社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

ソフトウェアバグと変数名の関係の分析

山本　博之† 　亀井　靖高† 　柗本　真佑† 　門田　暁人† 　松本　健一†

† 奈良先端科学技術大学院大学情報科学研究科　〒 630-0192 奈良県生駒市高山町 8916–5
E-mail: †{hiroyuki-y,yasuta-k,shinsuke-m,akito-m,matumoto}@is.naist.jp

あらまし　あるソフトウェアにおける，特定の変数名がバグと関わりがあることが分かれば，モジュールに含まれる変数名からバグの有無を予測し，テスト工程を効率化することができる．そこで，本稿ではモジュールに含まれる変数名に着目し，あるソフトウェアにおけるモジュール中の変数名とバグの関係を明らかにすることを目的とする．そのために，2 つの仮説（H1）変数名とバグには関係がある，仮説（H2）変数名とバグの関係は次のバージョンでも保たれる，について分析を行う．この 2 つの仮説を検証するために，変数名ごとのバグ密度を算出し，変数名によるバグ密度の違いを確認し，バグ密度のバージョン間における相関関係の分析を行った．Eclipse3.0 および Eclipse3.1 を対象として分析した結果，変数名ごとのバグ密度は，バージョン間で相関係数 0.621 と，やや強い相関が見られた．結果は，バグ密度の高いモジュールに含まれる変数名が，次のバージョンのあるモジュールに含まれていたとすると，そのモジュールはバグを含む可能性が高いことを示唆している．

キーワード　変数名，変数名とバグの関係，バグ密度

An analysis of relationship between software bug and variable name

Hiroyuki YAMAMOTO†, Yasutaka KAMEI†, Shinsuke MATSUMOTO†, Akito MONDEN†, and Ken-ichi MATSUMOTO†

† Graduate School of Information Science, Nara Institute of Science and Technology　Takayama 8916–5, Ikoma, Nara, 630–0192 Japan
E-mail: †{hiroyuki-y,yasuta-k,shinsuke-m,akito-m,matumoto}@is.naist.jp

Abstract　Variable names used in each software module could be used as indicators of software bugs in the module because (1) names are given by a programmer who is responsible of the module and some programmers tend to produce more bugs than others, and (2) names are usually related to some functions in a program and some functions are more difficult to implement (without injecting bugs) than others. So far, the relationship between variable names and software bugs have not investigated. In this paper, by analyzing Eclipse 3.0 and 3.1 projects, we experimentally validate two hypotheses; (H1) there is a relationship between variable names and software bugs, (H2) this relationship is preserved through different versions of a software product. In the experiment, for each variable name, we calculated the average bug density of modules in which the name appeared. The result supported both H1 and H2, and the coefficient of correlation of bug density (bound to each name) between Eclipse 3.0 and 3.1 was 0.621. This suggests that focusing on variable names related to high bug density in the previous version can help in identifying high risk modules of an ongoing project.

Key words　variable name, relationship between software bug and variable name, bug density

1. はじめに

ソフトウェア開発において，限られた開発期間で信頼性を確保するために，バグを含んでいる可能性の高いモジュールを特定し，テスト工数を重点的に割り当てることで，テスト工程を効率化することが重要であるとされている [5]．そのために，従来，モジュールの構造を数値化したメトリクス（ソースコード行数や，ループの数など）を説明変数とし，モジュールに含まれるバグの有無を目的変数とする判別モデルが線形判別分析，ロジスティック回帰分析，分類木など多数提案されている [2], [3], [6].

