Vol.2016-SE-194 No.6
2016/11/18

ソースコードの変更予測に向けた メソッド名の先頭単語に関する定量的調査

鈴木　翔[1,a)]　阿萬　裕久[2]　川原　稔[2]

概要：本稿では，Javaプログラムにおけるメソッド名の先頭単語に着目し，ソースコード品質との関係について定量的な調査を行っている．メソッドに付ける名前は動詞から始めるのが一般的であり，なおかつ，その動詞がそのメソッドの振舞いを表す上で重要な働きを持つと考えられる．そこで，五つの著名なオープンソースソフトウェアを対象としたデータ収集と分析を行い，Javaプログラミングで先頭単語として用いられる主要な英単語を把握し，またそこから予想される意味とソースコードの安定性についての分析を行っている．

A Quantitative Investigation of Method Name's First Word toward Source Code Change Prediction

Sho Suzuki[1,a)]　Hirohisa Aman[2]　Minoru Kawahara[2]

1. はじめに

一般に，ソフトウェアの保守においては，ソースコードの理解と変更が必要不可欠なものになってくる．それゆえ，ソースコードは可能な限り理解しやすいものに仕上げるべきであり，コード変更は新たなバグの作り込みを避けるためにも必要最小限に留めることが望ましい．そのためには，そのようなソースコードを作り上げるためのガイドラインを用意したり，ソースコードがそのようなものになっているか評価したりすることが重要となる．その種のガイドラインや評価基準としては，コーディング規約や各種メトリクスが考えられる．例えば，メソッドの命名に関する規則や，プログラムの構造を評価するメトリクスなどがその一例として挙げられる．

ガイドラインや評価基準を考える上で，開発者の経験や好みによる影響，すなわち個人差の影響を考慮する必要もあると筆者らは考える．なぜなら，プログラムの書き方には多少なりとも個人差があり，同じ仕様のプログラムを実装するとしても，開発者が異なれば出来上がるプログラムも同一になるとは限らないからである．そのため我々は，開発者間で個人差の影響しやすい要素に着目した研究を行ってきている [1], [2], [3]．文献 [3] では，メソッドの名前に着目し，その構成がバグの潜在性や変更の起こりやすさに関係していることを報告した．具体的には名前の構造（lowerCamelCase に従っているかなど）が一般的ではない場合，バグの潜在性が高くなっていたり，変更が起こりやすくなっていたりする傾向が確認された．本稿では，メソッド名に対する更なる調査及び分析として，メソッド名に使われる単語に注目する．メソッド名は動詞から開始することが一般的であり，その動詞がそのメソッドの働きを表現する上で，最も重要な要素であると考えられる．つまり，どういった動詞が実際に使われているか，更にはその違いが変更の起こりやすさなどと関係しているかどうかについて定量的な調査を行い，その結果について報告する．

以下，第 2 章では，関連研究とソースコードにおける識別子の重要性について述べる．そして，第 3 章は調査・分析の目的，手順等を述べ，第 5 章及び第 5 章で結果及び考

[1] 愛媛大学大学院理工学研究科
Graduate School of Science and Engineering, Ehime University, Matsuyama, Ehime 790–8577, Japan
[2] 愛媛大学総合情報メディアセンター
Center for Information Technology, Ehime University, Matsuyama, Ehime 790–8577, Japan
[a)] s_suzuki@se.cite.ehime-u.ac.jp

察をそれぞれ述べる．更に，第 6 章では妥当性への脅威について述べる．最後に第 7 章で本稿のまとめと今後の課題について述べる．

2. ソースコードにおける識別子

2.1 識別子とソースコード理解

ソフトウェアはリリース後も品質の維持や不具合の修正などのために保守作業が行われる．その作業工数の多くはソースコード理解に費やされることが知られており [4]，保守作業のコストを下げる有効な手段として，理解しやすいソースコードを書くことが挙げられる．そこで，ソースコード理解に関する研究がいくつか報告されている．Lawrie ら [5] はローカル変数名について，フルスペルや省略形といった表記法の違いがソースコードの理解容易性に影響することを報告している．また，本稿の先行研究 [3] において，コーディング規約に従っていない名前を持つメソッドや文字数・単語数の多い名前を持つメソッドは，ソースコードの修正率やバグ修正率が，そうでないものに比べて高い傾向にあることが確認されている．他にも，Høst ら [6] はメソッド名とその実装の特徴に着目し，メソッド名と実装内容の間に様々な関連性（パターン）があることを報告している．例えば，次の四つの特徴を有しているメソッドの場合は，何らかの探索を行っているものと考え，“find” で始まるメソッド名であるべきといったことを論じている：(1) オブジェクトへの参照を返す；(2) 仮引数を持つ；(3) ループ処理を持つ；(4) 複数のリターンポイントを持つ．また，名前とそのパターンにマッチしない場合は，そのメソッドは名前もしくは実装が適切ではないものと指摘しており，そのようなものを “naming bug” と呼んでいる．Kashiwabara ら [7] は，メソッドの名前に関する別の研究として，実装の特徴からメソッドの命名支援を行う試みを行っている．このように，識別子の良し悪しはソースコード理解において大きな関心事であり，保守性に影響を及ぼす重要なものの一つであると言える．

