情報処理学会論文誌 Vol.60 No.4 1040–1050 (Apr. 2019)

情報検索に基づくBug Localizationへの不吉な臭いの利用

高橋 碧[1,a)] セーリム ナッタウット[1,b)] 林 晋平[1,c)] 佐伯 元司[1,d)]

受付日 2018年8月1日, 採録日 2019年1月15日

概要：大規模なソフトウェア開発では，ある特定のバグを解決するために修正すべきソースコード箇所を見つける Bug Localization が必要である．情報検索に基づく Bug Localization 手法（IR 手法）は，バグに関して記述されたバグレポートとソースコード内のモジュールとのテキスト類似度を計算し，これに基づき修正すべきモジュールを特定する．しかし，この手法は各モジュールのバグ含有可能性を考慮していないため精度が低い．本論文では，ソースコード内のモジュールのバグ含有可能性として不吉な臭いを用い，これを既存の IR 手法と組み合わせた Bug Localization 手法を提案する．提案手法では，不吉な臭いの深刻度と，ベクトル空間モデルに基づくテキスト類似度を統合した新しい評価値を定義している．これは深刻度の高い不吉な臭いとバグレポートとの高いテキスト類似性の両方を持つモジュールを上位に位置付け，バグを解決するために修正すべきモジュールを予測する．4 つの OSS プロジェクトの過去のバグレポートを用いた評価では，いずれのプロジェクト，モジュール粒度においても提案手法の精度が既存の IR 手法を上回り，クラスレベルとメソッドレベルでそれぞれ平均 22%，137%の向上がみられた．また，不吉な臭いが Bug Localization に与える影響について調査を行った．

キーワード：Bug Localization，情報検索，不吉な臭い，バグ含有可能性

Using Code Smells to Improve Information Retrieval-based Bug Localization

Aoi Takahashi[1,a)] Natthawute Sae-Lim[1,b)] Shinpei Hayashi[1,c)] Motoshi Saeki[1,d)]

Received: August 1, 2018, Accepted: January 15, 2019

Abstract: Bug localization is a technique that has been proposed to support the process of identifying the locations of bugs specified in a bug report. For example, information retrieval (IR)-based bug localization approaches suggest potential locations of the bug based on the similarity between the bug description and the source code. However, while many approaches have been proposed to improve the accuracy, the likelihood of each module having a bug is often overlooked or they are treated equally, whereas this may not be the case. For example, modules having code smells have been found to be more prone to changes and bugs. Therefore, in this paper, we propose a technique to leverage code smells to improve bug localization. By combining the code smell severity with the textual similarity from IR-based bug localization, we can identify the modules that are not only similar to the bug description but also have a higher likelihood of containing bugs. Our case study on four open source projects shows that our technique can improve the baseline IR-based approach by 22% and 137% on average for class and method levels, respectively. In addition, we conducted investigations concerning the effect of code smell on bug localization.

Keywords: bug localization, information retrieval, code smell, bug proneness

1. はじめに

大規模なソフトウェア開発では，開発者は多くのバグを修正しなければならない．バグ報告に基づくバグレポートが多数作成され，それらを手動で解決するためには多くの時間を要する．また，バグを修正するために最も時間がか

[1] 東京工業大学情報理工学院
School of Computing, Tokyo Institute of Technology, Meguro, Tokyo 152–8550, Japan
[a)] takahashi-a-at@se.cs.titech.ac.jp
[b)] natthawute@se.cs.titech.ac.jp
[c)] hayashi@c.titech.ac.jp
[d)] saeki@c.titech.ac.jp

かる作業の 1 つは，バグレポートに書かれたバグの原因となるソースコード箇所の特定である．Thung ら [1] は，バグの位置特定が困難であることをソフトウェアの分析から明らかにしている．

上記の課題に対処する手段の 1 つに，Bug Localization がある．Bug Localization は，バグを解決するために修正すべきソースコード箇所を特定することを指し，情報検索 (Information Retrieval; IR) [2], [3], [4]，静的解析 [5]，動的解析 [6], [7] などに基づく手法がある．また，これらを統合した手法 [8], [9], [10] も提案されている．

情報検索に基づく手法（IR 手法）では，報告されたバグレポートとソースコードとのテキスト類似度を計算することで，バグの修正箇所を特定する．このように，IR 手法は，対象とするバグレポートとソースコードのみを入力として用いる，適用するための制約が少ない手法である一方，得られる精度に課題がある．その理由の 1 つに，IR 手法が主に入力のテキスト類似度に注目しており，結果として特定されるモジュールにどの程度バグが含まれやすいのかを考慮していないことがある（詳細は 2 章）．

本論文では，Bug Localization 手法の 1 つである IR 手法に加えて，不吉な臭いの情報も組み合わせることで IR 手法の精度を向上させる手法を提案する．以下に本論文の貢献を述べる．

- 既存の IR 手法の問題点を，オープンソースソフトウェアの実例を用いて示した．
- IR 手法とソースコードに存在する不吉な臭いの情報を組み合わせる Bug Localization 手法を提案した．本手法は，ソースコード以外の追加的な入力なしに，IR 手法の精度を向上できる．
- 4 つの OSS に関して評価実験を行い，IR 手法に比べて本手法ではすべてのプロジェクトで精度が向上することを示した．
- 不吉な臭いが Bug Localization の精度に影響を与えた要因について複数の観点から調査した．

本論文は，ICPC 2018 での我々の発表 [11] の内容を発展させたものである．既存の IR 手法の問題点を具体例を用いて示した点（2 章）と，不吉な臭いが Bug Localization の精度に影響を与えた要因について調査を行った点（5.5 節）が主要な拡張箇所である．

本論文の以降の構成を以下に示す．2 章では，提案手法を用いる動機を，例を用いて述べる．3 章では，Bug Localization に関する既存研究を紹介する．4 章では，Bug Localization と不吉な臭いを組み合わせた手法について説明する．5 章では，提案手法の有効性を検証するための評価実験の内容とその結果の考察を行う．6 章では，本論文のまとめと今後の課題について記す．

