社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

再利用に基づく自動プログラム修正における更新順を用いた挿入候補の絞込の提案

山本 将弘[†] 横山 晴樹[†] 肥後 芳樹[†] 楠本 真二[†]

† 大阪大学大学院情報科学研究科 〒 565–0871 大阪府吹田市山田丘 1–5
E-mail: †{m-yamamt,y-haruki,higo,kusumoto}@ist.osaka-u.ac.jp

あらまし 自動プログラム修正において，ソースコード中の行を再利用して修正を行う研究が行われている．再利用に基づく自動プログラム修正では，挿入する行をソースコード中からランダムに選択している．本研究では，ソースコード中の各行を最終更新日時の新しい順で選択する更新順という基準を提案し，ランダムではなくその基準により行を選択することで，修正に要する時間の削減を図る．本稿では，オープンソースソフトウェアで過去に行われた修正において用いられた行を調査することで，更新順が有用であることを示した．

キーワード デバッグ，自動プログラム修正，コード再利用

1. まえがき

ソフトウェア開発において，プログラムの信頼性を向上するためにデバッグを行うことが必要である．デバッグの主な工程は，欠陥の原因となる箇所の特定と，特定された箇所の修正である．デバッグは多くのコストを要する作業であるため，自動化できることが望ましい．自動で欠陥箇所の特定を行う研究が行われてきている [1], [2]．また，近年になり，自動で欠陥を修正する研究が活発に行われている [3]〜[6]．

自動で欠陥を修正する研究において，遺伝的アルゴリズムによって欠陥の修正を行う GenProg [3] が注目されている．GenProg は再利用に基づく自動プログラム修正手法である．再利用に基づく自動プログラム修正では，ソースコード中にある行を用いて欠陥の修正を行う．他に再利用に基づく自動プログラム修正手法として RSRepair [4] などがある．

GenProg や RSRepair はプログラム修正においてソースコード中にある行をランダムに選択して用いる．ランダムな選択では，修正対象のプログラムの規模が大きくなるにつれて修正に寄与しない行を選択する回数が多くなり，修正に時間がかかってしまうという課題がある．そのため，修正に寄与する行ができるだけ早く選択されるような基準が必要である．横山らは，欠陥が存在している箇所と類似度が高いコード片に存在している行を優先的に選択する類似度順という基準を提案し，調査を行った [7]．調査の結果，類似度順が行を選択する基準として有用であることを示した．しかし，修正に寄与する行であるにも関わらず，欠陥を含む箇所とは類似していないコード片に存在する場合があることが確認されている．

本研究では更新順という基準を提案する．更新順とは，ソースコード中の各行を最終更新日時の新しい順で選択する基準である．本稿では更新順の有用性の調査を行った結果について述べる．調査はオープンソースソフトウェアの過去に行われた欠陥の修正において，追加または変更された行を対象とした．実験の結果，対象の行のうちの 60%が更新順の上位 10%に存在していたことがわかった．また，同じ対象に対して横山らの手法を適用し，比較を行った．その結果，類似度順では優先的に選択されないが，更新順では優先的に選択される行が多数あることがわかった．

2. 既存研究

本章では本研究に関連する既存研究について述べる．

2.1 GenProg

GenProg [3] は遺伝的プログラミングを用いて再利用に基づく自動プログラム修正を行う手法である．GenProg は入力として欠陥を含むプログラムとテストケースを受け取り，全てのテストケースを通過するプログラムを出力する．図 1 に GenProg の動作の流れを示す．

遺伝的プログラミングは，選択・変異・交叉の 3 つの処理を繰り返し行うことにより，優秀な個体を生成する，生物の進化を模した手法である．選択・変異・交叉の処理は条件を満たす個体が生成されるまで繰り返し適用される．GenProg では，プログラムを個体とみなし，全てのテストケースを通過するプログラムが生成されるまで，3 つの処理を繰り返し適用する．