2.2 識別子とメソッド名

識別子はクラス，メソッド，変数などプログラムを構成する様々な要素に付けられる名前である．本稿はその中でも特にプログラムの動作に影響すると思われるメソッド名に着目する．Høst らはメソッド名について大きく，（1）unique labels，（2）mnemonic，（3）semantic role という三つの役割を指摘している [8]．この中でも特に，プログラム理解には振舞いを表現する（3）の役割が重要となる．一般的なプログラミング言語では，使用可能な文字や記法の制限の中では，開発者が自由に識別子を決定することができる．しかし，一般的にはソースコードの可読性を考慮すれば，意味があり具体的で誤解を招きにくい語句を用いた識別子が好ましいといわれている [9]．そして，実装を精読せずともその振舞いを容易に想像できる名前が理想的である．このようなことから，Java 言語仕様 [10] ではメソッド名が動詞あるいは動詞句であることが推奨されている．例えば，“getXXX”（XXX は任意の文字列）のような名前を持つメソッドがあったとすると，ソースコードの読み手は，そのメソッドが “XXX” についてのデータを返すものと考える．同様に，“isYYY” のような名前に対しては，条件 “YYY” を満たすかどうかを論理値で返すメソッドであると想像すると思われる．

更には，前小節 2.1 で述べたように，メソッドの名前の特徴には実装面にもある程度一貫したパターンがあることが報告されている [6]．このように名前が実装と関係を持つのであれば，メソッド名はソースコード品質を評価する面においても注目すべき要素の一つとなり得る．そこで，本稿はソースコード品質の分析にメソッド名を用いることが可能かどうか検討することを目的としている．

2.3 Java におけるコーディング規約と識別子

上述したように，一般的なプログラミング言語では識別子は自由に決定可能である．しかしながら，実際には開発グループなどによって命名に制限を課す場合もある．そのような場合，プログラミングの記法に統一性を持たせて，ソースコードの可読性を一定水準に保つために，コーディング規約を指定して開発を行うことが多い．Java のコーディング規約の例としては，Java コーディング標準（オブジェクト倶楽部）[11] や Google Java Style[12] などがある．そこでは識別子の命名規則についても言及されており，**表 1** に示す三つの形式が一般に知られている．

UpperCamelCase はクラスやインターフェースに用いられ，識別子を構成する全ての単語の頭文字を大文字で表記する形式である．lowerCamelCase は一つ目以外の単語の頭文字を大文字で表記する形式であり，メソッドや変数に用いられる．snake_case はアンダースコアで単語を区切る記法であり，定数に用いられる．ただし定数の定義はこれに加え，全てを大文字で綴ることも推奨されている．

識別子に対するコーディング規約の推奨事項は記法だけではない．例えば，次のような規則によって命名の統一化が図られている：

- 識別子を構成する単語は英単語を基本とすること．
- スコープによっては単語の省略形や意味のない単語を使わないようにすること．

表 1 Java 言語における命名規則
Table 1 Naming conventions for Java.

命名規則	特徴
UpperCamelCase	全ての単語の区切りが大文字
lowerCamelCase	先頭以外の全ての単語の区切りが大文字
snake_case	各単語をアンダースコアで区切る

- 機能的に対を成すメソッドの名前には set と get や insert と delete のような対照的な単語を用いること．

このような規則によって，メソッドの命名は，より制限された中で一定の規則に従って行われている可能性が高い．

3. 調査研究

3.1 目的と対象

本調査では Java メソッドの名前と開発履歴をもとに，メソッドの名前と，実装部のソースコード品質の関係について分析する．大きくは以下の内容について調査を行う．

RQ1: どのようなメソッド名が主流であるか？

RQ2: メソッド名の違いと，変更の起こりやすさ及びその規模は関係があるか？

具体的には個々のメソッドのリポジトリ上での変更を追跡し，変更やバグ修正との関連性について見ていく．分析対象については五つのオープンソースソフトウェアを用いた（**表 2**）．これらを対象とした理由は，以下の条件を全て満たしているからである：(1) Java で開発されている；(2) Git を用いてソースコードの管理が行われている；(3) SourceForge[*1]における Status が “5-Production/Stable” である．条件 (1) 及び (2) は，使用するツールの制約によるものである．条件 (3) は，本調査での結果の一般性を高めるため，著名でかつ安定的に開発されているプロジェクトを対象とすべきと考えたからである．

3.2 データ収集

本調査は Java メソッドを分析の対象とする．データ分析の手順を以下に示し，その流れを**図 1** に示す．

(1) リポジトリのクローンを取得
インターネット上に公開されている Git リポジトリをローカル環境にコピーする[*2]．

(2) Git リポジトリを Historage に変換
(1) で得たリポジトリを，プログラム要素単位の Git リポジトリ（Historage）に変換する．変換には専用のツール Kenja[*3]を使用した．

表 2 調査対象ソフトウェア
Table 2 Open source software products surveyed in this paper.

