Vol.2021-SE-207 No.35
2021/3/2

質問記事と回答記事のコード差分に着目したJavaランタイムエラー向けQ&A記事提案手法

木村　悠介[1,a)]　赤崎　拓未[1,b)]　菊池　慎司[1,c)]

概要：ソフトウェア開発中にエラーが発生した際に、開発者は Stack Overflow 等の Q&A サイトでエラー原因を探ることがある。コンパイラーメッセージだけを手がかりにするよりも早く問題が解決できるので、今日では多くの開発者が活用している。しかし、原因となる関数名などが明確に分からない限りキーワード検索するのは難しく、Web ブラウザと開発環境間を何度も行き来する必要もあるという問題を抱えている。本稿では、Java の実行時エラーを対象として、スタックトレースから関連する Q&A 記事のペアを検索する手法を紹介する。手法は、単に類似コードを含む記事を検索するのではなく、質問記事と回答記事内のコードの変更箇所を抽出して、質問記事と回答記事を１つずつセットで提案する点が特徴的である。提案手法は、Word-Diff と AST 木編集距離を組み合わせて記事内から精度良く変更箇所を得る技術や、2AST 間の厳密な比較よりも緩い基準で 2 コード間の類似度を計測する技術で構成されている。OSS コードを用いた実験では、データセットの約 73%に対して 0.1-3 秒程度で役立つ Q&A 記事ペアを提案することができることが分かった。また、Eclipse プラグインを用いた実験では、キーワード検索と比較して検索時間を数分から数秒に短縮できることが示された。

1. はじめに

様々なシステムのデジタル化が急速に進められている昨今、ソフトウェア生産性向上は重要な課題である。日本国内では 2030 年までに IT 人材は 16-79 万人不足すると試算されており、プログラミング言語教育の導入などによる人材供給力強化だけでは間に合わない [1]。従って生産性向上においては、開発者ひとり一人の開発時間を短縮する観点が重要であると考えられる。

開発者の生産性に関する調査としては、[2], [3], [4] が挙げられる。[2] では、マウスイベントやクリックされたウィンドウの情報を用いて、開発者の時間の使い方を調べている。結果では、全体のうちの 43%が、バグを取り除くための解決策を検索するために使われていることが示されている。検索作業には、Q&A サイトや GitHub の他、API ドキュメントの閲覧時間が含まれている。[3] では、80 名程度の開発者の IDE（統合開発環境）に監視ツールを導入し、時間の使い方を調べている。結果では、開発者が IDE に滞在している時間は、全体の仕事時間のうちの 4 割程度であることが示された。また。IDE に滞在している時間の内、コードを実際に編集しているのは 29%程度であることも示されている。[4] は、IDE プラグインを用いて、開発者が集中してコーディングしている際の時間の使い方について調査している。結果では、実際にコーディングしている時間は高々 5%程度であり、全体の内の 70%を「既存コードの理解」に費やしていることが示されている。また、IDE とブラウザの間を何度も行き来すると理解にかかる時間も長くなっていくことが示唆されている。

以上のように、開発者が実際にコードを編集している時間は非常に短く、コードや API ドキュメント・Q&A サイト等を読んだり理解したりするための時間が長いことが分かる。従って、開発者ひとり一人の生産性を向上させるにはこれらの作業を効率化させることが効果的である。

本稿では、エラーの発生したコードを素早く修正することを目的に、Java ランタイムエラーを解消するのに有効な質問記事と回答記事を１つずつセット（以降、Q&A 記事ペア）で提案する手法について扱う。提案手法はバグを含むコードの構造情報を用いて Q&A 記事ペアを検索するので、単にキーワード検索するよりも高い精度が期待できる。特に提案手法は「役立つ回答記事があること」を重視しており、単に類似コードを検索するのではなく、Q&A がセットで得られる点が既発表論文 [5] と異なる。

[1] 株式会社富士通研究所
FUJITSU LABORATORIES LTD., Kawasaki, Kanagawa 211–8858, Japan
[a)] yusuke-kimura@fujitsu.com
[b)] akazaki.takumi@fujitsu.com
[c)] skikuchi@fujitsu.com

以上の手法を、クライアントとなる IDE プラグインと、Web-API を提供するサーバーの 2 つに分けて実装した。実装の評価には、エラーを発生させる OSS の Java プログラム 87 個を使用した [6]。提案手法はこれらの約 73%で役に立つ Q&A 記事ペアを 0.1-3 秒以内に提案出来ることが示された。更に既発表論文 [5] と比較し、20-30%程度多くのプログラムに対して役立つ Q&A 記事ペアが提案できた。また、ブラウザを用いて検索することと比較し、IDE プラグインを用いると高速に検索でき、検索時間を数分から数秒に短縮できることが初期的な実験で示された。

