言語横断バグ箇所検索手法の日本語記述への適用可能性

林 晋平[1,a)]　小林 隆志[1,b)]　高井 康勢[2,c)]　加藤 正恭[2,d)]

概要：Xia らにより提案された言語横断バグ箇所検索手法を，日本語を用いた開発プロジェクトへ適用した結果を報告する．バグレポートおよびソースコード中の日本語記述に機械翻訳を適用し，それぞれを英語に統一したうえで，英語記述に基づくバグ箇所検索手法を適用することにより，言語横断バグ箇所検索を実現した．バグ箇所検索手法 BugLocator に基づくツールを試作し，2 つの企業プロジェクトに適用したところ，Xia らの結果とは異なり，良好な結果が得られた．

1. はじめに

ソフトウェア開発において，バグ修正には多くの時間を必要とする．開発者がバグのないソースコードを記述し続けることは困難であり，バグは頻繁に発生する．利用者や他の開発者によって発見されたバグは，バグレポートとして報告され，開発者はこれを確認しバグを修正する [1]．バグレポートを参照してバグを修正する場合，バグの原因となるソースコードを特定する必要がある．バグの情報をもとにバグの原因となるソースコードを絞り込む作業を，バグ箇所検索（bug localization; BL）と呼ぶ．この作業をそれぞれのバグに対して実施する必要があり，プロジェクトが大規模になるにつれ時間と労力のかかる作業となる [2]．

バグ修正作業の効率化のため，自動バグ箇所特定手法が研究されている．バグ情報に基づきバグ箇所の候補を出力するバグ箇所検索手法は，機能情報に基づき機能実装箇所を特定する機能捜索（feature location; FL）の手法に類似しており，Dit ら [3] によるとそれらは以下のように分類される．

- 動的解析手法：実行中のソフトウェアの挙動を分析する [4]．
- 静的解析手法：プログラム要素の構造情報や要素間の関係性を静的に分析する [5]．
- 情報検索（Information Retrieval; IR）手法：ソースコードを文書と見立て，文書検索を行う [6]．
- 履歴解析手法：版管理システムにより記録された変更履歴を解析する [7]．

これらの手法のうち，情報検索手法以外は実行シナリオ等の追加情報やプログラムの精密な解析が必要である．特に，動的解析手法には，テストケース毎にテスト実行時の特性とテスト結果を記録・分析することで不具合原因箇所の候補を提示する Fault Localization 技術などがあるが，こういった手法では不具合を再現させるテストケースを用意する必要がある．オープンソースソフトウェアと違い，利用者による不具合の再現条件の初期分析や原因の絞り込みが期待できない状況では，不具合に関する情報の一次情報はバグレポートだけであることも少なくない．本論文では，最も利用しやすく追加情報の不要な手法である情報検索手法を利用する．

これまでに提案されている，情報検索に基づくバグ箇所検索手法の多く [8], [9], [10] は，バグレポートが英語で記述されており，またソースコードが英語に基づく識別子を用いて記述されていることを前提としている．これは，ソースコード中の識別子を英単語の列とみなし，バグレポート中の英語記述との語彙的類似性に注目しているためである．しかし，英語以外を母国語とする開発者は，しばしば英語以外でバグレポートを記述する．英語以外で記述されたバグレポートを受け取った開発者が，英語を前提とする既存のバグ箇所検索手法を利用することは難しい．

この問題に対し，Xia らは，機械翻訳の適用により，英語以外で記述されたバグレポートに対してもバグ箇所検索を可能とする言語横断バグ箇所検索手法 CrosLocator を提案し，中国語で記述されたバグレポートでのバグ箇所検索性能を調査した [11]．その結果，機械翻訳後に利用してい

[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] 株式会社日立製作所 研究開発グループ デジタルサービス研究統括本部
Hitachi, Ltd., Research & Development Group, Center for Digital Services, Yoshida-cho 292, Totsuka, Yokohama, Kanagawa 244–0817, Japan
[a)] hayashi@c.titech.ac.jp
[b)] tkobaya@c.titech.ac.jp
[c)] yasunari.takai.kb@hitachi.com
[d)] tadahisa.kato.en@hitachi.com