2. 動機

IR 手法の精度が高くない理由の 1 つに，IR 手法が主に入力のテキスト類似度に注目しており，結果として特定されるモジュールにどの程度バグが含まれやすいのかを考慮していないことがある．**表 1** は，オープンソースソフトウェア（OSS）ArgoUML において，バグレポート[*1]（ID：3790）の要約（“Exception when saving to a readonly directory.”）とその説明を入力として，クラス粒度で IR 手法を適用した結果の抜粋である．ランキング上位のクラスは入力のバグレポートとのテキスト類似性が高く，たとえばクラス ActionSaveProject は `save` を含む多くの単語を共有している．しかし，実際の正解にこのクラスは含まれず，10 位の ProjectBrowser が正解に該当する．ここで注目すべき点は，このクラスは他のクラスよりもテキスト類似性に乏しいが，複数の不吉な臭い [12] を持っていることである．不吉な臭いとは，低品質な設計によりリファクタリングを必要とするソースコード箇所であり，その存在がソフトウェアの保守や理解に悪影響を与えることが示されている [13], [14]．また，多くの種類が Fowler の文献 [12] の中で紹介されているが，それらの不吉な臭いに密度 [15] や深刻度 [16] を割り当て，不吉な臭いに優先順位付けを行う手法もある．さらに，不吉な臭いを持つモジュールにはバグが含まれやすいことも示されている [17], [18]．ProjectBrowser クラスからは，行数の長さに起因する Blob Class と，責務の過多による複雑性に起因する God Class の 2 つの不吉な臭いが検出されており，さらにそれらの深刻度はそれぞれ 3，5 と高い．こういった不吉な臭いの特徴，特に詳細な順位付けが可能だと考えられる不吉な臭いの深刻度を，テキスト類似度に加えて考慮することで，このクラスを高順位に引き上げられる可能性がある．

表 1 IR 手法の適用例
Table 1 Application example of IR-based bug localization.

順位	クラス名	テキスト類似度
1	ActionSaveProject	0.179
2	AbstractFilePersister	0.174
3	ZipFilePersister	0.150
4	XmiFilePersister	0.150
5	UmlFilePersister	0.142
6	ProjectFilePersister	0.128
7	FileConstants	0.123
8	ActionSaveProjectAs	0.122
9	ActionOpenProject	0.121
10	ProjectBrowser	0.121

[*1] http://argouml.tigris.org/issues/show_bug.cgi?id=3790

3. 関連研究

3.1 IR に基づく Bug Localization 手法

IR 手法は Bug Localization の分野で幅広く用いられており，ソフトウェア開発の際に作成されたバグに関する記述であるバグレポートと，現在のソースコードの 2 つを入力として用いる．これまでに，Latent Dirichlet Allocation (LDA) [2], [19], Latent Semantic Indexing (LSI) [3], [20], Vector Space Model (VSM) [4] など，多くの IR 手法が提案されている．Rao ら [21] は，これらの代表的な手法の中でも VSM が最も効果的であると述べている．この VSM に加えて，過去の類似したバグレポートを用いた BugLocator [22] が Zhou らによって提案されている．

IR 手法では，バグレポートの質に強く依存するため，情報不足のバグレポートでは，良い結果が得られないことが問題となる．Kim ら [23] はバグレポートの質が十分かを判定し，十分な質であれば IR 手法を適用してバグの箇所を予測し，そうでなければ予測を行わない手法を提案している．Le ら [24] も IR 手法に基づくツールでの結果が効果的かを判定する手法を提案している．これらの手法を用いれば，開発者は IR 手法が誤って提案してしまったモジュールに時間をとられずに済む．ほかにも Chaparro ら [25] は，低品質なバグレポートを再構成する手法を提案している．

3.2 その他の Bug Localization 手法

静的解析手法は古くから用いられており，その典型的なものに静的プログラムスライシング [5] がある．静的プログラムスライシングでは，ある関心事に対応するスライス基準を与え，プログラム中の依存関係に注目して，関心事に関係のあるモジュール群を抽出する．この手法では，開発者がスライス基準を与えなければならない点，依存関係のあるモジュールが膨大となる点など，実践には多くの課題が残っている．Acharya ら [26] は，近年の大規模なソフトウェア開発に対するプログラムスライシング技術の問題点とその改善案について言及している．

動的解析手法では，実際にプログラムを実行した際に得られる情報を用いて，Bug Localization を行う．Wong ら [7] はスタックトレースを用いたツール BRTracer を提案した．スタックトレースには，テストに失敗するまでの間にどのような処理をどのような順番で呼び出したのかが表現されており，この情報を活用する．また，プログラムの実行トレースから，実行に成功したトレースと失敗したトレースの間のスペクトル統計情報を分析したスペクトルベースの Bug Localization も提案されている [27]．Dao ら [6] は，IR 手法に基づく Bug Localization に動的な実行情報がどのように役立つかを調査している．この調査では，失敗したテストのカバレッジや，動的なスライス情報が IR 手法の検索範囲を効果的に減らすなど，改善が見られることが報告されている．動的解析手法の特徴として，高い精度を持つ代わりに，開発者が実際にプログラムを実行する必要があり，開発者に高い知識と時間的な手間を要求することとなる．

過去の変更履歴を効果的に用いる手法も提案されている．Tantithamthavorn ら [28] は同時変更履歴を用いて，あるモジュールが変更された際に，同時に変更される可能性の高いモジュールを計算することで Bug Localization の精度を向上させている．

複数の手法を統合した手法も提案されている．Shi ら [29] は IR 手法に基づく Bug Localization に追加的な情報を付与し統合することが有益であると述べている．Saha ら [8] はソースコード構造に基づいた情報を用いて IR 手法を適用したうえで，利用可能であれば過去の類似のバグレポートを用いるツール BLUiR を提案した．ほかにも，Wang ら [9] は，BugLocator [22] と BLUiR [8]，さらに Bug Prediction ベースの変更履歴を統合したツール AmaLgam を提案している．さらに，Youm ら [10] はバグレポートやソースコード構造情報，変更履歴，スタックトレースを用いた動的情報を統合したツール BLIA を提案している．これらの追加的な情報を含める手法は IR 手法に比べて大きく精度を向上させてきた．ただし，これらの追加的な情報はつねに利用可能であるとは限らず，開発者の入力を必要とするものもある点に注意が必要である．

4. 提案手法

4.1 概要

提案手法の概要を図 1 に示す．提案手法では，バグレポートとソースコードを入力として IR 手法を適用し，ソースコード内のモジュールごとにバグレポートとのテキスト類似度を計算する．また，ソースコードを入力として不吉な臭いを検出し，不吉な臭いの深刻度 [16] を得る．不吉な臭いの深刻度とは，不吉な臭いの深刻さ・悪さを数値化し

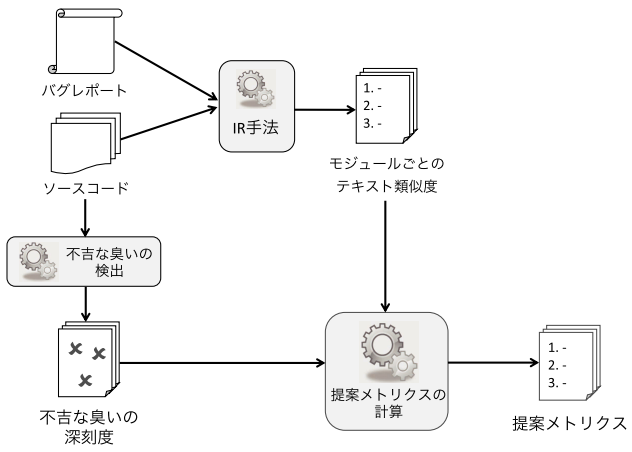

図 1 提案手法の概要
Fig. 1 Overview of proposed technique.

