Vol.2021-SE-207 No.29
2021/3/2

ソフトウェアドキュメントとバグ報告における単語の出現数比較によるバグ集中箇所の予測

山内泉水[1,a)]　森崎修司[1]

概要：バグレポートに出現する単語の出現頻度とソフトウェアドキュメントに出現する単語の出現頻度を比較し，バグレポートに頻出する単語から頻出するバグを予測する手法を提案する．形態素解析により，バグレポートと設計ドキュメントやマニュアルに出現する名詞を抽出し，出現頻度を比較する．バグレポートに多数出現する単語があれば，その単語を含むバグが頻出していると推測でき，機能名をはじめ実装が特定箇所に集中するような単語であれば，Fault-prone モジュール予測に使える可能性がある．実際に EC-CUBE と NetCommons のソフトウェアのバグレポートとドキュメントを用いて確かめたところ，EC-CUBE は 25 単語中 22 語，NetCommons は 18 単語中 16 語が実装の特定箇所に集中していた．

キーワード：形態素解析，設計ドキュメント，Fault-prone モジュール予測

1. はじめに

ソフトウェアテストや保守において，バグの集中している部分を特定することは，バグの修正はもちろんテスト工数割り当ての優先順位の決定に役立つ．しかし，実際にバグがある箇所を発見し，集中している箇所を予測するために十分なテストを行うには，多くの時間とコストがかかる．そのため，開発にかかる時間の短縮や，コストの削減が求められている．この問題を解決する方法として，時間的効率が高い方法でバグを検出する手法を考案することが重要である．そうした手法のひとつとして，fault-prone モジュールを予測する研究が行われている [1][2][3]．多くの研究で，モジュールのメトリクス値 (コードの行数や Cyclomatic 数など) を入力とすることで，欠陥の有無を予測している [4][5][6]．fault-prone モジュール予測では，過去のバグの報告によるモジュールの fault の有無と，メトリクス値を入力とすることでモデルを作成し，モデルにより fault の有無を予測する．

Fault-prone モジュール予測以外にも，過去のバグ報告の分析により，バグの集中している箇所を特定する方法が考えられる．バグの集中している箇所はバグ報告の数も多いと考えられる．したがって、バグ報告に多く出現する単語を取り出すことによって，バグの集中している箇所を予測できる可能性がある．ただし，バグが集中していない単語であっても，そのソフトウェアの対象領域を表す等していて，頻出している語も多く含まれることが推測される．単純にバグ報告の頻出単語にバグが集中しているわけではないと考えられる．たとえば，オンラインショッピングサイトを実現するソフトウェアのバグ報告で “商品” という語が頻出しているからといって，同様のバグが頻出しているかどうかは判断できない．

本研究では，ソフトウェアドキュメントとバグ報告の両方に出現する単語を対象として，ドキュメントよりもバグ報告での出現数が有意に大きい単語をバグ報告に頻出する単語と考える．そうした単語を含むバグ報告において修正されたソースコードファイルを調べ，実際に類似の原因で修正されているかどうかを実証的に評価する．また，他の方法と比較して予測の効果を確かめる．具体的には，2 件のオープンソースソフトウェアを対象として，設計書，ユーザマニュアルとバグ報告の両方に出現する単語のうち，バグ報告で有意に出現回数の多い単語を得る．それらの単語を含むバグ報告において修正されたソースコードファイルを確認し，類似のバグが集中しているかどうかを確かめる．また，予測モデルを作成せずソースコードメトリクスから

[1] 名古屋大学
Nagoya University, Nagoya, Aichi 464-8601, Japan
[a)] yizumy2112@gmail.com

欠陥を予測する簡易的な Fault-prone module 予測の結果，すべてのバグ報告の修正ソースコードファイルを集計した結果と比較する．

2. 提案手法

2.1 前提

提案手法では，あるソフトウェア x のマニュアル，設計書などのドキュメントにおける単語の集合 (D_x) の各出現数に対して，バグ報告に出現する単語の集合 (B_x) における出現数が多い単語を取り出すことで，その単語に関連するバグが多いと考える．そして，その単語に関連しているバグ報告を確認したとき，特定の部分に集中するバグがあったときバグが集中していると考える．出現数が多いかどうかを同一ソフトウェアのドキュメントでの単語の出現数と比較することで判断する．設計書は開発しているソフトウェアの機能を定義するドキュメントである．マニュアルはソフトウェアの機能をユーザ向けに説明したドキュメントである．本手法において，バグ報告，ドキュメントは自然言語で記述されていることを前提としている．単語は形態素解析ツールにより抽出した名詞とする．名詞は，機能名や処理名をはじめとして，開発間で認識を正しく共有するために，表現を統一し，同一の内容を表す際に，同一の語で繰り返して使われる可能性が高いと考えるからである．そして，バグ報告においても，同様の表現が用いられる可能性が高いと考えた．提案手法の手順を以下に示す．