ソフトウェア名	コミット数	調査期間
Eclipse Checkstyle Plug-in	984	2003-05 ～ 2016-07
Hibernate-orm	3781	2007-06 ～ 2012-03
FreeMind	1062	2011-02 ～ 2016-08
Dr Java	3466	2001-06 ～ 2014-10
PMD	8736	2002-06 ～ 2016-08

[*1] https://sourceforge.net
[*2] Hibernate-orm のみ http://kataribe.naist.jp/public より historage リポジトリを直接取得
[*3] https://github.com/niyaton/kenja

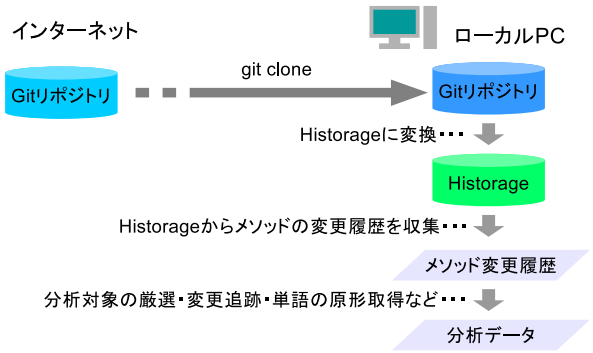

図 1 リポジトリごとの分析データ収集手順
Fig. 1 Procedure of data collection.

(3) Historage からメソッドの変更履歴を抽出
(2) で得た Historage から Git の各種コマンドを使用し，メソッドに対する変更履歴を抽出する．

(4) メソッド名からの先頭単語の抽出及びその原形の取得
本調査は，lowerCamelCase に従った名前を持つメソッドを対象としている．次の正規表現によってメソッド名の分割を行い先頭単語を取り出す．

- `(?<=[a-z])(?=[A-Z])`
- `(?<=[A-Z])(?=[A-Z][a-z])`
- `(?<=\D)(?=\d)|(?<=\d)(?=\D)`

単語の原形の取得には WordNet[13] を使用した．

3.3 Historage の構造とメソッドの変更追跡

Historage は**図 2** のようなディレクトリ構造を持つ．Git リポジトリを Historage に変換した場合，Historage のルート直下には全てのソースファイルのパスを名前に持つディレクトリが配置される．そして，そのディレクトリの直下にはクラスの情報をまとめるディレクトリ “[CN]” が存在する．“[CN]” 直下には，当該ファイルに記述されたクラスの名前を持つディレクトリが置かれている．このディレクトリの下には，当該クラスに定義されている各プログラム要素をまとめる “[CN]”，“[CS]”，“[FE]”，“[MT]” といったディレクトリが存在する．メソッドに関する情報は “[MT]” ディレクトリ以下に配置される．“[MT]” 直下にある body ファイルはメソッドの実装内容に関するファイル，parameters ファイルはメソッドの引数に関するファイルである．メソッドの変更追跡は，この body ファイルに対して行った．追跡は，100%の類似度に基づいたリネームも考慮に入れた．これにより，開発途中でプロジェクトのディレクトリ構造が変化する，所属するクラスが変化する，あるいはメソッドの宣言部分のみが変化するといった事象は全て，同一のメソッドに対する変更として扱われる．ただし，以下に示すようなメソッドは本調査の対象外とした．

- コンストラクタ
- 匿名クラスに属するメソッド

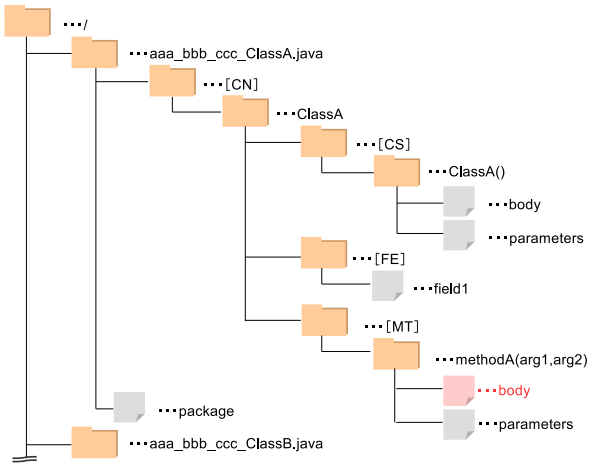

図 2 Historage のディレクトリ構造
Fig. 2 Structure of directories in Historage.

