまっている可能性は否定できない．

3.6.4 コーディング規約の影響

本調査では，ローカル変数の命名やコメントの記述はプログラマの裁量によるものであることを前提としている．しかしながら，分析対象の開発プロジェクトにおいてコーディング規約や命名規則といったプロジェクト内での実装ルールが定められていた場合，調査・分析の意義が失われてしまう危険性もある．筆者らが各プロジェクトの Web サイトを調べた範囲ではそういった実装ルールを見つけることはできなかったが，その種のルールが開発者間で定められていた可能性は否定できない．ただし，今回の分析では各プロジェクトからの無作為抽出を繰り返して行っているため，結果が特定のプロジェクトに大きく影響されている可能性は低いと考えられる．今後，コーディングスタイルや変数名の命名に関する個人差の分析も行うことで，この点に関する懸念を払拭していく必要があると思われる．これに関連して，コーディング規約違反に着目したフォールト潜在分析もいくつか報告されており [16], [17]，コーディング規約への違反状況を調べることで個人差の分析や本調査結果のさらなる考察も可能になることも考えられる．

3.6.5 自動生成コードの影響

今回収集した変数のスコープ長の中で最も長いものは 1313 行になっていた（表 3 参照）．プログラムの内容を詳細に追跡できているわけではないため断言は難しいが，そのような長いスコープの変数が登場した部分は，自動生成されたソースコードであった可能性がある．今回の分析ではそのようなプログラムが自動生成によるものであるかどうかを区別できておらず，人手によって書かれたプログラムと自動生成プログラムとが混在している可能性を否定できないという懸念がある．ただし，本論文ではスコープ長の外れ値は除外して分析を行っている（表 4 参照）ため，スコープが不自然に長く自動生成が疑われるようなプログラムが結果に影響を及ぼすことはないと考えられる．

4. おわりに

プログラミングにおいて，コメントの記述とローカル変数の命名はプログラマの裁量によるところが大きく，個人差が出やすい特徴であると考えられる．そのため，そのような特徴の違いが成果物の品質に影響を及ぼす可能性もある．これまで，プログラムにおけるコメント記述とフォールト潜在性の関係についてはいくつか実証研究が報告されており，コメントの多いプログラムの方がフォールト潜在性は高いという傾向が確認されている [3], [4], [5]．端的に言えば，コメントの存在はフォールト潜在の目印になるというものである．しかしながら，ローカル変数の名前やスコープの長さという観点については分析できていなかった．

そこで本論文では，コメントだけでなくローカル変数の名前の長さとスコープの長さについても考慮すべく，三つの著名なオープンソースソフトウェアを対象としてデータ収集を行った．そして，使われているローカル変数の名前の長さによって Java メソッドを三つのカテゴリに分類した：“長さ 1 ～ 2 文字（カテゴリ A）”，“長さ 3 ～ 5 文字（カテゴリ B）” 及び “長さ 6 ～ 19 文字（カテゴリ C）”．さらにカテゴリごとにスコープの長さにも注目し，スコープが （i）特に長い場合，（ii）中程度の場合，（iii）特に短い場合，にメソッドを分け，それぞれのメソッドにおけるフォールト潜在率を “コメント記述あり” のものと “コメント記述無し” のものに分けて算出した．

[*7] http://se.cite.ehime-u.ac.jp/tool/

結果として，スコープ長が特に長い場合や中程度の場合，カテゴリ A でコメント記述無しのメソッドでのフォールト潜在率はコメント記述ありのメソッドに比べて高いことが確認された．つまり，名前が 1 ～ 2 文字で構成されているローカル変数で，そのスコープ長が中程度以上であれば，コメントが書かれている方が望ましい（フォールト潜在性は低い）という傾向が確認された．これは，コメントとフォールト潜在性との関係に関する先行研究 [3], [4], [5] とは逆の傾向であった．一方，カテゴリ B ではコメント記述の有無でフォールト潜在率に差はほとんど見られず，カテゴリ C では逆にコメント記述ありメソッドの方がフォールト潜在率は高いという結果になった．カテゴリ B は 3 ～ 5 文字の変数，カテゴリ C は 6 文字以上の変数がそれぞれ使われているメソッドに相当し，カテゴリ A に比べて変数に関する補足説明の必要性は低いと考えられる．それでもなおコメントが書かれているということは，そのメソッドの内容について追加説明が必要な状況にあったと推察される．つまり，ソースコードの理解容易性に問題があり，結果的にフォールト潜在率が高かった可能性がある．このことは，これまでの実証研究の結果に符合するものであった．

