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と共起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：バグ報告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，バグ報告に頻出する単語とそれと共起する単語から頻出するバグを予測する手法を提案する．具体的には，形態素解析によりバグ報告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する．これらのうち，バグ報告に偏って多数出現する (カイ二乗値が大きい) 単語を選び，それらの単語を含むバグ報告が指す場所をバグの集中箇所とする．また，多数出現する単語が一般的なものである場合，集中箇所の予測に役立たない可能性があるので，共起する単語も合わせて計数し，頻出する共起単語により集中箇所を絞り込む．バグ報告に偏って多数出現する単語と共起する単語の両方を含むバグ報告をバグが集中している箇所の予測結果とする．NetCommons のバグ報告とユーザマニュアルに提案手法を適用したところ，単一の単語で多数出現すると判断した 16 語のうち，共起する単語も用いた場合に，9 語の予測精度が大きくなった．

キーワード：形態素解析，ソフトウェアドキュメント，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，開発の中で，実際にバグがある箇所を発見するためには，多くのコストがかかる．これ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ソースコードファイルを特定する方法が考えられる．バグの集中しているソースコードファイルはバグ報告の数も多く，それらのバグ報告に出現する単語も似通ったものになると考えられる．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ソースコードファイルを予測できる可能性がある．ただし，バグ報告に頻出している単語であっても，そのソフトウェアの対象領域を表す等していて，バグの集中しているソースコードファイルを予測できない単語も多く含まれることが推測される．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．筆者らは，文献 [7] にて，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た．そして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た．結果，実際に類似の原因で修正されており，ドキュメントとバグ報告の単語の出現数比較によるバグ集中箇所の予測に一定の効果があることをケーススタディを通して確認した．しかし，バグ報告に頻出する単語が一般的によく使わ

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

© 2021 Information Processing Society of Japan

れるものである場合，その単語を含むバグ報告の中には，類似の原因で修正されていないものも多く含まれていた．したがって，バグ報告に頻出する単語という基準だけで無く，その単語と共起する単語を併せて取り出し，単語二つの組み合わせについても加味することで，予測精度が向上することが期待される．

そこで本稿では，バグ報告に頻出する単語と共起する単語の 2 つの単語の組み合わせを利用したバグ集中箇所の予測手法を提案する．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，オープンソースソフトウェアを対象として，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．得られた単語に共起する単語を取り出し，バグ報告に頻出する単語と共起する単語 2 つの組み合わせの出現する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 概要

提案手法では，ソフトウェアのマニュアル，設計書などのドキュメントに出現する単語の集合 W_D の各出現数に対して，バグ報告に出現する単語の集合 W_B 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．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．出現数が多い単語に関連しているバグ報告を確認し，特定のソースコードファイルが複数のバグ報告で修正されている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．

単語は形態素解析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，開発者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．提案手法の手順を以下に示す．

手順 1 対象ソフトウェアのバグ報告の集合に出現する単語の集合 W_B のうち，ドキュメントに出現する単語 W_D と比べてバグ報告において出現頻度が大きい単語の集合 W_f および $w_f \in W_f$ の出現するバグ報告にて w_f と共起する単語の集合 $C(w_f)$ を得る．

手順 2 $w_f(\in W_f)$ の単語を含むバグ報告の集合 $B(w_f)$ 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ファイルが修正されているかどうか確かめる．

2.2 手順 1

$w \in W_B \cap W_D$ とし，w から，以下の基準でドキュメントと比較してバグ報告に多い単語の集合 W_f を得る．

手順 1.1 バグ報告とドキュメント間において出現頻度に統計的に有意な差がある単語を $W_B \cap W_D$ から取り出す．

$x(w)$ を帰無仮説 “単語 w のバグ報告 B とドキュメント D における出現数は独立である” としたときのカイ二乗値と定義する．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，$x(w) > 3.84$ のとき，単語 w がバグ報告とドキュメントにおいて出現頻度に統計的に有意な差があるとする．$x(w) > 3.84$ に加え，語 w のバグ報告での出現割合が，ドキュメントの出現割合と比較して大きければ，W_f に w を加える．$x(w) > 3.84$ を満たす単語 w は，バグ報告に多い単語とドキュメントに多い単語の両方が含まれているので，その中から，バグ報告の出現比率 $p(B, w)$ が高いもののみを取り出す．ある単語 w のバグ報告での出現回数を $b(w)$，ドキュメントでの出現回数を $d(w)$ とする．

