異なるバージョンの関数実行トレースの比較による フォールト特定及び修正方法の提案

M2015SE010 佐藤成
指導教員：蜂巣吉成

1 はじめに

ソフトウェアはリリース後に判明したフォールトの修正や，機能追加，環境の変化に対応するための修正などでソースコードの変更を伴う保守作業を行う必要がある．この一連の保守作業はバージョンアップと呼ばれている．バージョンアップで既存のフォールトの修正を行った結果，正常なプログラムの動作に影響を与えないようにするのが原則である．しかし実際には新たなフォールトが混入する可能性がある．

ソフトウェアをリリースした開発者であれば，どの部分にフォールトがあるのかをある程度推測できる．しかし，オープンソースでは，セキュリティホールの修正のようなバージョンアップが行われ，そのバージョンアップでフォールトが混入，顕在化した場合はバージョンダウンでは対応できず，開発者の修正を待たなければならない．オープンソースを用いたサービスを提供している場合は利用者の手で修正を行いたいが，内部構造の把握が不十分な状態で修正を行うのはコストがかかる．

本研究の目的は，バージョンアップによって混入したフォールトの特定及び修正手法の提案である．前提として，フォールトが混入していないバージョン (以後，旧バージョン) では正常に動作し，フォールトが混入したバージョン (以後，新バージョン) では故障が発生する入力 (以後，失敗テストケース) が判明しているとする．提案方法の基本的なアイディアは，失敗テストケースに対する旧バージョンと新バージョンでの関数実行トレースを比較してフォールトの箇所を特定し，フォールトがない旧バージョンのソースコードを用いてフォールトを修正することである．失敗テストケースと回帰テストがすべて成功したときに，フォールトが修正されたとする．関数実行トレースとはプログラムの実行において関数が実行された順序，及び関数単体の処理が完遂した順序が合わさった関数の呼出しの流れを示したログのことである。文単位で実行トレースを取得して比較するとコストがかかるので，関数呼出し単位で実行トレースの取得と比較を行う．

フォールトによって，必要な処理が行われない (必要な関数が呼び出されない)，必要のない処理が行われる (間違った関数が呼び出される) と考え，それらを関数実行トレースの比較により特定する．テストが成功するまで，フォールトを含む可能性のある関数を，旧バージョンのソースコードで置換または挿入して変更する．故障は特定の条件のときに発生すると考えて，新バージョンにおいて故障が発生するときの関数呼出しの実引数の値を取得し，そのときに限り旧バージョンのソースコードが実行されるような条件文を追加する．提案方法を実際のソフトウェアに適用し，フォールトが修正できることを確認した．

2 関連研究

2.1 GenProg，RSRepair，SemFix

既に存在する自動修正手法として GenProg[1] と RSRepair[2]，SemFix[3] が挙げられる．これらの手法は修正対象のソフトウェアのソースコードを入力し，コード片を改変したり，特定の行を書き換えることで変種プログラムを生成して，与えられたテストケースをすべて通過したプログラムを出力する．GenProg や RSRepair は，フォールト特定された行に対して，遺伝的アルゴリズムもしくはランダム探索に基づいて同一のソフトウェアのソースコードから一行を選択し，挿入，削除，もしくは置換を行い変種プログラムを生成する．また，SemFix ではフォールト特定した行に対して記号実行を用いることで，テストケースを満たすような制約を計算して，変種プログラムを生成する．この手法では修正に用いた行や書き換えた結果生成された行が修正に効果的かがわからない点があり，変種プログラムが大量に生成され，コストが増大する問題が存在する．

本研究ではフォールトが混入している新バージョンのソースコードを修正対象として，正常に動いていた旧バージョンのソースコードを用いて変種プログラムを生成する．旧バージョンは正しい実行を行っていたことから，効果的に修正が可能だと考えられる．さらに RSRepair や SemFix では修正は一行のみを変更していたが，本研究では関数のソースコードすべてを挿入置換して修正するので，複数行の変更を行う．