しかし，モジュールの複雑度を表すサイクロマティック数の

値が小さくても，扱っているライブラリの専門性が高いなどの理由から，モジュールの理解が困難になり，バグを作りこんでしまう可能性がある．このような理由から生じるバグは，従来のモジュールの構造のメトリクスからでは予測できない．

本稿では，キーアイデアとして，モジュール中の変数名に着目する．変数名とバグの関わりを調べるのは，2 つの理由を考えたためである．1 つ目は，変数名はモジュールの機能による影響を受けるからである．例えば，専門性の高いライブラリを用いることにより，そのライブラリの知識が不足していることが原因で，作りこんでしまうバグがあると考えられる．このような，専門性の高いライブラリを用いるモジュールにおいて，そのモジュールに含まれる変数名は名前中にそのモジュールの担う機能に関連した語を含む可能性が高いため，同じ機能のモジュールには同じ変数名が使用されると考えられる．2 つ目は，変数名は開発者の属人性の影響を受けるからである．例えば，経験が浅いなどの理由からバグを混入させやすい開発者は，変数名をつける際に，命名規則に反した名前をつけたり，その変数の役割を正確に表していない変数名をつけることによって，モジュールの理解が困難になり，バグが混入されやすくなると考えられる．これらのような理由からモジュールに含まれるバグは，次のバージョンでも同様の機能や，開発者が関わるモジュールに含まれると考える．

本稿では，変数名とバグの関係を明らかにすることを目的として，変数名とバグに関する 2 つの仮説について分析を行う．具体的には，仮説（H1）変数名とバグには関係がある，および仮説（H2）変数名とバグの関係は次のバージョンでも保たれる，の 2 つの仮説を実験的に確かめる．分析は，仮説（H1）を検証する単バージョン分析，及び仮説（H2）を検証するバージョン間分析の 2 つからなる．単バージョン分析では，まずある特定のバージョン中での各変数名のバグ密度の分布を分析する．次にバグ密度が高かった（低かった）変数名がどのようなものかを調べることにより，変数名によってバグ密度の違いがあるかを確かめる．バージョン間分析では，前のバージョンと次のバージョンの間での変数名ごとのバグ密度の相関関係を確かめることにより，バージョン間で変数名ごとのバグ密度が保たれているかを確かめる．

なお本稿では，モジュールごとの単純なバグの数ではなく，単位行数あたりのバグ数（以降，バグ密度）を用いて分析を行う．バグ密度を用いることによりソースコードの規模の要因を排除して，どの程度のバグが含まれているか確かめることが可能である．また，分析はオープンソースの統合開発環境 (IDE) である Eclipse のバージョン 3.0（以降，Eclipse3.0）とバージョン 3.1（以降，Eclipse3.1）を対象に行う．

以降，2 章では，分析方法と，手順を述べる．3 章で，分析結果を述べる．4 章で，まとめと今後の課題を述べる．

2. 分 析 方 法

2.1 仮　　説

仮説（H1）: 変数名とバグには関係がある

まず，仮説（H1）を実験的に確かめる．この仮説は以下の 2 つの理由に基づく．1 つ目は，特定の変数名には専門性の高いライブラリを用いているモジュールに含まれるなど，機能的な理由からで，2 つ目は，特定の変数名は経験が浅くバグを作りこみやすい開発者が開発したモジュールに含まれるなどの属人性からである．このような機能や属人性は，変数名により識別できると考える．

仮説（H2）: 変数名とバグの関係は次のバージョンでも保たれる

変数名による識別が可能なモジュールの機能や属人性があれば，その機能や開発者に関わりのあるバグは，次のバージョンでもバグが含まれる可能性があり，変数名に着目することにより検出できると考える．この仮説が支持されれば，前のバージョンでバグ密度の高かった変数名を含む次のバージョンのモジュールは，バグ密度が高く，逆に前のバージョンでバグ密度の低かった変数名を含む次のバージョンのモジュールは，バグ密度が低いと考えられる．これを利用して，バグ検出に役立てられる可能性がある．

2.2 分 析 概 要

2 つの仮説を検証するために，単バージョン分析とバージョン間分析を行う．

2.2.1 単バージョン分析

仮説（H1）の検証を行うために，単バージョン分析を行う．変数名とバグの関係を分析する方法として，個別の変数名のバグ密度の値を算出し，変数名のバグ密度の分布を箱ひげ図で示す．また，バグ密度上位／下位の変数名 10 個を表で示す．

