情報処理学会論文誌 Vol.65 No.4 792–807 (Apr. 2024)

情報検索を用いたBug Localization手法にモジュール粒度の違いが与える影響

積田 静夏[1,a)] 天嵜 聡介[2,b)] 林 晋平[1,c)]

受付日 2023年8月1日, 採録日 2024年1月10日

概要：Bug Localization とはバグの原因箇所を特定する，ソフトウェア保守において重要な作業である．メソッドレベルで自動で Bug Localization を行う手法は開発者にとって有用なものの，手法が少なく，評価可能なフレームワークも存在しないため知見が少ない．本論文では，既存の情報検索を用いた Bug Localization 手法をメソッドレベルで大規模に比較可能なフレームワーク FinerBench4BL を提案し，推薦モジュール粒度の違いが手法の精度や推薦時に考慮する追加情報，実行時間へ与える影響の調査を行う．データセットはリポジトリ変換により Bench4BL のプロジェクトをメソッドレベルに再構築したメソッドリポジトリから作成した．またメソッドリポジトリに基づき，既存のファイルレベルの手法を小さな修正でメソッドレベルに変更し，データセットと組み合わせて評価フレームワークとした．FinerBench4BL を利用した調査の結果，メソッドレベルの手法は精度が低下する一方で，デバッグに必要な労力が削減することが明らかになった．またメソッドレベルへの変更にともない，既存手法が考慮する追加情報の影響が小さくなることから，多くの種類の追加情報の考慮が精度向上につながることが分かった．

キーワード：Bug Localization，情報検索，リポジトリ変換

The Impact of Module Granularity in IR-based Bug Localization Techniques

Shizuka Tsumita[1,a)] Sousuke Amasaki[2,b)] Shinpei Hayashi[1,c)]

Received: August 1, 2023, Accepted: January 10, 2024

Abstract: Bug localization is an important aspect of software maintenance because it can locate modules that need to be changed to fix a specific bug. Although method-level bug localization is helpful for developers, there are only a few techniques, and there exists no large-scale framework for their evaluation. In this paper, we present FinerBench4BL, an evaluation framework for method-level information retrieval-based bug localization techniques, and investigate the impact of module granularities on the techniques' accuracy, additional information, and execution time. The dataset was constructed from a method repository where projects in Bench4BL were converted to the method level by repository transformation. In addition, based on the method repositories, we tailor the existing file-level bug localization technique implementations at the method level. By combining the generated dataset and implementations, we build a framework for method-level evaluation. We found that the change to method level reduces the influence of additional information considered by existing techniques and that considering much additional information leads to improved accuracy.

Keywords: bug localization, information retrieval, repository transformation

[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
[2] 岡山県立大学情報工学部
Computer Science and Systems Engineering, Okayama Prefectural University, Soja, Okayama 719–1197, Japan

[a)] tsumita@se.c.titech.ac.jp
[b)] amasaki@cse.oka-pu.ac.jp
[c)] hayashi@c.titech.ac.jp

1. はじめに

Bug Localization とは，バグの原因箇所を特定する作業のことである．大規模なソフトウェア開発プロジェクトでは，開発者は多くのバグを修正する必要があり，ソフトウェアのデバッグは困難で時間がかかることが分かっている [1]．そこで，この作業を自動化する Bug Localization 手法が数多く研究されている．たとえば，バグレポートの内容からバグの発生箇所を特定する手法や，情報検索（IR：Information Retrieval）を用いた手法 [2], [3]，実行トレースの動的解析を用いた手法 [4] などがある．また，Bug Localization の精度向上のため，IR 手法に追加情報を組み合わせた手法もいくつか提案されている．たとえば，BugLocator [5] は，過去に解決された類似のバグレポートを追加情報として用い，精度を改善させた．BLUiR [6] は，類似のバグレポートを使うだけでなく，ソースコードの構造情報を取り入れ，AmaLgam [7] は，バージョンの履歴情報と構造情報，類似バグレポートの情報を組み合わせた．

IR 手法を利用した Bug Localization（IRBL：IR-Based Bug Localization）手法の多くは推薦するモジュール粒度がファイルレベルであり，バグがある可能性が高い順にランク付ファイルリストを出力する．しかし，Java 言語などのソースファイルにおいて，1 つのファイルは複数のメソッドから構成されている．そのため，ファイルレベルでは多数のメソッドからなる大きなものが推薦される場合があり，開発者にとって有用な粒度ではない可能性がある．そこで，推薦するモジュール粒度がより細かくなるメソッドレベルであれば，開発者がデバッグに必要な労力を削減できると考える．

しかし，メソッドレベルでバグ箇所の推薦を行う IRBL 手法は少ない [8], [9], [10]．また我々が知る限り，メソッドレベル手法の包括的な分析は単純な IR 手法のみでしか行われていない [11]．メソッドレベルの IRBL 手法に対する分析は行われておらず，ファイルレベルとの結果の差異や結果に影響を与える要因などに関する知見は少ない．

本論文ではメソッドレベルの大規模 IRBL 評価フレームワーク FinerBench4BL を提案する．FinerBench4BL は，既存の IRBL 評価フレームワーク Bench4BL [12] をベースとしている．Bench4BL が提供するデータセットに対して，メソッドごとに分割されたメソッドリポジトリを生成するリポジトリ変換を適用し，メソッドレベルのデータセットを構築した．また，メソッドリポジトリを利用して，既存の IRBL 手法をメソッドレベルに変更し，データセットと合わせた評価フレームワークとした．

メソッドレベルに変更した IRBL 手法を用いて，モジュール粒度の違いによる性能の比較や影響を与える要因，および実行時間に関する分析を行った．メソッドレベルの IRBL 手法は，ファイルレベルの手法と比較して推薦精度が低下したものの，デバッグに必要な労力を削減することが判明した．また，メソッドレベルではバグレポートやソースコードのテキスト類似度以外に手法が考慮する追加情報が精度へ与える影響が小さく，メソッドレベル推薦における情報量が不足していることを明らかにした．さらに，IRBL 手法の実行時間は，リポジトリ変換によるソースファイル数の増加にともない，手法ごとに異なる傾向で増加することが分かった．

本論文の貢献を以下に示す．

- メソッドレベルで比較可能な IRBL 手法の評価フレームワーク FinerBench4BL の構築．
- メソッドレベルとファイルレベルにおける IRBL 手法の大規模な精度の評価．
- IRBL 手法のモジュール粒度の違いで追加情報が精度に与える影響の調査．
- IRBL 手法のモジュール粒度の違いによる実行時間の増加率の調査．

本論文は SANER 2023 での我々の発表 [13] の内容を発展させたものである．IRBL 手法の評価におけるより詳細な分析（RQ_2）や，粒度間における精度への追加情報の影響の分析（RQ_3），実行時間の増加率に関する分析（RQ_4）が主要な拡張箇所である．

本論文の構成は以下のとおりである．次章で IRBL 手法と評価フレームワークを紹介し，3 章で手法の推薦粒度の観点から課題を議論する．4 章で関連する既存の IRBL 手法や評価フレームワーク，データセットを説明する．5 章にて，提案するフレームワーク FinerBench4BL の詳細を述べる．6 章では実験設定および粒度間の手法の調査結果を説明し，7 章で妥当性の脅威を分析する．最後に 8 章にて本論文の結論と今後の課題をまとめる．

2. IRBL 手法とその評価フレームワーク

2.1 IRBL 手法

Bug Localization とはバグに関する情報に基づき，ソースコード中からバグの原因箇所を特定する作業である．本論文では特にテキスト類似度を用いてバグ箇所を特定する IRBL 手法を扱う．ソースコードとバグレポートが揃っていれば静的に実行可能な IRBL 手法は，多くのソフトウェア開発プロジェクトにおいて利用可能で，スケーラブルだという利点がある．また，IRBL 手法は Bench4BL [12] や Bugzbook [14] のような複数の手法の比較調査でも対象とされている一般的な Bug Localization 手法である．

IRBL 手法はバグレポートとソースコードを入力とし，それらのテキスト類似度に基づいたスコアを計算したうえで，修正すべきモジュールのランク付リストを出力する．IRBL 手法において，バグレポートがソースコードを検索するクエリとしての役割を担う．バグレポートはバグ ID，概要，バグの発生・修正日時，および詳細な説明を含む．

表 2 IRBL 手法が用いる追加情報
Table 2 Additional information that IR-based Bug Localization techniques use.

	BugLocator	BLUiR	BRTracer	AmaLgam	Locus	BLIA
過去のバグレポートとの類似度	✓		✓	✓		✓
ファイルサイズ	✓		✓			✓
ソースコードとバグレポートの構造		✓		✓		✓
ソースコードの履歴				✓	✓	✓
スタックトレース			✓			✓

表 1 BugLocator の CODEC-199 への適用結果
Table 1 Applying BugLocator to CODEC-199.