るバグ箇所検索手法 BugLocator の論文 [8] で示されている性能と比較し，CrosLocator はその性能が不十分だったとして，Xia らは言語横断バグ箇所検索は通常のバグ箇所検索と比較して難しいと結論付けている．しかし，言語横断バグ箇所検索の適用報告例は少ない．Xia らの調査以外には，複数の自然言語を対象とした三井らによる調査 [12] があるものの，これはオープンソースプロジェクトに対する適用結果の分析に留まっている．

本論文では，日本語を対象とした言語横断バグ箇所検索の実現可能性の調査結果を報告する．

本稿の貢献を以下に示す．

- ソースコード中の非英語に対しても機械翻訳を適用することによる言語横断バグ箇所検索の性能向上．
- 2 つの企業プロジェクトを対象とした性能の調査．

本稿の構成を以下に示す．2 章では，情報検索に基づくバグ箇所検索の既存手法について解説する．3 章では本論文で利用する言語横断バグ箇所検索法について説明する．4 章では言語横断バグ限局の精度について述べる．最後に，5 章で本論文をまとめ，今後の課題を整理する．

2. 準備

2.1 情報検索に基づくバグ箇所検索

情報検索に基づくバグ箇所検索では，ソースコードとバグレポートを文書と見立て，テキスト分析によりそれらの類似度を求める．与えられたバグレポートと各ソースコードとの類似度を求め，類似度に基づいて順位付けされたソースコードのリストを出力する．

2.1.1 VSM

ベクトル空間モデル（vector space model; VSM）は，情報検索手法の一つであり，文書をベクトル化することにより，文書間の類似度を計算する．ベクトル化においては，文書中に含まれる単語の重要度を考慮する tf-idf 手法などが用いられる．

複数ある tf-idf の計算方法のうち，本論文では Zhou ら [8] による以下の定義を用いる．

$$tf(t,d) = \log\left(\frac{c_{t,d}}{c_d} + 1\right)$$

$$idf(t) = \log\frac{|D|}{|\{\, d \mid c_{t,d} > 0 \,\}|}$$

ここで，$d \in D$ は文書，$t \in T$ は単語を表し，$c_{t,d}$ は文書 d における単語 t の出現数，c_d は d 中の全単語数である．出現頻度（term frequency）tf はその単語の出現数が高いほど大きく，逆文書頻度（inverse document frequency）idf はその単語が多くの文書に出現するほど小さくなる．一般性の高い単語の重要性は低いとして，d 中の t の重さに積 $tf(t,d) \cdot idf(t)$ を用いる．各単語の重さを要素とする T 次元のベクトル $\vec{d}$ として文書をベクトル化する．

与えられたソースコード d とバグレポート q の類似性スコアを，それらのベクトルのコサイン類似度として求める．

$$\cos(d_1, d_2) = \frac{\vec{d_1} \cdot \vec{d_2}}{|\vec{d_1}||\vec{d_2}|}$$

$$\mathrm{VSMScore}(q,d) = \cos(q,d)$$

ファイルサイズの大きいソースコードはバグを含む確率が高い．しかし単純な VSM においては，ファイルサイズの大きいソースコードのスコアが実際にバグを含む確率よりも低く評価されてしまう傾向にある．

2.1.2 rVSM

rVSM（revised VSM）[8] は，VSM の問題点を解決するために，ファイル内の単語数を考慮した VSM である．

$$N(c) = \frac{c - c_{\min}}{c_{\max} - c_{\min}}$$

$$\mathrm{rVSMScore}(q,d) = \frac{1}{1 + e^{-N(c_d)}}\mathrm{VSMScore}(q,d)$$

ここで，$N(c)$ は単語数 c の $[0,1]$ への線形正規化であり，$c_{\max}$，$c_{\min}$ はそれぞれ，対象とする文書集合の中で最大，最小の単語数を表す．単語数，すなわちファイルサイズが大きくなるほど $N(c)$ は 1 に近づき，VSMScore からの補正重みは大きくなる．$\mathrm{rVSMScore}(q,d)$ は，ソースコード d が大きくなるほど高い値を取ることから，ファイルサイズを考慮した類似度を算出する．