選択処理では，各個体のテストケース通過状況を基に適合値を求め，適合値の高い順に一定数の個体を取り出す．

変異処理では，各個体に対して変更を行う．変更を行う箇所は欠陥の原因として限局された箇所である．変更は，削除，挿

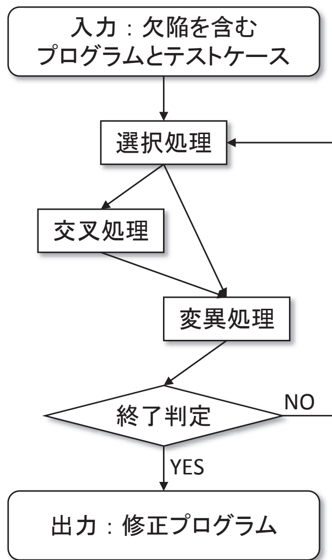

図 1　GenProg の動作の流れ

入，置換の 3 種類の操作である．削除操作では限局された箇所を削除する．挿入操作ではソースコード中からランダムに選択された行を限局された箇所に挿入する．置換操作では削除と挿入を行う．

交叉処理では，2 つの個体を混ぜ合わせることにより，新たな個体を生成する．2 個体は個体の集合からランダムに選択される．

GenProg はオープンソースソフトウェアで過去に修正が行われた 105 個の欠陥のうち，55 個を自動的に修正することに成功した [3].

2.2　横山らの手法

本節では，再利用に基づく自動プログラム修正が抱える課題に関する横山らの手法 [7] について説明する．はじめに横山らの研究で用いられた用語を説明する．

挿入候補　変異処理において挿入の対象となりうる行の集合

挿入行　挿入候補のうち，実際に挿入された行

周辺領域　ある行を中心とした前後数行

GenProg や RSRepair では，挿入行をソースコード中からランダムに選択している．ランダムな選択では，プログラムの規模が大きくなるにつれ，挿入候補の数が増大し修正に寄与しない行を選択する回数が増える．そのため，大規模なプログラムに対しては，プログラム修正に時間を要してしまうという課題点が存在する．この課題点を改善するために，横山らの研究では，ソースコードの各行に優先度付けを行い，優先度順に挿入する必要性があることを述べた．

横山らは，実際に行われたプログラム修正の例を元に，ソースコードの行のうち，その周辺領域と欠陥が存在している行の周辺領域が類似している場合は，欠陥を修正できる可能性が高いと考えた．オープンソースソフトウェアで過去に行われた修正で用いられた挿入行に対して調査を行い，挿入行を含む領域と欠陥が存在していた行の周辺領域の類似度は挿入行を含まない領域と欠陥が存在していた行の周辺領域の類似度よりも高くなる傾向があることを示した．

さらに横山らは，類似度の高い順を表す類似度順が，挿入行を選択する基準として有用であるかどうか確認するために追加調査を実施した．追加調査では，過去に行われた修正で用いられた挿入行のカバレッジを算出した．カバレッジとは，ソースコード行を欠陥を含む箇所との類似度が高い順に並べたときに，欠陥を修正できる行が全体のうちどの程度上位に来るのかを表す値である．カバレッジを算出することでソースコードの何%を調べると挿入行が選択されるのかが分かる．

追加調査の結果，類似度順を用いることで，調査対象とした挿入行のうち 75%はソースコード中の行の類似度順上位 10%に存在することを示した．

2.3　横山らの手法の課題

類似度順により挿入行を選択することが効果的なのは，欠陥が存在している行の周辺領域と類似したコード片にその欠陥の修正を行える行が存在している場合である．欠陥の修正を行える行が類似したコード片には存在していない場合は，横山らの手法は効果的ではない．図 2 に例を挙げる．この例は，オープンソースソフトウェアの Apache httpd のプログラムに対して実際に行われた修正である．