3 提案するフォールト特定及び修正方法

本研究では，実行トレースの比較を用いることで，フォールトが含まれている可能性がある関数の絞り込みを行い，旧バージョンのソースコードを挿入置換することでフォールトを修正する．修正の全体を図 1 に示す．入力は旧バージョンと新バージョンのソースコード，故障が発生したときの入力である．出力は回帰テストに成功した場合はフォールトを修正されて回帰テストに成功した新バージョンのソースコードを出力する．フォールト修正段階で失敗したならば，優先度リストや実行トレースを出力する．回帰テストの実行に失敗した場合はフォールトのみ修正した新バージョンのソースコードと優先度リストや実行トレースを出力することで，利用者のフォールト修正の支援を行う．

最初に関数実行トレースの差分を抽出する．差分から関数呼出しの増減を調べて，修正する関数の優先度を決定する．修正候補関数は呼出しが増減した関数そのもの

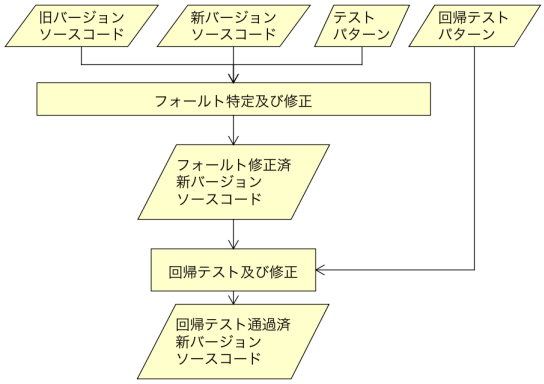

図 1 修正の全体図

や，その関数を呼出していた関数 (以後，呼出し元関数) とする．優先度の基準は呼出し増減の回数とする．呼出し元関数については，呼出した関数の増減回数を基準とする．このときに故障の種類から優先度を決める基準値に重みをつける．そして旧バージョンのソースコードを挿入してフォールトを修正してテストを行う．テストが成功した場合は，回帰テストを行う．

フォールト特定と修正の具体的な方法を 3.1 節，3.2 節，3.3 節で示す．回帰テストの具体的な方法を 3.4 節で示す．

3.1 実行トレースの抽出

フォールト特定及び修正を行う判断材料として関数実行トレースを抽出する．

関数は呼び出された順序だけでなく，関数単体の処理が完遂した順序も処理に大きな影響をもたらすので，動的な呼出し関係のトレースを出力する．さらに差分抽出では旧バージョンに対して新バージョンでどのように関数が増減したのかが関数の絞り込みには必要である．そのために Longest Common Subsequence(以後，LCS) と Shortest Edit Script(以後，SES) の観点によって差分を抽出する方法をとる．同名の関数が複数回呼ばれた際には，何回目に呼ばれている関数がフォールトを含有しているのかを知る必要があるので，差分に対して同名の関数に区別をつける情報を付与する必要がある．

3.2 優先度付け

抽出された差分を元に，新バージョンに移行した結果増減した関数呼出しがどの関数から呼ばれているのかを探し，修正を優先する関数を抽出する．フォールトがある可能性のある関数の抽出基準は SES の観点から行う．SES は挿入，削除，置換の 3 つの変更を用いてどのように旧バージョンのコードから新バージョンのコードに変更されたかを表している．この SES の内容からフォールトの原因になりそうな関数呼出しの増減があるかを推測する．

差分においてどの関数を修正候補関数として扱うかを以下，図 2 に示す．変更された関数の呼出し元が SES に影響をもたらしているので，関数の優先度をつけるための基準値は差が見られた関数とその関数を呼出している回数とする．その中で，故障の種類に応じて基準値に重