手順 1 ソフトウェアのバグ報告の集合に出現する単語 B_x のうち，ドキュメントに出現する単語 D_x と比べてバグ報告において出現頻度が多い単語の集合 W_{xf} を得る．

手順 2 W_{xf} の単語を含むバグ報告 B_{xf} を取り出して修正箇所を調べることで，バグが集中しているソースコードが修正されているかどうか確かめる．

2.2 手順 1

$W_x = B_x \cap D_x$ $w \in W_x$ とし，w から，以下の基準で D_x と比較して B_x に多い単語の集合 W_{xf} を得る．

手順 1.1 バグ報告 B_x とドキュメント D_x 間において出現頻度に統計的に有意な差がある単語を W_x から取り出す．自由度 1 のカイ二乗分布における有意水準 0.05 の棄却域は 3.84 以上であるため，バグ報告 B_x における出現頻度が大きい単語 $w(\in S_{xf})$ は，以下により求める．

$S_{xf} = \{w|(X_w > 3.84)\}$

ここで，X_w は帰無仮説 “単語 w のバグ報告 B_x とドキュメント D_x における出現数は独立である” としたときのカイ二乗値である．

手順 1.2 語 w のバグ報告 B_x での出現割合が，ドキュメント D_x の出現割合と比較して大きければ，W_{xf} に w を加える．

ある単語 w のバグ報告 B_x での出現頻度を b_w，ドキュメント D_x での出現頻度を d_w とする．バグ報告 B_x に出現する単語 $w(\in W)$ の出現回数の合計を b_{xall}，ドキュメント D_x に出現する単語 $w(\in W)$ の出現回数の合計を d_{xall} とすると，単語 w のバグ報告 B_x での出現割合 $P(B_x, w)$，ドキュメント D_x での出現割合 $P(D_x, w)$ を，以下のように定義する．

$P(B_x, w) = b_w / b_{xall}$

$P(D_x, w) = d_w / d_{xall}$

このとき，W_{xf} は，以下で定義できる．

$W_{xf} = \{w|(P(B_x, w) > P(D_x, w)) \wedge w \in S_{xf}\}$

手順 1.3 W_{xf} から以下の語を取り除く

- バグ報告のテンプレートで使われているもの (“確認”，“環境” 等)
- バグ報告においてバグの内容と関係なく出現しやすいもの (“失敗”，“エラー”，“発生” 等)
- 記号，一文字のアルファベットなど，語としての意味をなしていないもの

2.3 手順 2

$w(\in W_{xf})$ に対して，w の出現するバグ報告を選ぶ．その際に，以下の条件を満たすバグ報告を選ぶ．

(1) バグ報告において報告されている内容が解決済みである．

(2) バグ報告において報告されている内容がバグであったと確認されている．(機能改善や実際にはバグではないバグ報告があるため)

この基準を用いて取り出されたバグ報告の集合を B_{xf} とする．B_{xf} の各バグ報告の修正ファイルを確認して，特定の箇所 (ソースコード) に修正が集中しているバグが存在しているか確認する．本手法では，同一ソースコードファイルに 2 件以上の修正があった場合に，集中していると判断することとした．そして，B_{xf} の中で同一ソースコードファイルに 2 件以上の修正があればそれらのバグ報告の集合を B'_{xf} とする．

3. ケーススタディ

3.1 設定

本評価の目的は，提案手法によって取り出した語から，バグが集中しているソースコードを特定できるかどうかを確かめることである．提案手法で取り出した語 W_{xf} にバグが集中しているかを評価する手順を述べる．

3.1.1 評価 1 バグ集中箇所の予測

評価 1 では，ドキュメントと比較して，バグ報告に多く出現する語の出現するバグ報告 B_{xf} に記録された修正ソースコードファイル名を確認することで，バグが集中しているか確認することで評価を行う．また，バグが集中していると判断できた語がどれだけ存在するかを調査する．評価