以降の章立ては以下の通りである。第 2 節では、Q&A サイトや最小木編集距離・APG(Abstracted Program Graph) などの背景知識や関連既存研究を紹介する。第 3 節では、提案手法の大まかなフローの他に質問記事内のコードの変更箇所を抽出する手法について説明する。第 4 節では、提案手法の評価結果や IDE プラグインによる時間短縮の効果について説明する。最後に第 5 節で本論文を総括し、今後の課題について述べる。

2. 背景・関連研究

2.1 背景

2.1.1 Q&A サイト

質問を投稿したり、その質問に回答したりできる Web サービスは質問投稿サイト（以降、Q&A サイト）と呼ばれ、今日広く利用されている。利用は無料であることが多く、質問者にとっては気軽に回答を募集でき、回答者にとっては自身の存在感をアピール出来るなどのメリットがある。

特に開発者のための Q&A サイトとしては “Stack Overflow” [7] が非常に有名であり、利用者も多い。これはソフトウェア開発に特化した Q&A サイトであり、以下のような特徴がある。

- 質問・回答記事内でプログラムに該当する部分を分かりやすく表示できる
- プログラミング言語名やフレームワーク名のタグを付けられ、検索に役立てられる
- 1 つの質問記事に対して複数のユーザーが回答記事を投稿できる
- 質問・回答記事に Like できたり、質問者が回答記事の 1 つに Approved マークを付けられたりするので、良い記事を作成する動機付けとなっている

本研究の実装では、英語版の Stack Overflow を用いる。提案される質問記事は “java”,“android”,“spring” というタグの何れかが含まれる。記事から自然文とコードは簡単に分離できるので、そのうちのコードだけを用いて第 3 節で述べる解析を行える。

2.1.2 最小木編集距離

提案手法では変更箇所の検出や 2 コード間類似度の算出に最小木編集距離を使うため、本節で予め説明する。なお、木編集距離について述べる前に、編集距離 (edit distance) について説明する。

編集距離とは、ある文字列を別の文字列に変化させるために必要な一連の操作である。操作には以下の 4 種類がある。

- Match（両文字列に同じ文字が存在し、手を加えない）
- Delete（ある文字を削除）
- Insert（ある文字を挿入）
- Update（ある文字を別の文字に変更）

例えば、“calc()” という文字列を “cat()” という文字列に変化させるには、例えば以下のような編集を行えば良い。

- “c” が Match
- “a” が Match
- “l” を “t” に Update
- “c” を Delete
- “(” が Match
- “)” が Match

編集の仕方には様々な方法があるが、重要なのは最もシンプルな編集方法である。そこで、4 種類の操作にスコアを与え、その合計スコアが最も小さくなるものを最小編集距離と呼ぶ。$(Match, Delete, Insert, Update) = (0, 1, 1, 1)$ とスコアを与えれば、上記の編集距離は 2 であり、最小である。なお、“Match” は編集ではないので、スコアを 0 とすることが一般的である。“Match” は編集距離に影響しない操作だが、実際にどのような編集が行われるか知りたい場合や、編集されない部分を知りたい場合に役立つ。

最小編集距離を得るためのアルゴリズムの説明は省略するが、動的計画法を用いて $O(m \times n)$（ただし m, n は 2 つの文字列長）のオーダーで済む。2 ファイル間比較の diff コマンドのベースとして広く用いられている。

木編集距離 (tree edit distance) とは、上述の編集距離を木構造に拡張したものである。同様に 4 つの種類を仮定し、それぞれの操作にスコアを与えることで最小木編集距離を得ることが出来る。編集距離とは異なり、最小木編集距離は多項式時間で得ることはできず、長年アルゴリズムが改善されてきた。2 つの木のノード数をそれぞれ m, n とすると、最初期の研究 [8] では $O(m^3 \times n^3)$ の時間がかかるが、本研究の実装に用いたアルゴリズム [9] では $O(n^2 m(1 + \log \frac{m}{n}))$ である。

