R-2

チャットボットを利用した Java プログラムのデバッグ支援手法

橋浦研究室　　118I206 仲山 冬野 118I256 村田 匠

1. はじめに

ソフトウェアの品質を向上させる手法の一つとしてデバッグが挙げられる．本手法ではプログラム実行時の制御と流れを追跡することによって，欠陥を発見するデバッグ手法[1]に着目する．

この手法では，プログラムのトレースを見ることによって，不正な実行結果に至るデータと制御の流れを特定することができる．このためグラフィックスのあるプログラムやサイズの大きいプログラムなどではトレースが大規模になってしまい，ただトレースを検索するだけでも多くの時間がかかってしまうという問題がある．

また，大規模でないプログラムであっても，実行経路の誤りを手動で追う事はユーザにとって負担が大きい[2]. こういったソフトウェア開発に関連する問題をサポートするボットは，現代のソフトウェアエンジニアリングや開発に対処するための有望なアプローチとされている[3].

1.1. トレースとは

トレースとはユーザープログラムやオペレーティングシステムの動作の検証のために取得する，実行していた関数やメソッドなどの履歴のことである．

```
root
  thread-0
    test.main(args:<String[]:1>)
        System.in => <BufferedInputStream:1>
        new Scanner(<BufferedInputStream:1>) => <Scanner:1>
        System.out => <PrintStream:1>
        <PrintStream:1>.println("年齢を入力してください")
        scanner:<Scanner:1>.next() => "21"
        Integer.parseInt(old:"21") => 21
        if (age:21 >= 20)
        System.out => <PrintStream:1>
        <PrintStream:1>.println("成人です")
        return
```

図 1. トレースの例

これは，ソフトウェア開発者がプログラムを修正するデバッグ工程やフォールトローカライゼーションに用いられ，コード上のどのような実行経路を経て問題が発生したのかを知ることができ，不具合に関連する箇所の特定を容易に，デバッグにかかる時間を節約する[4].

1.2. Traceglasses[5]とは

Traceglasses[5]は Java プログラムのトレースを記録するデバッガである．本研究で使用するトレースは Traceglasses で記録したものを用いる．トレースを特定するためにタイムスタンプを利用する．しかし Traceglasses にはタイムスタンプを記録する機能がないため，タイムスタンプを記録できるようにカスタマイズして使用する．

2. 研究目的

本研究の目的はトレースの検索をより効率的かつ容易に行うことができる手法の実現，またはそれに寄与することである

手法の有効性を確認するため,以下のリサーチクエスチョン（以降 RQ という）を設定する．

RQ1. トレースから関連するソースコードを検索する際の負担軽減に寄与しているか

RQ2. トレース行とソースコードを検索する効率が上がっているか

無関係なソースコードを参照した数を集計することによって，バグ修正の迂回を減らすことができるかを確認できる．また検索時間の集計をすることによって開発におけるデバッグ時間の削減に貢献できるかを確認できる．

3. 関連研究

トレースに基づくデバッグ手法については，これまでも様々なものが提案されている．例えば Schulz と Bockisch [5]は Eclipse Java Development Tools (JDT)を拡張し，逆戻りデバッグが可能な Redshell を開発している．しかしながら Redshell を用いたデバッグでは，対象となるプログラムの実行に先立ってブレークポイントを設定しておく必要あるため，開発者はあらかじめプログラムの内部構造を把握しておく必要がある．

4. 提案手法

従来の Traceglasses の場合，プログラム操作中にその操作時点までのトレースを検索，その後またプログラム操作を継続しトレース検索をすることはできなかった．

そこで，本手法では検索したい箇所までのトレースのみで検索が可能となるような変更を加える．また検索後もプログラムを実行できるようなものとする．

トレースを区切る方法としてタイムスタンプ

を利用する．またプログラム実行中のタイムスタンプを取得する方法としてチャットを採用する．

プログラム実行中に検索したい箇所をチャットボットに送信し，チャットボットがメッセージとその送信時間を取得することによって，プログラムの実行開始から送信時間までのトレースのみで検索をすることできる．実行途中までのトレースから検索をすることによって，従来のツールよりも少ないトレースから検索をすることが可能となる．