ここで変数名ごとのバグ密度の算出方法について説明する．この変数名ごとのバグ密度は，ソフトウェア中のその変数名を含む全てのモジュールの総バグ数を総行数で割った値である．ある変数名を X，あるモジュールを M_i として，$bug(M_i)$ をモジュール M_i のバグ数，$SLOC(M_i)$ をモジュール M_i のソースコード行数，$module(X)$ を変数名 X を含むモジュールの集合とすると，変数名 X のバグ密度 $bug\_density(X)$ は，以下の式で表される．

$$bug\_density(X) = \frac{\sum bug(M_i)}{\sum SLOC(M_i) \times 1,000} (i \in module(X))$$

なお，本稿で扱うバグ密度は 1,000 行あたりいくつのバグが含まれていたかの値を指す．

2.2.2 バージョン間分析

次に，仮説（H2）の検証を行うためにバージョン間分析を行う．バージョン間における各変数名のバグ密度の相関を調べることにより，バージョン間で各変数名のバグ密度が保たれているかを調べる．まず，2.2.1 節において算出した両バージョンの各変数名のバグ密度から，変数名のバグ密度の変化を散布図で示す．次に，変数名のバグ密度のバージョン間における相関係数を算出する．

また，両バージョンでバグ密度が高かった変数名がどのような機能のモジュールに含まれているか調べる．調べる方法として，モジュールのパッケージに注目する．

表 1　分析対象の選別条件一覧

対象	条件
ソースコード行数	100 行以上
変数名	メンバ変数のみ
変数名の文字数	5 文字以上
変数名が跨るモジュール数	10 以上 100 以下

表 2　分析対象データの概要

	モジュール数	*SLOC*	総バグ数	バグ密度
Eclipse3.0	1,002	320,546	1,005	3.14
Eclipse3.1	1,188	390,188	1,286	3.30

2.3　分析対象のデータ

本稿は，分析の対象として，オープンソースの統合開発環境 (IDE) である Eclipse3.0 と Eclipse3.1 を用いる．分析を行う前にこれら両バージョンのモジュールと変数名に対して，データ選別を行う．

まず，分析対象とするモジュールは，ソースコード行数 100 行以上のモジュールのみとする．これは，バグ密度が，ソースコード行数が少ないことにより，非常に大きな値をとる [4] のを防ぐためである．次に，扱う変数はメンバ変数とする．メンバ変数を用いる理由は，メンバ変数はオブジェクトの状態を保持する変数のためモジュールの機能などを反映した名前が付けられているからと，開発者が名前を付けるので属人性の影響を受けるからである．変数名は，文字数 5 文字以上で，10 以上 100 以下のモジュールに含まれるものを対象とした．これは，“i” や “j”，“sum” といった短い文字数の変数名は複数のモジュールで使用される可能性の高い名前であり，これらを分析対象外とするためである．モジュール数に下限を設けることにより，バグがあるモジュールに偶然含まれていた変数名があった際に，その影響を抑えることができると考える．モジュール数に上限を設けたのは，あまりに多くのモジュールに含まれる変数名は，バグの有無や，モジュールの機能によらず多用される汎用的な変数名である可能性が高いためである，本稿では対象外にするためである．

以上の条件を両バージョンで満たす変数名は，125 個存在しており，この 125 個の変数名いずれかを含むモジュールは，Eclipse3.0 で 1,002 個，Eclipse3.1 で 1,188 個存在し，これらを分析対象とする．ここで，各モジュールのバグ数は，Gyimóthy ら [1] の方法を用いて Eclipse プロジェクトが運用している障害管理ツール Bugzilla から算出した．分析対象データの概要を，表 2 に示す．

また，本稿ではモジュールのパッケージについても調べた．パッケージとはクラスの集合を機能的な意味のある単位でまとめたものである．バグ密度上位の変数名が含まれるモジュールが，どのパッケージに属するかを調べることにより，バグ密度上位の変数名がどのような機能のモジュールに分布していたかを確かめることが出来る．このとき，モジュールの機能を大別するために，最も上位層のパッケージ，つまり org.eclipse 直下のサブパッケージに注目する．また，今回の分析で対象としていた Eclipse3.0 及び Eclipse3.1 には，org.eclipse 直下のサブパッケージと，少数の org.eclipse 以外のパッケージと合わせて 15 種類存在する．15 種類のパッケージを表 3 に示す．