提案手法の実装ではこの最小木編集距離の計算時間が支配的であるが、1 回の検索全体が高々 1 秒未満〜数秒程度あることには注意されたい。

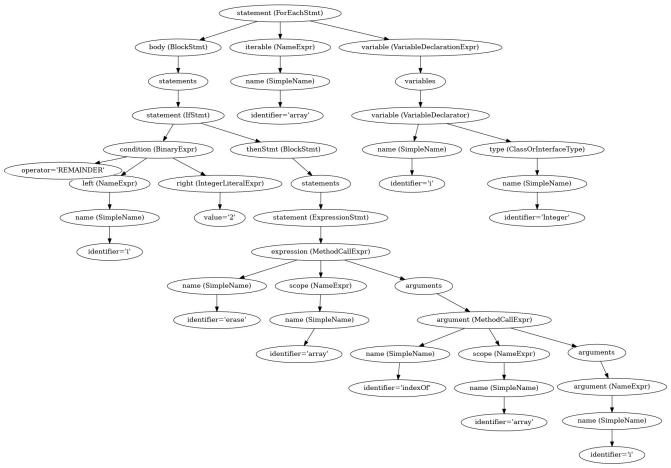
2.1.3 APG: Abstracted Program Graph

本節の内容は既発表論文 [5] と同様であるが、提案手法を理解する上で重要であるため、ポイントを説明する。

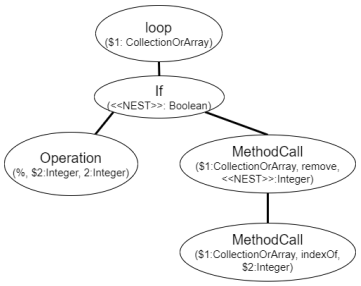
既存のコードクローン検出や Semantic Change Pattern 生成の研究の多くは、コードを AST や PDG に変換した上でサブグラフマイニングを行うというアプローチを取って

```
for(int i: array){
  if(i%2){
    array.remove(array.indexOf(i));
  }
}
```

(a) 元となるコード



(b) AST



(c) APG

図 1: AST と APG

いる。しかし Stack Overflow 内から類似のコードを見つける際には、コードの文法的な差異に過度に着目する必要はなく、より抽象化された形のデータ構造が必要である。

上述のような性質を満たすように設計されたデータ構造が APG である。APG は AST を変換する形で生成され、以下のようないくつかのステップを経て生成される。

(1) 関数呼び出し等の一連のサブグラフを１つのノードにまとめて分かりやすい構造にする
(2) 型情報のない変数に情報を追加する。（例えば、x=1 なら x は便宜的に Integer とする）
(3) 意味的に似たノードを変換する。（例えば、for, while, for-each 等を “loop” ノードとして扱う）
(4) ユーザー定義のクラス名を汎用名 “appClass1” などに変更する

コード片から AST と APG を生成した例を図 1 に示す。図中の AST は、JavaParser[10] で得られた木構造を図示したものである。APG は AST と比較して詳細な情報が省かれてシンプルになっていることが分かる。

２つの APG の類似度は以下の２つの指標を使用して算出できる。

- 構造上の類似度
- 変数の類似度

構造上の類似度は、２つの APG 間の最小木編集距離（第 2.1.2 節）をもとに算出し、“Match” と “Update” の操作数に応じて大きくなる指標である。変数の類似度は、２つの APG で使われている変数名が似ているほど大きくなる指標である。これらの２つの指標にそれぞれ定数を乗じて加えたものが、最終的な２コード間の類似度となる。なお最小木編集距離の計算に当たっては、“Delete” が質問コードを抜粋したことによるものかコード編集によるものか機械的に特定するのが難しい、という問題点がある。実装では “Delete” の距離を 0 としている。

2.2 関連研究

Q&A 記事を検索するという観点では、以下の２つが代表的である。[11] は、IDE で編集中のコード片に類似する Q&A 記事を検索する手法を提案している。この手法ではプログラムの構造情報は用いておらず、キーワード検索で類似記事内のコードを提案する。[12] は、コード片に類似する Q&A 記事内のコードを検索する手法を提案している。記事内の自然言語に注目し、それらをコードの説明と捉えて検索に活用している点が特徴的である。第 3 節で説明する提案手法は Q&A 記事内のコードの構造情報を活用しているので、[11], [12] とはアプローチが異なっている。

本研究に関連するテーマとして、コードクローン検出が挙げられる。コードクローンとは、ソフトウェア内で共通のコードを探すことである。同じコードが複数箇所にあると保守性が悪いという理由で、コードクローンはリファクタリングに活用される。[13] は、複数のコードを AST に変換し、共通のコードクローンを探す手法である。検出を効率化するために、ハッシュを用いている点が特徴的である。

更に関連するテーマとして、Semantic Change Pattern 検出が挙げられる。[14] は、PDG を活用してパターンを発見している。パターンは複数グラフ間の共通サブグラフと考えることが出来るので、少しずつノード数を増やしながら大きなパターンを得るアルゴリズムを採用している。同様の手法だが CDFG(Control Data Flow Graph) を活用しているものとして [15]、AST を用いているものとして [16] が挙げられる。