- 無名メソッド
- 命名が lowerCamelCase に従っていないメソッド
- 実装を持たないメソッド
- jUnit のテスト用ディレクトリに属するメソッド
- 名前の文字数が 1 文字のメソッド
- 180 日間（約半年間）以上存在しなかったメソッド

3.4 評価方法

ここでは，前小節 3.1 で述べた調査項目の評価方法について説明する．本稿では次の二つの評価基準を用いる．

- 変更行数
- 変更率

まず，変更行数とは，各メソッドにおいて追加された行と削除された行の数であり，Git の`--numstat` で収集することができる．

次に，変更率とは，与えられたメソッド集合において，変更の起こったメソッドの割合である．後述するように，本分析では各メソッドをその名前によってグループ分けし，変更率によってグループ間の比較を行う．

4. 結果

本調査は五つのプロジェクトから前述の条件を満たす全メソッドを抽出したのち，それぞれのプロジェクトのメソッド集合から 2,500 個ずつ無作為抽出して 12,500 個のデータ集合を作成して行った（**図 3**）．

本調査で抽出したデータ集合の中で，分析の対象としたメソッドを**表 3** に示す．これはメソッドの先頭単語に着目して出現数を計上し，上位 50 位までの計 51 個の単語に関するデータである．

図 4 は本調査で着目した 51 個のメソッドの初版の行数の箱ひげ図である．各単語は左からメソッド行数の第三四分位数の昇順に並んでいる．**図 5** 及び**図 6** は，それぞれ

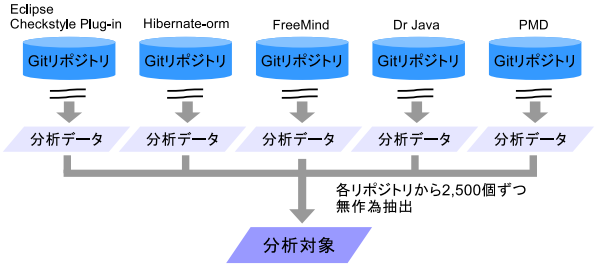

図 3 本調査における分析対象収集手順
Fig. 3 Procedure of sampling in our survey.

表 3 頻出単語を先頭単語とするメソッドの変更率及び変更行数（第三四分位数）
Table 3 Change rates and number of changed lines (the third quartile) in methods having major first words.

単語	メソッド数	行数	変更率	変更行数
get	3,205	6	0.4624	2
set	967	5	0.4767	2
be	634	3	0.5047	2
visit	539	8	0.7161	7
create	442	25	0.5249	14.25
add	371	9	0.5660	4
to	179	6	0.6257	2
remove	136	9	0.4338	8.5
build	120	18	0.7417	10
handle	106	25.25	0.4057	21.5
have	99	6	0.6061	2
update	95	17.5	0.5263	17
run	94	21	0.4574	10
check	91	10	0.5165	8.5
write	85	23	0.5294	22
save	76	19	0.4737	12
initialize	75	19.5	0.5867	16.25
make	67	15.5	0.3284	51.75
equal	65	11	0.5538	4
value	65	9	0.5231	9.75
resolve	62	15	0.6129	9.75
apply	60	14	0.5833	14.5
hash	60	5.25	0.6667	5
load	59	26.5	0.5593	20
find	57	23	0.6316	16.5
process	52	24	0.6538	32
accept	51	8.5	0.4314	2.75
init	51	20	0.7255	12
new	49	18	0.5714	9
parse	49	18	0.6327	8.5
show	47	11.5	0.5532	13.75
contain	45	3	0.3778	2
select	45	15	0.5333	5
compare	44	12.25	0.4091	3
register	44	6	0.6129	3.75
reset	42	7	0.5952	10
clear	40	6	0.4750	6
configure	40	14	0.5250	14
do	40	27.25	0.6750	25.5
on	40	13	0.4250	10
generate	37	15	0.7027	34.75
mouse	37	16	0.2432	23
start	37	23	0.5676	34
ok	36	13	0.3056	24
transform	36	16.25	0.7500	10
clone	34	13.75	0.3529	12.5
perform	34	20.5	0.6765	17.5
insert	33	12	0.4848	15.5
open	33	16	0.6061	17.75
read	33	16	0.6364	10
size	33	3	0.3636	3

メソッド集合（先頭単語による）での変更率の棒グラフ及び変更行数の箱ひげ図である．単語の並び順は図 4 と同じである．

5. 考察

5.1 RQ1: どのようなメソッド名が主流であるか？

Java 言語によるプログラミングでは，メソッド名は動詞句であることが推奨されているが，具体的にどのような動詞を用いるかについては，getter などの一部を除きルールは存在していない．そこで実際のソフトウェアではどのような単語が好んで使用されるか調査を行った．本調査で作