順位	ファイル名	スコア
1	`Soundex.java`	0.800
2	`DaitchMokotoffSoundex.java`	0.618
3	`RefinedSoundex.java`	0.564
4	`SoundexTest.java`	0.500
5	`RefinedSoundexTest.java`	0.388

一部の IRBL 手法はバグレポートとソースコードのテキスト類似度に加え，過去のバグレポートやソースコードの履歴情報などの様々な追加情報を考慮する．

IRBL 手法の出力例として，BugLocator [5] を CODEC-199 [*1] に適用した結果を**表 1** に示す．各列が出力のランク付リストの順位，ファイル名および該当ファイルとバグレポートとの類似度のスコアを示している．ハイライトされている `Soundex.java` がバグ修正時に実際に変更されたファイルである．順位が高いほどバグを含む可能性が高いとして，開発者はこの結果を基に `Soundex.java` から順にバグの原因箇所を捜索する．

精度向上のため，様々な追加情報を考慮してバグ箇所を推薦する IRBL 手法が多く研究されている．過去のバグレポートとの類似度やファイルサイズを利用する BugLocator [5] や，ソースコードとバグレポートの構造情報を利用する BLUiR [6]，履歴情報を用いる AmaLgam [7] や Locus [15]，バグレポートに記載されたスタックトレースを考慮する BRTracer [4] などがあり，これらの情報をすべて組み合わせた BLIA [16] なども提案されている．各手法が扱う追加情報を**表 2** に示す．これらの手法はいずれもファイルレベルで推薦を行い，各手法の性能評価には限られたデータ数のプロジェクトが用いられている．表から分かるように，BLIA が最も多くの追加情報を用いている．

2.2 IRBL 手法の評価フレームワーク

IRBL 手法の評価では，すでに解決されたバグを利用し，実際に修正されたモジュールを正解と見なす．IRBL 手法の性能は解決済バグレポートに対して出力したランク付リストと正解モジュールを比較して評価される．この解決済バグレポートと正解モジュールの組合せを評価用データセットとし，評価可能な手法を組み合わせたものを評価フレームワークとする．

ファイルレベルの IRBL 手法の大規模評価フレームワークとして，Lee らによって提案された Bench4BL [12] がある．46 プロジェクトに対してイシュートラッキングシステムから解決済バグレポートと実際に修正されたファイルを取得し，評価用データセットとして提供している．彼らはバグレポートが提出された際に指定されたリリースバージョンを検索対象とすることで，より正確に手法を評価した．データセットに加えて，彼らは BugLocator [5], BLUiR [6]，BRTracer [4], AmaLgam [7]，BLIA [16]，Locus [15] の実装も提供した．

メソッドレベルにおいては，追加情報を加味せずに推薦を行う IR 手法の評価は Tantithamthavorn らにより行われている一方 [11]，よく知られたフレームワークでのメソッドレベルの IRBL 手法の評価は著者らの知る限り存在しない．Youm らの BLIA1.5 [10] や Amasaki らの拡張した BLUiR [17]，Razzaq らの BoostNSift [8] など，メソッドレベル IRBL 手法の評価においても，多くの手法を大規模なプロジェクトに対して適用したものはない．

3. 動機

3.1 ファイルレベル推薦における課題

ファイルレベルでバグ箇所の推薦を行う際，バグと無関係なメソッドを含む大きなファイルが推薦され，バグ箇所の特定が困難となる場合がある．例として CODEC-221 [*2] を考える．このバグでは，`HmacUtils.java` が含む 3 つのメソッドを実際に修正された正解のメソッドとしている．`HmacUtils.java` は 40 個のメソッドを含む 729 行のファイルであり，バグ箇所の特定には労力を要するため，メソッドレベルで推薦されれば少ない労力でデバッグが済むと考える．

表 3 は IRBL 手法の 1 つである BLIA をこのバグレポートに対して両方の粒度で適用した結果である．BLIA は本論文で扱う IRBL 手法において，最も多くの種類の追加情報を扱う．この表は BLIA が推薦したモジュールの名前と順位，バグレポートとのテキスト類似度や追加情報に基づき算出されたスコアおよび各モジュールの行数（LOC）を示している．実際に修正されていた正解のモジュールをハ

[*1] https://issues.apache.org/jira/browse/CODEC-199

[*2] https://issues.apache.org/jira/browse/CODEC-221

表 3 BLIA の CODEC-221 に対するメソッドレベルとファイルレベルでの適用結果
Table 3 Applying BLIA at file and method levels to CODEC-221.

	ファイルレベル			メソッドレベル		
順位	モジュール名	スコア	LOC	モジュール名	スコア	LOC
1	HmacUtils.java	0.640	729	BaseNCodecInputStream#reset()	0.640	15
2	DigestUtils.java	0.251	752	HmacUtils#updateHmac(Mac,InputStream)	0.573	29
3	HmacUtilsTest.java	0.095	237	HmacUtils#updateHmac(Mac,byte[])	0.565	19
4	BaseNCodecInputStream.java	0.048	184	HmacUtils#updateHmac(Mac,String)	0.565	19

イライトした．本例にて，ファイルレベルでは正解となる HmacUtils.java が 1 位に推薦されているものの，700 行を超えるファイルの中でどのメソッドを修正すべきかの特定は容易ではない．一方，メソッドレベルの推薦結果では，2，3，4 位にすべての正解メソッドが推薦されており，1 位の不正解メソッドが不適であることを判断するために読むことを含め，4 位までのメソッドを読む必要がある．すべてのメソッドを読んだとしても，計 82 行でバグ箇所が特定でき，729 行の HmacUtils.java を読む労力を 11%まで削減させている．

また，ファイルが含むメソッド数に対してバグがあるメソッドは少ないことが知られている．Hata らはバグ予測の文脈において，細粒度予測の有効性を確かめるため，対象プロジェクトのファイルが含むメソッド数とバグを含むメソッド数を調査した [18]．その結果，ファイルが含むメソッド数の中央値に対して，バグのあるメソッド数の中央値は 1〜2 と小さいことを示した．したがってファイルレベルの推薦では，バグ発生箇所の特定に無駄な捜索を強いる可能性がある．

3.2 メソッドレベル推薦における課題

メソッドレベルの推薦が開発者の労力を削減する可能性が高い一方，メソッドレベルで推薦を行う Bug Localization 手法は少なく，知見が蓄積されていない．著者らの知る限り，BLIA を改良した BLIA1.5 [10] のほか，Zhang らの FineLocator [9]，Razzaq らの BoostNSift [8] などの近年発表された IRBL 手法しかメソッドレベルの推薦を行っていない．また，メソッドレベルの推薦手法を評価する，よく知られた大規模なフレームワークも存在しない．したがって各手法が異なる推薦粒度の間で示す性能差は不明であり，メソッドレベルへの変更により生じる，手法が考慮する追加情報の精度への影響や実行時間の増加率へ関する知見も不足している．これらを明らかにするため，メソッドレベルの結果に対する詳細な分析が必要である．

4. 関連研究

4.1 メソッドレベルの IRBL 手法

メソッドレベルで Bug Localization を行う手法として，Youm らが BLIA を拡張した BLIA1.5 [10] や，Amasaki らがメソッドの外の情報を利用しつつメソッドレベルで Bug Localization をするように BLUiR を修正したもの [17]，Razzaq らが発表したバグレポートの情報からソースコードをフィルタリングしてメソッドレベルで推薦を行う BoostNSift [8] などがある．

メソッドレベルで推薦を行う Bug Localization 手法はそれぞれ異なるデータセットを用いて評価を行っている．BLIA1.5 は AspectJ，SWT，ZXing の 3 つのプロジェクトを利用しており，Amasaki らはメソッドレベルで推薦を行う BLUiR を Bench4BL のプロジェクトを利用して評価した．BoostNSift はメソッドレベルに実装を修正した BLUiR，BugLocator と比較し，データセットとしては BLIA1.5 が利用したものに Eclipse を加えたものを使用した．いずれの評価においても多くのメソッドレベル手法を大規模なプロジェクトに対して適用してはおらず，評価に用いるデータセットも統一されていない．

一方，追加情報をともなわない純粋な IR 手法のみに限った場合には，メソッドレベルのフレームワーク上での評価がある．Tantithamthavorn らは，VSM，LDA，LSI，Entity Metric の 4 つの IR 手法の有効性をメソッドレベルにおいて調査した [11]．彼らは開発者がデバッグに要する労力を測る指標として top-k LOC を提案した．top-k LOC は，手法が出力するランク付リストから，累積の LOC が k 行となるより上位のモジュールから少なくとも 1 つのバグの正解モジュールが見つかるバグレポートの割合を示す．彼らはメソッドレベルの IR 手法においてテキストの前処理の方法やバグレポートの利用箇所のような設定が手法の性能に大きな影響を与えることや，良い性能を示す設定がいずれの粒度でも共通することを明らかにした．また，同程度の精度を示したとしても，バグを発見するまでに要する労力が異なる可能性があることから，LOC を用いた性能評価も必要だとしている．

表 4 に本論文が提案するフレームワークと先行研究との関連性を示した．追加情報の列は，その手法がテキスト類似度だけではなく追加の情報を考慮しているかを示している．メソッドレベルの列はその手法がメソッドレベルで推薦を行うかを示す．手法数，プロジェクト数はそれぞれ先行研究において評価対象となった手法とプロジェクトの数を表す．本論文は，メソッドレベルにおいて追加情報を用

表 4 IRBL 手法評価の比較
Table 4 Comparison of IRBL studies.

	追加情報	メソッドレベル	手法数	プロジェクト数
Lee ら（Bench4BL）[12]	✓		6	46
Youm ら（BLIA1.5）[10]	✓	✓	1	3
Amasaki ら [17]	✓	✓	1	43
Razzaq ら（BoostNSift）[8]	✓	✓	4	4
Tantithamthavorn ら [11]		✓	4	2
本論文（FinerBench4BL）	✓	✓	5	37

表 5 IRBL 手法の評価データセットの比較
Table 5 Comparison of IRBL datasets.

名前	粒度	プロジェクト数	バグ数
iBUGS [19]	ファイル	3	390
MoreBugs [20]	ファイル	2	902
BugLinks [21]	ファイル	2	5,046
Bench4BL [12]	ファイル	46	9,459
Bugzbook [14]	ファイル	29	21,253
FDS [22]	ファイル	11	4,429
MDS [22]	メソッド	14	360
FinerBench4BL	メソッド	37	3,344

いた Bug Localization 手法を評価した研究の中でも，手法数が最も多く，大規模な数のプロジェクトで分析を行ったものである．

4.2 IRBL 手法の評価データセット

IRBL 手法はすでに解決済みのバグレポートに手法を適用して得られるランク付リストと，実際に修正されたモジュールを正解と見なして，これらを比較することで評価を行う．これまで複数の解決済バグレポートと修正されたモジュールを結び付けたものがまとめられ，Bug Localization の評価データセットとして提案されている．

表 5 に評価に用いるデータセットの概要を示す．iBUGS [19] は Dallmeier らによって開発されたデータセットである．Rao らによる MoreBugs [20] は，iBUGS が採用したプロジェクトにバージョン履歴情報を追加した拡張データセットである．BugLinks [21] は，Sisman らが提案した非 Java プロジェクトも含むデータセットである．iBUGS，MoreBugs および BugLinks は，比較的小規模なデータセットで，2〜3 のプロジェクトを含む．

Bench4BL は Lee らにより提案された大規模な IRBL 評価フレームワークである．Bench4BL には 46 の Java プロジェクトと 9,459 のバグレポートが含まれている．さらに Bench4BL には，複数の IRBL 手法の実装が含まれており，提供されたプロジェクトに対して容易に手法を適用できる．Akbar らは Bugzbook [14] という，Bench4BL とは別の大規模な IRBL 手法の評価データセットを提案している．このデータセットには Java，C/C++，Python で開発された 29 のプロジェクトから 20,000 を超えるバグレポートが含まれている．