3. 提案手法

本稿で紹介する提案手法は既発表論文 [5] を発展させたものであるが、次の点で大きく異なる。

- 質問記事だけでなく、質問記事と回答記事のペアを提案している点（第 3.1 節の全体フローに対応）
- 質問記事内のコードのどの場所が変更されたかを詳細に解析している点（第 3.2 節のコード変更箇所抽出手法に対応）

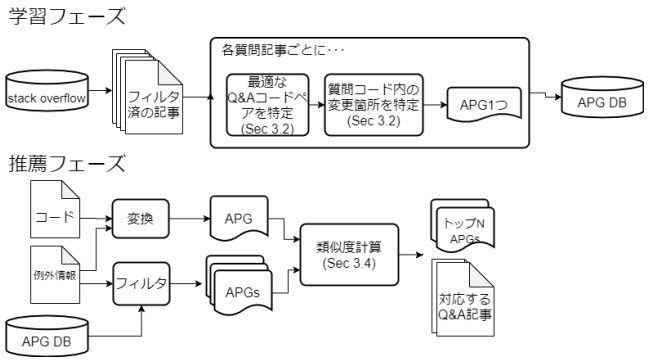


図 2: 全体フロー

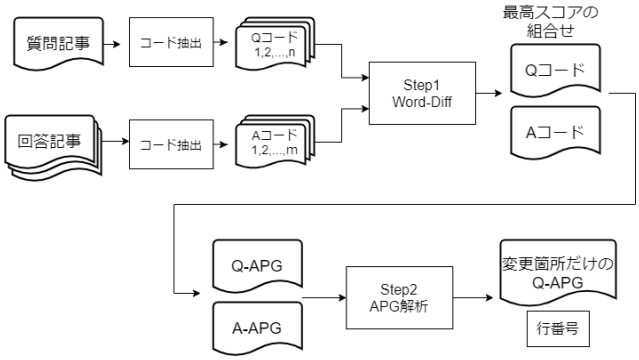


図 3: 質問記事内のコード変更箇所の抽出手法

本節では以上の 2 点を順に説明する。提案手法中で使われる APG については第 2.1.3 節を参照されたい。

3.1 全体の流れ

提案手法は、大きく学習フェーズと推薦フェーズに分けられる。学習フェーズは事前にまとめて行っておく作業であり、推薦フェーズは Q&A 記事ペアを提案するたびに行う作業である。図 2 も参照されたい。

3.1.1 学習フェーズ

学習フェーズでは、Stack Overflow 記事群を入力として、複数の APG を得てデータベースに格納する作業が行われる。Stack Overflow の記事群は解析用にまとめられたものが公開されている [17], [18]。

まずは Stack Overflow の全質問記事のうち、Java ランタイムエラーに関連する記事のみを抽出しそれ以外を削除した。以下のような条件をすべて満たすものが解析対象の質問記事である。

- “java”,“android”,“spring” など Java や著名なフレームワーク名に関するタグが付けられている
- タイトルに “Exception” という単語が含まれる
- 質問記事内に<pre><code></pre></code>HTML タグで囲われたコードがある
- １つ以上のコードを含む回答記事が存在する

残った質問記事には、それぞれに回答記事が存在する。質問記事内に M 個のコードが、対応する全回答記事内に N 個のコードが書かれているとすると、Q&A コードの組合せは $M \times N$ 個存在する。本手法では、精度良く変更すべき箇所を特定するために、第 3.2 節の手法を用いて、ただ１つの最適な Q&A コードペアを抽出し、それをもとに質問コード内の変更箇所を特定して１つの APG を得る。質問記事の数だけ APG が得られ、それらはデータベース内に格納されて推薦フェーズで活用されることになる。APG が格納される際には、対応する質問・回答記事 ID や記事から得られた Exception 名も記録される。

[5] では常に質問記事内のコード M 個分の APG が得られるようになっており、類似コードを含む記事が得られる可能性は高まるが、修正と無関係のコードまでマッチしやすい。著者が確認する限り、Stack Overflow 質問記事内の複数コードが同時に誤っていることは少なく、誤ったコードは１つであることが多い。従って、本提案手法では、ただ１つの最適な Q&A コードペアを探すこととした。

3.1.2 推薦フェーズ

推薦フェーズでは、以下の入力をもとに、データベースから類似の APG が検索されて「関連する Q&A 記事ペア（１つまたは複数）」が出力される。

- エラーのあるコード (`class Test{int calc(){...}}` など)
- Exception 名 (ArithmeticException など)
- エラーが発生した行番号