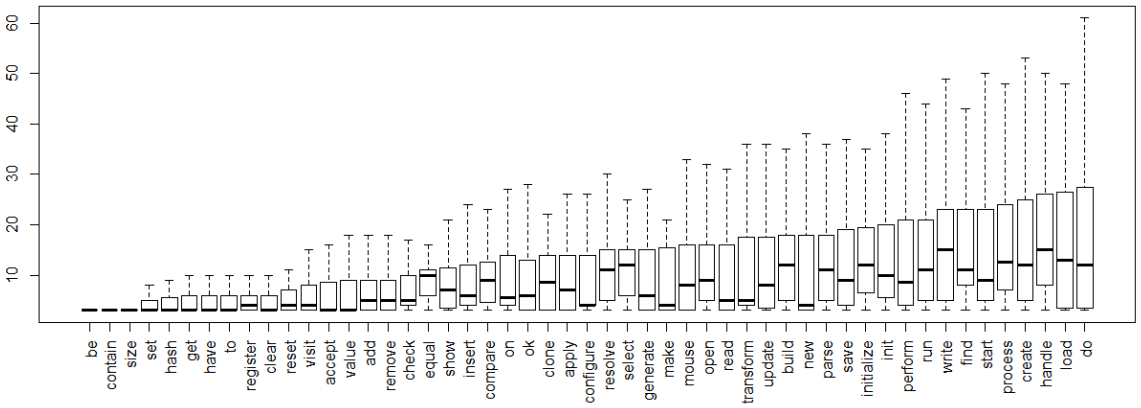

図 4 先頭単語別に見たメソッドの行数（第三四分位数の昇順）

Fig. 4 LOCs of methods grouped by their first words (increasing order of the third quartile).

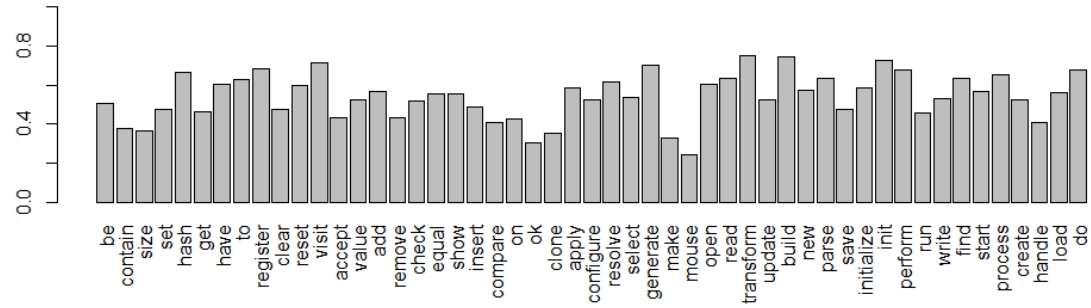

図 5 先頭単語別に見たメソッドの変更率

Fig. 5 Change rates of methods grouped by their first words.

成した 12,500 個のメソッド集合では 734 種の先頭単語が見られたが，表 3 に示す上位 50 位 51 種の単語のみで全体の 7 割に及ぶ．つまり，メソッドの先頭単語の種類は多数に及ぶが，一般的には限られた語彙の中からメソッド名の先頭単語が使われているということである．

最も先頭単語として多く出現した単語は “get”，次に多く出現した単語は “set” であり，それぞれ 3,205 個と 967 個であった．Java 言語仕様やコーディング規約においては，オブジェクトの属性値を取得・設定するためのメソッドは，“get-*”，“set-*” の様な名前を付けることが推奨されており，プログラミングで馴染み深い振舞いが実際に結果に現れていると言える．同様に，条件の判定を行うメソッドは “is-*”，オブジェクトを別の形式に変換するメソッドは “to-*” とするような命名も言語仕様上推奨されており，これらの特徴を持つメソッドも上位に出現している．Java の標準ライブラリのメソッドに対して同様のデータ収集を行ったところ，当該ライブラリの上位 50 位までの単語は調査対象のオープンソースソフトウェアで収集した単語の上位 50 位までに 60% 出現しており，Java の標準ライブラリでよく利用される単語が，オープンソースソフトウェアのソースコードでもおおむね利用されていることが分かる．一方，“build” や “hundle” などのように，調査対象のオープンソースソフトウェアでは多く見られたものの Java 標準ライブラリの上位には見られない単語も存在した．これらについては，まず標準ライブラリと調査対象のオープンソースソフトウェア（アプリケーション）の立場の違いが理由として考えられる．標準ライブラリは，様々なプログラムから利用されることが想定されているが，アプリケーションはそのような想定になっていない．つまり，その状況に特化された一連の操作を実行するといった意味の単語（“build” や “hundle” など）は，Java 標準ライブラリには多くは登場しなかったと思われる．また，標準ライブラリはドメインの観点からも幅が広いことが考えられ，“paint”，“decode” のような特定のドメインに限定されることが予想される単語は反対にアプリケーションでは多くは出現しなかったのではないかと考えられる．