図 2 では，ある欠陥に対して，代入文（灰色の行）を他のメソッドに移動することで修正を行っている．つまり，移動先に対する行の挿入と移動元の行の削除により修正が完了する．この代入文の移動元と移動先の周辺コードは類似度が高くなく，横山らの手法では効果的に欠陥の修正を完了させることができない．このような課題点があるため，別の基準によるアプローチを行う必要がある．

3.　アプローチ

前章において，横山らの手法における挿入行を選択する基準である類似度順とは別の基準を設ける必要性を述べた．そこで本研究では，挿入行を選択する基準として，更新順というものを提案する．更新順の定義は以下の通りである．

更新順　ソースコード中の各行を更新日時の新しい順で選択する基準

本研究において，行の更新日時は行に対して最後に加えられた変更の日時を表すこととする．ソースコードに追加されて以降 1 度も変更されていない行については，追加された日時を更新日時とする．行の更新日時は，バージョン管理システムを利用して取得する．

再利用に基づく自動プログラム修正において，挿入候補から更新順により行を選択する手法を提案する．この提案は，最近行われた変更が修正に寄与するのではないかという仮定に基づいている．

実際に更新順の上位から順に挿入行を選択することでうまくいく例として，前章の既存研究の課題における例が挙げられる．図 2 では，ある 1 行を他のメソッドに移動することで修正を行っている．この行はリビジョン r88904 のコミットにおける修正の少し前のコミットにおいて追加された行であるため，更新順で優先的に選択される．このように，最近追加されたある

```
mod_log_config.c
212  static apr_hash_t *log_hash;
     …
1154 static void log_pre_config(apr_pool_t *p, apr_pool_t *plog, apr_pool_t *ptemp)
1155 {
1156    static APR_OPTIONAL_FN_TYPE(ap_register_log_handler) *log_pfn_register;
1157
1158    log_hash = apr_hash_make(p);
1159    log_pfn_register = APR_RETRIEVE_OPTIONAL_FN(ap_register_log_handler);
1160    if (log_pfn_register) {
1161        log_pfn_register(p, "h", log_remote_host, 0);
            …
1188    }
1189 }
1190
1191 static void register_hooks(apr_pool_t *p)
1192 {
1193    ap_hook_pre_config(log_pre_config,NULL,NULL,APR_HOOK_REALLY_FIRST);
1194    ap_hook_child_init(init_child,NULL,NULL,APR_HOOK_MIDDLE);
1195    ap_hook_open_logs(init_config_log,NULL,NULL,APR_HOOK_MIDDLE);
1196    ap_hook_log_transaction(multi_log_transaction,NULL,NULL,APR_HOOK_MIDDLE);
1197
1198    APR_REGISTER_OPTIONAL_FN(ap_register_log_handler);
1199 }
```

(a) 修正前（r88903）

```
mod_log_config.c
212  static apr_hash_t *log_hash;
     …
1154 static void log_pre_config(apr_pool_t *p, apr_pool_t *plog, apr_pool_t *ptemp)
1155 {
1156    static APR_OPTIONAL_FN_TYPE(ap_register_log_handler) *log_pfn_register;
1157
1158    log_pfn_register = APR_RETRIEVE_OPTIONAL_FN(ap_register_log_handler);
1159
1160    if (log_pfn_register) {
1161        log_pfn_register(p, "h", log_remote_host, 0);
            …
1187    }
1188 }
1189
1190 static void register_hooks(apr_pool_t *p)
1191 {
1192    ap_hook_pre_config(log_pre_config,NULL,NULL,APR_HOOK_REALLY_FIRST);
1193    ap_hook_child_init(init_child,NULL,NULL,APR_HOOK_MIDDLE);
1194    ap_hook_open_logs(init_config_log,NULL,NULL,APR_HOOK_MIDDLE);
1195    ap_hook_log_transaction(multi_log_transaction,NULL,NULL,APR_HOOK_MIDDLE);
1196
1197    /* Init log_hash before we register the optional function. It is
        …
1202    */
1203    log_hash = apr_hash_make(p);
1204    APR_REGISTER_OPTIONAL_FN(ap_register_log_handler);
1205 }
```