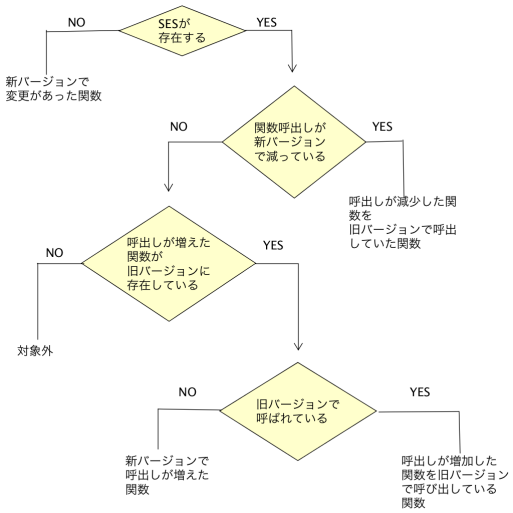

図 2 優先度リストに取り上げる関数

みをつける．例えば，出力内容が期待される結果よりも少ない場合，新バージョンで出力を行っている関数の実行が旧バージョンより減少しているのが原因であると考えられる．このように故障の種類に対して呼出しの回数の集計に重みをつけることで，故障に関係のある関数の優先度を高く設定することができる．以上の理由から，重みを合わせて優先度リストを生成する．

修正候補関数 x の総数 x_n において，x の関数呼出しの変化について，挿入された関数の個数を a_0，削除された関数の個数を a_1，置換された関数の個数を a_2 とした場合の重みは以下の基準で設けるものとする．w は $w > 1$ の正数とする

$$x_n = \begin{cases} wa_0 + a_1 + a_2 & \text{(不要な出力が行われる場合)} \\ a_0 + wa_1 + a_2 & \text{(必要な出力が行われない場合)} \end{cases}$$

3.3 修正

修正は旧バージョンのソースコードを新バージョンへ置換することで実現する．

手順としては旧バージョンの修正対象の関数内部のソースコードを新バージョンに存在する同一の関数と置換する．置換は次の手順で行う．

1. 旧バージョンにおける修正対象の関数のソースコードを新バージョンのソースコードと置換する．
2. コンパイルを行う
3. 成功した場合，失敗テストケースを実行し，テストを行う．
4. 成功した場合は回帰テストに移行する．失敗した場合は次に優先度が高い関数を対象として最初の処理に戻る

優先度リストに記載された関数すべてで失敗した場合は故障が起きたときの引数のときのみ旧バージョンのコードが実行されるようにする．この場合の処理を次に示す．

1. 故障が発生したときの引数の値を条件式とした if 文

で旧バージョンを条件分岐させる．

2. 条件分岐させた旧バージョンのソースコードを新バージョンの関数に挿入する．
3. コンパイルを行う
4. 成功した場合，失敗テストケースを実行し，テストを行う．
5. 成功した場合は，回帰テストに移行し，正常になっていないのならば，次に優先度が高い関数に対して同じ処理を実行する．

優先度リストすべてに処理を行った場合は，修正に失敗したとし，関数実行トレースや優先度リスト，引数の情報を出力し，利用者の手で修正を行うことに対する支援を行う．

3.4 回帰テスト

修正によってソフトウェア本来の動作を損なっていないかを回帰テストで調べる．回帰テストが通らなければ，その原因を調べる必要がある．回帰テストが失敗した場合は，修正が適応される範囲が広いのが回帰テスト失敗の原因であることが考えられるので，旧バージョンのソースコードが実行される状況を限定する．ソースコード内部に存在する変数の値や関数を if 文の条件式とし，旧バージョンのソースコードがその if 文の内部で実行されるようにコードを修正する．条件分岐に用いる変数の値や関数は，定義されている変数，他の if 文の条件式に代入されている実引数や関数，変数に代入されている値を選択する．成功した場合は，修正が完了したソースコードを出力し，失敗した場合はフォールトの修正のみ完了しているソースコードと実行トレースや優先度リストを出力する．

4 実験

提案手法の有効性を評価するために実験を行った．対象は nkf(Network Kanji Filter) を使用した．nkf は異なる文字コード間の変換を行うソフトウェアであり，バージョン 2.0.6 にて UTF-8 の日本語文字列を MIME エンコードすると，最後の ?= と改行が出力されないというソフトウェア故障が発生した．この故障はバージョン 2.0.7 でも発生しており，バージョン 2.0.8 で修正された．

nkf のソフトウェア故障

```
$ echo ’名古屋’ | nkf − w | nkf-2.0.5/nkf − M
=?ISO-2022-JP?B?GyRCTD44RTIwGyhC?=
(バージョン 2.0.5 では正常)
$ echo ’名古屋’ | nkf − w | nkf-2.0.6/nkf − M
=?ISO-2022-JP?B?GyRCTD44RTIwGyhC
(バージョン 2.0.6 では異常)
```