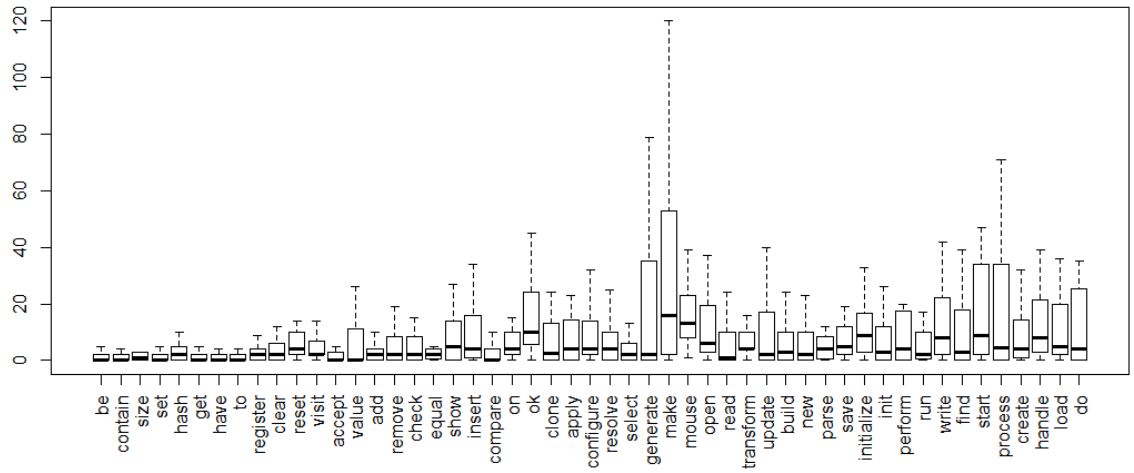

図 6 先頭単語別に見たメソッドの変更行数
Fig. 6 Number of changed lines in the methods grouped by their first words.

5.2 RQ2: メソッド名の違いと，変更の起こりやすさ及びその規模は関係があるか？

先頭単語でグループ分けしたメソッド集合について，各グループにおける変更率及び変更行数の傾向を掴むため，スピアマンの順位相関係数を求めた結果を**表 4** に示す．順位相関係数は，メソッドの初版の行数，変更率及び変更行数に関する 3 通りの組合せに対して，初版行数と変更行数の中央値を用いたものと第三四分位数を用いたものの 2 通りの計 6 通りの相関係数を計算してある．

一般に，ソースコードの変更の起こりやすさは，その行数と強く関連していることが知られている．つまり，長いプログラムほど変更も起こりやすいという傾向がある．したがって，メソッドの初版行数が多くなれば，そのメソッドの変更の起こりやすさも高くなることが予想される．これら 2 要素間の順位相関係数は，中央値に着目したもので 0.1866，第三四分位数に着目したもので 0.2442 と，ほとんど相関は見られなかった（表 4（A））．したがって，メソッドの変更の起こりやすさは初版の行数だけではなく，メソッドの名前ごとに着目することについても関係があることが考えられる．

初版行数と変更行数については中央値，第三四分位数のいずれに着目した場合も 0.5748，0.7608 と正の相関関係が見られた（表 4（B））．変更が起こったメソッドでの変更行数は，そのメソッドの初版行数の影響を強く受けるようである．しかし，変更の起こりやすさと変更の行数には相関性は見られず（表 4（C）），変更が起こりやすいメソッドが必ずしも変更の規模が大きくなるわけではないということが伺えた．

次に，先頭単語でグループ分けした各メソッド集合における変更行数について見ていく．図 6 の全データをメソッドの初版行数（第三四分位数）によって三つのグループに分けた：(1) 10 行未満（**図 7**），(2) 10 行以上 20 行未満（**図 8**）及び (3) 20 行以上（**図 9**）．

前小節で述べた，Java 言語で馴染みのある “get”，“set”，“be”，“have” のような先頭単語を持つメソッドは，初版のメソッド行数が少なく，変更が起こった場合の行数も少な

表 4 初版行数，変更率，及び変更行数間の順位相関係数
Table 4 Rank correlation coefficients among LOC, change rate and number of changed line.

組合せ	中央値	第三四分位数
(A) 初版行数と変更率	0.1866	0.2442
(B) 初版行数と変更行数	0.5748	0.7608
(C) 変更率と変更行数	0.0507	0.0823

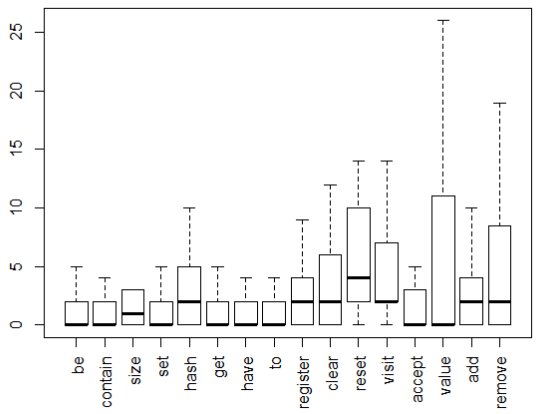

図 7 変更行数（初版行数の第三四分位数が 10 未満）
Fig. 7 Number of changed lines (the third quartiles of LOCs are less than 10).