表 1　バグ報告の件数，出現単語数

	EC-CUBE	NetCommons
バグ報告件数	531	1093
出現単語数	111581	62888

の対象は，バグ報告とそのバグ報告によって修正されたファイルと修正箇所がオープンソースとして公開されているソフトウェアとした．

3.1.2　評価 2 ソースコードメトリクスによる予測との比較

評価 2 では，評価 1 と同じソフトウェアのソースコードからソースコードメトリクス値を算出し，cyclomatic complexity の大きい順にならべたときの順位と評価 1 の結果と比較する．fault-prone モジュール予測のように，過去のバグとソースコードメトリクスを学習モデルに与えて予測する方法は使わなかった．Cyclomatic Complexity は，モジュールの分岐の数に比例して大きくなる数であり，一般的に，値が大きいほど複雑になることが知られている．そのため，欠陥も含まれやすいことが経験的に知られている．

3.1.3　評価 3 ソースコード修正数の単純比較による予測との比較

評価 3 では，評価 1 と同じソフトウェアから出現単語を考慮せず，すべてのバグ報告におけるソースコード修正数を集計し，降順にならべたときの順位と評価 1 の結果を比較する．この値はソースコードにバグがどの程度存在していたかを示すものであり，この値の上位は実際にバグが多いソースコードファイルを示しているといえる．

3.1.4　評価対象

評価 1，評価 2，評価 3 ともに，github 上でソースコードが公開されていて，マニュアルや設計書などのドキュメントが公開されている次のソフトウェアとした．オープンソースの電子商取引のコンテンツ管理システムである EC-CUBE，NetCommons プロジェクトが開発しているコンテンツ管理システムである NetCommons である．EC-CUBE ではドキュメントを設計ドキュメント (D_e) とした．バグ報告は Github の issue(B_e) を使った．NetCommons ではドキュメントをユーザマニュアル (D_n) とした．バグ報告は Github の issue(B_n) を使った．

D_e, B_e, D_n, B_n とも，少数の海外ユーザのバグ報告を除いて，すべて日本語で記述されている．その概要を表 1 に示す．

3.1.5　手順

評価 1.1

提案手法の**手順 1** にしたがって，B_e と D_e を対象に，ドキュメントにおける出現頻度のほうがバグ報告における出現頻度より大きい語の集合 W_{ef} を得る．単語の出現数の取り出しは，形態解析素ツール KH-coder を利用した．日本語の文書の場合，取り出しの対象となる単語は KH-coder 標準のシステム辞書において名詞として定義される単語とした．

また B_n，D_n についても同様に行い，ユーザマニュアルにおける出現頻度のほうがバグ報告における出現頻度より大きい語の集合 W_{nf} を得る．

評価 1.2

取り出した語の集合 W_{ef}, W_{nf} の単語を対象にして，提案手法を適用する事で，バグが集中している箇所が特定できるかについて評価する．

本評価において対象としたバグの条件を以下に示す．なお，カッコ内は，提案手法の手順で示した条件である．

(1) issue がクローズされている (バグ報告において報告されている内容が解決済みである)

(2) issue にバグである事を示すタグ (bug-critical, bug-high, bug-low, bug-middle, bug) がついている (バグ報告において報告されている内容がバグであったと確認されている．)

評価 1.3

バグ報告において出現頻度が大きい単語を含むバグ報告 B_{xf} に記録された修正ファイルを確認し，特定の箇所 (ソースコード) に修正が集中しているバグが存在しているか確認する．本評価では，同一ソースコードファイルに 2 件以上の修正があった場合に，集中していると判断することとした．B_{xf} の中で特定のソースコードファイルが 2 件以上修正されているバグ報告がある場合，その修正が記録されているバグ報告を B'_{xf} とし W_{xf} に含まれる単語ごとに件数を調べる．

評価 1.4 バグが集中していると特定できた単語はどの程度存在するか

W_{xf} に含まれる単語において，その単語が出現している B'_{xf} の数が 2 つ以上である場合，バグが集中していると特定できた単語とする．そのような単語が W_{xf} にどの程度の割合存在しているか調べる．

評価 2 ソースコードメトリクスによる予測との比較

EC-CUBE, NetCommons について, ソースコードを Github から取得し，モジュールごとのメトリクス値を計測する．本評価では，phpmetrics を利用した．評価 1 における，特定の場所にバグが集中していると判断できるバグ報告の集合 B'_{xf} におけるバグ報告の共通する修正ファイルを取り出す．Cyclomatic Complexity 値が大きいソースコードを確認することで，提案手法との重複を確かめる．具体的には，以下により確かめる．