入力されたコードから APG が生成される際には、エラーが発生した行を含む関数だけに着目し、それ以外の関数部分を除いて APG にする。また、学習フェーズで作成したデータベースから与えられた Exception 名を元に該当する APG のみを取り出す。エラーが発生したコードに対応する１つの APG とデータベースから得られる複数の APG が得られるが、第 2.1.3 節の手法に基づいて 1 組ずつ類似度を計算し、最もスコアの良い APG に対応する Q&A 記事ペアを回答として出力する。実装では、上位 5 つの Q&A 記事ペアを出力している。

なお、各 APG には質問記事 ID と回答記事 ID が対応づけられている点に注意されたい。多くの既存研究では単に類似コードを探していたり、質問コードだけに着目していることが多く、提案された質問記事を読んで良い回答記事を探さなくてはならない。本提案手法は予め Q&A 記事ペアが提案されることになるので、より開発者に役立つ形で情報が提供できる。

3.2 質問記事内のコード変更箇所の抽出手法

Q&A 記事は自然言語とコードの組合せで質問と回答がやりとりされているので、機械的に修正内容を取り出すことは出来ない。本節では、質問コード内のどの部分（何行目）を修正すべきかを得るための技術を説明する。どの程度正確に変更箇所が得られるかによって最終的な Q&A 記

事ペアの提案精度が変化するので、本技術は提案手法の重要な要素の１つである。

Stack Overflow では、１つの質問記事に対して複数の回答記事が付くことが頻繁にある。また、質問者は自分のコードのどこに誤りがあるか分からないので、複数のコードを掲載することも多い。質問記事内のコード数を M、全回答記事内のコード数を N とすると、$M \times N$ 個の組合せが存在する。それらすべての情報から質問コード内の修正すべき箇所を機械的に得るのは困難である。

従って、以下のような２ステップに分けて修正すべき箇所を得ることを提案する。図 3 も参照されたい。

Step.1 $M \times N$ 個の組合せすべてに対して Word-Diff[19] を行い、変更されずに残った文字数を数える

Step.2 最も多くの文字が残った組合せだけに着目し、得られた２コードの APG を解析して修正箇所（行番号）と APG を得る

Step.1 では、修正されるべき質問コードは高々１つと仮定し、質問コードと最もよく似た回答コードを Word-Diff を使って探索している。Word-Diff はコードを文字列として取り扱うので、木構造である APG の編集距離を計算するよりも高速である。

Step.2 では、Step.1 で得られた質問コード・回答コード１つずつだけを使う。得られた２コードは APG に変換され、質問コードの APG を回答コードの APG に変換するための最小木編集距離を計算する。木編集距離で “Update”, “Insert”, “Delete” となった部分が編集箇所と考えることができるが、回答コードは修正すべき箇所の周辺だけを切り取って掲載されることが多いため、Delete が実際の編集箇所かどうか見極めるのは難しい。本技術では、Update や Insert となっている部分を修正と考え、対応する行番号を修正箇所とすることとした。質問コードの APG から、得られた行番号やその周辺だけ残したものを得ることができ、これがデータベースに保存される APG である。

4. 実験

4.1 実装

第 3 節で説明した内容を Java で実装した。サーバー部とクライアント部に分かれており、クライアント部は RESTful-API を通してサーバー上で役立つ記事を検索する。やりとりされる情報は以下の通りである。

入力１ エラーが発生した場所のソースコード１ファイル

入力２ エラーが発生した行番号

入力３ Exception 名 (java.lang.NullPointerException など)

出力１ Stack Overflow の Q&A 記事ペア（５つ程度）

出力２ 提出コード内で特に良くマッチした部分の行番号

検索を行うサーバー部では、Java プログラムの構文解析に JavaParser[10] を、木編集距離の算出に apted[9] を用いている。

4.2 評価１：提案精度

4.2.1 データセット・評価方法

データセットは [5] で用いられているオープンソースソフトウェア [6] を利用した。78 個の Java プログラムに、それぞれエラーが発生する行番号と Exception 名が与えられている。Exception の種類は 19 種類で、合計 87 個のエラーを用意した。詳細は表 1 も参照されたい。なお、プログラムを選出する際には、GitHub 上のパッチが修正する Exception の比率に似せた構成となるようにしている。各 Exception ごとのデータ数は表 1 の２列目に記されている。