表 3　パッケージ一覧とモジュール数

パッケージ	Eclipse3.0	Eclipse3.1
org.eclipse.ui	347	422
org.eclipse.team	171	196
org.eclipse.jface	91	105
org.eclipse.core	84	95
org.eclipse.swt	84	96
org.eclipse.debug	63	77
org.eclipse.update	52	56
org.eclipse.ant	40	43
org.eclipse.compare	22	22
org.eclipse.help	20	46
org.eclipse.search	15	15
org.eclipse.search2	6	8
org.eclipse.text	4	4
org.apache.lucene	2	2
org.eclipse.tomcat	1	1
合計	1,002	1,188

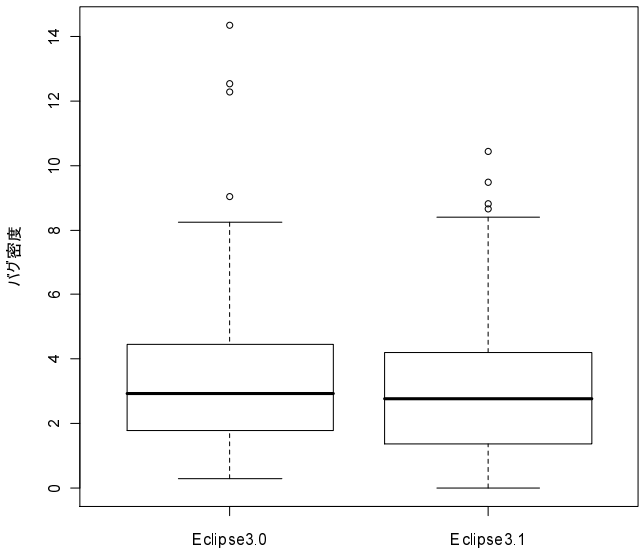

図 1　各変数名のバグ密度の分布

3.　分 析 結 果

3.1　単バージョン分析

Eclipse3.0，Eclipse3.1 の両バージョンの各変数名のバグ密度の分布を図 1 に示す．図 1 から，各変数名のバグ密度は，Eclipse3.0 から Eclipse3.1 で，バグ密度の分布はほとんど変わっていないことが分かる．

次に，バグ密度の上位／下位 10 個の変数名を Eclipse3.0 については表 4 と表 5 に，Eclipse3.1 については表 6 と表 7 にそ

表 4　Eclipse3.0 におけるバグ密度上位の変数名

変数名	バグ密度	モジュール数	パッケージ数
WorkbenchMessages	14.35	17	1
right	12.56	10	4
PlatformUI	12.28	18	3
ImageDescriptor	9.03	17	6
element	8.23	13	4
adapter	8.05	10	5
result	7.92	37	8
window	7.66	33	6
fView	7.44	10	3
provider	7.34	10	4
平均値	9.49	17.5	4.4

表 5　Eclipse3.0 におけるバグ密度下位の変数名

変数名	バグ密度	モジュール数	パッケージ数
CVSProviderPlugin	0.26	12	1
LRESULT	0.40	10	1
lpWndClass	0.42	15	1
refCount	0.48	11	2
IProject	0.61	10	3
index	0.74	26	7
buffer	0.88	14	8
fModel	0.90	10	4
handle	1.00	13	2
column	1.01	10	3
平均値	0.67	13.1	3.2

表 6　Eclipse3.1 におけるバグ密度上位の変数名

変数名	バグ密度	モジュール数	パッケージ数
plugin	10.45	15	7
adapter	9.47	12	5
initialized	8.80	12	5
context	8.64	23	6
ImageDescriptor	8.41	19	6
SIZING_TEXT _FIELD_WIDTH	8.25	10	3
descriptor	8.15	21	7
folder	7.84	11	3
element	7.48	20	6
WorkbenchMessages	7.39	18	1
平均値	8.49	16.1	4.9