- **評価 2.1** B'_{xf} に記録されている修正モジュールが，メトリクス値上位のソースコードとどの程度重複していたか
- **評価 2.2** メトリクス値下位のソースコードが提案手法により得られた B'_{xf} にどの程度存在していたか

評価 3 すべてのバグ報告から集計した修正頻度の大きいソースコードファイルとの比較

対応が完了したバグ報告には修正ソースコードが記録されている．これらを集計するとバグが集中しているソースコードファイルを得られる可能性がある．すべてのバグ報告から集計したソースコードファイルの集計結果と提案手法によって予測したソースコードファイルに違いがあるかどうかを確かめる．

3.2 評価結果

3.2.1 評価 1 バグ集中箇所の予測

評価 1.1，1.2 EC-CUBE のバグ報告に多く出現した語 W_{ef} を表 2 に示す．表に示している語は，バグ報告における出現頻度の降順となっている．NetCommons についても，バグ報告に多く出現した語 W_{nf} を表 3 に示す．

評価 1.3 バグ報告の分析結果

B'_{wf} は，バグが集中しているバグ報告であり，特定の場所にバグが集中していると判断したバグ報告のことを指す．本研究においては，このカテゴリのバグ報告の数が 2 つ以上存在するとき，その語を含むバグ報告は特定箇所に集中したバグ修正を記録に含むバグ報告が含む語と考える．

EC-CUBE において B_{wf} の中で，W_{ef} に含まれるそれぞれの単語が出現するバグ報告で B'_{wf} に分類されるバグ報告の数を表 4 に示す．列 “バグ報告数” は，同じソースコードを修正しているバグ報告の数を指す．

NetCommons においても同様に，B_{wf} の中で，W_{ef} に含まれるそれぞれの単語が出現するバグ報告で B'_{wf} に分類されるバグ報告の数を表 5 に示す．

評価 1.4 バグが集中していると特定できた語はどの程度存在するか

EC-CUBE の W_{xf} において，バグが集中していると判断できた語は 25 語中 22 語であり，その比率は 88%であった．NetCommons においては，18 語中 16 語の 89%がバグが集中していると判断できた語であった．

3.2.2 評価 2 ソースコードメトリクスによる予測との比較

評価 2.1 B'_{xf} に記録されている修正モジュールが，メトリクス値上位のモジュールとどの程度重複していたか

静的解析ツール phpmetrics を利用して 2 つのソフトウェアにおける php ファイルのメトリクスを計測した．EC-CUBE は 583 モジュール，NetCommons は 816 モジュールのメトリクス値を計測した．Cyclomatic Complexity の値 (以下 cc 値) の高いモジュールに，提案手法の予測結果がどの程度含まれていたかを確認した．結果を図 1，図 2 に示す．グラフの横軸は cc 値の大きさの順位であり，値が大きいほどグラフの左側になる．縦軸は，提案手法によりバグが集中していると予測されたモジュールが何件含まれていたかを表している．図 1 の 254 位〜303 位のモジュールの cc 値は全て 2，304 位〜は全て 3 で同率であるためまとめて表記した．

cc 値の低いモジュールに，どの程度提案手法によってバ

表 2 W_{ef} に含まれる語と出現数 (EC-CUBE)

W_{ef}	バグ報告	ドキュメント	$X_{w'}$
ORDER	297	2	34.96
ID	425	14	30.42
登録	619	33	27.34
受注	381	13	26.60
MySQL	133	1	15.45
CSV	125	1	14.40
PRODUCT	155	4	12.86
マスター	107	1	12.05
APP	105	1	11.78
PostgreSQL	104	1	11.65
クリック	80	1	8.52
ADMIN	116	5	6.62
vendor	62	1	6.19
SESSION	61	1	6.05
Array	83	3	5.55
件数	56	1	5.41
CODE	55	1	5.28
遷移	68	2	5.25
画面	994	104	4.96
Template	78	3	4.96
マスタ	50	1	4.64
検索	229	18	4.53
FAX	49	1	4.51
config	70	3	4.02
Resource	45	1	4.00

表 3 W_{nf} に含まれる語と出現数 (NetCommons)