表 2 ArgoUML に対する適用例
Table 2 Application example to ArgoUML.

(a) IR 手法（$\alpha = 0$）

順位	クラス名	*nSim*	*nSev*	*BLI*
1	ActionSaveProject	1	0	1
2	AbstractFilePersister	0.971	0	0.971
3	ZipFilePersister	0.842	0	0.842
4	XmiFilePersister	0.841	0	0.841
5	UmlFilePersister	0.795	0	0.795
6	ProjectFilePersister	0.715	0	0.715
7	FileConstants	0.691	0	0.691
8	ActionSaveProjectAs	0.683	0	0.683
9	ActionOpenProject	0.679	0	0.679
10	ProjectBrowser	0.679	0.667	0.679

(b) 提案手法（$\alpha = 0.42$）

順位	クラス名	*nSim*	*nSev*	*BLI*
1	ProjectBrowser	0.679	0.667	0.674
2	ActionSaveProject	1	0	0.580
3	AbstractFilePersister	0.971	0	0.563
4	ZipFilePersister	0.843	0	0.489
5	XmiFilePersister	0.842	0	0.488
6	Modeller	0.222	0.833	0.479
7	UmlFilePersister	0.794	0	0.461
8	Import	0.430	0.500	0.460
9	ParserDisplay	0.051	1	0.449
10	StylePanel	0.586	0.250	0.445

たものである．テキスト類似度と不吉な臭いの深刻度の両方を組み合わせることで提案メトリクスが計算される．各モジュールを提案メトリクスの高い順に並べ替えたリストを作成し，そのリストを参考に開発者はバグの修正を試みる．

4.2 IR 手法

IR 手法では，対象プロジェクトのバグの内容が記述されたバグレポートとソースコードを入力とする．これらの入力から，ソースコードのモジュールごとにバグレポートとのテキスト類似度が計算される．対象とするモジュール群 M のうち，特定のモジュール m とバグレポートとのテキスト類似度を $Sim(m)$ として得る．この $Sim(m)$ を，以下のように [0, 1] に線形正規化して用いる．

$$nSim(m) = \frac{Sim(m)}{\max_{m' \in M} Sim(m')}$$

4.3 不吉な臭いの検出

対象プロジェクトのソースコードを入力として，不吉な臭いを検出する．検出された不吉な臭いは，その種類，対象モジュール，深刻度（Severity）などの情報をともなっている．提案手法では不吉な臭いの情報の定量化方法としてこの深刻度を用いる．これは，単に不吉な臭いを持つか否かよりも細かく優先順位付けを行える分解能を与えるためである．

対象とするモジュール群 M のうち，特定のモジュール m で検出されたすべての不吉な臭いの深刻度の合計 $Sev(m)$ を計算する．ただし，Bug Localization の対象モジュール粒度と同じ粒度の不吉な臭いのみを用いる．この $Sev(m)$ を，以下のように [0, 1] に線形正規化して用いる．

$$nSev(m) = \frac{Sev(m)}{\max_{m' \in M} Sev(m')}$$

4.4 提案メトリクスの計算

モジュールごとに，*nSim* と *nSev* を用いて以下のように提案メトリクスを表す *Bug Likelihood Index*（*BLI*）を計算する．

$$BLI(m) = (1 - \alpha) \times nSim(m) + \alpha \times nSev(m)$$

ここで α（$0 \leq \alpha \leq 1$）は用いている *nSim* と *nSev* の重要度の分配を表すパラメータであり，利用者が経験的に与えたり，適用したいプロジェクトの過去のデータや他のプロジェクトの値などから定める．$\alpha = 0$ のとき，*BLI* が表すモジュール順位は IR 手法の結果と等しくなる．一方，$\alpha = 1$ のとき，順位には不吉な臭いの深刻度のみが反映される．

4.5 適用例

VSM に基づくテキスト類似度の計算に TraceLab [30]，不吉な臭いの検出に inFusion [*2] を用いて，提案手法を適用した例を示す．付録 A.1 章には，inFusion により検出される，粒度がクラスレベルとメソッドレベルの不吉な臭いの一覧が含まれている．**表 2** は，表 1 で入力として用いたバグレポートに対する適用結果の上位 10 を示しており，IR 手法（$\alpha = 0$）と提案手法（$\alpha = 0.42$）の結果を比較している．バグレポートに対して実際に変更された正解クラスは灰色で強調されている．IR 手法では正解のクラス ProjectBrowser が 10 位に位置付けられているが，このクラスは不吉な臭いを持っており，*nSev* の値が高かったため，提案手法を用いた場合，1 位に位置付けられるようになった．この例では，不吉な臭いの深刻度を用いることで，実際に変更されたクラスを IR 手法より高順位に位置付けることができている．

5. 評価実験

提案手法の評価のため，以下の 3 つの Research Question（RQ）を定めた．

[*2] http://www.intooitus.com/products/infusion
（販売は中止されており，現在はアクセスできない）

表 3 プロジェクトの情報
Table 3 Projects used.

プロジェクト	バージョン	バグレポート数	クラス数	メソッド数	クラスの臭い数	メソッドの臭い数
ArgoUML	0.20–0.24	74	1,476	12,131	61	411
JabRef	2.0–2.6	36	374	2,947	37	60
jEdit	4.2–4.3	86	406	5,276	51	185
muCommander	0.8.0–0.8.5	81	529	3,916	41	44

RQ_1：重要度のパラメータ α は手法の精度にどのように影響するか？ α は提案手法において不吉な臭いの情報と既存の IR 手法の情報の結合割合であり，この値の影響を調べることは 2 つの情報のそれぞれの重要度を求めることに相応する．また，実際に手法を利用する際に α を定めるためにも，最適な α を求めておくことは重要である．

RQ_2：既存の IR 手法と比べ，提案手法は優れているか？ 実際に提案手法を用いることで既存の IR 手法よりも優れているかを調べることにより，不吉な臭いの情報を既存の IR 手法に組み合わせることが効果的であり，精度を向上させるかを検証する．α を定める際には，RQ_1 の結果を利用する．具体的には，複数の既存プロジェクトの α の最適値が既知である状況を想定し，特定のプロジェクトの α には，他の 3 プロジェクトの α の最適値の平均を用いる．

RQ_3：不吉な臭いの利用による精度変化の要因は何か？ 提案手法による不吉な臭いの利用がどのような要因により精度の変化に影響を及ぼしたのかを，不吉な臭いのモジュール分布およびバグレポートごとの精度の変化の 2 つの観点から分析する．また，これらの分析により，提案手法の特徴や今後の課題を明らかにする．