2.1.3 BugLocator

BugLocator[8] は，過去に修正されたバグレポートを考慮した rVSM である．過去のバグ情報との類似度を計算することにより，rVSM の精度向上を目指す．

$$\mathrm{SimiScore}(q,d) = \sum_{b \in B_d} \frac{\cos(q,b)}{n_b}$$

$$\mathrm{BugLocator}(q,d) = (1-\alpha)\, N(\mathrm{rVSMScore}(q,d)) + \alpha\, N(\mathrm{SimiScore}(q,d))$$

ここで，B_d は d を修正ファイルに持つ過去のバグレポートの集合，n_b はバグ b の解決に要した修正ファイル数である．$\alpha \in [0,1]$ は rVSMScore と SimiScore の混合割合を決定するパラメータであり，$\alpha = 0$ のときは rVSM 同様となり過去のバグ情報を一切考慮しない．

2.2 言語横断バグ箇所検索

言語横断情報検索（Cross-Language Information Retrieval; CLIR）では，入力クエリと異なる言語で記述された文書を検索する [13]．これまでに様々な言語横断情報検索手法が提案されている．Hayes らは，翻訳ベースの手法として Google 翻訳によるイタリア語–英語間の文章翻訳を行い，追跡性の回復を行った [14]．Xu らは，事前に Web サイトのクローリングによってコーパスを作成しておき，入力となるテキストのクエリ化を行いコーパスベースの翻訳を行うことで，中国語の検索クエリによる英語情報サイ

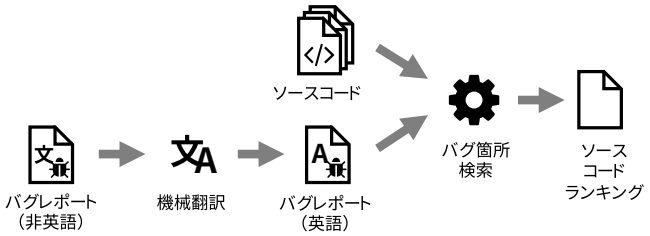

図 1 言語横断バグ箇所検索の概要

トでの質問検索を可能とした [15]．Tonoike らは，辞書に存在しない複合名詞に対し，コーパスを作成することにより辞書の拡張を行い，日本語–英語間での翻訳推定の性能向上を達成した [16]．

Xia らは，入力となるバグレポートに対して機械翻訳を適用して得られたテキストを，既存のバグ箇所検索手法 BugLocator[8] に入力することにより，英語以外の言語で記述されたバグレポートから言語横断バグ箇所検索を行う手法 CrosLocator を提案した [11]．CrosLocator に基づく言語横断バグ箇所検索の概要を**図 1** に示す．入力は，英語に基づくソースコードと，非英語で記述されたバグレポートである．まず，バグレポート中の自然言語記述に機械翻訳器を適用し，英語のバグレポートに変換する．次に，ソースコードと英語バグレポートを入力として情報検索に基づくバグ箇所検索を実行し，修正すべきと推定されたソースファイルのランキングを得る．なお，CrosLocator では複数の機械翻訳器を用いて得られたそれぞれの英語テキストを用いた検索結果を融合することによりスコアを求めているが，図にはこれは表現されていない．本論文では後述する理由のため，1 種類の機械翻訳器のみの利用を想定し，融合アプローチは用いない．

3. 日本語を対象とした CrosLocator の拡張

本論文では，CrosLocator を拡張し，日本語で記述されたバグレポートを対象とした言語横断バグ箇所検索を試みる．

我々がいくつかの日本語で開発された開発プロジェクトを分析したところ，ソースコードにも日本語記述が含まれていた．日本語記述は，ソースコード中のコメントや，文字列リテラルに含まれていた．このコメントや文字列リテラルに含まれる日本語は，後段の BugLocator にそのまま入力される．一方で，バグレポート中の日本語は英語に翻訳された形で BugLocator に入力されるため，バグレポートとソースコードの双方に類似の日本語記述が含まれていたとしても，それらは類似度に貢献しなかった．

これらに対して，いずれも英語に統一する形でそれらの類似性を求めた．（機械翻訳器を 1 種類に限定した）CrosLocator 法に対して，ソースコードにも日本語が含まれる場合の対処を追加した手法の概要を**図 2** に示す．ソースコードの構文解析を行い，文字列リテラルとコメント記述を抽出する．抽出された記述のうち，少なくとも 1 文字の日本語を含む記述を日本語記述とみなし，機械翻訳を適用して英語に翻訳する．翻訳して得た英語記述を，ソースコードの抽出元となった位置に埋め込む．ソースコード全体を機械翻訳しない理由は，日英機械翻訳がソースコード中の識別子やソースコードの構文を破壊する恐れがあるためである．なお，日本語リテラルを含むソースコードがコメントアウトされている場合，コメント全体を翻訳することは同様の不具合につながるため，コメントが日本語リテラルに相当する表現を含む場合，リテラルだけを抽出して翻訳するようにした．