本実験では、最も類似度スコアが高いと判断された Q&A 記事ペア５つについて、著者が「修正に役立つかどうか」という観点で評価した。１つでも役立つ記事ペアがあれば提案手法は正しい答えが得られたものとする。参考として、最高スコアの Q&A 記事ペアだけ考えた場合の結果も掲載する。提案手法の他に、同グループの既存研究である [5] とも比較した。こちらは質問記事しか提案しないので、質問記事と最も良い回答記事を選んで評価した。

なお、本実験の評価は [5] の評価者とは異なる人物が異なる基準で行っている。[5] の実験結果と本実験結果は単純に比較できない点に注意されたい。

4.2.2 実験結果

実験結果を表 1 に示す。実験結果は Exception 名ごとに示されており、収録数順に整列されている。“Proposed”, “[5]” がそれぞれ提案手法と既存手法の実験結果である。“Top.5” (もしくは “Top.1”) は類似度トップ 5(1) の Q&A 記事ペアが役立つかどうか判断し、役立つとされたものの数が示されている。たとえば提案手法は IllegalArgumentException では、８つのエラー中３つ（Top1 の場合は２つ）で役立つ Q&A 記事ペアを提案することが出来ている。“#Indexed” は学習フェーズでデータベースに収録した APG の数であり、多いほど実行時間が長くなると考えられる。“Time” は平均実行時間である。

提案手法は、[5] と比較して 2-3 割程度多くの役立つ Q&A 記事ペアを提案できており、全体の 73%程度に対して役立つ Q&A 記事ペアが提案できている。実行時間は概ね 1 秒未満であり、最も APG 数が多く時間のかかる NullPointerException でも 2.4 秒程度で提案が終了する。

既存手法よりも良い結果が得られているのは、第 3 節で説明されている通り、より回答記事に着目しているからと考えられる。また不要な記事を除去しているためインデックスされる APG 数は減少しているが、結果の精度を落とさずに実行時間が大幅に短縮出来ている。一方で、RejectedExecutionException や CMMException では収録 APG 数が極端に少なくなってしまう問題も見受けられ、[5] の方が良い記事を提案できていた。

表 1: OSS データセットを用いた評価

Name	# Errors	Proposed				[5]			
		Top.5	Top.1	# Indexed	Time	Top.5	Top.1	# Indexed	Time
ClassCast	8	8	5	1090	0.23	5	5	3001	4.9
ConcurrentModification	8	8	8	512	0.28	8	8	1054	1.5
IllegalArgument	8	3	2	826	0.29	2	2	2173	4
IllegalState	8	4	2	930	0.35	3	3	2364	4.7
IndexOutOfBounds	8	8	3	512	0.37	3	2	3115	6.6
NullPointer	8	8	5	5714	2.4	6	2	18305	59
Arithmetic	4	4	3	32	0.05	4	3	101	0.18
NoSuchElement	4	4	3	462	0.18	4	0	717	1.5
RejectedExecution	4	1	1	11	0.04	2	0	47	0.11
Security	4	1	0	64	0.26	0	0	361	0.99
UnsupportedOperation	4	4	3	177	0.05	4	0	378	0.83
EmptyStack	2	2	2	7	0.11	2	1	29	0.23
NegativeArraySize	2	1	0	14	0.31	1	1	34	0.35
ArrayStore	1	1	1	67	0.22	1	1	78	0.37
BufferOverflow	1	0	0	6	0.03	0	0	45	0.29
BufferUnderflow	1	0	0	7	0.08	0	0	24	0.23
CMM	1	0	0	1	0.09	1	1	3	0.17
IllegalMonitorState	1	0	0	76	0.09	0	0	165	0.49
MissingResource	1	0	0	15	0.07	0	0	42	0.31
TOTAL	78	57	38	10523	0.45	46	29	32036	8.5

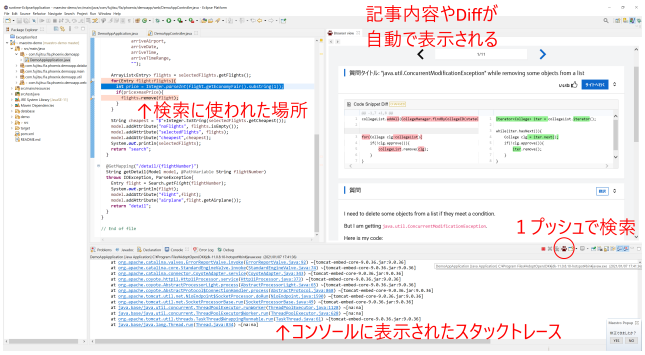


図 4: Eclipse プラグイン

4.3 評価 2：IDE プラグインの効果

本節の内容は IDE プラグインの初期的な評価であり、定量的な指標は設定できていない。参考として掲載する。

4.3.1 データセット・IDE プラグインについて・評価方法

データセットとして、表 2 に示すようなバグを混入させたデモ用 Web アプリを用意した。アプリは SpringBoot フレームワークを用いて実装されている。本実験の目的は、提案手法を活用する IDE プラグインと手動の検索とでの検索時間を比較することである。