5.1 データ収集

本実験では，Dit らの変更影響分析のデータセット [31] を用いた．このデータセットには，4 つのオープンソースソフトウェアプロジェクトに対して，特定バージョンのソースコード，次期バージョンまでに解決されたイシュー，イシュー解決時に変更されたモジュール（正解セット）が含まれている．本実験では，バグカテゴリに属するイシューをバグレポートとして使用した．各プロジェクトの情報を**表 3** に示す．不吉な臭いは，データセットに含まれる変更前バージョンのソースコードを inFusion に入力として与えて得た．各モジュール粒度で得られた不吉な臭いの検出数も表 3 に示している．また変更前バージョンのソースコードは，バグレポートとの VSM を用いたテキスト類似度の計算にも用いた．

5.2 評価メトリクス

提案手法では，提案メトリクス *BLI* に基づいて各モジュールを順位付けしたリストが得られる．これと，データセットから得られる正解セットとを比較し，順位付けリストの精度の評価を行う．これには，順序付きランキングを評価するために用いられる精度メトリクスである Mean Average Precision（MAP）を用いた．MAP はすべてのバグレポートの Average Precision（AP）の平均である．1 つのバグレポートの AP は以下のように計算される．

$$AP = \sum_{i=1}^{N} \frac{Pre(i) \times pos(i)}{\text{正解セットのモジュール数}}$$

ここで，i は順序付けされたモジュールの順位，N は順序付けしたモジュールの総数を表す．$Pre(i)$ は i を含む i よりも順位の高いモジュールにおける正解セットのモジュールの割合を表す．$pos(i)$ は i 番目にランク付けされたモジュールが正解セットに含まれていれば 1，さもなくば 0 を返す．MAP は，与えられたすべてのバグレポートでの AP の平均値となる．

5.3 RQ_1：重要度のパラメータ α は手法の精度にどのように影響するか？

5.3.1 調査方法

RQ_1 に答えるために，α を横軸，MAP を縦軸とするグラフを描く．α を 0.01 刻みで 0 から 1 まで変化させ，それぞれの *BLI* を計算し各モジュールを順位付けた場合の MAP を計算する．このグラフをモジュール粒度がクラスレベルとメソッドレベルの両方で対象プロジェクトに関して作成する．これらにより得られたグラフの形状および MAP が最大値となる α の値について議論する．

5.3.2 結果と考察

結果を**図 2** に示す．また，それぞれのグラフの α の最適値，すなわち MAP が最大となる α の値を**表 4** に示す．最も重要な結果として，既存の IR 手法のみを用いたり（$\alpha = 0$），不吉な臭いのみを用いたり（$\alpha = 1$）するのではなく，それら 2 つを組み合わせた場合の方が効果的だったことがある．クラスレベルの結果では，α が 0.25–0.46 の値で MAP が最大となった．メソッドレベルの結果では，α が 0.41–0.47 の値で MAP が最大となった．メソッドレベルの結果では，クラスレベルの結果に比べて α の最適値が広範囲にわたっていない．一方で，クラスレベルでは，JabRef のみ他のプロジェクトに比べて α の値が明らかに小さい．その理由として，表 3 を見ると分かるよう

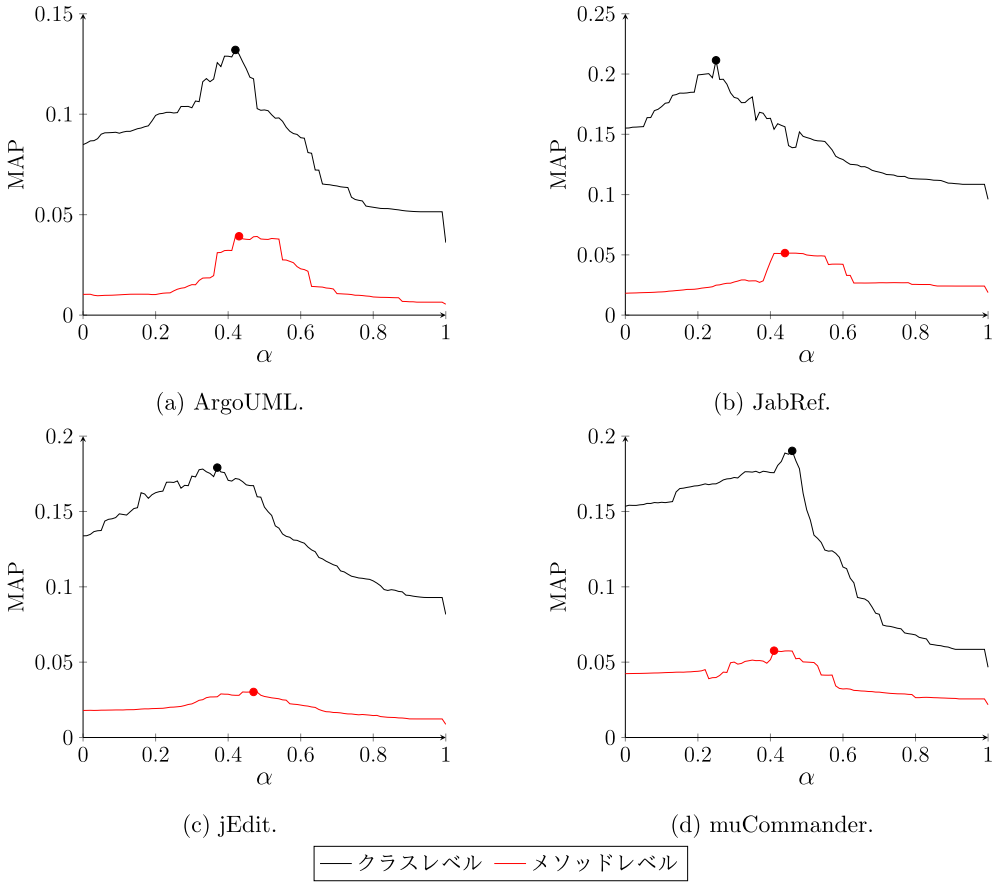

図 2 α ごとの MAP の値
Fig. 2 MAP values for each α.

表 4 α の最適値
Table 4 Most appropriate α values.