W_{nf}	バグ報告	ユーザマニュアル	X_w
汎用	273	7	34.44
完了	265	13	23.35
更新	116	2	16.45
指定	196	12	13.98
ログイン	263	21	12.98
配置	106	3	12.88
登録	570	63	12.76
タグ	153	9	11.40
新着	129	7	10.42
時間	117	6	9.94
ブロック	173	14	8.35
デフォルト	114	7	8.11
パスワード	103	6	7.76
ボタン	228	23	6.78
項目	232	24	6.43
コンテンツ	141	12	6.20
ダウンロード	154	14	5.91
お知らせ	144	15	3.91

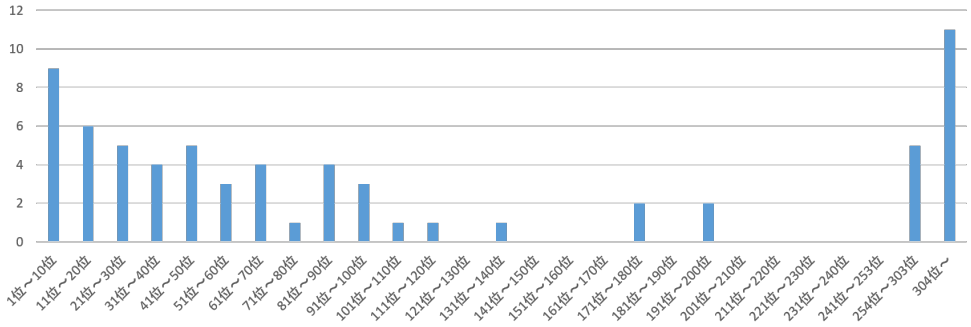

図 1　Cyclomatic Complexity の値の高い順に並べたモジュールとの比較 (EC-CUBE)

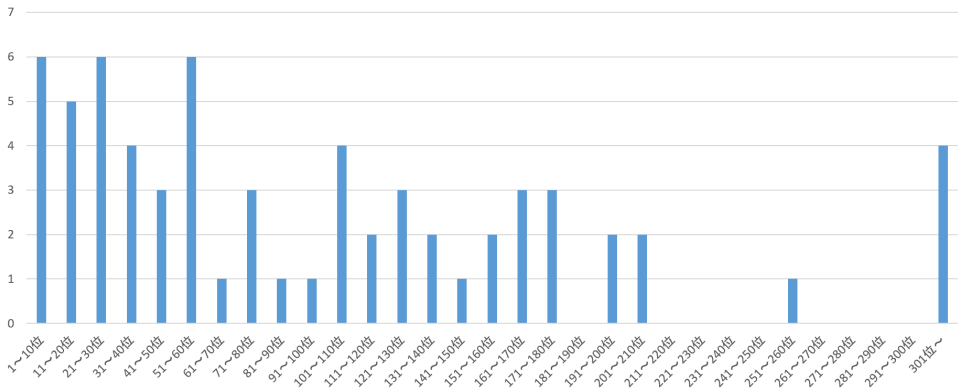

図 2　Cyclomatic Complexity の値の高い順に並べたモジュールとの比較 (NetCommons)

グが集中していると予測されたモジュールが含まれているか確かめたところ，EC-CUBE では cc 値が 2 のモジュールが 5 つ，1 のモジュールが 11，NetCommons では，cc 値が 2 のモジュール，1 のモジュールがそれぞれ 1 つずつ含まれていた．

評価 2.2 メトリクス値下位のモジュールが提案手法により得られた B'_{xf} にどの程度存在していたか

バグ報告に記録されている修正ソースコードを集計し修正の多かったファイル上位 20 位までを対象として，提案手法と Cyclomatic Complexity による予測の結果を調べた．結果を表 6，表 7 に示す．表の “提案手法” の列は，同じ単語の出現するバグ報告における同一ソースコードファイルの修正件数の最大値を示している．この値が 2 以上の場合，提案手法において B'_{xf} に含まれていたことを表す．“cc 値” の列は，Cyclomatic Complexity の値を表す．“cc 値” の列が空白になっていることは，Cyclomatic Complexity 値が取得できなかったことを表している．提案手法で得られた B'_{ef} を含むバグ報告に記録された修正履歴に同一のソースコードファイルが 2 回以上出現するものが，上位 20 位までのソースコード 24 個のうち全てに含まれていた．提案手法で得られた B'_{nf} を含むバグ報告に記録された修正履歴に同一のソースコードファイルが 2 回以上出現するものが，上位 20 位までのソースコード 21 個のうち 18 個含まれていた．