プラグインは、Java プログラム実行時に発生したエラーが出力したスタックトレースをコンソールから読み取り、エラー解消に役立つ Q&A 記事ペアの検索を 1 クリックで行うことが出来る。スタックには複数のフレームがあるが、Java や Spring などのライブラリを除いて、プロジェクト固有のコードで一番先頭に表示されたフレームで Exception が発生したものとして検索する。動作の様子は図 4 も参考にされたい。

手動で検索を行う際は、表 2 最終列のキーワードを用いて Google 検索した。1 つ目のバグでは発生箇所に for 文しか存在せず疑わしい関数がなかったため、Exception 名だけで検索を行った。2 つ目のバグでは、substring という関数名もクエリに追加した。

4.3.2 実験結果

プラグインを使った場合には、1 秒程度でバグの原因と解決策・コードを説明した Q&A 記事ペアを得ることが出来た。No.1 のバグでは Iterator を使ってコレクション内をループするのが一般的な方法であることが分かる。No.2 のバグでは、文字列への範囲外アクセスが発生してしまっているので、ロジックを修正する必要がある。これらの「気付き」が、Google 検索ではどの程度の時間をかけて得られるか、著者が実際に検索を行って調査した。なお、Google 検索は同じ検索クエリでも異なる結果を返すことがあるため、再現性が低い点に注意されたい。

表 2 中の No.1 の ConcurrentModificationException を Google 検索した場合、最初の 3 つの結果は以下の通りである。「ループ内でコレクションを操作している」という構造的な特徴そのものはキーワードにしにくく、検索に数分以上かかった。

- `stackoverflow.com/q/39927223` 質問コード中で It-

表 2: Web アプリに混入したバグとエラー

No.	Exception 名	バグの説明	検索クエリ
1	ConcurrentModification	ループ内でループ対象のアイテムを削除	ConcurrentModificationException site:stackoverflow.com
2	IndexOutOfBounds	substring で文字列外を指定	IndexOutOfBoundsException substring site:stackoverflow.com

erator が既に使われている記事。異なる質問である

- `stackoverflow.com/q/27262602` subList 関数を使ったことで、Exception の原因究明が難しくなってしまった場合の記事。初学者には難しい内容の可能性がある
- `stackoverflow.com/q/16852469` Iterator を使うべき、ということについて丁寧に解説された記事。これが最初に得られると有用であった。ただし回答記事は修正コード例が付いていない

No.2 の IndexOutOfBoundsException を Google 検索した場合、最初の 3 つの結果は以下の通りである。substring という関数名が得られているため、所望の記事にすぐに辿り着くことが出来ている。

- `stackoverflow.com/q/42778401` substring 関数が Exception を返す場合のシチュエーションについて丁寧に説明がある記事
- `stackoverflow.com/q/42778414` substring の引数に負数を与えた場合の挙動について説明している記事
- `stackoverflow.com/q/48241594` substring 関数によって Exception が起きている具体的なコードの修正方法を議論した記事

以上のように、バグの特性によって、キーワード検索は検索時間に幅があることが示された。提案手法と比較して、数分程度の時間差が生まれる可能性があることが示された。

ところで、[4] では IDE とブラウザの間を何度も行き来するとコード理解にかかる時間も長くなっていくことが示唆されている。IDE プラグインを用いれば IDE 上で作業を完結することが出来るため、開発者のバグ理解の時間短縮に通じる可能性がある。定量的な調査については、今後の課題としたい。

5. まとめと今後の課題

本稿では、Java の実行時エラーを対象として、関連 Q&A 記事ペアを提案する手法を紹介した。提案手法は単に類似コードを検索するのではなく、質問記事と回答記事のペアを出力するように作られており、既存研究よりも精度良く提案することが期待される。実装を用いた評価では、エラーを発生させるオープンソースのコードに対して役立つ Q&A 記事ペアを 73%の精度で提案でき、既発表論文 [5] よりも 20-30%多くのエラーに対して役立つ Q&A 記事ペアを出力することも示された。IDE プラグインを用いた評価では、バグ理解のための時間が大幅に短縮する可能性が示された。