プロジェクト	クラスレベル	メソッドレベル
ArgoUML	0.42	0.43
JabRef	0.25	0.44
jEdit	0.37	0.47
muCommander	0.46	0.41

に，JabRef ではクラスレベルの不吉な臭いの検出数が全プロジェクトの中で最も少なく，不吉な臭いの影響が大きくならなかった可能性がある．しかし，muCommander との検出数の差はわずか 4 であり，さらにクラス数に対する不吉な臭いの検出数の割合は ArgoUML が最も少ないため，原因の特定にはより詳細な分析が必要である．JabRef のクラスレベルでの結果を除き，α の最適値は 0.40 付近に集中しており，この値は経験的に他のプロジェクトにも利用できる可能性があると考える．

> いずれの試行においても，テキスト類似度，不吉な臭いの両情報を適切に組み合わせた際（クラスレベルでは $\alpha \in [0.25, 0.46]$，メソッドレベルでは $\alpha \in [0.41, 0.47]$）に MAP が最大となった．

表 5 RQ_2 で用いた α の値
Table 5 α values used in answering RQ_2.

プロジェクト	クラスレベル	メソッドレベル
ArgoUML	0.36	0.44
JabRef	0.42	0.44
jEdit	0.38	0.43
muCommander	0.35	0.45

5.4 RQ_2：既存の IR 手法と比べ，提案手法は優れているか？

5.4.1 調査方法

RQ_1 の結果を用いて α を実際に定め，提案手法を適用する．ここでは，あるプロジェクトを予測する際には，他の 3 プロジェクトの α の最適値の平均を用いた．たとえば，対象プロジェクトが ArgoUML の場合には，JabRef, jEdit, muCommander の 3 つのプロジェクトの RQ_1 で求めた α の最適値の平均を α として定めて提案手法を適用した．これは，複数の既存プロジェクトの α の最適値が既知である状況で対象プロジェクトの α を定める場合を想定している．実際に用いた α の値を**表 5** に示す．このようにして MAP を求め，提案手法が既存の IR 手法（$\alpha = 0$）に比べて優れているかを検証する．

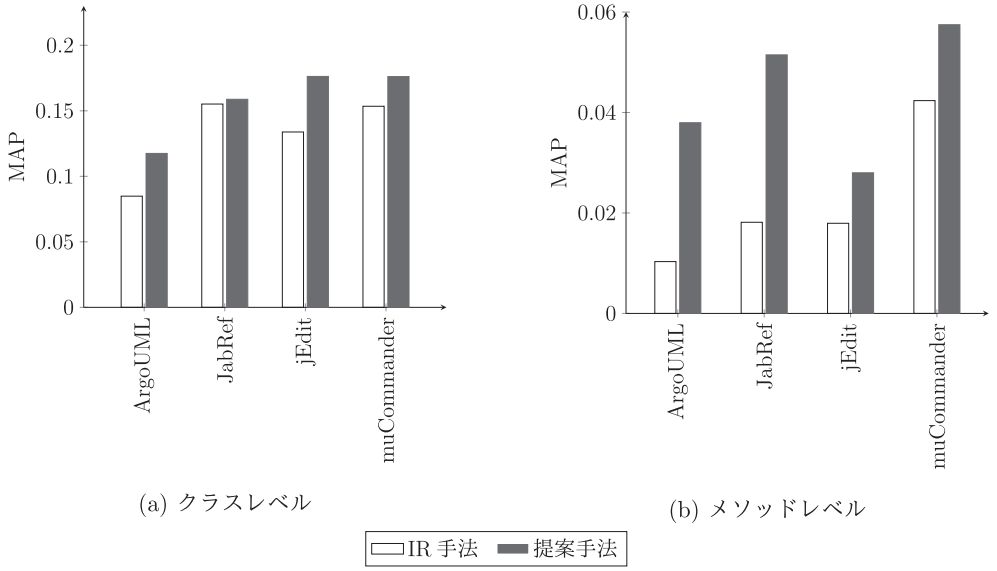

図 3 MAP の値の比較
Fig. 3 Comparison of MAP values.

5.4.2 結果と考察

結果を**図 3** に示す．既存の IR 手法に対する提案手法の MAP の上昇率は，クラスレベルで 39%，2%，32%，15%，メソッドレベルで 269%，185%，56%，36%であった．いずれのプロジェクト，モジュール粒度でも，既存手法に比べて提案手法の MAP が向上した．ただし，全体としてクラスレベルの方が，上昇率が低かった．この原因に，α の最適値の多様性の差がある．クラスレベルの JabRef から得た α の最適値は表 4 から 0.25 であるが，この値は他の 3 プロジェクトから得た α とは大きく離れている．メソッドレベルでは，α の最適値は全プロジェクトで近い値をとっているため，この差が上昇率の差を生んでいる．また，図 2 からも分かるように，メソッドレベルよりクラスレベルの方が MAP の絶対値が高いこともあげられる．一方で，メソッドレベルではプロジェクトごとの α の最適値が近いため，MAP の上昇率が高くなったと考える．

> すべての試行において，提案手法の MAP が既存の IR 手法を上回った．クラスレベルでは平均 22%（最大では 39%），メソッドレベルでは平均 137%（最大では 269%）の向上があった．

5.5 RQ_3：不吉な臭いの利用による精度変化の要因は何か？

前述した，提案手法による MAP の向上がどのような要因によって起こったのかを詳細に知るために，いくつかの分析を行った．

5.5.1 不吉な臭いの分布

提案手法がなぜ効果的に機能したかを直感的に示すために，バグレポートに対して実際に変更されたモジュール（正解セット）とそうでないモジュールが，どの程度不吉な臭いを持っていたかを調べた．具体的には，正解セットに含まれるモジュール，および，正解セットに含まれないモジュールの中で，不吉な臭いが検出されたものの割合

$$SR = \frac{\sum_{r \in R} |O_r \cap M^s|}{\sum_{r \in R} |O_r|}, \quad SR^* = \frac{\sum_{r \in R} |M^s \setminus O_r|}{\sum_{r \in R} |M \setminus O_r|}$$

をそれぞれ求めた．ここで，R はバグレポート全体の集合，M はモジュール全体の集合，M^s は不吉な臭いを持つモジュールの集合，O_r はバグレポート r の正解セットを指す．

表 6 不吉な臭いを含むモジュールの割合
Table 6 Ratios of modules having a code smell.

	クラスレベル		メソッドレベル	
プロジェクト	SR	SR^*	SR	SR^*
ArgoUML	21.43%	4.11%	7.10%	3.99%
JabRef	36.07%	9.03%	11.11%	2.77%
jEdit	51.14%	9.13%	15.57%	3.25%
muCommander	31.71%	6.59%	7.22%	1.35%
全プロジェクト	32.63%	5.86%	9.52%	3.26%

結果を**表 6** に示す．表の値から，いずれのプロジェクトにおいても，SR^* に比べて SR が自明に高いことが分かる．つまり，正解セットに含まれるモジュールはより高い割合で不吉な臭いを持っているということであり，これは Bug Localization に不吉な臭いの情報を用いることが効果的であることを示唆している．

また，不吉な臭いの分布に関して，臭いの種類が提案手法に与える影響についてより詳細に分析するために，同様の調査を，不吉な臭いの種類ごとにも行った．具体的には，前述した，不吉な臭いが検出されたモジュールの割合 SR,

表 7 不吉な臭いを含むモジュールの割合（種類ごと）
Table 7 Ratios of modules having a code smell per type.