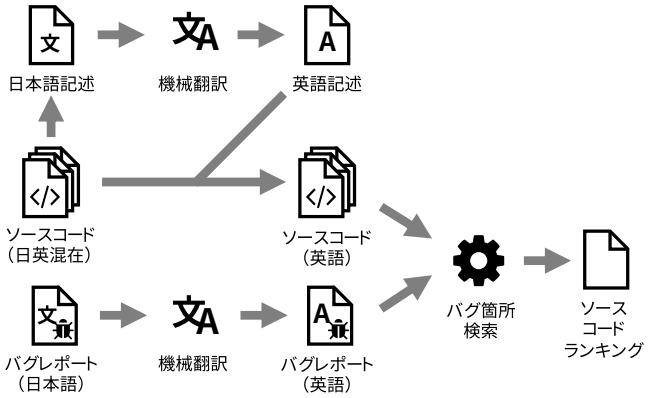

図 2 ソースコードに日本語が含まれる場合の対処

表 1 分析対象プロジェクト

プロジェクト	主要開発言語	ソースファイル数	バグレポート数
P_1	C#	929	8/186
P_2	Java	591	267/316

Bench4BL [17] に添付された BugLocator の実装を拡張する形で実装した．この実装は，入力ソースコードとして Java プログラムのみを受け付けていたが，他の言語も入力できるよう修正した．また，機密保持と実業務での利用の双方の観点から，著者らの一部の所属である日立製作所の社内で業務に利用している機械翻訳器を使用した．

4. 実験

本実験では，以下の 2 点のリサーチクエスチョン（RQ）に答える．

- RQ_1（CrosLocator アプローチの追試）言語横断バグ箇所検索は，日本語に基づく企業プロジェクトに対しても有効か?
- RQ_2（改善法の有効性）ソースコード中の日本語の翻訳はバグ箇所検索性能の改善に貢献するか?

4.1 データセット

本実験に用いたプロジェクトを**表 1** に示す．P_1 は企業で開発されたソフトウェアであり，主に C#で記述されている．P_2 も企業で開発されたソフトウェアであり，Java

で記述されている．

各プロジェクトで管理されていたバグレポートのうち，一部を用いた．P_1 では，後述するように開発者へのヒアリングを行ったため，対象とするバグレポートを少数に絞る必要があった．管理されていたバグレポート全 186 件を分類し，機能バグを扱っているもの（13 件），かつバグ修正が完了しているもの（9 件）のみを抽出した．さらに，C#ソースファイルが正解修正ファイルに含まれなかった 1 件を除外した 8 件を対象とした．P_2 では，バグレポート全 316 件のうち，Java ソースファイルが正解修正ファイルに含まれる 267 件を対象とした．

これらのプロジェクトでは，正解ファイルの準備方法が異なる．P_2 では，バグの解決記録に基づく自動的な特定を行っている．バグはイシュー管理システム，ソースコードは版管理システムで管理されており，バグ修正がどのコミットで行われたかがイシュー管理システムで記録されている．該当のコミットで修正されたソースファイルを特定することにより，バグ修正時に修正を要したソースファイルを特定できる．これらを関連付けることにより，バグレポートと，それに対応する正解のバグ箇所としてのソースファイルを決定できる．この方式では，バグレポートに対する正解をデータに基づき自動的に特定できるという利点がある一方，常に真の正解を特定するわけではないという欠点もある．誤った正解につながる要因には，イシュー管理システム上に記録されたバグと修正コミットの対応付けの誤りや漏れ，もつれた変更の構成による該当のバグ修正以外の意図の変更の同一コミットへの混在 [18]，などがある．また，バグ箇所検索においては，直接修正すべきバグ箇所を必ずしも特定しなくとも，開発者にとって該当バグを修正するために有用なファイルを推薦できれば有意義であり，修正されたファイルのみを正解とみなすと，こういった観点での評価が行えない，という問題もある．