バグ報告に出現する単語 $w_B(\in W_B)$ の出現回数の合計を b_{all}，ドキュメントに出現する単語 $w_D(\in W_D)$ の出現回数の合計を d_{all} とし，それぞれ以下の式で表す．

$b_{all} = \sum_{w_B \in W_B} b(w_B)$

$d_{all} = \sum_{w_D \in W_D} d(w_D)$

単語 w のバグ報告での出現割合 $p(B, w)$，ドキュメントでの出現割合 $p(D, w)$ を，以下のように定義する．

$p(B, w) = b(w)/b_{all}$

$p(D, w) = d(w)/d_{all}$

このとき，W_f は，以下のように表す．

$W_f = \{w | (p(B, w) > p(D, w)) \wedge (x(w) > 3.84)\}$

手順 1.2 W_f 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

手順 1.3 W_f と共起する単語を取り出す

$w_f(\in W_f)$ について，共起する単語 $C(w_f)$ を得る．共起する単語は，単語 $w_f(\in W_f)$ が出現しているバグ報告 $B(w_f)$ のうち，自由記述部分に含まれる単語の集合 W_c から取り出す．単語 $w_f(\in W_f)$，$w_c(\in W_c)$ が出

現しているバグ報告の件数は $|B(w_f) \cap B(w_c)|$ と表せる．バグ報告内の出現位置，出現回数は考慮しない．

2.3 手順 2

$w_f(\in W_f)$ に対して，w_f 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 $B(w_f)$ とする．$B(w_f)$ の各バグ報告の修正ソースコードファイルを確認して，特定の箇所 (ソースコードファイル) 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．バグ報告 $b \in B$ の修正ソースコードファイルの集合を $S(b)$，単語 w_f を含むバグ報告で修正されているソースコードファイルの集合を $\bigcup_{b \in B(w_f)} S(b)$ と定義する．そして，$B(w_f)$ の各バグ報告の修正ソースコードファイルを確認して，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，そのソースコードファイルの集合を $\bigcup_{b \in B(w_f)} S'(b)$ と定義する．$\bigcup_{b \in B(w_f)} S'(b)$ は，以下の式で表せる．

$\bigcup_{b \in B(w_f)} S'(b) = \{s | s \in S(b_n) \wedge S(b_m), b_n \in B(w_f), b_m \in B(w_f), n \neq m\}$

$\bigcup_{b \in B(w_f)} S'(b) \neq \emptyset$ であれば，$\bigcup_{b \in B(w_f)} S'(b)$ に含まれるソースコードファイルを修正しているバグ報告の集合を $B'(w_f)$ とし，w_f はバグが集中していると特定できた単語とする．同様に，手法の実施者が $(w_f, w_c)(w_f \in W_f, w_c \in C(w_f))$ の組み合わせに対して，出現するバグ報告を選び，特定のソースコードファイルに修正が集中しているバグが存在しているか確認する．

3. ケーススタディ

3.1 設定

本評価の目的は，手順 1.1，1.2 を実施した場合より，1.3 まで実施した場合で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ファイルをより効率的に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_f と $C(w_f)$ を含むバグ報告にバグ報告にバグが集中しているソースコードファイルが含まれているかを評価する手順を述べる．

3.1.1 評価 1 w_f のみを用いた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 $w_f(\in W_f)$ の出現するバグ報告 $B(w_f)$ に記録された修正ソースコードファイル名を確認し，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．

表 1 バグ報告の件数，出現単語数

	NetCommons
バグ報告件数	1093
出現単語数	62888

3.1.2 評価 2 w_f と w_c を用いた集中箇所の予測

評価 2 では，w_f と w_c の組み合わせについて，評価 1 と同様に，修正ソースコードファイル名を確認し，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組み合わせがどれだけ存在するかを調査する．また，評価 1 の結果と適合率の比較を行う．適合率は，以下の式で計算する．

$$1 - \frac{|\bigcup_{b \in B(w_f)} S(b)| - |\bigcup_{b \in B(w_f)} S'(b)|}{\sum_{b \in B(w_f)} |S(b)|}$$