表 7　Eclipse3.1 におけるバグ密度下位の変数名

変数名	バグ密度	モジュール数	パッケージ数
fModel	0.00	11	4
HASH_FACTOR	0.00	12	2
HASH_INITIAL	0.00	12	2
hashCode	0.00	13	3
string	0.51	12	2
buffer	0.60	16	9
images	0.64	13	4
resources	0.66	10	3
height	0.66	14	5
model	0.68	11	4
平均値	0.37	12.4	3.8

れぞれ示す．　表 4，表 5 から，Eclipse3.0 のバグ密度は，バグ密度の低い変数名とバグ密度の高い変数名では，10 倍以上の差があった．同様に，Eclipse3.1 についても，表 6，表 7 から，各変数名のバグ密度は，上位と下位では 10 倍以上の差があった．このことから，変数名によってバグ密度は大きく異なると言える．また，表 4 と，表 6 から，“adapter” や “result” などの変数名は，両バージョンでバグ密度が高いという結果が得られた．このことから，バージョンに関わらずバグ密度の高い変数名が存在する可能性がある．また，Eclipse3.0 で最もバグ密度が高かった “WorkbenchMessages” については表 4 より 1 パッケージでのみ出現していた変数名であった．変数名に着目することによりバグが多い一部の機能を特定できていた可能性がある．

3.2　バージョン間分析

Eclipse3.0 と Eclipse3.1 のバージョン間で，各変数名のバグ密度の関係を分析した結果を説明する．各変数名それぞれの Eclipse3.0 でのバグ密度と Eclipse3.1 でのバグ密度の散布図を図 2 に示す．図の各点は各変数名を表す．バージョン間の相関係数は，0.621 となり，やや高い値が得られた．無相関検定による p 値は，1.09×10^{-14} であり，有意であるといえる．このことから，バグと変数名の関係は，2 つのバージョンの間で保たれていると言える．この結果から，バグが含まれていやすい特定の変数名を確認し，その変数名が含まれるモジュールを重点的にテストすることにより，効率的な品質改善ができる可能性がある

また，Eclipse3.0 と Eclipse3.1 の両バージョンでバグ密度の高かった変数名 5 つについて，Eclipse3.0 におけるモジュールのパッケージの割合を図 3 に示す．図 3 から，変数名が含まれるモジュールの機能について，“right”，“adapter” といった，どのような機能のモジュールでも使用されそうな変数名は，複数の機能に偏ることなく分布していることが分かる．また，“ImageDescriptor”，“WorkbenchMessages” といった，複数語から成り，特定の機能を表している変数名は，その変数名を含むモジュールの過半数が 1 つの機能に集中していることが分かる．

4.　まとめと今後の課題

本稿では，キーアイデアとして変数名に着目し，変数名とバグの関係を明らかにすることを目的として，2 つの仮説について分析を行った．

Eclipse3.0 と Eclipse3.1 のデータを用いた分析の結果，変数名のバグ密度には，変数名によっては 10 倍以上の差があること，及び変数名のバグ密度は，前のバージョンと次のバージョンでやや強い相関があることが分かった．前のバージョンから，バグが含まれていやすい変数名を特定し，次のバージョンでその変数名を含むモジュールを重点的にテストすることにより，効率的な品質改善ができる可能性がある．