また，チャットボットがいるチャットルームに検索メッセージを送信することによって他の開発者に，デバッグしているプログラムの何を検索しているのかの共有ができる．課題や進捗の共有などが円滑になりチーム開発でのコミュニケーションに貢献することが期待できる．

5. ツールの実装

実装は Traceglasses のトレース検索ボックスにボットに送信されたメッセージの取得とタイムスタンプを参照した検索結果を表示する機能を追加する．チャットに送信されたワードをボットのボタンを押すことによって送信時間までのトレースで検索をすることができる．実装は Traceglasses の Search タブにボットの機能を追加する．チャットに送信されたメッセージをボットのボタンを押すことによって，送信時間までのトレースをメッセージで検索をすることができる．

6. 実験

ツールの有効性を確認するため，Traceglasses と本手法によって拡張した Traceglasses を使用し比較実験を行う．日本工業大学先進工学部情報メディア工学科に所属する学生とプログラミング学習者の計 7 名を対象とする．

実験に使用する問題プログラムは 3 種類用意し，意図的に挿入したバグに関係のないソースコードを参照した数(以降，誤ファイル参照数という)とトレース検索時の検索時間を集計する．

RQ1 についてツールを用いてデバッグをした際に，関係のないソースコードを検索・参照した数の計測を行う．RQ2 についてはトレース検索した際の検索時間を集計することによって確認する．

実験の流れについては以下の通りである．

1. 実験の説明と使用するプログラムの説明
2. トレースを取りつつプログラムを実行
3. バグに関係するトレースの検索
4. 検索が完了したらグループのツール使用の有り無しで交代し再度 1〜3 を実行
5. 検索を終えたら実験を終了

実験に使用する問題はコマンドライン上で動作する 3 種類のボードゲームプログラムである．これらの問題に意図的に混入したバグは，実行時出力から発見できるようなものとした．

7. 結果と考察

実験結果について表 1,2 に示す．また表中語句の取り扱いに関しては以下の通りとした．

1) 誤ファイル参照数 バグに該当するソースコードを検索する際に該当するソースコードだと誤って参照した数．
2) 検索時間[s] トレースの検索時間，トレースの検索開始から計測を開始し，バグに該当するトレースの発見かギブアップ, 10 分の時間制限を計測終了判定とした．
3) 発見 実際にトレースからバグに該当するソースコードを発見できたかどうかについて○×として表に示した．
4) -(ハイフン) バグを発見できずファイルを参照することがなかった事例．
5) ギブアップ バグを発見できたがトレースからソースコードを発見することを被験者が不可能と判断し実験を終了した事例．

表 1. 拡張ありの実験結果

#	実験種類	誤ファイル参照数	検索時間[s]	発見
1	ブラックジャック	2	156	○
2	マインスイーパー	1	390	○
3	2048	10	304	○
4	マインスイーパー	1	147	○
5	2048	2	509	○
6	ブラックジャック	-	-	×
7	ブラックジャック	-	-	×
8	2048	19	600	×

表 2. 拡張なしの実験結果

#	実験種類	誤ファイル参照数	検索時間[s]	発見
1	ブラックジャック	14	540	○
2	マインスイーパー	6	506	○
3	2048	ギブアップ	150	×
4	ブラックジャック	3	600	×
5	マインスイーパー	4	593	×
6	マインスイーパー	6	480	×
7	ブラックジャック	10	480	×
8	2048	4	600	○

表 1,2 にある実験結果の誤ファイル参照数とトレース検索時間に対してそれぞれ箱ひげ図を作成しマン・ホイットニーの U 検定を行なった．

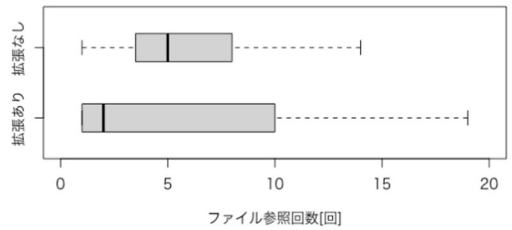

図 2．誤ファイル参照数の比較

本検定では，帰無仮説 H_0 を「拡張ありと拡張なしに差はない」，対立仮説 H_1 を「拡張ありと拡張なしに差がある」，有意水準αは 0.05 とした．表 2 より $p<0.05$ を満たさないため，H_0 を採用する．