粒度	不吉な臭いの種類	SR_t	SR_t^*
クラスレベル	God Class	25.59%	3.09%
	Blob Class	7.04%	0.72%
	Schizophrenic Class	4.23%	0.96%
	Refused Parent Bequest	0.70%	0.12%
	Tradition Breaker	0.23%	0.37%
	Data Class	0.23%	1.31%
メソッドレベル	Blob Operation	7.00%	1.06%
	Intensive Coupling	2.51%	0.64%
	Internal Duplication	0.36%	0.84%
	Sibling Duplication	0.36%	0.37%
	Feature Envy	0.36%	0.10%
	Data Clumps	0.00%	0.44%
	External Duplication	0.00%	0.32%
	Shotgun Surgery	0.00%	0.01%

SR^* を，特定の臭いの種類に限定したもの

$$SR_t = \frac{\sum_{r \in R} |O_r \cap M_t^s|}{\sum_{r \in R} |O_r|}, \quad SR_t^* = \frac{\sum_{r \in R} |M_t^s \setminus O_r|}{\sum_{r \in R} |M \setminus O_r|}$$

を求めた．ここで，M_t^s は種類 t の不吉な臭いを持つモジュールの集合を指す．

結果を**表 7** に示す．この表では，検出された不吉な臭いのみを表記している．表の値より，クラスレベルでは God Class，メソッドレベルでは Blob Operation が多く検出されていることが分かる．これらは主に規模に関する臭いであり，正解セットに特に多く含まれるこれらの不吉な臭いが結果に寄与していることは，規模の大きいモジュールから多くのバグが検出されているとみることもでき，これは驚くべき結果ではない．しかし，Schizophrenic Class や Instensive Coupling など，必ずしも規模のみに関連するわけではない不吉な臭いも，正解セットから大きな割合で検出されている．また，その他にも，多くの種類の不吉な臭いが正解セットに含まれている．このことは，必ずしも規模の観点に留まらない不吉な臭いが Bug Localization の精度向上に貢献している可能性を示唆している．

一方で，クラスレベルにおける Data Class などは，正解セットに含まれていないクラスで多く検出された．これは，Data Class が振舞いよりもデータの管理に特化したモジュールに対する不吉な臭いであり，バグ修正の際の変更対象とはなりづらいことに起因するかもしれない．このような種類の不吉な臭いは，提案手法における検出対象から除くことで，さらなる精度の向上が可能かもしれない．

正解セットに含まれるモジュールは多くの種類の不吉な臭いを持っており，それらのうち不吉な臭いを持つものの割合は，正解セットに含まれないモジュールのうち臭いを持つものの割合よりも大きかった．

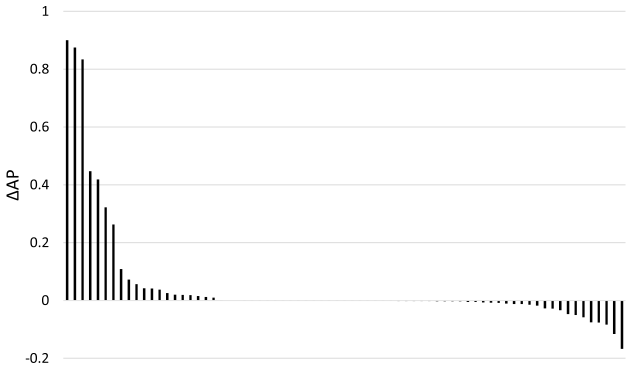

図 4 ArgoUML でのバグレポートごとの AP の変化
Fig. 4 Delta of AP for each bug report of ArgoUML.

5.5.2 バグレポートごとの AP の調査

図 4 は，ArgoUML においてクラスレベルの Bug Localization を行った際に，既存の IR 手法（$\alpha = 0$）に比べて提案手法（$\alpha = 0.42$）がどの程度 AP を上昇させたかを示したものである．横軸はそれぞれのバグレポート，縦軸は AP の上昇量（$\Delta AP = AP_{\alpha=0.42} - AP_{\alpha=0}$）を表しており，$\Delta AP$ が高い順に左からバグレポートを並べている．グラフから，それぞれのバグレポートがバランスよく AP を向上させたわけではなく，偏りがあることが分かる．また，中央付近には変化がほとんどみられないバグレポートがあり，右端の AP 減少量は左端の AP 上昇量に比べて小さい．0.01 以下の微小な変化を無視したとして，ΔAP が 0.01 より大きい（増加）バグレポートは 20，-0.01 より小さい（減少）バグレポートは 17 あり，減少したものよりも増加したものが多かった．また，AP の定義により，ΔAP の値には上位の正解モジュールの順位変動が強く影響し，下位の正解モジュールの順位変動の影響が小さい．これらのことから，提案手法の導入が，ランキング上位の正解モジュールをより高い順位に引き上げる効果を与えたものと考える．

提案手法では，精度低下を起こしたバグレポートよりも，精度向上を起こしたバグレポートが多数だった．また，精度向上例においては，ランキング上位の正解モジュールをより高順位に引き上げていた．

5.6 妥当性の脅威

内的妥当性．本実験では 4 つのプロジェクトを用いたが，プロジェクトのバグレポートやソースコードの質が脅威となる．バグレポートの質が悪ければ悪いほど，既存の IR 手法の重要度が下がる．また，ソースコードの質が悪ければ悪いほど，不吉な臭いが多く検出される可能性が高いため，不吉な臭いの重要度が上がる．しかし，データセットとして整備されているプロジェクトのバグレポートやソースコードを用いたため，この脅威は小さいと考える．さらに，不吉な臭いはいくつかのソフトウェアメトリクス

を元に検出されるため，不吉な臭いの情報ではなく，ソフトウェアメトリクスがBug Localizationに効果的であった可能性がある．そのため，ソフトウェアメトリクスについても同様の検証を行う必要がある．

外的妥当性．本実験ではソースコードがJava言語で書かれているプロジェクトに関して検証したが，他のプログラミング言語でも同じように手法を用いてMAPが上がるとは限らない．また，4つのプロジェクトに対してしか検証できていないため，どのようなプロジェクトでもMAPが向上するとはいえない．そのため，他のプログラミング言語や他の多くのプロジェクトで検証する必要がある．