我々の知る限りでは，利用可能な状態で公開されているデータセットのほとんどはファイルレベルである．上記で述べたように，いくつかの既存のメソッドレベルの IRBL 手法に関する研究では，それぞれが内部的にメソッドレベルのデータセットを構築・分析している．しかし，これらのデータセットは他の研究者が利用可能な形式で公開されていない．IR 手法のみに着目すれば，Chaparro らにより Bug Localization を行うためのクエリの再構築に関する研究の一環として，クラス，ファイル（FDS），メソッドレベル（MDS）で構築されたデータセットが，要求に応じて提供されている [22]．一方で，IRBL 手法に対してはこうしたメソッドレベルのデータセットや評価フレームワークが存在しないことが，メソッドレベルの Bug Localization 手法の研究を妨げていると考える．

5. FinerBench4BL

3 章で述べた課題解決のため，リポジトリ変換で作成する**メソッドリポジトリ**を利用して，既存の IRBL 手法のメソッドレベルへの変換，およびメソッドレベルのデータセットの構築を行い，メソッドレベルの評価フレームワーク FinerBench4BL を構築した．ここで，**メソッドリポジトリ**とは，元リポジトリが含む情報ごと，メソッドごとのソースコードをファイルとして保持するよう変換された版管理リポジトリを指す．ファイルレベルからメソッドレベルへの手法の推薦粒度の変換はソースファイルの構文解析が行えれば実施可能である．なお，メソッドレベルの手法をファイルレベル手法として用いることも可能だったが，採用しなかった．これは，メソッドレベルの IRBL には表 4 や表 5 に記載したほどの手法やデータセットしかなく，一般的なものとは考えられなかったためである．

IRBL 手法は，ソースコードとバグレポートの類似度のほか，手法により過去のバグレポートや履歴情報などの情報を加味して算出するスコア順にモジュールのランク付リストを出力する．手法の評価時には，解決済のバグへ手法を適用して得られるリストと，実際に修正されたファイルとを比較する．したがって，メソッドレベルの評価フレームワークは以下の要件を満たす必要がある．

- ランク付リストがメソッドレベルで出力される．
- 解決済バグレポートで実際に修正された箇所がメソッドレベルで対応付けされる．
- 各メソッドに対応する履歴情報が取得できる．

手法をメソッドレベルに変更する際，一般的にはスコアの計算をファイルが含むメソッドごとに処理できるよう，手法ごとに特別な修正が必要であるが，それには多大な実装コストがかかる．

一方で我々の提案する，各ファイルが持つメソッドごと

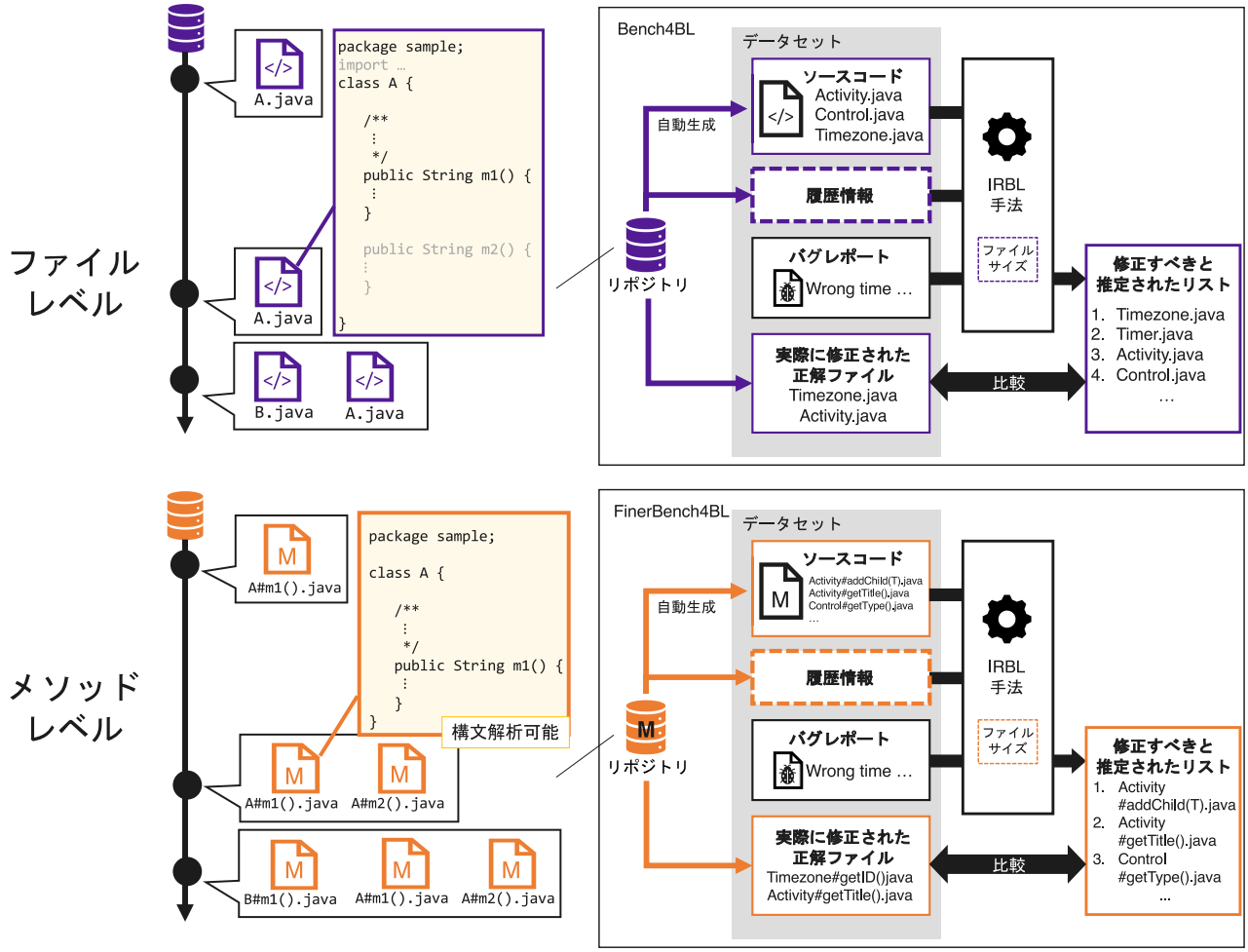


図 1 FinerBench4BL の概要
Fig. 1 Overview of FinerBench4BL.

に切り出された**メソッドファイル**を用いるアプローチは，既存の IRBL 手法に各メソッドファイルを通常のソースファイルのように誤認させ，実装への大きな修正なしに各メソッドに対するスコアの計算を可能にする．具体的には，入力となるソースファイルを各メソッドを含む複数のファイルに分割することにより，メソッドレベルでランク付けされた出力が得られる．またメソッドファイルを，ソースファイルの形式を保ちつつ，分割前のメソッドの履歴情報を持つように切り出すことから，解決済のバグレポートに対して実際に修正されたメソッドを容易に対応付けられる．メソッドファイルは意味解析上では問題がある Java ソースコードとなるが，IRBL 手法における類似度計算では表面的なテキスト解析のみを行うため，多くの場合で解析に問題が生じない．したがって，ほぼメソッドファイルの準備のみで既存 IRBL 手法をメソッドレベルに変更可能にし，再実装に必要なコストを大きく削減したうえで，粒度の違いの要因を排除した状態での比較が可能である．

変換により作成された FinerBench4BL の概要を**図 1** に示す．既存の評価フレームワークに対して，データセットのリポジトリを変換し，メソッドリポジトリを生成する．メソッドレベルのリポジトリを入力とすれば，メソッドレベルのデータセット構築が容易となり，各手法をメソッドレベル推薦へ変更する修正量も削減する．

5.1 メソッドリポジトリの作成

メソッドレベルへ変換する既存の評価フレームワークとして，Bench4BL を選定した．Bench4BL は各バグレポートに対する実際の修正ファイルを対応付けることが可能である．また，評価に用いる IRBL 手法の実装を提供しており，リポジトリ変換を利用した推薦粒度の変更に最適なフレームワークと考えた．既存の大規模データセットにはほかにも Bugzbook [14] があるが，実際の修正ファイルを対応付けるためのプログラムや手法の実装が公開されていないことから本論文では採用しなかった．

Bench4BL が提供するデータセットに，リポジトリ変換ツール Historinc [23] を適用することにより，対象のリポジトリをメソッドレベルに変換し，メソッドファイルを生成する．Historinc は変換に，定評のある Java 構文解析ライブラリ Eclipse JDT を用いており，変換の精度はこのライブラリの精度に従う．Apache Commons CODEC の

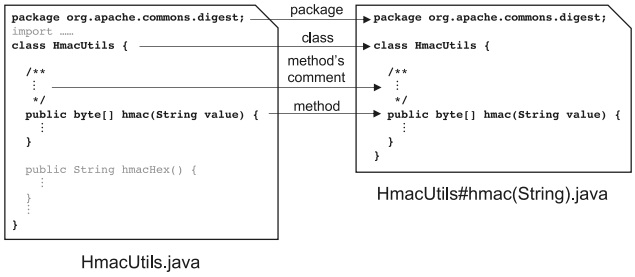


図 2 リポジトリ変換の例
Fig. 2 Example of repository transformation.