3.1.3 評価 3 ソースコードメトリクスによる予測との比較

評価 3 では，評価 2 と同じソフトウェアのソースコードファイル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 2 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．複数のメトリクス値を利用した予測との比較を行う場合， 過去のバグとソースコードファイルを対応づけ予測モデルに与える等の手間が必要となり，研究の初期段階では，大きいほどバグが含まれやすいということが経験的に知られている cyclomatic complexity 値を利用した予測との単純比較で足りるものと判断した．

3.1.4 評価 4 ソースコードファイル修正数の単純比較による予測との比較

評価 4 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ファイル修正数を集計し，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ファイルにバグがどの程度存在していたかを示すものであり，この値の上位ソースコードファイルは実際にバグが多い．

3.1.5 評価対象

評価の対象は，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．評価 1，評価 2，評価 3，評価 4 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 NetCommons とした．NetCommons は国立情報学研究所 NetCommons プロジェクトの教育機関向けサイト構築用システムである．NetCommons ではユーザマニュアルをドキュメント，Github の issue をバグ報告とした．

ドキュメント，バグ報告ともに，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．

© 2021 Information Processing Society of Japan

3.1.6 評価手順

評価 1 w_f のみを用いたバグ集中箇所の予測

評価手順 1.1 提案手法の**手順 1.1〜1.2** にしたがって，B と D を対象に，集合 W_f を得る．単語と出現数の取り出しに，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．取り出した語の集合 W_f 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ソースコードファイルが特定できるかを評価する．本評価において対象としたバグの条件は，**手順 2** の条件に従う．

評価手順 1.2 $B(w_f)$ に含まれる $B'(w_f)$ の数を単語 $w_f(\in W_f)$ ごとに調べる．

評価手順 1.3 $|B'(w_f)| \geq 2$ となる単語が W_f にどの程度の割合存在しているか調べる．

評価 2 w_f と w_c を用いた集中箇所の予測

評価手順 2.1 Netcommons を対象として, 提案手法の**手順 1.3** にしたがって, $w_f(\in W_f)$ と共起する単語の集合 $C(w_f)$ を得る．提案手法の**手順 2** にしたがって，$w_f(\in W_f)$ と $w_c(\in C(w_f))$ の組み合わせについても評価 1.1，評価 1.2 と同様に，バグが集中している箇所を特定できているかを確認する．

評価手順 2.2 $w_f(\in W_f)$ のバグが集中していると判断した単語それぞれが単一で出現している出現バグ報告の修正ソースコードファイルと，共起する単語との組み合わせで出現しているバグ報告の修正ソースコードファイルについて，異なるバグ報告で 2 回以上修正されているソースコードファイルの割合の比較を行う．

共起する語との組み合わせで出現しているバグ報告については，1 位から 5 位のものを合わせて割合を計算する．$w_c(\in C(w_f))$ の 1 位から 5 位を w_{c1}〜w_{c5} とする．$w_f(\in W_f)$ と $w_c(\in C(w_f))$ の出現する異なるバグ報告で 2 回以上修正されているソースコードファイルの集合は $\bigcup_{b\in B(w_f)} S'(b) \wedge \bigcup_{b\in B(w_c} S(b)$ となる．共起する語との組み合わせで出現している異なるバグ報告で 2 回以上修正されているソースコードファイルの数は以下のように計算する．

$$\bigcup_{b\in B(w_f)} S'(b) \ \wedge \ (\bigcup_{b\in B(w_{c1}} S(b) \ \vee \ \bigcup_{b\in B(w_{c2}} S(b) \ \vee \ \bigcup_{b\in B(w_{c3}} S(b) \vee \bigcup_{b\in B(w_{c4}} S(b) \vee \bigcup_{b\in B(w_{c5}} S(b))$$