一方で P_1 では，開発者へのヒアリングに基づきバグレポートと正解との対応付けを特定できた．対象とした 8 バグレポートに対して，以下の 2 種類のファイルを特定させた．

- O^d: 直接のバグ箇所を含むファイル
- O^i: 直接のバグ箇所を含むわけではないものの，間接的に関係がありバグの特定に貢献するファイル

前者だけでなく，後者も含めて特定できるかも確認するため，正解として直接のバグ箇所を含むファイルのみを対象とする実験設定（P_1^d）に加え，正解として間接的に関係のあるファイルも含めた実験設定（P_1^i）を用意した．後者においては，正解として $O = O^d \cup O^i$ を用いた．

なお，P_2 はソースコードに日本語記述が含まれないため，3 章で導入した拡張の効果は P_1 を対象に検証した．

4.2 評価指標

バグ箇所検索をはじめとする，ランキングを出力する手法の評価指標として用いられる以下を用いた．

- MAP（mean average precision）[19].
- MRR（mean reciprocal rank）[20]．MRR は，与えられたバグレポート集合における逆数順位の平均．ここで逆数順位（reciprocal rank; RR）とは，ランキング中で最も高順位の正解ファイルの順位の逆数を表す．
- 正規化減損累積利得（normalized discounted cumulative gain; nDCG）[21].
- N 位再現率（$\text{Recall}_{@N}$）．ランキングの上位 N 位以内のエントリを対象とした再現率．本実験では $N = 20$ を用いる．
- N 位成功率（$\text{Success}_{@N}$）．ランキングの上位 N 位以内に正解ファイルが少なくとも 1 つ存在する割合．本実験では $N = 10$ を用いる．

nDCG の計測においては，正解ファイルの重み付けが必要となる．P_1^d および P_2 では，すべての正解ファイルに重み 1 を与えた．P_1^i では，O^d に含まれる正解ファイルには重み 1 を，O^i に含まれる正解ファイルには重み 0.5 を与えた．

4.3 RQ_1：言語横断バグ箇所検索は，日本語に基づく企業プロジェクトに対しても有効か?

P_1^d，P_1^i，および P_2 に対して拡張 CrosLocator を適用した．結果を**表 2** に示す．

P_1^i では，$\text{Recall}_{@20}$ が 0.652 であり，重要なファイルの大半は 20 位以内で推薦できていることがわかる．また，$\text{Success}_{@10}$ が 0.500 であり，半数の結果では上位 10 位以内に少なくとも 1 つの正解ファイルを推薦できていることがわかる．P_1^d では，P_1^i に比べて MRR が上昇している．また，P_1^i に比べて $\text{Recall}_{@20}$ が 0.373 に低下している．このことから，間接的な正解ファイルはソースコードランキングの 20 位以下に多く出現していることが読み取れる．

全体を通して，rVSM に比べて，BugLocator を採用した場合の精度が向上している．つまり，BugLocator による類似バグを用いた性能向上が有効に機能していることがわかる．また，P_2 に対する結果も，全体的に P_1 に対するものに類似している．

Xia らは，適用対象のプロジェクトは異なるものの，CrosLocator の適用結果の精度が，英語を対象としている BugLocator を提案する論文で報告されている精度に劣っていることから，言語横断バグ箇所検索の難しさを報告している．本実験で得た値は，BugLocator における値には至らないものの，CrosLocator の適用結果として報告された値よりは改善されている．前述する他の指標での観点も踏まえれば，日本語を対象としたバグ箇所検索には一定の適用可能性があるものと考える．

表 2 適用結果

プロジェクト	コード翻訳	バグ箇所検索器	MAP	MRR	nDCG	$\text{Recall}_{@20}$	$\text{Success}_{@10}$
P_1^d	あり	BugLocator	0.157	0.276	0.443	0.373	0.625
	あり	rVSM	0.145	0.265	0.430	0.287	0.500
	なし	BugLocator	0.112	0.139	0.393	0.277	0.375
P_1^i	あり	BugLocator	0.164	0.259	0.388	0.652	0.500
	あり	rVSM	0.158	0.248	0.379	0.464	0.375
	なし	BugLocator	0.101	0.137	0.324	0.483	0.375
P_2	（なし）	BugLocator	0.174	0.270	0.272	0.347	0.461
	（なし）	rVSM	0.145	0.220	0.244	0.295	0.397