表 3．誤ファイル参照数 検定

統計量 U	5.5
p 値	0.051

検定の結果，拡張ありと拡張なしで誤ったファイルの参照回数に差は見られなかった．

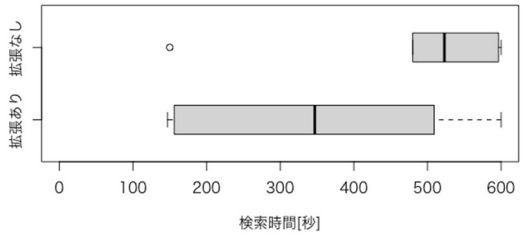

図 3．トレース検索時間の比較

本検定では，帰無仮説 H_0 を「拡張ありと拡張なしに差はない」，対立仮説 H_1 を「拡張ありと拡張なしに差がある」，有意水準αは 0.05 とした．表 2 より $p<0.05$ を満たさないため，H_0 を棄却し H_1 を採用する．

表 4．トレース検索時間 検定

統計量 U	3.0
p 値	0.018

検定の結果，検索時間には有意差があるという結果になった．5 人の被験者はバグが見つからずにギブアップしていたため，拡張なしの実験記録のなかでは検索時間が短い傾向にある．3 人の被験者はバグを発見するまでに 500 秒以上かかっており，拡張ありでバグを発見した被験者よりも時間がかかっていることがわかる．

拡張ありのツールで実験を行った被験者は，検索候補の後方からトレースとソースコードを確認し比較的早い時間でバグを発見しており，期待通りにツールを使用していた．拡張なしのツールで実験を行った被験者は，手当たり次第にトレースを確認していたが，時間がたつにつれて諦めてトレースを眺めている時間が増えていた．

前述した結果を元に，RQ について回答する．RQ1 について，実験より本ツールではトレースから関連するソースコードを検索する際の負担軽減に寄与していないことを確認した．また，RQ2 について，実験より本ツールではトレース行とソースコードを検索する効率が上がっていることを確認した．したがって，ツールによってトレース行とソースコードを検索する効率を上げることは可能である．

8. まとめと今後の課題

本研究ではトレースを用いるデバッグにおいて，チャットボットを利用し効率的にトレースを検索する手法を提案した．実験によって本手法が該当トレースに関連するソースコードの検索を助けることを明らかにした．今後は提案手法の改良および，チャットボットを利用することによるデバッグ手法への効果など実践的な環境での評価を実施していきたい．

文　献

[1] 松村俊徳,石尾隆,鹿島悠, “REMViewer :複数回実行された Java メソッドの実行経路可視化ツール,” コンピュータソフトウェア,Vol.32, No.3, pp. 137-148, Sep. 2015.

[2] 松下 圭吾, 松本 真樹, 大野 和彦, 佐々木 敬泰, 近藤 利夫, 中島 浩, “実行トレースの比較を用いたデバッグ手法の提案及び評価, ”先進的計算基盤システムシンポジウム論文集, pp. 152-159, May 2011.

[3] L. Erlenhov, F. Gomes de Oliveira Neto, R. Scandariato, and P. Leitner, “Current and future bots in software development,” Proc. of the 1st International Workshop on Bots in Software Engineering (BotSE '19), pp. 7-11, May 2019.

[4] S. Pearson et al., “Evaluating and Improving Fault Localization,” Proc. of IEEE/ACM 39th International Conference on Software Engineering (ICSE 2017), pp.609-620, May 2017, doi: 10.1109/ICSE.2017.62.

[5] 櫻井 孝平, 増原 英彦, 古宮 誠一, “Traceglasses : 欠陥の効率良い発見手法を実現するトレースに基づくデバッガ,” 情報処理学会論文誌プログラミング, 3 巻, 3 号, pp. 1-17, Jun. 2010.

[6] Stefan Schulz, Christoph Bockisch, “RedShell: OnlineBack-In-Time Debugging,” Proc. of Companion to the first International Conference on the Art, Science and Engineering of Programming (Programming ‘17), pp. 1-2, Apr. 2017.