評価 3 ソースコードメトリクスによる予測との比較

NetCommons について, ソースコードファイルを Github から取得し，ファイルごとのメトリクス値を計測する．本評価では，殆どのソースコードファイルが php で記述されていたため，.php ファイルについては phpmetrics というツールを利用して，メトリクス値を計測した．ツールの適用外である一部の.js ファイルについては計測は行っていない．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 $B'(w_f)$ 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ファイルを確認することで，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 3.1** $B'(w_f)$ に記録されている修正ソースコードファイルが，メトリクス値上位のソースコードファイルとどの程度重複していたか
- **評価 3.2** メトリクス値下位のソースコードファイルが提案手法により得られた $B'(w_f)$ にどの程度存在していたか

表 2 W_f に含まれる語 w_f と出現数 (NetCommons)

w_f	バグ報告	ユーザマニュアル	$x(w_f)$
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

評価 4 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ファイルが記録されている．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

評価手順 1.1 NetCommons のバグ報告に多く出現した語 $w_f(\in W_f)$ を表 2 に示す．表に示している語は，カイ 2 乗値の降順となっている．

評価手順 1.2 $B'(w_f)$ 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．バグ報告 $B(w_f)$ には，それぞれ修正したソースコードファイルが含まれている．その修正されたソースコードの集合は $\bigcup_{b\in B(w_f)} S(b)$ となる．2 つのバグ報告 $b_1, b_2(\in B(w_f))$ が，ソースコードファイル $s(\in \bigcup_{b\in B(w_f)} S(b))$ を修正しているとき，$b_1, b_2(\in B'(w_f))$

表 3 バグが集中していると判断した語 w_f(NetCommons)

w_f	重複バグ報告数
汎用	29
完了	49
更新	6
指定	18
ログイン	23
配置	9
登録	27
タグ	7
新着	12
時間	0
ブロック	12
デフォルト	10
パスワード	10
ボタン	13
項目	23
コンテンツ	4
ダウンロード	16
お知らせ	0

となる．本研究においては，**手順 2** の通り，$B'(w_f)$ の数が 2 つ以上であるとき，単語 w_f を含むバグ報告は特定箇所に集中したバグを含むと考える．

$B(w_f)$ の中で，W_f に含まれるそれぞれの単語が出現するバグ報告で $B'(w_f)$ に分類されるバグ報告の数を表 3 に示す．列 “重複バグ報告数” は，$B'(w_f)$ の数を指す．

評価手順 1.3 NetCommons の W_f において，バグが集中していると判断できた語は 18 語中 16 語であり，その比率は 89%であった．

3.2.2 評価 2 w_f と w_c を用いた集中箇所の予測

評価手順 2.1 $w_f(\in W_f)$ と共起する単語の集合 $C(w_f)$ の上位 5 位と，それぞれの同じソースコードファイルを修正しているバグ報告の数をそれぞれ表 4 に表す．W_f のうち，バグが集中していると判断できなかった語（時間，お知らせ）は除外している．列 $C(w_f)$ の各単語の横の括弧内の数字は，$w_f, w_c(w_f \in W_f, w_c \in C(w_f))$ の両方の出現するバグ報告のうち，同じソースコードファイルを修正しているバグ報告の数を示している．バグが集中していると判断した 16 語と共起する単語の上位 5 位，80 の組み合わせのうち，73 の組み合わせにおいて，w_f が同一であっても共起する単語との組合わせにより別のソースコードファイルが予測できていた．単一の単語の場合には，これらのソースコードファイルは同一の w_f とともに得られるため，共起する単語によりより詳細な集中箇所を予測できていると言える．また，一部の w_f, w_c の組み合わせについて，共起を利用した予測を用いることで，それぞれ別のソースコードファイルを予測できていることを確認した．たとえば，ソースコードファイル Model/MultidatabaseContent.php，Model/AutoUserRegist.php，は，単語「項目」のバグ報告においてバグが集中していると予測したソースコードファイルに含まれるが，このうち Model/AutoUserRegist.php は「項目」「設定」の組み合わせのバグ報告で，Model/MultidatabaseContent.php は「項目」「追加」の組み合わせのバグ報告で予測している．