6. おわりに

本論文では，Bug Localization手法の中でも基盤となるIR手法に対して，不吉な臭いの情報を組み合わせた新しいメトリクスおよびそれを用いたモジュールの推薦手法を提案した．4つのプロジェクト，クラスレベルとメソッドレベルの2粒度に関して，既存のIR手法と提案手法を比較したところ，既存のIR手法に比べてMAPの向上がみられ，それぞれの粒度に関して，平均22%，137%の向上がみられた．また，どのプロジェクトに関してもMAPの向上がみられた．本手法で用いた不吉な臭いはソースコードのみを入力とするため，IR手法を適用できる状況では適用が可能である．すなわち，これまでにIR手法をベースの手法とした発展的なBug Localization手法においても，ベースとするIR手法として本論文の提案手法を用いることが可能である．

本論文の今後の課題を以下にあげる．

パラメータαのより厳密な設定．提案手法を実際に用いるためには，パラメータαの値を定める必要がある．たとえば，対象プロジェクトの過去のバグレポートとその際に変更されたモジュールの情報を用いて，RQ_1での調査と同様にαの最適値を計算しておくことにより，プロジェクト特有のαを得られると考える．こういった試みを含めた，パラメータ調整のための枠組みが必要である．

より多くの評価実験の実施．本論文での評価では4つのプロジェクトのみを対象としており，またそれらのいずれもJava言語を用いたものである．そのため，本手法がどのようなプロジェクトに対しても効果的な手法であることを十分には示せていない．より多くのプロジェクトに関して提案手法を検証し，効果的であることを示す必要がある．また，Zhouら[22]がBug Localization手法の評価をするためのデータセットを提案しており，提案手法もこういったデータセットを用いて評価したい．さらに，本論文ではIR手法のみに不吉な臭いの深刻度を組み合わせた手法が精度向上につながることを示したものの，IR手法よりも高精度の手法に対しても同様に精度が向上することを示す必要がある．IR手法に実行情報や過去の変更履歴などの追加的な情報を組み合わせた手法に対しても，不吉な臭いの深刻度を組み合わせた際に精度が向上するかを検証することが望ましい．

ソフトウェアメトリクスを用いた検証．不吉な臭いはソフトウェアメトリクス，特にプロダクトメトリクスを元に検出されている．プロダクトメトリクスとはソースコード上のモジュールの特性を定量化した尺度であり，たとえば行数を示すLines of Code（LOC）やメソッドの数を示すNumber of Methods（NOM）などがある．こういったメトリクスの組合せに基づき，不吉な臭いが定義されている[32]．不吉な臭いそのものではなく，ソフトウェアメトリクスがBug Localizationを改良した可能性もあるため，ソフトウェアメトリクスを用いた，同様の検証を行い分析することで，Bug Localizationを改良した要因を明らかにすることができる．

提案手法の改良．現在利用している不吉な臭いは，Bug Localizationの対象モジュール粒度と同じ粒度の不吉な臭いのみである．しかし，対象モジュール粒度に限らない粒度の不吉な臭いを用いることで，精度を改善できる可能性がある．

謝辞　本研究の一部は，日本学術振興会科学研究費補助金（JP15K15970，JP15H02683，JP15H02685，JP18K11238）の助成を受けた．

参考文献

[1] Lucia, Thung, F., Lo, D. and Jiang, L.: Are faults localizable?, *Proc. 9th Working Conference on Mining Software Repositories*, pp.74–77 (2012).
[2] Nguyen, A.T., Nguyen, T.T., Al-Kofahi, J., Nguyen, H.V. and Nguyen, T.N.: A topic-based approach for narrowing the search space of buggy files from a bug report, *Proc. 26th International Conference on Automated Software Engineering*, pp.263–272 (2011).
[3] Marcus, A., Sergeyev, A., Rajlich, V. and Maletic, J.I.: An information retrieval approach to concept location in source code, *Proc. 11th Working Conference on Reverse Engineering*, pp.214–223 (2004).
[4] Gay, G., Haiduc, S., Marcus, A. and Menzies, T.: On the use of relevance feedback in IR-based concept location, *Proc. 25th International Conference on Software Maintenance*, pp.351–360 (2009).
[5] Weiser, M.: Programmers use slices when debugging, *Comm. ACM*, Vol.25, No.7, pp.446–452 (1982).
[6] Dao, T., Zhang, L. and Meng, N.: How does execution information help with information-retrieval based bug localization?, *Proc. 25th International Conference on Program Comprehension*, pp.241–250 (2017).
[7] Wong, C.-P., Xiong, Y., Zhang, H., Hao, D., Zhang, L. and Mei, H.: Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis, *Proc. 30th International Conference on Software Maintenance and Evolution*, pp.181–190 (2014).
[8] Saha, R.K., Lease, M., Khurshid, S. and Perry, D.E.: Improving bug localization using structured information retrieval, *Proc. 28th International Conference on Automated Software Engineering*, pp.345–355 (2013).