今後の課題として、Semantic Change Pattern のアイデアを取り入れることが挙げられる。実験では、NullPointerException など Stack Overflow 上で質問数の多い記事は、検索時間が長くなってしまっている傾向が見られた。また、そのような質問記事の中には同じ内容なのに別記事として投稿されているものが非常に多い。似た記事同士をクラスタリングすることによって検索時間を高速化し、1 つ目の提案が誤っていた際に次も似た記事を提案することを避けることができる。

もう 1 つの課題はツールの評価である。Eclipse プラグインは初期的な実験をしただけで、実際の開発者にとってどれだけ役立つかが評価できていない。また、企業の開発現場などではオープンソースとは異なるエラーが多く発生する傾向があることも考えられるため、提案精度が変化する可能性がある。より規模の大きな実験を通してツールを評価する予定である。

参考文献

[1] みずほ情報総研：IT 人材需給に関する調査，`www.meti.go.jp/policy/it_policy/jinzai/` (2019).
[2] Xia, X., Bao, L., Lo, D., Xing, Z., Hassan, A. E. and Li, S.: Measuring Program Comprehension: A Large-Scale Field Study with Professionals, *IEEE Transactions on Software Engineering*, Vol. 44, No. 10, pp. 951–976 (2018).
[3] Amann, S., Proksch, S., Nadi, S. and Mezini, M.: A Study of Visual Studio Usage in Practice, *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1, pp. 124–134 (2016).
[4] Minelli, R., Mocci, A. and Lanza, M.: I Know What You Did Last Summer - An Investigation of How Developers Spend Their Time, *2015 IEEE 23rd International Conference on Program Comprehension*, pp. 25–35 (2015).
[5] Mahajan, S., Abolhassani, N. and Prasad, M. R.: *Recommending Stack Overflow Posts for Fixing Runtime Exceptions Using Failure Scenario Matching*, p. 1052–1064, Association for Computing Machinery (2020).
[6] Sonal, M., Negarsadat, A. and Mukul, P.: Artifacts for MAESTRO: A Tool for Recommending Stack Overflow Posts for Fixing Runtime Exceptions, `https://doi.org/10.6084/m9.figshare.11948619.v1` (2020).
[7] Stack Exchange Inc.: Stack Overflow, `https://stackoverflow.com/`.
[8] Tai, K.-C.: The Tree-to-Tree Correction Problem, *J. ACM*, Vol. 26, No. 3, p. 422–433 (1979).
[9] Pawlik, M. and Augsten, N.: Efficient Computation of

the Tree Edit Distance, *ACM Trans. Database Syst.*, Vol. 40, No. 1 (2015).

[10] javaprser's Github Repository: javaparser, `https://github.com/javaparser/javaparser`.

[11] Ponzanelli, L., Bavota, G., Di Penta, M., Oliveto, R. and Lanza, M.: Mining StackOverflow to Turn the IDE into a Self-Confident Programming Prompter, *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, New York, NY, USA, Association for Computing Machinery, p. 102–111 (2014).

[12] Kim, K., Kim, D., Bissyandé, T. F., Choi, E., Li, L., Klein, J. and Traon, Y. L.: FaCoY: A Code-to-Code Search Engine, *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, New York, NY, USA, Association for Computing Machinery, p. 946–957 (2018).

[13] Jiang, L., Misherghi, G., Su, Z. and Glondu, S.: DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones, ICSE '07, USA, IEEE Computer Society, p. 96–105 (2007).

[14] Nguyen, H. A., Nguyen, T. N., Dig, D., Nguyen, S., Tran, H. and Hilton, M.: Graph-Based Mining of in-the-Wild, Fine-Grained, Semantic Code Change Patterns, *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, IEEE Press, p. 819–830 (2019).

[15] Amann, S., Nguyen, H. A., Nadi, S., Nguyen, T. N. and Mezini, M.: Investigating next Steps in Static API-Misuse Detection, *Proceedings of the 16th International Conference on Mining Software Repositories*, MSR '19, IEEE Press, p. 265–275 (2019).

[16] Dotzler, G., Kamp, M., Kreutzer, P. and Philippsen, M.: More Accurate Recommendations for Method-Level Changes, *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, New York, NY, USA, Association for Computing Machinery, p. 798–808 (2017).

[17] Stack Exchange Inc.: Stack Exchange Data Dump, `https://archive.org/details/stackexchange` (2020).

[18] Baltes, S., Dumani, L., Treude, C. and Diehl, S.: SOTorrent: reconstructing and analyzing the evolution of stack overflow posts, *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018* (Zaidman, A., Kamei, Y. and Hill, E., eds.), ACM, pp. 319–330 (2018).

[19] GNU: Wdiff, `https://www.gnu.org/software/wdiff/`.