`HmacUtils.java` に対する変換例を**図 2** に示す．このファイルは変換後，`HmacUtils#hmac(String).java` や `HmacUtils#hmacHex().java` といったメソッドファイルに分割される．パッケージ宣言，クラス宣言，メソッドの Javadoc コメント，メソッド本体をメソッドファイルとして得るため，フィールドや import 文などメソッド外のコード要素は除外される．そのため，これらが原因となるバグは評価の対象外とした．

5.2 バグレポートへの正解メソッドの対応付け

Bench4BL フレームワークはバグレポートに実際の修正ファイルを対応付けるスクリプトを含む．このスクリプトは，コミットメッセージに記載のバグ ID を参照してバグの修正コミットを対応付けることで正解のファイルを特定している．そこで，変換後のメソッドリポジトリとバグレポートとで再度対応付けを行い，正解ファイルをメソッドファイルに更新することで，メソッドレベルの IRBL 手法を比較可能にする．

5.3 手法のメソッドレベルへの変更

既存のファイルレベルの IRBL 手法をメソッドレベルへ変更する．リポジトリ変換によるアプローチは，ファイルを構文解析可能な Java ソースファイルとしてメソッドごとに分割するため，メソッド内部までの構文解析結果を擬似的に再現できる．したがって，手法が扱う追加情報がメソッドファイルからも取得できる情報であれば，変更にあたり実装の修正は必要ない．例として BugLocator の変更を考える．BugLocator はソースコードとバグレポートのほか，ソースコードのファイルサイズと過去のバグレポートとの類似度を加味してファイルリストを出力する手法である．ファイルサイズや過去のバグレポートとの類似度はメソッドファイルからそのまま計算できるため，手法への修正は不要である．

実装の修正量が実際にどれほどだったか，各手法の実装の Java ファイルの LOC の合計に対する，メソッドレベルの推薦に変更するために必要な修正の LOC とその割合を**表 6** に示す．実際に修正が必要な手法は BRTracer と BLIA の 2 手法のみだった．2 手法はスタックトレースにあるファイルがバグを含みやすいとしてスコアを加算する．このファイル名の取得時にスタックトレースに記載があるファイルのメソッド名までを取得するように変更した．たとえば COMPRESS-203 [*3] に記載されたスタックトレースに含まれる行 “at org.apache.commons.compress.archivers.tar.TarArchiveOutputStream.writePaxHeaders(TarArchiveOutputStream.java:485)” から，メソッド名の `writePaxHeaders` までを取得し，対応するメソッドのスコアを加算するように修正した．スタックトレースからファイル名を抽出する機能の実装箇所を特定し，メソッドファイル名までを抽出する修正は容易であり，わずか 4 行の修正でメソッドレベルに変更できた．

表 6 メソッドレベル手法に変更するための修正量
Table 6 Changes required to fix techniques for method level.

IRBL 手法	全体の LOC	修正した LOC	割合 (%)
BugLocator	3,180	0	0
BLUiR	10,469	0	0
BRTracer	10,530	4	0.038
AmaLgam	10,469	0	0
BLIA	10,474	4	0.038

修正量はそれぞれ 0.038%と非常に少なかった．修正が必要な手法もあったが，ファイルサイズや履歴情報などのファイルに直接関連する情報を用いた手法には修正は不要だった．本論文のアプローチにより，ファイルレベルの IRBL 手法は非常に少ない修正でメソッドレベルに変更可能だった．すなわち，メソッドレベルの手法を必ずしも実装し直さなくとも，ファイルレベルの手法研究で得られた知見を細粒度化し確認可能なアプローチだと考える．

5.4 FinerBench4BL のデータセット

Bench4BL が提供する 46 プロジェクトのうち，実行時間が過大となった 9 プロジェクトを除いた 37 プロジェクトを用いて FinerBench4BL のデータセットを構築した．比較のため，両方の粒度で 5 つすべての IRBL 手法の出力が揃わなかったバグを除去した．たとえば，解決済のバグの修正箇所がすべてメソッド外にあり，メソッドレベルでの正解メソッドが空となったバグを除いた．全 479 バージョンから 25,966 個のソースファイルと 211,581 個のメソッドファイルを抽出し，3,344 個のバグレポートを得た．実験に用いたプロジェクトの詳細を**表 7** に示す．

本実験で扱った FinerBench4BL は GitHub にて公開している [*4]．

6. 実証的調査

本論文では FinerBench4BL を用いてメソッドレベルの IRBL 手法に関する調査を行い，以下のリサーチクエスチョ

*3 https://issues.apache.org/jira/browse/COMPRESS-203
*4 https://github.com/salab/FinerBench4BL

表 7 FinerBench4BL が含むプロジェクト
Table 7 Projects in FinerBench4BL.

	グループ・プロジェクト	F	M	V	B
Commons	CODEC	115	1,310	6	27
	COLLECTIONS	525	6,997	5	59
	COMPRESS	265	2,591	15	105
	CONFIGURATION	447	6,073	11	107
	CRYPTO	82	488	1	4
	CSV	29	452	3	6
	IO	227	2,608	12	70
	LANG	305	6,336	15	158
	MATH	1,617	15,695	15	175
	WEAVER	113	473	1	1
Jboss	ENTESB	252	3,210	1	4
	JBMETA	858	4,834	3	15
Spring	AMQP	408	3,996	32	86
	ANDROID	305	3,582	2	8
	BATCH	1,732	10,071	33	335
	BATCHADM	243	1,298	4	16
	DATACMNS	604	4,512	30	104
	DATAGRAPH	848	5,190	14	43
	DATAJPA	330	2,002	32	107
	DATAMONGO	622	6,703	40	209
	DATAREDIS	551	9,488	15	44
	DATAREST	414	2,183	23	89
	LDAP	566	3,556	5	46
	MOBILE	64	814	3	8
	ROO	1,109	7,803	15	568
	SEC	1,618	9,295	41	422
	SECOAUTH	726	3,912	6	61
	SGF	695	5,790	19	83
	SHDP	1,102	6,348	8	37
	SHL	151	749	2	6
	SOCIAL	212	1,344	4	10
	SOCIALFB	253	1,786	4	11
	SOCIALLI	180	830	1	2
	SOCIALTW	153	1,197	5	6
	SPR	6,512	57,696	10	89
	SWF	808	6,864	19	101
	SWS	925	3,505	24	122
	合計	25,966	211,581	479	3,344

F：ファイル数，M：メソッド数，V：バージョン数，B：バグ数

ン（RQ）に答える．

- RQ_1：メソッドレベルの IRBL 手法の精度はどれほどか
- RQ_2：メソッドレベルの IRBL 手法の労力はどれほどか
- RQ_3：IRBL 手法の粒度による追加情報の影響はどれほどか
- RQ_4：粒度により IRBL 手法の実行時間はどれほど変化するか

調査に用いるのは Bench4BL が提供した 6 手法の内，Locus [15] を除いた BugLocator [5]，BLUiR [6]，BRTracer [4]，AmaLgam [7]，BLIA [16] の 5 手法である．Bench4BL に含まれる Locus は実行途中にクラッシュし，結果の測定ができなかったため除外した．また，調査にあたり 5 手法の類似度計算に関わる欠陥を修正した．特に，BLUiR，AmaLgam，BRTracer には追加情報の計算に問題があったため，これらを一般化した手法である BLIA の実装を用いて再現した．

6.1 RQ_1：メソッドレベルの IRBL 手法の精度はどれほどか

6.1.1 動機

メソッドレベルの推薦における新たな知見を得るため，メソッドレベル推薦に修正した手法を精度の観点から評価し，性能の変化を明らかにする．

6.1.2 実験設定

同じ粒度内と異なる粒度間で手法の性能を精度の観点から比較する．指標として各プロジェクト内のバグレポートの Average Precision（AP）の平均である MAP（Mean Average Precision）と Reciprocal Rank（RR）の平均である MRR（Mean Reciprocal Rank），および top-N を用いる．top-N とは，ランク付リストの上位 N 番目までに 1 つ以上の正解を含むバグの割合を示す．

両側の Wilcoxon 符合順位和検定（$\alpha = 0.05$）を用いて 2 つの粒度間での手法の結果を比較した．

6.1.3 評価と考察

MAP の結果を元に精度の評価を行う．本項では FinerBench4BL のすべてのバグレポートを用いる場合と，修正メソッドが多すぎることでメソッドレベルの評価において不利に働く可能性があるバグレポートを除外した場合の 2 つに分けて分析を行った．

6.1.3.1 すべてのバグレポートを用いる場合

各プロジェクトの MAP の結果を**表 8** に示す．MRR の結果は MAP の結果と似た傾向を示したため，以下では MAP の分析結果のみを扱う．

表 8 に示すように，BLIA が最も MAP の中央値が高く，37 プロジェクト中メソッドレベルでは 26，ファイルレベルでは 20 で最大値を示した．また，18 プロジェクト（全体の 48.6%）に対しては，ファイルレベルで最も高い値を示した手法が，メソッドレベルでも最高の値で推薦した．この結果は，ファイルレベルでの精度の高さがメソッドレベルに引き継がれており，ファイルレベルの推薦で有効だった手法の選択がメソッドレベルでの推薦にも役立つことを示唆している．

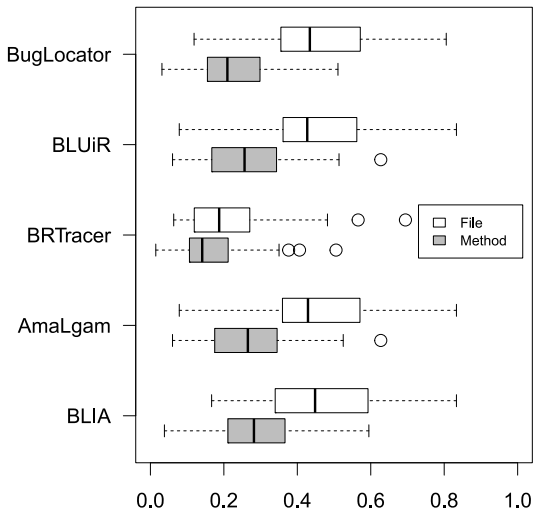
次に各手法の精度のメソッドレベルへの変更にともなう変化を調査した．ファイルレベルとメソッドレベルの結果を比較した MAP と MRR の箱ひげ図を**図 3** (a)，(b) に示す．各手法間の MAP の中央値にはすべて有意な差があり，ファイルレベルでの結果はメソッドレベルから見て，最小

表 8 MAP の結果
Table 8 Results of MAP.