3.2.3 評価 3 すべてのバグ報告から集計した修正頻度の大きいソースコードファイルとの比較

提案手法とすべてのバグ報告から集計した修正頻度の大きいソースコードファイルの比較結果を評価 2 と同じく表 6，表 7，に示す．表 6 では，“提案手法” の 1 位は src/Eccube/Controller/ShoppingController.php で，“バグ報告数” では 2 位であった．また表 7 では，“提案手法” の 1 位は view/helper/multidatabasecontentviewelementhelper.php で，“バグ報告数” では 3 位であった．

4. 考察

4.1 評価 1 バグ集中箇所の予測

評価 1 の結果より，予測したソースコードの多くがバグが集中しているソースコードであった．したがって，ドキュメントにおける出現頻度と比較して，バグ報告における出現頻度の割合が大きい語は，同様のバグが報告されているバグ報告を選別するために役立つと考えられる．また，評価では，バグ報告に記録された修正ソースコードの

表 4　バグが集中していると判断した語 W_{ef}(EC-CUBE)

W_{ef}	バグ報告数
ORDER	20
ID	0
登録	6
受注	14
MySQL	4
CSV	17
PRODUCT	12
マスター	21
APP	8
PostgreSQL	4
クリック	13
ADMIN	9
vendor	0
SESSION	6
Array	0
件数	5
CODE	2
遷移	9
画面	18
Template	4
マスタ	5
検索	20
FAX	2
config	6
Resource	6

表 5　バグが集中していると判断した語 W_{nF}(NetCommons)

W_{nf}	バグ報告数
汎用	26
完了	49
更新	2
指定	10
ログイン	14
配置	6
登録	27
タグ	4
新着	5
時間	0
ブロック	9
デフォルト	9
パスワード	2
ボタン	13
項目	20
コンテンツ	4
ダウンロード	11
お知らせ	0

履歴から修正箇所を追跡したが，実際に対象ソフトウェアを開発しているメンバであれば，得られた語からソースコードファイルを特定できるものもあると期待される．たとえば，得られた語のうち，“受注”，“検索”，“ログイン”といった語に関係するソースコードファイルの数はそれほど多くない．本論文の評価では，バグ報告の修正記録を調べて 2 回以上同一箇所を修正していれば，集中していると判断したが，実際に開発しているメンバが提案手法を利用する場合には，そうした手間を省ける可能性がある．

今回の評価では，ドキュメントとして，設計ドキュメントとユーザマニュアルとし，それらにおける出現頻度とバグ報告における出現頻度を比較した．ソフトウェアによっては，設計ドキュメントが存在しなかったり，バージョンアップを重ねれば，古いバージョンのまま更新されていなかったりすることが起こりうるが，ユーザマニュアルは入手，更新がしやすいため，提案手法の利用範囲は大きいと考えられる．

4.2　評価 2 ソースコードメトリクスによる予測との比較

評価対象の 2 件のソフトウェアにおいて，ソースコードファイルごとに Cyclomatic Complexity を計測し，その値が上位であるソースコードファイル 50 件に対して提案手法による予測結果と比較したところ，一致するソースコードファイルが EC-CUBE では 30 件，NetCommons では 24 件含まれていた．また，Cyclomatic Complexity の値が小さいもののうち，提案手法がバグの集中を予測したモジュールも存在した．提案手法では，コードの複雑さ以外の観点でバグが集中しやすいモジュールを予測できる可能性がある．他のソフトウェアで同様の比較をすることや，学習モデルを用いた fault-prone モジュール予測との比較をすることは今後の課題である．

4.3　評価 3 すべてのバグ報告から集計した修正頻度の大きいソースコードファイルとの比較

提案手法による予測 1 位のソースコードの，すべてのバグ報告から集計した修正頻度の順位は，EC-CUBE では 2 位，NetCommons では 3 位であった．このことから，提案手法による予測はすべてのバグ報告から集計した修正頻度と完全には一致していないことが分かる．また，表 6 より，提案手法による予測 1 位のソースコードファイルはすべてのバグ報告から集計した修正頻度第 1 位のソースコードファイルと比較して，“cc 順位” も上位である．このことから，すべてのバグ報告の修正ソースコードを集計した場合，すべてのバグ報告から各ソースコードファイルの修正数を数えるため，バグが集中しているとは言えない，いろいろな機能に関わるソースコードも上位になりやすい傾向にあると考えられる．それに対し，提案手法はバグ報告に多かった単語が出現しているバグ報告から各ソースコー