[9] Wang, S. and Lo, D.: Version history, similar report, and structure: Putting them together for improved bug localization, *Proc. 22nd International Conference on Program Comprehension*, pp.53–63 (2014).
[10] Youm, K.C., Ahn, J. and Lee, E.: Improved bug localization based on code change histories and bug reports, *Information and Software Technology*, Vol.82, pp.177–192 (2017).
[11] Takahashi, A., Sae-Lim, N., Hayashi, S. and Saeki, M.: A preliminary study on using code smells to improve bug localization, *Proc. 26th International Conference on Program Comprehension*, pp.324–327 (2018).
[12] Fowler, M.: *Refactoring: Improving the Design of Existing Code*, Addison-Wesley Professional (1999).
[13] Yamashita, A. and Moonen, L.: Do code smells reflect important maintainability aspects?, *Proc. 28th International Conference on Software Maintenance*, pp.306–315 (2012).
[14] Yamashita, A. and Moonen, L.: Exploring the impact of inter-smell relations on software maintainability: An empirical study, *Proc. 35th International Conference on Software Engineering*, pp.682–691 (2013).
[15] Fontana, F.A., Ferme, V., Zanoni, M. and Roveda, R.: Towards a prioritization of code debt: A code smell intensity index, *Proc. 7th International Workshop on Managing Technical Debt*, pp.16–24 (2015).
[16] Marinescu, R.: Assessing technical debt by identifying design flaws in software systems, *IBM Journal of Research and Development*, Vol.56, No.5, pp.9:1–9:13 (2012).
[17] Khomh, F., Di Penta, M., Guéhéneuc, Y.-G. and Antoniol, G.: An exploratory study of the impact of antipatterns on class change-and fault-proneness, *Empirical Software Engineering*, Vol.17, No.3, pp.243–275 (2012).
[18] Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R. and De Lucia, A.: On the diffuseness and the impact on maintainability of code smells: A large scale empirical investigation, *Empirical Software Engineering*, Vol.23, No.3, pp.1188–1221 (2018).
[19] Lukins, S.K., Kraft, N.A. and Etzkorn, L.H.: Bug localization using latent dirichlet allocation, *Information and Software Technology*, Vol.52, No.9, pp.972–990 (2010).
[20] Poshyvanyk, D., Marcus, A., Rajlich, V., Gueheneuc, Y.-G. and Antoniol, G.: Combining probabilistic ranking and latent semantic indexing for feature identification, *Proc. 14th International Conference on Program Comprehension*, pp.137–148 (2006).
[21] Rao, S. and Kak, A.: Retrieval from software libraries for bug localization: A comparative study of generic and composite text models, *Proc. 8th Working Conference on Mining Software Repositories*, pp.43–52 (2011).
[22] Zhou, J., Zhang, H. and Lo, D.: Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports, *Proc. 34th International Conference on Software Engineering*, pp.14–24 (2012).
[23] Kim, D., Tao, Y., Kim, S. and Zeller, A.: Where should we fix this bug? A two-phase recommendation model, *IEEE Trans. Software Engineering*, Vol.39, No.11, pp.1597–1610 (2013).
[24] Le, T.-D.B., Thung, F. and Lo, D.: Will this localization tool be effective for this bug? Mitigating the impact of unreliability of information retrieval based bug localization tools, *Empirical Software Engineering*, Vol.22, No.4, pp.2237–2279 (2017).
[25] Chaparro, O., Florez, J.M. and Marcus, A.: Using Observed Behavior to Reformulate Queries during Text Retrieval-based Bug Localization, *Proc. 33rd International Conference on Software Maintenance and Evolution*, pp.376–387 (2017).
[26] Acharya, M. and Robinson, B.: Practical change impact analysis based on static program slicing for industrial software systems, *Proc. 33rd International Conference on Software Engineering*, pp.746–755 (2011).
[27] Naish, L., Lee, H.J. and Ramamohanarao, K.: A model for spectra-based software diagnosis, *ACM Trans. Software Engineering and Methodology*, Vol.20, No.3, pp.11:1–11:32 (2011).
[28] Tantithamthavorn, C., Ihara, A. and Matsumoto, K.: Using co-change histories to improve bug localization performance, *Proc. 14th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, pp.543–548 (2013).
[29] Shi, Z., Keung, J., Bennin, K.E. and Zhang, X.: Comparing learning to rank techniques in hybrid bug localization, *Applied Soft Computing*, Vol.62, pp.636–648 (2018).
[30] Dit, B., Moritz, E. and Poshyvanyk, D.: A TraceLab-based solution for creating, conducting, and sharing feature location experiments, *Proc. 20th International Conference on Program Comprehension*, pp.203–208 (2012).
[31] Dit, B., Revelle, M., Gethers, M. and Poshyvanyk, D.: Feature location in source code: A taxonomy and survey, *Journal of Software: Evolution and Process*, Vol.25, No.1, pp.53–95 (2013).
[32] Lanza, M. and Marinescu, R.: *Object-Oriented Metrics in Practice*, Springer Science & Business Media (2007).
[33] Brown, W.H., Malveau, R.C., McCormick, H.W. and Mowbray, T.J.: *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*, John Wiley & Sons, Inc. (1998).
[34] Riel, A.J.: *Object-Oriented Design Heuristics*, Addison-Wesley (1996).
[35] Martin, R.C.: *Agile Software Development: Principles, Patterns, and Practices*, Prentice Hall (2007).
[36] Hunt, A. and Thomas, D.: *The Pragmatic Programmer: From Journeyman to Master*, Addison-Wesley (2000).

付　　録

A.1 検出される不吉な臭い

表 A·1 には，inFusion により検出される，オブジェクト指向言語を対象とした，粒度がクラスレベルとメソッドレベルの不吉な臭いとそれらの説明が含まれている．

表 A·1 inFusion により検出される不吉な臭い
Table A·1 Code smells detected by inFusion.

粒度	種類	説明
クラスレベル	Blob Class [12], [32], [33]	非常に大きく複雑なクラス
	Data Class [12], [32], [34]	機能を持たずデータばかりを持ち，他クラスから頻繁に参照されるクラス
	Distorted Hierarchy [34]	継承階層が非常に狭く深いクラス
	God Class [12], [32], [34]	他クラスが持つデータを処理するクラス
	Refused Parent Bequest [32], [34], [35]	基底クラスから継承するメンバをほとんど使わないクラス
	Schizophrenic Class [34], [35]	複数の概念を表現しているクラス
	Tradition Breakers [32], [34]	基底クラスが定める規約に違反するクラス
メソッドレベル	Blob Operation [12], [32], [33]	大きく複雑なメソッド
	Data Clumps [12]	数個のデータがグループとして現れているメソッド
	External Duplication [12], [33], [36]	関係性のないクラスとの間に重複したコードを含むメソッド
	Feature Envy [12], [32], [34]	自クラスのデータより他クラスのデータとの関連性が高いメソッド
	Intensive Coupling [12], [32], [34]	他の多くのメソッドに対して，結合性が高いメソッド
	Internal Duplication [12], [33], [36]	同一クラス内に重複したコードを含むメソッド
	Message Chains [12], [32], [33]	連鎖的に多くのメソッド呼び出しが起こるメソッド
	Shotgun Surgery [12], [32]	変更をする際に，多くの他のメソッドに変更が伝播するメソッド
	Sibling Duplication [12], [33], [36]	兄弟クラス間に重複したコードを含むメソッド

高橋 碧 （学生会員）

2018 年東京工業大学工学部情報工学科卒業．現在，同大学情報理工学院情報工学系修士課程在学中．学士（工学）．Bug Localization に関する研究に従事．IEEE 会員．

セーリム ナッタウット

2012 年キングモンクット工科大学ラカバン校工学部コンピュータ工学科卒業．2016 年東京工業大学大学院情報理工学研究科計算工学専攻修士課程修了．現在，同大学情報理工学院情報工学系博士後期課程在学中．修士（工学）．ソースコードの不吉な臭いに関する研究に従事．

林 晋平 （正会員）

2008 年東京工業大学大学院情報理工学研究科計算工学専攻博士後期課程修了．2009 年同専攻助教．2018 年同大学情報理工学院准教授．博士（工学）．ソフトウェア変更やソフトウェア開発環境の研究に従事．電子情報通信学会，日本ソフトウェア科学会，IEEE-CS，ACM 各会員．

佐伯 元司 （正会員）

1983 年東京工業大学大学院工学研究科情報工学専攻博士後期課程修了．同大学助手，助教授を経て，2000 年同大学大学院情報理工学研究科計算工学専攻教授．現在，同大学情報理工学院教授．工学博士．要求工学やソフトウェア開発技法等の研究に従事．電子情報通信学会，人工知能学会，日本ソフトウェア科学会，IEEE，ACM 各会員．