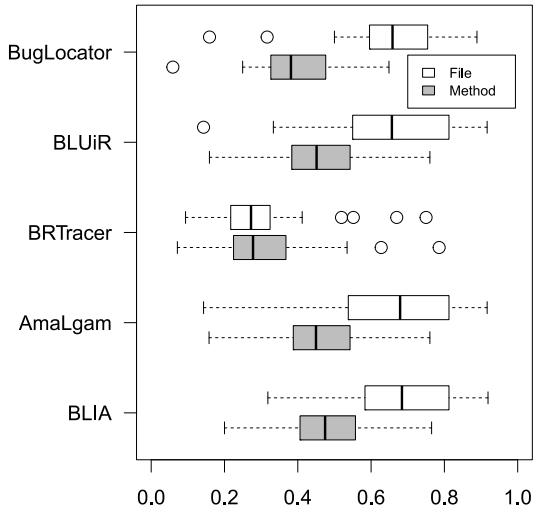
	ファイルレベル					メソッドレベル				
プロジェクト名	BugLocator	BLUiR	BRTracer	AmaLgam	BLIA	BugLocator	BLUiR	BRTracer	AmaLgam	BLIA
CODEC	**0.631**	0.623	0.283	0.629	0.621	0.192	0.352	0.094	0.345	**0.381**
COLLECTIONS	0.572	0.604	0.283	0.585	**0.614**	0.366	0.369	0.094	0.377	**0.394**
COMPRESS	0.631	**0.703**	0.263	0.678	0.696	0.281	0.318	0.170	0.302	**0.349**
CONFIGURATION	0.715	0.735	0.250	0.734	**0.776**	0.210	0.329	0.138	0.313	**0.363**
CRYPTO	0.314	0.313	0.063	0.322	**0.323**	0.106	0.060	**0.264**	0.060	0.058
CSV	0.806	**0.833**	0.482	**0.833**	**0.833**	0.155	0.423	0.154	0.437	**0.453**
IO	0.742	0.761	0.271	0.764	**0.772**	0.400	**0.506**	0.227	0.499	0.495
LANG	0.696	0.710	0.277	0.707	**0.738**	0.374	0.503	0.167	0.525	**0.536**
MATH	0.499	0.553	0.145	0.571	**0.578**	0.263	0.355	0.122	0.355	**0.375**
WEAVER	**0.321**	0.079	0.125	0.079	0.167	0.068	**0.083**	0.015	**0.083**	0.038
ENTESB	0.119	**0.431**	0.399	0.429	0.329	0.031	0.167	**0.350**	0.166	0.214
JBMETA	**0.359**	0.278	0.150	0.318	0.315	0.146	**0.226**	0.035	**0.226**	0.214
AMQP	0.404	0.421	0.086	0.422	**0.434**	0.155	0.167	0.106	0.165	**0.187**
ANDROID	0.540	0.562	**0.566**	0.499	0.556	0.298	0.343	0.237	0.353	**0.367**
BATCH	0.414	0.436	0.122	**0.448**	**0.448**	0.216	0.222	0.138	0.227	**0.241**
BATCHADM	0.438	0.553	0.276	0.549	**0.606**	**0.256**	0.223	0.136	0.248	0.225
DATACMNS	0.300	0.365	0.136	**0.369**	0.362	0.125	0.204	0.124	0.201	**0.214**
DATAGRAPH	0.145	0.171	0.107	0.182	**0.197**	0.145	0.171	0.106	0.182	**0.197**
DATAJPA	0.355	0.369	0.110	0.380	**0.394**	0.169	0.173	0.115	0.175	**0.183**
DATAMONGO	0.296	0.317	0.111	0.316	**0.340**	0.114	0.144	0.096	0.142	**0.165**
DATAREDIS	0.382	**0.410**	0.114	0.406	0.391	0.163	0.160	0.123	0.152	**0.190**
DATAREST	0.264	0.272	0.119	**0.290**	0.289	0.100	0.127	0.077	**0.128**	**0.128**
LDAP	0.498	0.476	0.223	0.487	**0.508**	0.227	0.300	0.212	**0.309**	**0.309**
MOBILE	**0.530**	0.427	0.251	0.427	0.427	0.372	**0.627**	0.376	**0.627**	0.595
ROO	**0.414**	0.375	0.127	0.384	**0.414**	0.200	0.193	0.087	0.200	**0.228**
SEC	0.505	0.533	0.221	0.545	**0.559**	0.299	0.334	0.186	0.339	**0.362**
SECOAUTH	0.434	0.426	0.157	0.429	**0.457**	0.301	0.275	0.176	0.285	**0.318**
SGF	**0.415**	0.380	0.155	0.384	0.404	**0.182**	0.159	0.123	0.168	0.169
SHDP	**0.387**	0.361	0.187	0.360	0.379	0.200	0.161	0.166	0.160	**0.211**
SHL	0.503	**0.593**	0.217	**0.593**	**0.593**	0.227	**0.327**	0.226	**0.327**	0.314
SOCIAL	**0.692**	0.570	0.292	0.600	0.684	0.316	0.499	0.406	0.499	**0.501**
SOCIALFB	**0.666**	0.470	0.222	0.495	0.520	0.184	0.300	**0.328**	0.318	0.318
SOCIALLI	**0.327**	0.300	0.084	0.300	0.303	0.511	0.514	0.505	0.514	**0.515**
SOCIALTW	0.693	0.462	**0.694**	0.462	0.512	**0.488**	0.320	0.193	0.320	0.282
SPR	**0.415**	0.277	0.119	0.321	0.339	0.203	0.167	0.141	0.227	**0.254**
SWF	0.461	0.424	0.253	0.411	**0.476**	0.212	0.243	0.191	0.229	**0.271**
SWS	0.270	0.257	0.105	0.264	**0.285**	0.272	0.256	0.095	0.265	**0.283**

で BRTracer の 1.33 倍，最大で BugLocator の 2.07 倍となり，総じてメソッドレベルでの精度はファイルレベルよりも低くなった．これは，メソッドレベルとなった結果，類似度を高く保てなくなったケースがある点に加え，メソッドレベルでは検索対象のファイル数と正解メソッド数のいずれもが増加するため，ファイルレベルよりもバグ箇所の特定が難化した点も影響していると考える．また，両粒度において BLIA の示す MAP の中央値が最大となり，以降 BLIA，AmaLgam，BLUiR，BugLocator，BRTracer という順となっており，BRTracer を除いて，扱う追加情報の数に応じて精度が低下した．追加情報をすべて利用する BLIA の精度が高いことは，メソッドレベルの Bug Localization において不足する情報を様々な観点から補うことで精度の維持が可能なことを示唆している．対して，構造情報のみを追加情報に用いる BLUiR の精度が，構造情報以外の情報も加味する AmaLgam や BLIA と同等だったことから，メソッドレベル推薦において構造情報の利用が有効な可能性がある．

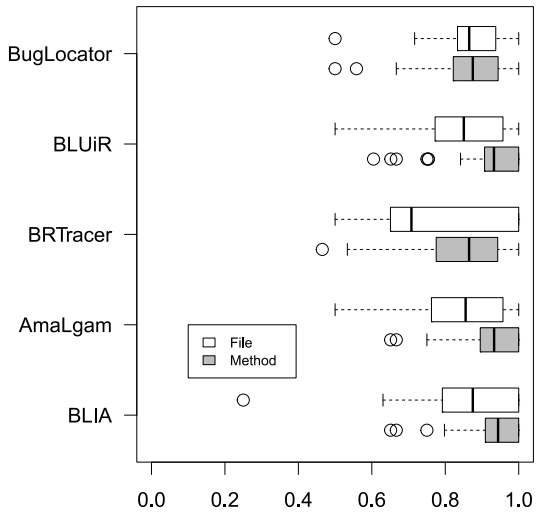
両レベルの結果を比較した top-N（$N \in \{5, 10\}$）の箱ひげ図の結果を**図 4** (a)，(b) に示す．Kochhar らの調査によれば，約 80%の開発者らはランク付リストの上位 5 位以内に，約 98%の開発者は上位 10 位以内に正解モジュールが



(a) MAP.

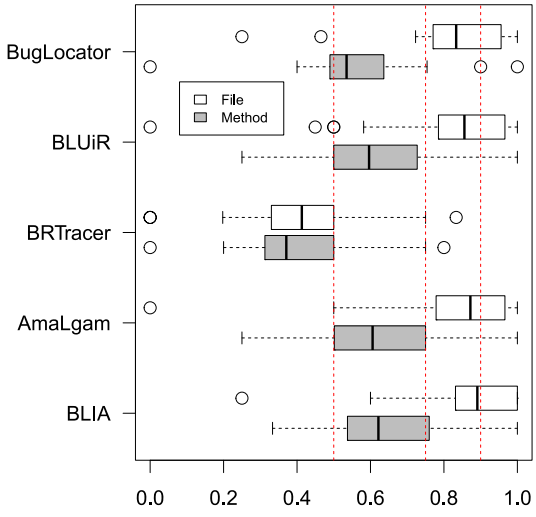


(b) MRR.

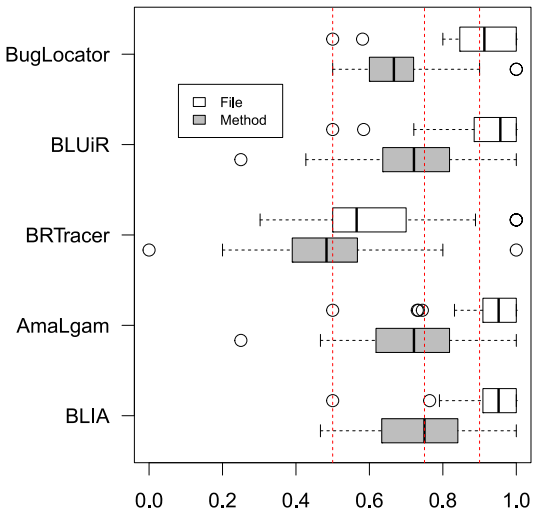


(c) Top-1,000 LOC.

図 3 IRBL 手法のそれぞれの粒度での評価
Fig. 3 Evaluation of IRBL techniques at each level.



(a) Top-5.



(b) Top-10.

図 4 IRBL 手法のそれぞれの粒度での top-N
Fig. 4 Top-N of the IRBL techniques at each level.