次に，スコープが特に短い場合，他の二つとは逆の傾向が確認された．つまり，カテゴリ A でコメント記述ありの場合に最もフォールト潜在性が高く，変数名が長くなるにつれてコメント記述無しのメソッドの方がフォールト潜在性が高くなる傾向にあった．通常，スコープが短ければ変数名も短く単純なものでもよいと思われる [6] が，それでもなおコメントが書かれている場合，開発者にとってその部分に着目すべき特別な理由が存在していた可能性もある．

以上のように，変数名とスコープの長さを考慮することで “コメントがフォールト潜在の目印になる” という傾向にあるメソッドとその逆の傾向にあるメソッドを切り分けることができた．これまでの研究ではコメントの有無や行数によって画一的にフォールト潜在性を分析していたため，相異なる傾向を持った二種類のメソッドが混在したまま扱われていたことが推察される．つまり，変数名とスコープの長さを考慮した分類を行うことで，コメントの功罪に関するより詳細な分析とフォールト潜在予測の精度向上が可能になると期待される．今回は定量的な調査と傾向の分析に留まっており，個別のフォールト案件に対する定性的な考察や実際にどのようにしてフォールト検出に活用していくかという観点での検討には至っていない．今後，さらに多くのデータを収集・分析して結果の一般性を高めていくとともに，データの定性的な分析や予測モデルの構築・評価も行っていく予定である．また，コメント文の内容や変数名そのものの分析も重要な今後の課題となっている．例えば，省略形・非省略形といった変数名の形式の違いに着目した分析やコメント文に対する自然言語処理を通じた分析を行うことで，新たな知見が得られることも期待される．

謝辞 本論文の初版について，有益な助言を下さった査読者の皆様に感謝致します．本研究は JSPS 科研費 25330083（基盤研究 （C））の助成を受けたものです．

参考文献

[1] 野中 誠，水野 修：fault-prone モジュール予測技法の基礎と研究動向，ソフトウェアエンジニアリングシンポジウム 2010 チュートリアル資料，Vol. 2010, No. 2, 情報処理学会シンポジウムシリーズ (2010).

[2] 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 (2012).

[3] 阿萬裕久：オープンソースソフトウェアにおけるコメント記述およびコメントアウトとフォールト潜在との関係に関する定量分析，情報処理学会論文誌，Vol. 53, No. 2, pp. 612–621 (2012).

[4] Aman, H.: An Empirical Analysis on Fault-proneness of Well-Commented Modules, *Proc. 4th Int'l Workshop on Empirical Softw. Eng. in Practice*, pp. 3–9 (2012).

[5] Aman, H., Amasaki, S., Sasaki, T. and Kawahara, M.: Empirical Analysis of Fault-proneness in Methods by Focusing on their Comment Lines, *Proc. 21st Asia-Pacific Softw. Eng. Conf., vol.2*, Vol. 2, pp. 51–56 (2014).

[6] Kernighan, B. W. and Pike, R.: *The practice of programming*, Addison-Wesley Longman, Boston, MA (1999).

[7] Sousa, M. J. and Moreira, H.: A Survey on the Software Maintenance Process, *Proc. Inte'l Conf. Softw. Maintenance*, pp. 265–274 (1998).

[8] Tenny, T.: Program Readability: Procedures Versus Comments, *IEEE Trans. Softw. Eng.*, Vol. 14, No. 9, pp. 1271–1279 (1988).

[9] Woodfield, S. N., Dunsmore, H. E. and Shen, V. Y.: The Effect of Modularization and Comments on Program Comprehension, *Proc. 5th Int'l Conf. Softw. Eng.*, pp. 215–223 (1981).

[10] Fowler, M.: *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Longman, Boston, MA (1999).

[11] Steidl, D., Hummel, B. and Juergens, E.: Quality analysis of source code comments, *Proc. 21st Int'l Conf. Program Comprehension*, pp. 83–92 (2013).

[12] 森崎雅稔：最新 Java コーディング作法，日経 BP 社，東京 (2011).

[13] Lawrie, D., Morrell, C., Feild, H. and Binkley, D.: What's in a name? a study of identifiers, *Proc. 14th Int'l Conf. Program Comprehension*, pp. 3–12 (2006).

[14] 縣 俊貴：良いコードを書く技術，技術評論社，東京 (2011).

[15] Kawamoto, K. and Mizuno, O.: Predicting Fault-prone Modules Using the Length of Identifiers, *Proc. 4th Int'l Workshop on Empirical Softw. Eng. in Practice*, pp. 30–34 (2012).

[16] Boogerd, C. and Moonen, L.: Assessing the value of coding standards: An empirical study, *Proc. 24th Int'l Conf. Softw. Maintenance*, (2012).

[17] Mizuno, O. and Nakai, M.: Can Faulty Modules be Predicted by Warning Messages of Static Code Analyzer?, *Advances in Softw. Eng.*, vol.2012, no.924923 (2012).