実験ではバージョン 2.0.5 を旧バージョン，バージョン 2.0.7 を新バージョンとして修正を行う．バージョン 2.0.5 は 12209 行，バージョン 2.0.7 は 14347 行のコードで実装されており，バージョン 2.0.7 ではバージョン 2.0.5 に対して 2240 行の行の追加と削除，4539 行の置換が行われていた．この故障に対し，本研究の手法を優先度リストの作成までをツールで行い，修正については自動化ができていないので，手動で原因となるフォールトを取り除いた．今回重みとして必要な出力が行われない場合の式に w=2 を代入して優先度を計算した．

関数実行トレースと引数の値を抽出するツールとして dtrace を用いた．dtrace は汎用情報採取のフレームワークであり，C/C++言語の動的なトレースを取得することが可能である．実行トレースの抽出では dtrace 向けのスクリプトとしてトレース採取用のスクリプトをあらかじめ作成し，故障が発生したときの入力と共に dtrace で実行することで抽出する．このスクリプトの実行を旧バージョンと新バージョンの両方に行う．

両バージョンの関数実行トレースの差分抽出を行い，呼び出された個数順に関数名と差分における行数のリスト (以後，行あり優先度リスト) を生成する．この後の修正に用いるために行数を削除しもう一度ソートして関数名のみのリスト (以後，行なし優先度リスト) も生成する．優先度リストから w_iconv と w_status を修正候補関数として扱う．

優先度リストから w_iconv と w_status を修正対象として旧バージョンのソースコードを新バージョンと置換しコンパイルを行った結果，コンパイルに失敗した．手順に従い dtrace の実行時の引数の状態を抽出する機能を用いて，w_iconv と w_status の新バージョンでの引数の値を抽出した．

行あり優先度リストからどの引数が対応するのかを調べ，関数名とその関数が呼出した関数の個数，そして引数の値のリストを生成した．

引数付き優先度リスト

```
w_iconv 20 個 args(4294967295, 0,0)
//dtrace の仕様から − 1 は符号なし整数で表現される
w_iconv 4 個 args(229, 184,130)
w_iconv 4 個 args(229, 177,139)
w_iconv 4 個 args(229, 143,164)
w_iconv 4 個 args(229, 144,141)
w_status 4 個
args(4443477728, 10,140735231599304)
————以下省略———
```

ソースコード 1 は旧バージョンの w_iconv のソースコードであり，ソースコード 2 は新バージョンの修正後の w_iconv のソースコードである．ソースコード 2 の 5 行目から 17 行目が提案方法により挿入されたコードである．引数が c2=− 1，c1=0，c0=0 のときを、if 文の条件式として条件分岐させ，旧バージョンの w_iconv のコードをソースコード 2 の 8 行目から 14 行目に挿入することでフォールトの修正が確認できた．その後回帰テストに失敗したので，修正として MIME と名がついている変数や関数を変数定義から一行ずつ挿入した．if 文の条件式に mimeout_mode を挿入して条件分岐したコードを 2 の 6 行目から 16 行目までに挿入した結果，回帰テスト

を通過した．

ソースコード 1 旧バージョンのコード

```
int w_iconv(c2, c1, c0)
    int c2,c1, c0;
{
    int ret = w2e_conv(c2, c1, c0, &c2, &c1);
    if (ret == 0){
        (*oconv)(c2, c1);
    }
    return ret;
}
```

ソースコード 2 修正後のコード

```
1  nkf_char w_iconv(nkf_char c2, nkf_char c1,
       nkf_char c0)
2  {
3      nkf_char ret = 0;
4
5      //修正箇所
6      if (mimeout_mode) {//回帰テストの修正
7      //フォールト修正
8          if(c2==− 1&&c1==0&&c0==0){
9              ret = w2e_conv(c2, c1, c0, &c2, &c1);
10             if (ret == 0){
11                 (*oconv)(c2, c1);
12             }
13             return ret;
14         }
15         //フォールト修正終了
16     }
17     //回帰テストの修正終了
18
19     /* throw away ZERO WIDTH NOBREAK SPACE
           (U+FEFF) */
20
21     if(ignore_zwnbsp_f){
22         ignore_zwnbsp_f = FALSE;
23         if(c2 == 0xef && c1 == 0xbb && c0 == 0
               xbf)
24             return 0;
25     }
26     以下省略
```