表 4 バグが集中していると判断した組み合わせ (NetCommons)

w_f	$C(w_f)$ の上位 5 位
汎用	DB(17)，Multidatabases(15)，inbox(12)，作業 (14)，確認 (12)
完了	TO(18)，作業 (26)，修正 (17)，表示 (14)，登録 (12)
更新	思う (2)，修正 (2)，対応 (4)，表示 (2)，設定 (4)
指定	表示 (6)，対応 (7)，思う (0)，修正 (4)，エラー (4)
ログイン	表示 (11)，画面 (5)，管理 (9)，設定 (6)，修正 (4)
配置	プラグイン (6)，誤り (3)，表示 (3)，修正 (3)，ルーム (2)
登録	完了 (18)，修正 (19)，表示 (12)，リリース (10)，画面 (16)
タグ	対応 (5)，設定 (3)，表示 (5)，画面 (2)，修正 (3)
新着	表示 (6)，完了 (5)，修正 (6)，プラグイン (4)，作業 (6)
ブロック	表示 (5)，設定 (2)，画面 (0)，一覧 (3)，プラグイン (0)
デフォルト	表示 (2)，設定 (2)，修正 (4)，変更 (0)，管理 (2)
パスワード	設定 (2)，確認 (2)，場合 (2)，画面 (0)，ログイン (3)
ボタン	表示 (7)，画面 (6)，修正 (6)，対応 (5)，編集 (4)
項目	設定 (7)，表示 (16)，修正 (10)，完了 (12)，追加 (10)
コンテンツ	修正 (2)，対応 (0)，確認 (0)，設定 (0)，表示 (0)
ダウンロード	対応 (5)，修正 (8)，ファイル (11)，確認 (6)，表示 (3)

『列 $C(w_f)$ の上位 5 位』の各単語の横の括弧内の数字は，$w_f, w_c(w_f \in W_f, w_c \in C(w_f)$ の両方に出現するバグ報告のうち，同じソースコードファイルを修正しているバグ報告の数を示している．

評価手順 2.2 単一の単語の場合との適合率の比較の結果を表 5 に示す．この表は，単一の単語を用いた従来手法と共起する単語も用いた本提案手法の比較を行うことで，従来手法にて含まれていたバグが集中していると判断できないバグ報告の割合がどの程度変化したかを示している．結果，共起する単語を用いた場合の予測が単一の単語の場合より，適合率が上回っていたのは 16 語中 9 語であった．

3.2.3 評価 3 ソースコードメトリクスによる予測との比較

評価 3.1 静的解析ツール phpmetrics を利用して NetCommons における php ファイルのメトリクスを計測した．NetCommons では 816 個のソースコードファイルのメトリクス値を計測した．cyclomatic complexity の値 (以下 cc 値) の高いソースコードファイルに，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．cc 値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

表 5 2 回以上修正されているソースコードファイルの割合 W_f(NetCommons)

w_f	単一の単語での予測	共起使用
汎用	0.537	0.550
完了	0.087	0.100
更新	0.100	0.091
指定	0.148	0.134
ログイン	0.370	0.412
配置	0.163	0.168
登録	0.173	0.198
タグ	0.265	0.231
新着	0.220	0.233
ブロック	0.132	0.126
デフォルト	0.171	0.200
パスワード	0.241	0.286
ボタン	0.173	0.160
項目	0.351	0.373
コンテンツ	0.063	0.048
ダウンロード	0.278	0.269

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 $w_f(\in W_f)$ のみを使用した予測の場合 24 件，$w_f(\in W_f)$，$w_c(\in C(w_f))$ を利用した予測の場合 15 件含まれていた．

評価 3.2 cc 値の低いソースコードファイルに，どの程度提案手法によってバグが集中していると予測されたソースコードファイルが含まれているか確かめたところ，NetCommons では，cc 値が 2 のソースコードファイルが 1 つ，1 のソースコードファイルが 4 つ含まれていた．

バグの集中箇所を予測する本研究の手法を，実際にバグが集中しており修正の多かったソースコードファイルの上位において，メトリクス値を利用した予測と精度を比較することで，本研究の予測の精度を評価した．そのうち上位約 2.5%である 20 位までの結果を表 6 に示す．表の “共起 (単一)” “提案手法での順位” の列は，$w_f(\in W_f)$，$w_c(\in C(w_f))$ を用いた場合の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と順位を示している．この値が 2 以上の場合，提案手法において $B'(w_f)$ に含まれていたことを表す．括弧内の数字は，$w_f(\in W_f)$ のみを用いた場合の同一ソースコードファイルの修正件数の最大値と順位である．