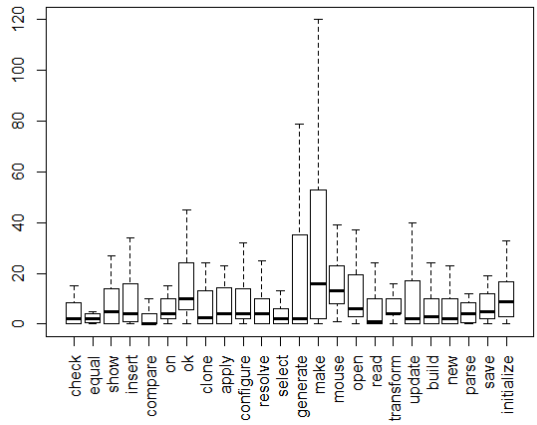

図 8 変更行数（初版行数の第三四分位数が 20 未満）
Fig. 8 Number of changed lines (the third quartiles of LOCs are less than 20).

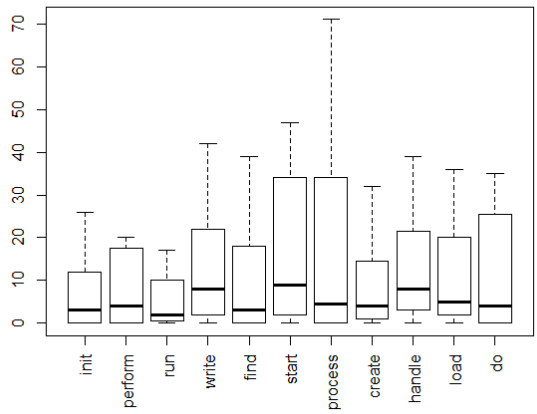

図 9 変更行数（初版行数の第三四分位数が 20 以上）
Fig. 9 Number of changed lines (the third quartiles of LOCs are equal to or greater than 20).

いことが見て取れる．これらの名前を持つメソッドは振舞いが自明であることが多く，結果として初版行数及び変更行数が少なかったと考えられる．

一方，‘make” や “generate” のようなオブジェクトの生成を行うことが予想されるメソッドや，“start”，“process” のような複数の処理が集約されていることが予想されるメソッドに関しては変更が起こった場合の変更行数が分析対象の中で比較的大きい傾向であることが分かった（図 8, 図 9）．前者のように，オブジェクトの生成が大きく関係している場合，型の確認や例外の処理など，そこで実装される処理が複雑化している可能性がある．また，後者のように，複数の処理が集約されていた場合，当該メソッドの外に修正があった場合であっても，その影響を間接的に受けやすくなることが考えられる．

特に，変更の規模について，“make” で名前が始まっているメソッドでは特異な性質が見られた．そのようなメソッドは，上述のようにオブジェクトを新たに生成するような振舞いが連想でき，それに伴う処理の複雑さなどが変更規模を拡大させた要因として考えられる．また “make” という単語は英単語としての特徴においても，他の 50 単語と比較して大きな違いが見られた : WordNet により，本調査の対象である 51 種の単語それぞれについて，（動詞としての）意味の個数を調べたところ，平均は 9.3 個であったのに対し，“make” の持つ意味の数は最も多く 49 個であった（**表 5**）．つまり，“make” は調査対象とした単語の中でも多様な意味を読み手に与えるということである．そのような “曖昧さ” が，他のメソッドよりも大きな修正を引き起こしやすい要因の一つであったと推察される．更には，目的が具体的に決まっていないメソッドに対しても当てはまりやすく，後に修正が起こりやすい，作り込みの甘いメソッドになってしまっていることも考えられる．

ただし，本調査はあくまでメソッド名の先頭単語のみに着目しており，それに続く単語や，引数の型と名前，戻り値の型などといった要素もまた，ソースコードレビューの際に重要な要素となり得ることが予想される．これらの点については今後更なる調査が必要と考える．

6. 妥当性への脅威

本調査におけるメソッド変更履歴の追跡精度は Historage の精度に依存している．ただし，Historage については文献 [14] において有用性が示されていることから，本調査における追跡精度についても大きな問題はないと考えられる．

今回の調査は Java 言語で開発されたオープンソースソフトウェアに限定しているため，他の言語で開発されたソフトウェアでは，各単語の出現数や修正に関して異なった傾向が現れる可能性もある．オブジェクト指向型言語であれば，Java と同様の概念の下にメソッドの命名が行われていることが期待されるが，ここでは断言は難しい．言語の違いが及ぼす影響については，今後の課題として検討していきたい．

表 5 単語の持つ意味の数（一部抜粋）
Table 5 Number of sences which the word has.