本稿では，各モジュールに含まれる変数名のみに着目したが，モジュールごとに含まれる変数名の組み合わせについては考慮

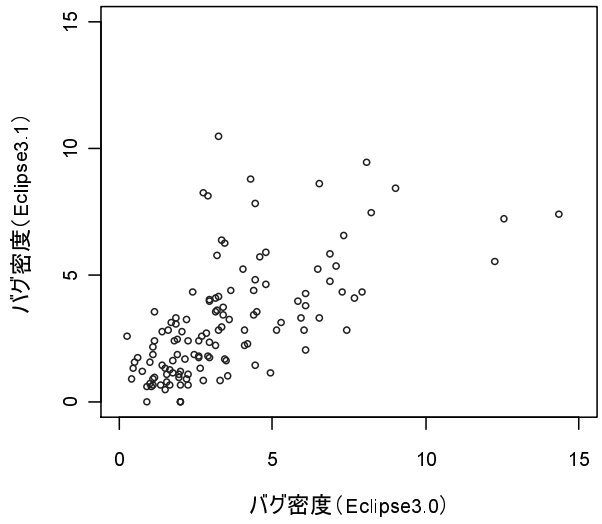

図 2　バージョン間のバグ密度

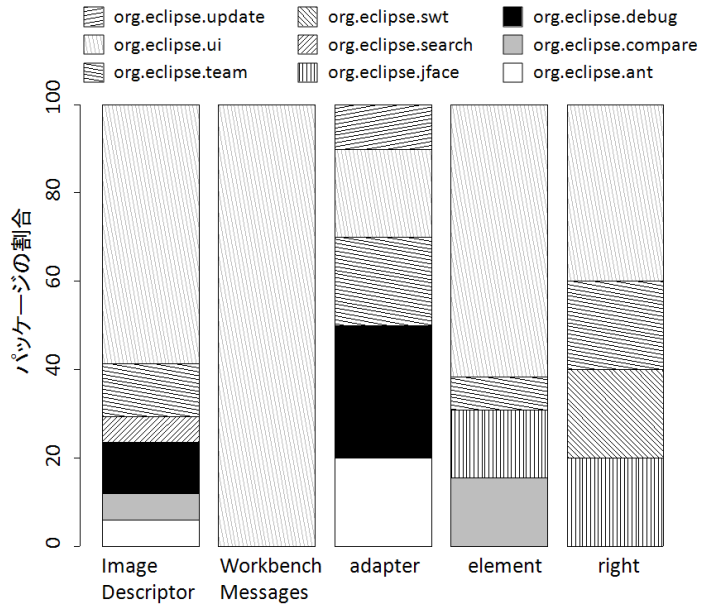

図 3　両バージョンでバグ密度が高い変数名のパッケージ

していない．例えば，“proxy” や “connection”, “protocol” といった変数名が同時に出現するモジュールはネットワークに関する機能を有する可能性が高い．このように変数名の組み合わせに着目し，バグ密度の分析をすることでバグが含まれやすい機能を特定できる可能性がある．また，今後は本稿での分析結果に基づき，変数名を用いることでバグ密度の推定が可能かどうかを実験的に確かめる予定である．

謝辞　本研究の一部は，文部科学省「次世代 IT 基盤構築のための研究開発」の委託に基づいて行われた．また特別研究員奨励費（課題番号：20009220，21・8995）の研究助成を受けて行われた．

文　　献

[1] T. Gyimóthy, R. Ferenc and I. Siket, “Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction” IEEE Trans. Software Engineering, vol.31, no.10, pp.897-910, 2005.

[2] D. W. Hosmer and S. Lemeshow, “Applied Logistic Regression,” Wiley, 1989.

[3] T. M. Khoshgoftaar and E. B. Allen, “Modeling Software Quality with Classification Trees,” Recent Advances in Reliability and Quality Engineering, World Scientific , pp.247-270, 1999.

[4] A. G. Koru, D. Zhang, K. E. Emam and H. Liu, “An Investigation into the Functional Form of the Size-Defect Relationship for Software Modules,” IEEE Trans. Software Engineering, vol.35, no.2, pp.293-304, 2009.

[5] P. L. Li, J. Herbsleb, M. Shaw and B. Robinson, “Experiences and Results from Initiating Field Defect Prediction and Product Test Prioritization Efforts at ABB Inc.,” Proc. 28th Int’l Conf. on Software Engineering (ICSE’ 06), pp.413-422, 2006.

[6] J. C. Munson and T. M. Khoshgoftaar, “The Detection of Fault-Prone Programs,” IEEE Trans. Software Engineering, vol.18, no.5, pp.423-433, 1992.