“cc 値” の列は，cc 値を表す．

$w_f(\in W_f)$ のみを用いた予測で得られた $B'(w_f)$ 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ファイル 21 個のうち 19 個含まれていた．一方で，$w_f(\in W_f)$，$w_c(\in C(w_f))$ を用いた予測では，21 個のうち 17 個含まれていた．

3.2.4 評価 4 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 3 と同じく表 6 に示す．表 6 では，共起の一位は view/helper/- multidatabasecontentviewelementhelper.php で，“バグ報告数” では 3 位であった．これは単一の場合とも同様である．

4. 考察

4.1 $w_f(\in W_f)$ と $w_c(\in C(w_f))$ を用いたバグ集中箇所の予測 (評価 2)

共起を用いて評価を行ったとき，バグが集中していると判断した 16 語と共起する単語の上位 5 位，80 の組み合わせのうち，73 の組み合わせが，バグの集中箇所を予測できていた．2 つの単語の組み合わせが出現するバグ報告という条件であっても，ある程度，バグの集中箇所を予測できると考えられる．また，一部の組み合わせにおいて，共起を利用した予測を用いることで，それぞれ別のソースコードファイルを予測していた．本研究の意義として，従来の fault-prone 予測と違い，バグがどのようなふるまいをしているかが予測できる可能性がある，というものが挙げられる．表 6 の単純なバグ報告数と提案手法での順位が異なることからも分かるように，実際に，本研究では，一部のバグ報告に頻出する単語から，それぞれ異なったソースコードファイルにバグが頻出していると予測できている．したがって，共起する単語を利用し，さらに対象を細分化した結果，別々で取り出すことができているという結果は，バグのふるまいに対し共起する単語を使用しない場合より具体的に言及できる可能性を示唆していると考えられる．

また，単一の単語での予測と適合率の比較を行った結果，バグが集中していると判断した 16 語のうち 9 語が，共起する単語を用いた場合の予測が単一の単語の場合より，適合率が上回っていた．この結果から，共起する単語を用いた場合の予測が単一の単語の場合より正確な場合もあるということがいえる．共起で取り出した単語は，「表示」や「設定」など同じ単語が多く見られた．これは，共起の特性上同じバグ報告に多く含まれていれば取り出されるので，単純にバグ報告に多く出現する単語が多くなると考えられる．したがって，特定の単語により関連した取り出し方を行うことで，また違った結果が得られる可能性がある．他のソフトウェアについても同様の比較を行うことや，共起以外の関連する単語の取り出しを行って評価を行うことは今後の課題である．

4.2 ソースコードメトリクスの予測との比較 (評価 3)

評価対象のソフトウェア NetCommons において，ソースコードファイルごとに cc 値を計測し，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 $w_f(\in W_f)$ のみを使用した予測の場合 24 件，$w_f(\in W_f)$，$w_c(\in C(w_f))$ を利用した予測の場合 15 件含まれていた．また，cc 値が小さいもののうち，提案手法がバグの集中

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

順位	ファイル名	バグ報告数	共起 (単一)	提案手法での順位	cc 値	cc 順位
1	configschema/schema.php	20	2(3)	29(15)	1	817
2	utility/currentframe.php	11	2(3)	29(15)	23	119
3	view/helper/multidatabasecontentviewelementhelper.php	10	8(10)	1(1)	22	126
3	utility/current.php	10	3(3)	13(15)	75	3
5	utility/installutil.php	9	2(2)	29(37)	53	8
5	model/taskcontent.php	9	3(3)	13(15)	50	12
5	model/behavior/saveuserbehavior.php	9	7(7)	2(2)	46	18
5	model/user.php	9	2(2)	29(37)	39	31
5	model/behavior/nc2tonc3wyswygbehavior.php	9	2(2)	29(37)	26	102
5	view/helper/menuformhelper.php	9	7(7)	2(2)	17	168
5	config/route.php	9	1(1)	55(45)	1	817
5	controller/component/downloadcomponent.php	9	1(5)	55(7)	11	260
5	testsuite/netcommonssavetest.php	9	1(1)	55(45)	6	364
14	controller/pageseditcontroller.php	8	2(2)	29(37)	58	5
14	model/usersearch.php	8	1(2)	55(37)	52	9
14	controller/cabinetfileseditcontroller.php	8	2(2)	29(37)	42	22
14	model/multidatabasecontent.php	8	7(7)	2(2)	35	48
14	controller/component/roomsrolesformcomponent.php	8	3(3)	13(15)	33	59
14	controller/taskcontentscontroller.php	8	2(2)	29(37)	30	77
14	test/case/config/routestest.php	8	3(3)	13(15)	1	817
14	utility/currentpage.php	8	2(2)	29(37)	58	5