表 6　提案手法によるバグ集中箇所の予測と Cyclomatic complexity による予測の比較 (EC-CUBE)

順位	ファイル名	バグ報告数	提案手法	手法順位	cc 値	cc 順位
1	src/Eccube/Application.php	39	17	3	7	120
2	src/Eccube/Controller/ShoppingController.php	33	22	1	66	5
3	src/Eccube/Controller/Admin/Product/ProductController.php	25	15	4	86	3
4	src/Eccube/Controller/Admin/Order/EditController.php	20	18	2	44	12
5	src/Eccube/Controller/Install/InstallController.php	17	2	59	121	2
5	src/Eccube/Service/ShoppingService.php	17	10	7		
7	src/Eccube/Service/PluginService.php	16	5	22	78	4
7	src/Eccube/ServiceProvider/EccubeServiceProvider.php	16	12	6	2	254
9	src/Eccube/Controller/Admin/Product/CsvImportController.php	15	10	7	222	1
10	src/Eccube/Repository/CustomerRepository.php	13	8	12	44	12
10	src/Eccube/Repository/ProductRepository.php	13	5	22	39	15
10	src/Eccube/Service/CartService.php	13	9	9	49	9
13	src/Eccube/Controller/Admin/Customer/CustomerController.php	11	9	9	15	60
13	src/Eccube/Controller/Admin/Store/PluginController.php	11	5	22	57	7
13	tests/Eccube/Tests/Web/Admin/Order/EditControllerTest.php	11	9	9		
16	src/Eccube/Controller/ProductController.php	10	5	22	29	23
16	src/Eccube/Repository/OrderRepository.php	10	7	15	60	6
18	src/Eccube/Controller/Admin/Content/BlockController.php	9	3	38	11	82
18	src/Eccube/Controller/Admin/Order/OrderController.php	9	8	12	47	11
18	src/Eccube/Entity/Product.php	9	5	22	38	17
18	src/Eccube/Form/Type/Admin/SearchCustomerType.php	9	8	12	1	304
18	tests/Eccube/Tests/Repository/CustomerRepositoryGetQueryBuilder…	9	6	18		
18	tests/Eccube/Tests/Service/PluginServiceTest.php	9	2	59		
18	tests/Eccube/Tests/Web/Admin/Product/ProductContorllerTest.php	9	6	18		

表 7　提案手法によるバグ集中箇所の予測と Cyclomatic complexity による予測の比較 (NetCommons)

順位	ファイル名	バグ報告数	提案手法	手法順位	cc 値	cc 順位
1	configschema/schema.php	20	3	15		
2	utility/currentframe.php	11	3	15		
3	view/helper/multidatabasecontentviewelementhelper.php	10	10	1	22	126
3	utility/current.php	10	3	15		
5	utility/installutil.php	9	2	37	53	8
5	model/taskcontent.php	9	3	15	50	12
5	model/behavior/saveuserbehavior.php	9	7	2	46	18
5	model/user.php	9	2	37	39	31
5	model/behavior/nc2tonc3wyswygbehavior.php	9	2	37	26	102
5	view/helper/menuformhelper.php	9	7	2	17	168
5	config/route.php	9	1	45		
5	controller/component/downloadcomponent.php	9	5	7		
5	testsuite/netcommonssavetest.php	9	1	45		
14	controller/pageseditcontroller.php	8	2	37	58	5
14	model/usersearch.php	8	2	37	52	9
14	controller/cabinetfileseditcontroller.php	8	2	37	42	22
14	model/multidatabasecontent.php	8	7	1	35	48
14	controller/component/roomsrolesformcomponent.php	8	3	15	33	59
14	controller/taskcontentscontroller.php	8	2	37	30	77
14	test/case/config/routestest.php	8	3	15		
14	utility/currentpage.php	8	1	45		

ドファイルの修正数を数えている．したがって，提案手法で上位となるソースコードは出現数の多い単語に関するバグであり，バグが集中しているソースコードが取り出しやすいと考えられる．評価 2 と同様に他のソフトウェアで同様の比較をすることは今後の課題である．