5 考察

今回は nkf を用いて実験を行ったが，今回は入力に対して出力を返す処理に対してフォールトが見つかった，しかし UI やファイル保存などの処理でフォールトが起こる可能性もあり，関数の呼出しが複雑になる可能性が考えられる．その場合は関数実行トレースの内容をグラフ化することで可視化を行い，自動修正が困難な場合でも利用者の修正コストを減少させる工夫が必要となる．

重みを故障の種類で設定したが，関数の機能を考慮できれば，重みをさらに詳細に設定できる．関数単位で修正に失敗した場合は，優先度リストに記載された関数に対して文単位で比較して修正する方法も考えらえる．異常な入力の差分で失敗した場合や，差分が多い場合は，正常な入力を実行してトレースを抽出し，その差分内容をフォールト特定に用いることも考えられる．

本研究の提案方法では，回帰テストで発生した故障を修正するために，修正対象プログラムのソースコード中に存在している関数や変数の値を if 文の条件式に当てはめて，旧バージョンのソースコードが実行される条件をさらに定義する．この手法は GenProg で修正に用いるコード片を捜索する方法と類似しているが，文献 [1] と文献 [4] から，GenProg を始めとする再利用を用いた手法では修正できるフォールトは約半数であることが示されている．

自動プログラム修正の修正可能バグ数に関する考察 [6] では再利用に基づく手法で修正できるフォールトの数を約半数から増やすために，大規模データセットを用いている，さらには変数名を正規化してソースコード行を再利用し，フォールトを含む箇所の周辺から変数名を復元することで再利用可能なソースコード行を増加させる手法を提案し，修正できたことを示している．この研究で用いられている変数名を正規化する手法を用いることで，フォールト修正に利用できそうな関数や変数を増加させることで，回帰テストの修正の成功率を高めることが考えられる．

6 おわりに

本研究では，ソフトウェアの関数実行トレースの差分からフォールトが含まれている可能性がある関数に対して優先度付けを行い，旧バージョンのソースコードを挿入，置換して修正を行う方法を提案した．

今後の課題として nkf 以外の他のプログラムでも実験を行うことや，優先度付けの重みの設定の拡張や回帰テストの自動化，バージョンアップによる関数の仮引数の変更に対応することである．

参考文献

[1] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer, “A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each,” ICSE ’12, pp.3-13, IEEE Press, Piscataway, NJ, USA, 2012. http://dl.acm.org/citation.cfm?id=2337223.2337225

[2] Y. Qi, X. Mao, Y. Lei, Z. Dai, and C. Wang, “The strength of random search on automated program repair,” ICSE ’14, pp.254-265, ACM, New York, NY, USA, 2014. http://doi.acm.org/10.1145/2568225.2568254

[3] H.D.T. Nguyen, D. Qi, A. Roychoudhury, and S. Chandra, “Semfix: program repair via semantic analysis,” In 35th International Conference on Software Engineering (ICSE) 2013., pp.772âĂŞ781, May 2013.

[4] E. T. Barr, Y. Brun, P. Devanbu, M. Harman, and F. Sarro, ”The Plastic Surgery Hypothesis,” FSE 2014, pp.306-317, 2014.

[5] Sun Microsystems,USENIX “04-Technical Paper,General Track”, 2004. http://www.usenix.org/events/usenix04/tech/general/full_papers/cantrill/cantrill_html/index.html

[6] 鷲見 創一, 肥後 芳樹, 堀田 圭佑, 楠本 真二 “自動プログラム修正の修正可能バグ数に関する考察，”コンピュータソフトウェア，Vol. 33 no. 3 p. 81-87，Jul. 2016.