4.4 RQ_2：ソースコード中の日本語の翻訳はバグ箇所検索性能の改善に貢献するか?

P_1^d，P_1^i，および P_2 に対して拡張 CrosLocator を適用した．表 2 の「コード翻訳」列には，3 章で導入したソースコード翻訳の適用の可否が示されている．P_1 に BugLocator を適用した場合において，コード翻訳の適用の有無を比較したところ，いずれの評価指標においてもその値が大きく向上していた．このことは，ソースコード翻訳が有効に機能したことを示している．

5. おわりに

本論文では，日本語記述を対象とした言語横断バグ箇所検索の適用可能性について議論した．2 つの企業プロジェクトに対して適用したところ，良好な結果が得られた．今後の課題として，他のプロジェクトへの適用や，検索結果の開発者の利用事例の収集を進めたい．

参考文献

[1] Anvik, J., Hiew, L. and Murphy, G. C.: Coping with an open bug repository, *Proc. OOPSLA Workshop on Eclipse Technology eXchange*, pp. 35–39 (2005).
[2] Tassey, G.: The Economic Impacts of Inadequate Infrastructure for Software Testing, Technical report, National Institute of Standards and Technology (2002).
[3] 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).
[4] Cornelissen, B., Zaidman, A., Van Deursen, A., Moonen, L. and Koschke, R.: A systematic survey of program comprehension through dynamic analysis, *IEEE Transactions on Software Engineering*, Vol. 35, No. 5, pp. 684–702 (2009).
[5] Chen, K. and Rajlich, V.: Case study of feature location using dependence graph, *Proc. 8th International Workshop on Program Comprehension*, pp. 241–247 (2000).
[6] 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).
[7] Ratanotayanon, S., Choi, H. J. and Sim, S. E.: Using transitive changesets to support feature location, *Proc. 25th IEEE/ACM International Conference on Automated Software Engineering*, pp. 341–344 (2010).
[8] 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).
[9] 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*, pp. 345–355 (2013).
[10] 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*, pp. 181–190 (2014).
[11] Xia, X., Lo, D., Wang, X., Zhang, C. and Wang, X.: Cross-language bug localization, *Proc. 22nd International Conference on Program Comprehension*, pp. 275–278 (2014).
[12] 三井亮称，佐伯元司，林　晋平：多言語データセットを用いた言語横断バグ限局の性能評価，電子情報通信学会技術研究報告， Vol. 120, No. 343, pp. 31–36 (2021).
[13] Kishida, K.: Technical issues of cross-language information retrieval: A review, *Information Processing & Management*, Vol. 41, No. 3, pp. 433–455 (2005).
[14] Hayes, J. H., Sultanov, H., Kong, W.-K. and Li, W.: Software verification and validation research laboratory (SVVRL) of the University of Kentucky: traceability challenge 2011: language translation, *Proc. 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, pp. 50–53 (2011).
[15] Xu, B., Xing, Z., Xia, X., Lo, D. and Li, S.: Domain-specific cross-language relevant question retrieval, *Empirical Software Engineering*, Vol. 23, No. 2, pp. 1084–1122 (2018).
[16] Tonoike, M., Kida, M., Takagi, T., Sasaki, Y., Utsuro, T. and Sato, S.: A comparative study on compositional translation estimation using a domain/topic-specific corpus collected from the web, *Proc. 2nd International Workshop on Web as Corpus*, pp. 11–18 (2006).
[17] 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 International Symposium on Software Testing and Analysis*, pp. 61–72 (2018).
[18] Herzig, K. and Zeller, A.: The impact of tangled code changes, *Proc. 10th Working Conference on Mining Software Repositories*, pp. 121–130 (2013).
[19] Cormack, G. V. and Lynam, T. R.: Statistical precision of information retrieval evaluation, *Proc. 29th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 533–540 (2006).
[20] Craswell, N.: Mean reciprocal rank, *Encyclopedia of*

Database Systems, p. 1703 (2009).
[21] Järvelin, K. and Kekäläinen, J.: Cumulated Gain-based Evaluation of IR Techniques, *ACM Transactions on Information Systems*, Vol. 20, No. 4, pp. 422–446 (2002).