7N-04

自動バグ修正技術の企業内ソースコードへの適用に向けて

池田翔† 中野大扉† 亀井靖高† 佐藤亮介† 鵜林尚靖† 吉武浩‡ 矢川博文‡
†九州大学 ‡富士通九州ネットワークテクノロジーズ株式会社

1 はじめに

ソフトウェア開発においてバグの修正は避けて通れないものであり，時間のかかる作業である．自動バグ修正に関するサーベイ論文 [1] では「デバッグに関する作業はいくつかあるが，その中でも大きな割合を占めるのは広範囲に及ぶマニュアル作業を要するバグの特定とバグを取り除く作業である」と述べられている．

バグを修正する作業を自動化することによって生産性の向上が期待できるため，自動バグ修正の研究は盛んに行われている．自動バグ修正ではデバッグ作業の大部分を占めるバグの特定とバグを取り除く作業の両方について研究が進められている．例えば，Prophet[2] はバグを取り除く作業を自動化する手法で，実際のソフトウェアに対する自動バグ修正に成功している．

自動バグ修正の手法には様々な方法が存在するが，そのほとんどはオープンソースソフトウェア（OSS）を修正対象とし，企業内ソースコードを対象としているものは少ない．デバッグに関する作業は企業内開発も例外ではなく，企業内ソースコードに対して自動バグ修正が適用できるか試みることは有用であると考えられる．

本研究では富士通九州ネットワークテクノロジーズ株式会社の企業内ソースコードに対して自動バグ修正を行って自動バグ修正を適用する上での OSS との差異を明らかにすることを目標とする．本稿ではそのアプローチを述べる．

2 背景

2.1 自動バグ修正

自動バグ修正手法の多くは，バグを修正するために実際にバグを検出するテストを必要とする．それぞれの手法で生成したパッチが上記のテストでも正常に動作したかによって，バグ修正ができたかどうかを判断する．また，テストはバグの場所を特定するのにも使用される．

Applying Automatic Bug Repair Techniques to Enterprise Source Code Files

Sho Ikeda† Daito Nakano† Yasutaka Kamei† Ryosuke Sato†
Naoyasu Ubayashi† Hiroshi Yoshitake‡ Hirofumi Yagawa‡
†Kyushu University ‡FUJITSU KYUSHU NETWORK TECHNOLOGIES LIMITED
{ikeda, nakano} @posl.ait.kyushu-u.ac.jp
{kamei, sato, ubayashi} @ait.kyushu-u.ac.jp
{yoshitake.hiro, yagawa.hirofumi}@jp.fujitsu.com

本研究で用いる企業内ソースコードは C 言語であるため，サーベイ論文内で紹介されている既存研究の中でも C 言語に適用可能でかつツールが公開されている先行研究から，Prophet を企業内ソースコードに適用する．

Long らが提案した Prophet は OSS から得られたバグ修正を行ったパッチを元に学習を行い，テンプレートを用いてパッチを生成する手法である．学習に用いた OSS は言語，サーバー等様々な分野から 8 つの OSS を選択し，普遍的なバグ修正の特徴を抽出することを目指している．GenProg[3] などの既存手法と比較し OSS を対象により多くのバグを修正し，開発者に受け入れられるパッチを生成することに成功している．Prophet を OSS に適用した場合は開発者にバグの修正結果が受け入れられたが，企業内のソースコードに適用した場合も同様に開発者に受け入れられるかを調査するために Prophet を選択した．

2.2 本研究の目的

先行研究で紹介したような自動バグ修正技術を企業内ソースコードに対して適用することで，自動バグ修正を企業内ソースコードに適用する上での課題を明らかにすることができる．既存研究では，自動バグ修正技術を OSS に適用することで評価を行っているが，実際の企業内ソースコードに適用しているものは少ない．本研究では，自動バグ修正技術が企業内ソースコードに適用可能か，及び修正結果が開発者に受け入れられるかを調査する．

3 アプローチ

本研究では自動バグ修正が企業内ソースコードに対しても適用可能かを調査するために 2 つの研究課題（RQ: Research Question）を挙げる．

RQ1 実際に発生したバグを自動バグ修正で修正できるか

RQ2 開発者に受け入れられるパッチを生成できるか

データセット ソースコードはバージョン管理システムによって管理され，総コミット数は 759，最終コミットでのソースコードの行数は約 16 万行である．コミットメッセージに「バグ・修正」等のキーワードが含まれるものは 759 コミット中 130 コミットであり，目視調査でバグ修正を行なったコミットと判断できるものは 21 コミットであった．

```
    void main(void){
        int config_val = read_config();
        //設定ファイルの値の大きさでメモリ確保
        memory_allocation(config_val);
        ...
        print_val();
    }

    void print_val(void){
-       for (i = 0; i < 256; i++) {
+       for (i = 0; i < config_val; i++) {
            printf("%d", data[i].val);
        }
    }
```