含まれることを期待している [24]．図 4 には，この調査結果に基づき，top-5 へ期待する最小値に対応する開発者の割合 50%，75%，90%の値に補助線を引いている．

図 4 (a) から，BLIA，BLUiR，AmaLgam の 3 手法で 0.6 以上の中央値を示しており，60%以上のバグレポートに対して上位 5 位以内に正解メソッドを推薦したことが分かる．また，図 4 (b) からは，これらの 3 手法の中央値が 0.7 を超え，70%以上のバグレポートで正解メソッドを上位 10 位以内に推薦したことが分かる．図 3 (a) に示した MAP の評価ではメソッドレベルの精度はファイルレベルと比較して低下したが，この top-5，top-10 の結果からは，ランク付リストの上位に正解が推薦されていることが分かる．また，メソッドレベルでは推薦モジュールのファイルサイズがファイルレベルよりも小さくなることから，開発者がランク付リストを処理する際の労力が軽減されることも考慮すると，MAP の低下ほどメソッドレベルで推薦する手

法の実用性は下がっていないと考える．

中央値に基づき，top-5，top-10 の値とそれらに対する開発者の期待を比較する．ファイルレベルの IRBL 手法は，top-5 において BRTracer を除く 4 手法が 75%の，top-10 では 90%の開発者の期待を満たす．一方，精度が下がったメソッドレベルにおいても，top-5 では BRTracer 以外の 4 手法で 50%の開発者の期待を満たす．top-10 においても，BLUiR，AmaLgam，BLIA の 3 手法は 75%程度の開発者を満足させる値が得られ，現状のメソッドレベル手法でも多くの開発者が満足する精度で推薦が可能であることを示唆している．

これまでの調査の中で，推薦の粒度の変化にともなって精度が大きく変動したバグがいくつか存在した．BugLocator の結果において粒度間で結果が異なった 2 例を紹介する．

まずは精度が下がった例を示す．COLLECTIONS-220 [*5]では，`UnboundedFifoBuffer.java` が含む `writeObject()` がバグの原因とされている．ファイルレベルでは対象ファイルが 1 位に推薦されたが，メソッドレベルでは対象メソッドが 789 位に推薦されていた．入力バグレポート中の “increment”，“tail”，“head” といった単語は，7 行の小さなメソッドである `writeObject()` にいっさい含まれておらず，同じファイル内の別のメソッドである `add()` や `remove()` に多く含まれていた．このことからファイルレベルの推薦では，無関係なメソッドの情報が推薦の精度を押し上げていたと考える．

CONFIGURATION-558 [*6]では精度が向上しており，ファイルレベルでは 22 位だった正解をメソッドレベルでは 7 位まで向上させた．バグレポートの内容は少ないものの，対象のメソッドに直接関係する引数や名前を含んでおり，これらのコード要素がメソッドレベルでの精度を高める要因になったと考える．これは Wang らや Rahman らが Program Entity を含むバグレポートが IRBL に適していると結論付けたことからも裏付けられる [25], [26]．

精度が大きく変動した 2 つのバグレポートの例から，ファイルレベルでは実際に修正されたメソッド以外の情報を情報検索に利用しており，開発者が扱う出力のランク付リストには無関係な情報が多く含まれることが示唆される．一方で妥当な推薦粒度といえるメソッドレベルでの推薦は情報検索で扱える情報が大きく不足する．このことから，メソッドレベルの IRBL には，Amasaki らが提案したメソッド外の情報を Bug Localization に利用するような [17]，ハイブリッドなアプローチが必要だと考える．

MAP の評価結果から，ファイルレベルで精度が良い手法はメソッドレベルでも精度良く推薦できることが分かった．したがって IRBL 手法のファイルレベルでの性能の向上は，メソッドレベル手法の性能の向上や評価の見積りにつなげることができ，有効に活用できると考える．また，ファイルレベルからメソッドレベルに変更したことで MAP の指標は大きく低下したものの，top-5，top-10 の指標ではそれぞれ 0.6，0.7 以上を示したことから，リポジトリ変換によりモジュール数が大きく増加してもランキングの上位に正解モジュールを推薦できる割合は高い．IRBL 手法を利用してバグ箇所を探す際にはランキングを参照することや，メソッドレベルではファイルレベルと比較してモジュール数が増加することを考慮すると，単純な精度の指標を表す MAP より，正解が上位に推薦されているかを表す top-N は重要である．今後のメソッドレベルの評価には，MAP だけではなく上位に正解モジュールが推薦されているか，top-N の観点からも継続的に評価すべきだと考える．

> 両方の粒度で BLIA の MAP の中央値が最も高かった．48.6%のプロジェクトで両粒度で最高精度の手法が共通しており，精度の良さはメソッドレベルにも引き継がれる．MAP の中央値には最大で 2.07 倍の落差があったものの，追加情報の利用がメソッドレベルの推薦にも有効だった．メソッドレベルの手法の多くが top-5 では 0.6 以上，top-10 では 0.7 以上の値を示し，現状の手法もある程度実用的な推薦が可能だと分かった．

6.1.3.2 正解モジュールをフィルタリングした場合

推薦粒度の変化にともない精度が大きく変動したバグの中には，メソッドレベルの推薦で的中させるべき正解モジュール数がファイルレベルよりも極端に多いものが含まれていた．たとえば CODEC-61 [*7]では，ファイルレベルでの正解が `Base64.java` のみだった一方．メソッドレベルでは `Base64.java` が含む 12 個のメソッドが正解となり，的中させる必要がある正解モジュール数に大きな差が生じている．メソッドレベル化にともない正解モジュール数が増大した場合，各メソッドにおける修正箇所が局所化し，バグレポートとのテキスト類似度が低くなるため，よりバグ箇所の特定が難化すると考える．また，Hata らによれば，バグのあるメソッド数の中央値は 1〜2 であり [18]，これと比較しても 12 個の正解メソッドは評価に適切とはいえない．

したがって正解のモジュール数が多いバグレポートを除去した条件下で，両粒度のより均一な比較を行う．Lucia らの調査によると，83%のバグの原因となるメソッドは 6 つ以内で収まることが分かっているため [27]，正解メソッド数が 6 つ以下となるバグレポート 3,344 件中 2,322 件にフィルタリングしたうえで MAP の結果を調査した．

結果を**図 5** に示す．各粒度において下にある箱ひげ図が

[*5] https://issues.apache.org/jira/browse/COLLECTIONS-220
[*6] https://issues.apache.org/jira/browse/CONFIGURATION-558
[*7] https://issues.apache.org/jira/browse/CODEC-61

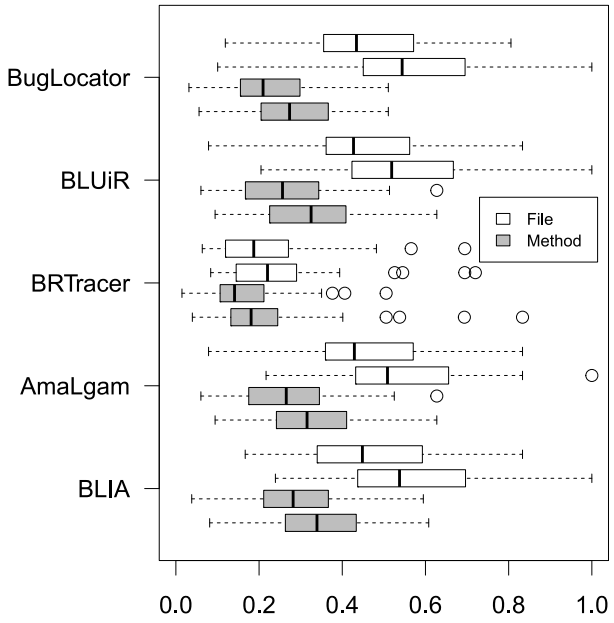


図 5 正解モジュール数が 6 以下の MAP の比較
Fig. 5 Comparison of MAP with ≤ 6 fixed modules.

フィルタリング後の結果である．すべての手法と粒度において，フィルタリング後の MAP の中央値はフィルタリング前より上昇した．

6.1.3 節から，メソッドレベルの結果はファイルレベルと比較して MAP の中央値に 1.33〜2.07 倍の精度低下があった．しかし，フィルタリングを行ったことで，精度低下は 1.22〜1.99 倍にとどまった．また，メソッドレベルにおいて，BLUiR，AmaLgam，BLIA の 3 手法はフィルタリングにより MAP の中央値が 0.3 を超えた．以上により，正解モジュール数が多いバグレポートは精度の評価に悪影響を与えていたと考える．

フィルタリングされたバグレポートのうち，正解数が 100 を超えるようなものには，修正にともないメソッド名や引数の変更などにより，多くの場所で同様の修正を要するバグが存在した．こうした systematic edits [28] を正解に含むバグレポートは IRBL 手法の評価に適しておらず，データセットから除外するべきである．正解モジュール数によるフィルタリングは systematic edits を含む可能性が高いバグレポートを除いた評価が可能だと考える．

手法が的中させるべき正解メソッド数が多いバグレポートは手法の精度の評価に悪影響を与える．また，メソッドレベルの精度だけではなくファイルレベルの精度も大きく向上したことから，正解メソッド数が多くフィルタリングされたバグレポートは systematic edits を正解に含む Bug Localization 手法を用いた推薦に適さない可能性が高い．IRBL 手法の評価時には，正解モジュール数が多く評価に悪影響を与えうるバグレポートを除く必要があり，多くのメソッドが関連する可能性が高いバグレポートを事前に予測することで，IRBL 手法の適用可否の考慮や高精度な推薦につながる可能性がある．

表 9 top-1,000 LOC の中央値
Table 9 Median of top-1,000 LOC.

	ファイル	メソッド	p 値	Cliff's d
BugLocator	0.865	0.875	0.644	0.063 (negligible)
BLUiR	0.850	0.933	0.026	0.297 (small)
BRTracer	0.708	0.865	0.024	0.301 (small)
AmaLgam	0.855	0.933	0.022	0.305 (small)
BLIA	0.875	0.944	0.027	0.295 (small)

正解数が 6 つ以下であるバグレポートのみを対象としたところ，すべての手法の両粒度において MAP の中央値は大きく向上した．正解モジュール数が多いバグレポートは systematic edits を正解に含み，IRBL 手法を用いた推薦に向かない可能性が高い．

6.2 RQ_2：メソッドレベルの IRBL 手法の労力はどれほどか

6.2.1 動機

Bug Localization における開発者がデバッグに必要な労力に関する新たな知見を得るため，メソッドレベル推薦に修正した手法を労力の点から評価し，性能の変化を明らかにする．

6.2.2 実験設定

同じ粒度内と異なる粒度間で手法の性能を労力の観点から比較する．指標として top-k LOC [11] を用いる．top-k LOC とは，ランク付リスト上位からたかだか k 行のコードを読むまでに正解モジュールが少なくとも 1 つ出現するようなバグレポートの割合を表す．たとえば top-1,000 LOC = 0.1 のとき，10%のバグレポートに関してリストの上位 1,000 行以内に正解のソースファイルが存在することを示す．