順位	単語	意味の数
1	make	49
2	run	41
3	get	36
4	check	25
5	set	24
5	clear	24
7	have	19
⋮	⋮	⋮

調査対象のプロジェクトは，著名でかつ安定的に開発されている Java ソフトウェアを対象としたが，ドメインについては考慮できていない．メソッドの名前は，そのソフトウェアのドメインと関係性が深いことが考えられるため，ドメインの違いが結果の違いをもたらすことも懸念される．今回はその違いを分析するには至っておらず，今後の課題としたい．

データ収集では，テストコードを対象外とするため，テスト用と思われるディレクトリの判別を行った．手法としては，ソースパス名に基づいた簡易的なパターンマッチであったため，厳密とは言い難いが，対象外とすべきデータの除外を可能な範囲で試みた上で実験を実施している．

7. おわりに

本稿では，Java メソッド名の先頭単語に着目し，著名な五つのオープンソースソフトウェアを対象としたデータ収集と分析を行った．

まず，先頭単語にのみ限定した場合であっても多くの単語が見られたが，プログラミングを行う上で主にメソッド名の先頭に利用される単語は，多くが限定的な範囲から選択されたものであった．また，Java の言語仕様やコーディング規約に見られる “get-*” などの名前を持つメソッドはオープンソースソフトウェアにおいても多く登場し，Java 標準ライブラリと概ね同様の命名が行われていることを確認できた．

メソッドの変更率と修正行数については，先頭単語により違いが見られた．単語ごとに分けた状態で分析したところ，次の傾向が見て取れた．

- “get”，“set”，“be”，“have” のような先頭単語を持つメソッドは，初版のメソッド行数が少なく，変更が起こった場合の行数も少ない傾向にあった．
- “make” や “generate” のようなオブジェクトの生成を行うことが予想されるメソッドや，“start”，“process” のような複数の処理が集約されていることが予想されるメソッドに関しては変更が起こった場合の変更行数が比較的大きい傾向にあった．

また，名前が “make” で始まるメソッドは，他のメソッドに比べて変更の行数が特に多くなる場合があった．これは，読み手に対して複雑なオブジェクト生成を行っている印象を与えるだけでなく，英単語そのものの特徴として，他の意味を持つことも否定できず，比較的 “曖昧さ” が高いことがその一因として考えられる．

今後の課題として，先頭単語だけでなく，仮引数の型や名前，戻り値の型などのような要素についても調査及び分析を行う必要があると考えている．

謝辞 本研究は JSPS 科研費（16K00099）の助成を受けたものです．

参考文献

[1] Aman, H., Amasaki, S., Sasaki, T. and Kawahara, M.: Empirical Analysis of Change-Proneness in Methods Having Local Variables with Long Names and Comments, *Proc. 9th Int'l Symp. Empir. Softw. Eng. & Measurement*, pp. 50–53 (2015).

[2] Aman, H., Amasaki, S., Sasaki, T. and Kawahara, M.: Lines of Comments as a Noteworthy Metric for Analyzing Fault-Proneness in Methods, *IEICE Trans. Inf. & Syst.*, Vol. E98-D, No. 12, pp. 2218–2228 (2015).

[3] 鈴木　翔，阿萬裕久，川原　稔：メソッド名の長さと構成に着目したソースコード品質に関する定量的調査，電子情報通信学会技術報告，Vol. 116, No. 128, pp. 137–142 (2016).

[4] Murphy, G. C., Kersten, M., Robillard, M. P. and Čubranić, D.: The Emergent Structure of Development Tasks, *ECOOP 2005 - Object-Oriented Programming* (Black, A. P., ed.), Lecture Notes in Computer Science, Vol. 3586, Springer Berlin Heidelberg, pp. 33–48 (2005).

[5] 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).

[6] Høst, E. W. and Østvold, B. M.: Debugging Method Names, *ECOOP 2009 Object-Oriented Programming* (Drossopoulou, S., ed.), Lecture Notes in Computer Science, Vol. 5653, Springer Berlin Heidelberg, pp. 294–317 (2009).

[7] Kashiwabara, Y., Ishio, T., Hata, H. and Inoue, K.: Method Verb Recommendation Using Association Rule Mining in a Set of Existing Projects, *IEICE Trans. Inf. & Syst.*, Vol. E98-D, No. 3, pp. 627–636 (2015).

[8] Høst, E. W. and Østvold, B. M.: The Java Programmer' s Phrase Book, *Software Language Engineering* (Gašević, D., Lämmel, R. and Wyk, E. V., eds.), Lecture Notes in Computer Science, Vol. 5452, Springer Berlin Heidelberg, pp. 322–341 (2009).

[9] Boswell, D.: リーダブルコード，オライリージャパン，東京 (2012).

[10] Oracle: The Java Language Specification, Java SE 8 Edition, http://docs.oracle.com/javase/specs/jls/se8/html/ (2015).

[11] 平鍋健児：Java コーディング標準, http://objectclub.jp/community/codingstandard/CodingStd.pdf (2006).

[12] Google: Google Java Style, https://google.github.io/styleguide/javaguide.html (2014).

[13] Miller, G. A.: WordNet: A Lexical Database for English, *Communications of the ACM*, Vol. 38, No. 11, pp. 39–41 (1995).

[14] Hata, H., Mizuno, O. and Kikuno, T.: Historage: Fine-grained Version Control System for Java, *Proc. 12th Int'l Workshop Principles of Softw. Evolution & 7th ERCIM Workshop Softw. Evolution*, pp. 96–100 (2011).