図 1: プログラムの概要と開発者によるバグ修正

RQ1 実際に発生したバグを自動バグ修正で修正できるか

先行研究では OSS に対して自動バグ修正が有効であることを示しているが，企業内ソースコードに対しても有効であるかは示されていない．

RQ1 では実際に企業内ソースコードで発生したバグをコミット履歴を元に手動で特定し，Prophet を用いて自動バグ修正を試みる．Prophet は修正の際にバグを検出するテストと検出しないテストを必要とするため，各バグに対して著者らが手動でテストを作成する．

RQ2 開発者に受け入れられるパッチを生成できるか

サーベイ論文ではバグの修正だけでなく開発者に受け入れられるパッチを生成するべきだと述べられている．

RQ2 では RQ1 でパッチを生成することに成功した場合，開発者が実際に行った修正と生成したパッチを比較する．生成したパッチに対して実際の機能や非機能要件を満たしているか調査し，企業内アンケート等を実施することで開発者に受け入れられるパッチを生成できたか評価する．

4 初期調査結果

RQ1，RQ2 の初期調査として実際のバグ修正コミット 21 件のうち 1 件について Prophet を適用した結果を記す．

初期調査に用いたプログラムの概要を図 1 に示す．このプログラムでは設定ファイルの値を読み込み，その値によって`memory_allocation()` でメモリを確保している．設定ファイルが 256 未満の場合，`print_val()` 内でメモリ外を参照してしまうため，コアダンプが起きてしまう．対象のバグ修正コミットでは図 1 のようにメモリ外への参照を防ぐためにループ文の回数を設定ファイルの値によって制限を加えていた．修正範囲は 1 行であり，Prophet によって修正可能であると判断した．

筆者らは表 1 のように設定ファイルの値を変更しコアダンプを引き起こすテストを含むテストスイートとバグ修正前のプログラムを用意し，Prophet を適用した．

RQ1 Prophet によって生成したパッチは設定ファイルを読み込む部分を無効化し，デフォルトの最大値を利用しメモリを確保するように変更したものであった．このパッチは表 1 の全てのテストに対し期待値を出力し，コアダンプを回避しているためバグを修正していると判断した．

表 1: テストスイート

テスト番号	修正前	期待値	設定ファイルの値
1	正常出力	正常出力	256
2	コアダンプ	正常出力	4

```
    int read_config(void){
        int config_val = 256;
        ...
-       fgets(line, n, file);
+       //設定ファイル読み込みを無効化
        ...
        return config_val;
    }
```

図 2: Prophet によって生成したパッチ

RQ2 Prophet はテストスイートを全て通過するパッチを生成することには成功したが，設定ファイルを読み込む部分を無効化しているため開発者に受け入れられるパッチは生成できていない．

考察 今回作成したテストスイートでは設定ファイルを正常に読み込んでいるかを確認するテストが含まれていないため，図 2 のような開発者に受け入れられないパッチが生成されたと考えられる．今後開発者に受け入れられるパッチを生成するためにはバグの箇所のテストスイートだけでなく元々のプログラムの機能を損なっていないか確認するテストが必要であり，自動バグ修正とテストスイートの洗練を繰り返し行うことが求められる．

5 おわりに

本稿では本研究の目的･貢献と企業内ソースコードへの自動バグ修正適用に向けてのアプローチを説明した．今後の課題としては，データセットの残り 20 件についても自動バグ修正を試みること，自動バグ修正を企業内ソースコードと OSS に適用する上での差異はあるのかについての分析を進めることなどが挙げられる．また，Prophet は OSS から得られたバグ修正を行ったパッチを元に学習しているため，学習を企業内ソースコードで行った場合にバグ修正にどのような影響があるかも調査する．

参考文献

[1] Gazzola, L., Micucci, D. and Mariani, L.: Automatic software repair: a survey, *IEEE Transactions on Software Engineering*, pp. 1–1 (2017).

[2] Long, F. and Rinard, M.: Automatic patch generation by learning correct code, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '16*, pp. 298–312 (2016).

[3] Weimer, W., Nguyen, T., Le Goues, C. and Forrest, S.: Automatically finding patches using genetic programming, *Proceedings of the 31st International Conference on Software Engineering, ICSE '09*, pp. 364–374 (2009).