5. 関連研究

ソフトウェアのモジュールから得られるメトリクス値を用いて，そのモジュールに欠陥が含まれるかどうかを予測する手法を提案した研究がある [1][2][3]. これらの手法では，過去のプロジェクトのモジュールのコードの行数や Cyclomatic Complexity の値といったメトリクス値と，そのモジュールに実際にバグが存在したかどうかを記録したデータを使って，fault-prone モジュール判別モデルを構築する．提案手法では，バグの多い場所をバグ報告を分析することで予測するという点で，これらの手法とは異なる．

バグ報告を対象としてテキストマイニング手法を適用し，バグ報告の自動分類や重複バグ報告 (同一のバグを報告した複数のバグ報告) を特定する手法がある [7][8][9]．これらの手法は，バグ報告を自動分類することでバグの傾向分析をしたり，重複バグ報告を見つける手間を減らすことが目的であり，提案手法とは目的が異なる．

過去の多くのバグ報告のデータベースからデータマイニングを行って，バグ報告の傾向を分析することでバグに関する情報を得るツール BugMiner がある [10]．BugMiner はバグ報告の完了確認，冗長性確認，傾向分析を行う．ツールによって得られた結果をバグトラッキングツールに与えることで，将来のバグ報告の件数を予測し，バグ管理の正確性をより高めることを目的としている．提案手法のようにバグが集中している箇所を予測していない点で異なる．

6. まとめ

同一ソフトウェアにおけるバグ報告とドキュメントの両方に出現する単語における出現数の違いを利用することで，バグの集中している箇所を予測する手法を提案し，評価した．提案手法はバグ報告における出現頻度が他のドキュメントよりも大きい語を選ぶことで，バグ報告における出現頻度が大きい語を選ぶ．出現頻度が大きい語を含むバグ報告に記録されたソースコード修正履歴を確認することで，バグが集中しているソースコードを特定する．

ドキュメントが公開されているオープンソースのソフトウェア 2 件 (EC-CUBE, NetCoomons) を対象として，提案手法を評価した．バグ報告において出現頻度が大きかった語が EC-CUBE は 25 語，NetCommons は 18 語存在した．それらの語を対象として，その語を含むバグ報告の修正箇所を確かめた．その結果，同一ソースコードの特定箇所を 2 回以上修正しているものが，EC-CUBE では 22 件，NetCommons では 16 件あることがわかった．

さらに，各ソフトウェアのソースコードモジュールの Cyclomatic Complexity を求め，上位 50 件のソースコードが提案手法による予測箇所と一致しているかどうか確かめた．その結果，EC-CUBE では 30 件，NetCommons では 24 件のソースコードが提案手法の予測結果にも含まれていた．また，すべてのバグ報告に記録された修正ソースコードファイルを集計したものと比較した場合にも，提案手法のほうが類似のバグが集中しているソースコードファイルを予測することができた．

参考文献

[1] W John Wilbur and Karl Sirotkin. "The automatic identification of stop words. "*journal of information science*, 18(1):pp. 45-55, 1992.

[2] Norman E. Fenton and Niclas Ohlsson. "Quantitative analysis of faults and failures in a complex software system. "*IEEE Transactions on Software engineering*, 26(8):pp. 797-814, 2000.

[3] Tim Menzies, Jeremy Greenwald, and Art Frank. "Data mining static code attributes to learn defect predictors. "*IEEE transactions on software engineering*, 33(1):pp. 2-13,2007.

[4] 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. Software Engineering(ICSE'06), pp. 413-423, 2006.

[5] A. Mockus and D.M. Weiss, "Predicting risk of software changes, "Bell Labs Tech. J., 5(2), pp. 169-180, 2000.

[6] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures, "Proc. 28th Int'l Conf. Software Engineering(ICSE'06), pp. 452-461, 2006.

[7] Yu Zhou, Yanxiang Tong, Ruihang Gu, and Harald Gall. "Combining text mining and data mining for bug report classification. "Journal of Software: Evolution and Process. 28(3): pp. 150-176, 2016.

[8] Ashish Sureka and Pankaj Jalote. "Detecting Duplicate Bug Report Using Character N-Gram-Based Features. "Software Engineering, 17th Asia-Pacific Conference. pp. 366-374, 2010.

[9] Mamdouh Alenezi, Kenneth Magel and Shadi Banitaan. "Efficient Bug Triaging Using Text Mining. "Journal of Software: pp. 2185-2190, 2013.

[10] Leon Wu, Boyi Xie, Gail Kaiser, Rebecca Passonneau. "BugMiner: Softwere Reliability Analysis Via Data Mining of Bug Reports. " Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering: pp. 95-100, 2011.