両側の Wilcoxon 符合順位和検定（$\alpha = 0.05$）を用いて 2 つの粒度間での手法の結果を比較した．

6.2.3 評価と考察

図 3 (c) はランク付リストからバグ箇所を特定するまでにかかる労力の指標である top-1,000 LOC の結果を示している．**表 9** は top-1,000 LOC の中央値，両側の Wilcoxon 符合順位和検定における p 値，Cliff の delta [29] の値を示す．すべての手法において，メソッドレベルへの変更で top-1,000 LOC の値は向上した．このうち BugLocator 以外の手法には，粒度間での値に有意な差が存在した．

ファイルレベルの性能と比較したとき，値は 2.2%（BugLocator）から 22.2%（BRTracer）の範囲で増加している．また，両方の粒度において，BLIA が最も良い結果を示し，メソッドレベルでは 94.4%のバグレポートに対して出力のランクリストの 1,000 行以内にバグの原因となるメ

ソッドファイルを特定していた．

ファイルレベルの top-1,000 LOC の性能がメソッドレベルより低くなった理由として，精度の良さに対して各ファイルの LOC が大きい分，ランク付リストを読む行数が多くなるためだと考える．このことから，現状の高精度ではないメソッドレベルの推薦でも開発者がデバッグ時に読む行数を削減可能だと考える．

手法の序列をみると，両粒度において BLIA の値が最も大きく，BRTracer が最も小さい値を示した．有意差が得られなかった BugLocator を除けば，各手法の top-1,000 LOC の序列は推薦粒度間で変化せず，BLIA，AmaLgam，BLUiR，BRTracer の順となった．ファイルレベルでの性能の良さはメソッドレベルでも保たれていることが分かる．また，いずれの粒度においても構造情報のみを追加情報に用いる BLUiR の性能が良いことから，構造情報の考慮が top-1,000 LOC の向上につながると考える．

top-1,000 LOC の比較により，メソッドレベル手法はファイルレベルでのバグ箇所の推薦と比べて，開発者がランキングを読み，正解モジュールにたどり着くまでの労力を削減したことが分かった．より高精度にメソッドレベルでもバグ箇所の推薦を可能にする手法であれば，さらに少ない労力で正解モジュールを発見できると期待される．精度における top-5，top-10 の評価と同様に，MAP が低下したとしても手法が開発者に与える価値が低下するとは限らず，正解のモジュールへたどり着くための労力の計測は今後のメソッドレベルの IRBL 手法の評価でも継続して行うべきだと考える．

> 1 手法を除いて，メソッドレベル手法は top-1,000LOC を有意に向上させた．特に BLIA は 94%以上のバグレポートにおいて正解メソッドを 1,000 行以内に特定可能だった．

6.3 RQ_3：IRBL 手法の粒度による追加情報の影響はどれほどか

6.3.1 動機

メソッドレベル推薦への変更にともない，各メソッドファイルが持つ履歴長や関連するバグレポートの現象などにより，追加情報の精度への影響が小さくなる可能性がある．メソッドファイルから得られる追加情報がどれほど精度に寄与できたのかを明らかにする．

6.3.2 実験設定

BLIA を用いて，モジュールとバグレポートとの類似度の計算過程で加算する追加情報のスコアを分析する．BLIA は表 2 に示すように，FinerBench4BL が含む手法が考慮する追加情報をすべて扱うことから，BLIA を対象とすることですべての追加情報の分析が可能だと考えた．本実験では計算過程で得られる追加情報なしの類似度スコアと，過去のバグレポートとの類似度，スタックトレース，履歴情報を考慮する際に加算する各スコアを用いる．メソッドレベルでの AP の値の増加率に対する各スコアの増加率を分析する．

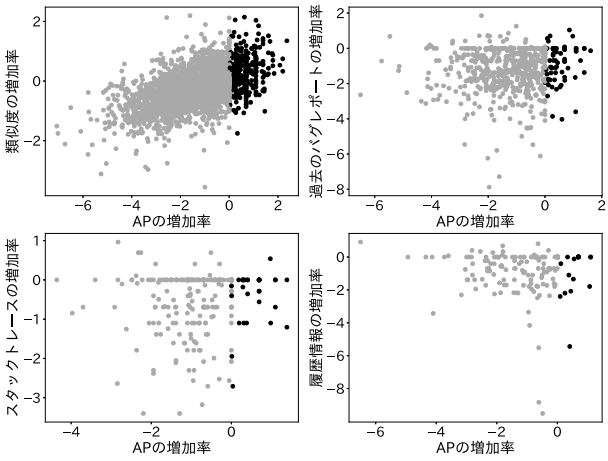


図 6 BLIA の精度の増加率に対する追加情報の増加率
Fig. 6 BLIA's additional information increase per AP.

6.3.3 評価と考察

図 6 に，各バグレポートについて BLIA の AP の増加率に対する追加情報の増加率を表す散布図を示す．ファイルレベルでスコアが計算されなかったバグレポートを除外し，増加率はそれぞれ対数をとった．黒色・灰色のプロットは，メソッドレベルにしたことで AP が向上・低下したバグレポートをそれぞれ表している．

過去のバグレポートとの類似度，スタックトレース，履歴情報を加味したスコアにおいて，スコアの増加率が 1 を上回ったバグレポートはそれぞれ 40，10，13 件だった．したがって大多数のバグレポートにおいて，ファイルレベルからメソッドレベルに変更したことで追加情報のスコアは下がり，メソッドレベルにおける追加情報の精度への影響はファイルレベルと比較して小さくなっている．

一方，すべての追加情報において正の相関があり，追加情報が活用されているバグレポートであるほど精度が高く，AP が増加したバグレポートを表す黒色のプロットの方がスコアの増加率のばらつきが小さい．黒色のプロットに着目すると，精度向上の原因が追加情報にあるとはいい切れず，追加情報なしで類似度を計算したスコアが最も相関が強いことから，元のメソッドとバグレポートとの類似度が高いものが，特にメソッドレベルにしたことで精度が向上したと考える．

すべての追加情報に AP との正の相関があるものの，相関は弱く，大きく精度に貢献したとはいい切れない．これは，ファイルレベルと同様の追加情報の利用方法では，検索対象であるソースコード自体が小さいメソッドレベルで情報を十分に活用できていないからだと考える．様々な追加情報の考慮に加え，不足するテキスト情報を他のメソッ

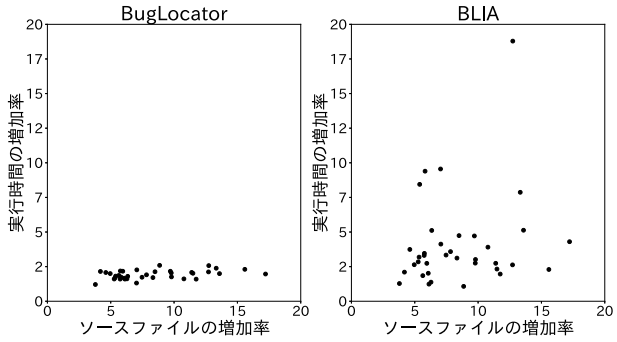


図 7 ファイル数の増加率に対する実行時間増加率
Fig. 7 Increase in execution time per number of files.

ドから補ったり，追加情報を考慮する際の重みを変更したりすることでさらなる精度の向上につながると考える．

メソッドレベルへの変更により追加情報の精度への影響は低下した．一方で，追加情報が活用されたバグレポートであるほど精度が向上した．よって，BLIA のように多くの追加情報を考慮し，メソッドファイルへの分割により失われた情報を多角的に補填することが望ましい．

6.4 RQ_4：粒度により IRBL 手法の実行時間はどれほど変化するか

6.4.1 動機

IRBL 手法をメソッドレベルに変更したことで処理する必要があるファイル数が増加し，実行に時間がかかることが予想されるが，手法により増加幅の程度が明確ではない．メソッドレベルに変更後も IRBL 手法が実用可能な範囲で実行可能かどうかを調査する．

6.4.2 実験設定

BugLocator と BLIA における，各 37 プロジェクトの FinerBench4BL が含む最新リリースバージョンの実行時間を用いる．

BugLocator は少数の追加情報のみを扱う比較的単純な手法として，BLIA は最も多くの追加情報を扱う複雑な手法として選択した．Bench4BL が提供した BLUiR，AmaLgam，BRTracer のオリジナル実装には追加情報の計算に問題があり，本論文では BLIA の実装を用いて再現したため評価対象から除外した．メソッドファイルに分割したことで増えた入力のソースファイル数の増加率に対して，メソッドレベルの実行時間のファイルレベルの実行時間からの増加率を比較し，傾向を分析する．

6.4.3 評価と考察

図 7 は各プロジェクトにおける実行時間の増加率を示す散布図である．左は BugLocator，右は BLIA の結果を表す．縦軸はメソッドレベル手法への変更時の実行時間の増加率を示し，横軸はリポジトリ変換にともなうソースファイルの増加率を示す．

すべてのプロジェクトにおいて，メソッドレベルに変更したことで実行時間が増加した．実行時間が増加したメソッドレベルにおいて，BugLocator は最も実行時間がかかるプロジェクトで 66.31 秒，BLIA は 563.89 秒で実行が完了した．最も時間がかかったプロジェクトでも実行時間が 10 分以内に収まることから，メソッドレベルで推薦を行っても十分に実用可能だと考える．

BugLocator は，ファイル数の増加率に対して実行時間の増加率が一定だった．ファイル数がどれだけ倍増しても実行時間の増加率はたかだか 2.59 であり，ファイルレベルでの実行時間から予想できる範囲内に収まる．BugLocator は各ファイルに対してファイルサイズと過去のバグレポートへの類似度を考慮するが，ファイル数が増加した分，ファイルサイズが小さくなったことでそれほど実行時間に影響が出なかったと考える．

一方，BLIA は実行時間の増加率にばらつきがあったが，おおむねファイル数の増加率より小さい増加率で実行時間が増えた．最も実行時間の増加率が高いプロジェクトでは，ファイル数の増加率が 12.74 に対して実行時間の増加率は 18.79 となった．BugLocator と比較して扱う追加情報の数も多く，バグレポートによってはスタックトレースを含まないものもあることから，実行時間が増加する要因がファイル数のみに依存しているわけではないため，明確な傾向が現れていない可能性がある．

2 手法とも，メソッドレベルへの変更で増加したソースファイル数ほど，実行時間が増加してはいないことが分かった．また．図 7 における各プロットはいずれもプロジェクト単位であり，1 バグレポートに対する実行時間はさらに短い．以上により，メソッドレベル手法では実行時間が増加するとはいえ，十分実用に足る速度にとどまっていると考える．