を予測したソースコードファイルも存在した．$w_f(\in W_f)$，$w_c(\in C(w_f))$ を利用した予測の場合，$w_f(\in W_f)$ のみを使用した予測と比較して，cc 値が高いソースコードファイルが含まれていない．

一方で，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ソースコードファイルを予測できる可能性がある．また，$w_f(\in W_f)$ のみを使用した予測，$w_f(\in W_f)$，$w_c(\in C(w_f))$ を利用した予測それぞれについて，ソースコードファイルを修正回数に基づいて順位付けし，cc 値の順位と比較したところ，順位に特に大きな変化は見られなかった．ただし，$w_f(\in W_f)$，$w_c(\in C(w_f))$ を利用した予測の方が修正回数が減るため，横並びの順位になりやすい傾向があると考えられる．

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 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 (評価 4)

提案手法による予測 1 位のソースコードファイルの，すべてのバグ報告から集計した修正頻度の順位は，$w_f(\in W_f)$ のみを使用した予測において 3 位であった．また，$w_f(\in W_f)$，$w_c(\in C(w_f))$ を利用した予測の場合においても同じ順位であった．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．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ことを意味する．ゆえに，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ファイル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ファイル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ファイルが取り出しやすいと考えられる．こちらも，cc 値との比較と同様に，また，$w_f(\in W_f)$ のみを使用した予測，$w_f(\in W_f)$，$w_c(\in C(w_f))$ を利用した予測それぞれについて，ソースコードファイルを修正回数に基づいて順位付けし，順位を比較したところ，共起する単語を用いた事に対して順位に特に大きな変化は見られなかった．評価 3 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 cc 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

© 2021 Information Processing Society of Japan

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 [8][9][10]．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 [11]．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を他のドキュメントと比較することで，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ファイル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ファイルを特定する．また，共起する単語も計数し，集中箇所の予測精度を評価した．その結果，同一ソースコードファイルの特定箇所を 2 回以上修正しているものが 16 語あった．

提案手法では 16 語それぞれと共起する単語上位 5 語を取り出し，その語の組み合わせを含むバグ報告の修正箇所を確かめた．提案手法でも，単一の単語による予測と同様に，16 語全てが同一ソースコードファイルの特定箇所を 2 回以上修正していることが確認できた．提案手法では，80 通りの組み合わせのうち 73 通りが同一ソースコードファイルを 2 回以上修正していた．この結果より，バグ報告をフィルタリングしてもバグ集中箇所を予測できることから，確認するべきバグ報告を減らすことができると考えられる．

また，単一の単語による予測と比較して，提案手法では 16 語のうち 9 語が予測ソースコードファイルの適合率が高いことが分かった．このことから，提案手法を利用する事で，予測精度が向上する可能性がある．さらに，各ソフトウェアのソースコードモジュールの cc 値を求め，上位 50 件のソースコードファイル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，単一の単語での予測の場合 24 件，共起を用いた予測の場合 15 件一致していた．cc 値による Fault-prone モジュール予測でもバグを含む可能性が高いと予測されたソースコードファイルを提案手法でも予測できており、Fault-prone モジュール予測では予測できないソースコードファイルも含まれていた．

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] 山内泉水，森崎修司　「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」，第 207 回ソフトウェア工学研究発表会，in March ,2021.

[8] 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.

[9] 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.

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

[11] 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.

© 2021 Information Processing Society of Japan