推薦粒度をメソッドレベルにした結果，すべてのプロジェクトで実行時間が増加したものの，すべて 10 分以内に実行が完了し，実用可能な範囲に収まった．各ファイルに対して追加情報の処理を行う BugLocator はファイル数の増加率に対して実行時間の増加率が一定だった一方，バグレポートにより処理数が変動する BLIA ではより長い実行時間を必要とした．

7. 妥当性の脅威

バグレポートに紐づく正解ファイルの正しさによる妥当性の脅威がある．本論文では Bench4BL が提供するスクリプトを用いてバグレポートと正解ファイルを対応付けたが，目視で確認しておらず，コミットログから特定できていないバグレポートや，バグ修正以外のファイルを正解に含むバグレポートが存在する可能性がある．

FinerBench4BL を構築するにあたり，規模が大きく，実

行時間が非常に長いプロジェクトを省いて評価を行った．Bench4BL と比較するとデータセットの規模が小さく，省いたプロジェクトの中に結果に影響を及ぼすものが存在した可能性がある．

また，労力を評価する指標として top-k LOC を採用したが，バグ箇所の特定に必要な労力への妥当性は明らかでない．今回の評価では出力のリストのランク順に，正解の 1 つ前までのファイルの行数の累計を用いており，正解が 1 位に出力されると必要な LOC が 0 となるため，実際の労力とは異なる．また，開発者はメソッドだけを見てバグを発見できるとは限らず，LOC の算出自体も労力による比較に影響を与えうる．

RQ_4 において，手法の実行時間の調査対象として BLIA と BugLocator のみを用いた．これは，Bench4BL が提供する BLUiR，BRTracer，AmaLgam のオリジナル実装に問題があり，使用を控えたためである．しかし，代替として用いた BLIA とそれぞれの手法の実装の実行時間は異なる可能性があり，3 手法に関して正しく実行時間を分析できたとはいえない．ただし，これらの 3 手法は BugLocator よりも複雑で，かつ BLIA よりも少ない追加情報を扱う．3 手法が扱う追加情報はすべて BLIA に含まれており，実行時間はこれら 2 手法の結果の間におさまると予測している．

8. おわりに

本論文では，リポジトリ変換を用いて少ない修正でメソッドレベルへ変換した IRBL 手法を用い，各モジュール粒度で様々な観点から評価を行った．

評価の結果，既存手法をメソッドレベルに変換すると精度が低下する一方で，デバッグにかかる労力を削減することが分かった．推薦粒度をメソッドレベルにしたことで各手法が扱う追加情報が精度へ与える影響も小さくなるものの，追加情報を多く考慮することが精度向上につながる．したがって，Amasaki らが述べるように，メソッドレベル手法において不足する情報量をメソッド外の情報で補うことにより，さらなる性能向上の余地がある [17]．また，メソッドレベルへ変換した IRBL 手法の実行時間は増加してもたかだか 10 分であり，実用可能な時間で推薦が可能である．

本研究の今後の課題を以下に示す．

- **メソッドレベル手法のパラメータの調整**．本論文のメソッドレベル手法はファイルレベル用に調整されたパラメータで実行したが，最適な調整を明らかにすればさらなる性能向上が可能である．
- **メソッド外のコード要素の考慮**．本実験では除外したフィールド変数や import などのメソッド外の要素を考慮することで異なる知見が得られる可能性がある．
- **実行時間に与える要因の分析**．BLIA の実行時間の増加率にばらつきがあった原因を，各追加情報の処理時間に着目して詳細に分析する．

謝辞 本研究の一部は日本学術振興会科学研究費補助金（JP21H04877, JP21K18302, JP21KK0179, JP21K11833, JP22H03567）の助成を受けた．

参考文献

[1] Zeller, A.: *Why Programs Fail: A Guide to Systematic Debugging*, Morgan Kaufmann (2006).
[2] 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).
[3] 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 IEEE/ACM International Conference on Automated Software Engineering* (*ASE'11*), pp.263–272 (2011).
[4] 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 IEEE International Conference on Software Maintenance and Evolution* (*ICSME'14*), pp.181–190 (2014).
[5] 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* (*ICSE'12*), pp.14–24 (2012).
[6] Saha, R.K., Lease, M., Khurshid, S. and Perry, D.E.: Improving bug localization using structured information retrieval, *Proc. 28th IEEE/ACM International Conference on Automated Software Engineering* (*ASE'13*), pp.345–355 (2013).
[7] 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* (*ICPC'14*), pp.53–63 (2014).
[8] Razzaq, A., Buckley, J., Patten, J.V., Chochlov, M. and Sai, A.R.: BoostNSift: A Query Boosting and Code Sifting Technique for Method Level Bug Localization, *Proc. 21st IEEE International Working Conference on Source Code Analysis and Manipulation* (*SCAM'21*), pp.81–91 (2021).
[9] Zhang, W., Li, Z., Wang, Q. and Li, J.: FineLocator: A novel approach to method-level fine-grained bug localization by query expansion, *Information and Software Technology*, Vol.110, pp.121–135 (2019).
[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] Tantithamthavorn, C., Lemma Abebe, S., Hassan, A.E., Ihara, A. and Matsumoto, K.: The impact of IR-based classifier configuration on the performance and the effort of method-level bug localization, *Information and Software Technology*, Vol.102, pp.160–174 (2018).
[12] Lee, J., Kim, D., Bissyandé, T.F., Jung, W. and Le Traon, Y.: Bench4BL: Reproducibility Study on the Performance of IR-Based Bug Localization, *Proc. 27th ACM SIGSOFT International Symposium on Software Testing and Analysis* (*ISSTA'18*), pp.61–72 (2018).
[13] Tsumita, S., Hayashi, S. and Amasaki, S.: Large-

Scale Evaluation of Method-Level Bug Localization with FinerBench4BL, *Proc. 30th IEEE International Conference on Software Analysis, Evolution and Reengineering* (*SANER'23*), pp.815–824 (2023).
[14] Akbar, S.A. and Kak, A.C.: A large-scale comparative evaluation of IR-based tools for bug localization, *Proc. 17th International Conference on Mining Software Repositories* (*MSR'20*), pp.21–31 (2020).
[15] Wen, M., Wu, R. and Cheung, S.-C.: Locus: Locating Bugs from Software Changes, *Proc. 31st IEEE/ACM International Conference on Automated Software Engineering* (*ASE'16*), pp.262–273 (2016).
[16] Youm, K.C., Ahn, J., Kim, J. and Lee, E.: Bug Localization Based on Code Change Histories and Bug Reports, *Proc. 22nd Asia-Pacific Software Engineering Conference* (*APSEC'15*), pp.190–197 (2015).
[17] Amasaki, S., Aman, H. and Yokogawa, T.: On the Effects of File-level Information on Method-level Bug Localization, *Prec. 46th Euromicro Conference on Software Engineering and Advanced Applications* (*SEAA'20*), pp.314–321 (2020).
[18] Hata, H., Mizuno, O. and Kikuno, T.: Bug prediction based on fine-grained module histories, *Proc. 34th International Conference on Software Engineering* (*ICSE'12*), pp.200–210 (2012).
[19] Dallmeier, V. and Zimmermann, T.: Extraction of bug localization benchmarks from history, *Proc. 22nd IEEE/ACM International Conference on Automated Software Engineering* (*ASE'07*), pp.433–436 (2007).
[20] Rao, S. and Kak, A.: moreBugs: A New Dataset for Benchmarking Algorithms for Information Retrieval from Software Repositories, Technical Report TR-ECE-13-07, Purdue University, School of Electrical and Computer Engineering (2013).
[21] Sisman, B. and Kak, A.C.: Assisting code search with automatic query reformulation for bug localization, *Proc. 10th Working Conference on Mining Software Repositories* (*MSR'13*), pp.309–318 (2013).
[22] Chaparro, O., Florez, J.M. and Marcus, A.: Using bug descriptions to reformulate queries during text-retrieval-based bug localization, *Empirical Software Engineering*, Vol.24, No.5, pp.2947–3007 (2019).
[23] 柴 駿太，林 晋平：Historinc: 細粒度履歴追跡のための増分的なリポジトリ変換ツール，コンピュータソフトウェア，Vol.39, No.4, pp.75–85 (2022).
[24] Kochhar, P.S., Xia, X., Lo, D. and Li, S.: Practitioners' Expectations on Automated Fault Localization, *Proc. 25th International Symposium on Software Testing and Analysis* (*ISSTA'16*), pp.165–176 (2016).
[25] Wang, Q., Parnin, C. and Orso, A.: Evaluating the Usefulness of IR-Based Fault Localization Techniques, *Proc. 24th International Symposium on Software Testing and Analysis* (*ISSTA'15*), pp.1–11 (2015).
[26] Rahman, M.M. and Roy, C.K.: Improving IR-Based Bug Localization with Context-Aware Query Reformulation, *Proc. 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (*ESEC/FSE'18*), pp.621–632 (2018).
[27] Lucia, Thung, F., Lo, D. and Jiang, L.: Are faults localizable?, *Proc. 9th IEEE Working Conference on Mining Software Repositories* (*MSR'12*), pp.74–77 (2012).
[28] Meng, N., Kim, M. and McKinley, K.S.: Systematic Editing: Generating Program Transformations from an Example, *Proc. 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation* (*PLDI'11*), pp.329–342 (2011).
[29] Romano, J., Kromrey, J.D., Coraggio, J. and Skowronek, J.: Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen's d for evaluating group differences on the NSSE and other surveys?, *Proc. 2006 Annual Meeting of the Florida Association of Institutional Research* (*FAIR'06*), pp.1–33 (2006).

積田 静夏

2022 年東京工業大学情報理工学院情報工学系学士課程卒業．2024 年同大学情報理工学院情報工学系情報工学コース修士課程修了．修士（工学）．Bug Localization に関する研究に従事．IEEE 会員．

天嵜 聡介 （正会員）

2006 年大阪大学大学院情報科学研究科後期課程修了．岡山県立大学准教授．工数見積もり手法や不具合モジュール判別手法等の研究に従事．博士（情報科学）．IEEE-CS，ACM 各会員．

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．同大学助教を経て，2018 年より同大学情報理工学院准教授．博士（工学）．ソフトウェア進化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．