(b) 修正後（r88904）

図 2　類似度順で挿入行を選択するのが適さない変更例

行が欠陥の原因であり，その行を移動することで修正が実現するような場合，更新順による選択は適している．

その他にも更新順による選択が有用である例として，2 箇所ある同様の欠陥に対してどちらか一方のみプログラム修正が行われた場合などが挙げられる．この場合，一方の修正直後であれば，更新順を用いるともう一方の修正に必要な行は優先的に選択されるため，有用である．

4. 調　　査

本章では，調査の目的，手順，対象について述べる．

4.1 調査目的

前章では，更新順という基準を提案した．本稿では，再利用に基づく自動プログラム修正手法に更新順による行の選択を実装する前に，更新順が挿入行を選択する基準として有用であるかを調査する．更新順においてプログラムを修正する行が優先的に選択されていれば，更新順が有用であるということを示せ

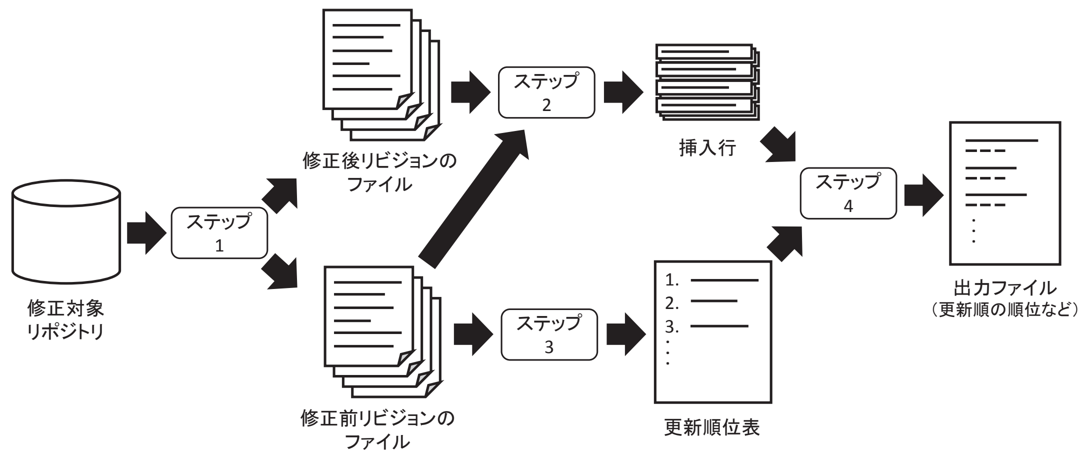

図 3　修正コミットごとの処理

る．調査の結果，更新順が有用であることが示せれば，今後の研究で再利用に基づく自動プログラム修正手法に更新順を実装する価値が有ることになる．

4.2　調 査 手 順

本研究ではオープンソースソフトウェアのプログラムで過去に実際に行われた修正で用いられた挿入行に対して調査を行う．過去のプログラム修正の情報はオープンソースソフトウェアのリポジトリを解析することにより収集する．調査対象としたオープンソースソフトウェアについては 4.3 節で述べる．

調査の手順について述べる．まずはじめに，リポジトリの全コミットに対してコミットログを調べ，修正コミットを特定する．修正コミットとは，コミットログに “fix”，“fixed”および “fixing”という単語が含まれているコミットのことである．特定した全ての修正コミットに対して，以下の 4 つの処理を行う．

ステップ 1　修正前後のリビジョンのファイル取得
ステップ 2　挿入行の抽出
ステップ 3　更新順位表の作成
ステップ 4　挿入行の更新順における順位取得

上記の手順を図 3 に示す．それぞれのステップについて説明する．

ステップ 1：修正前後のリビジョンのファイルの取得

修正コミットで行われた修正の前後のリビジョンからソースコードを取得する．修正前のリビジョンでは，各行の更新日時も取得する．

ステップ 2：挿入行の抽出

修正前後のリビジョンにおけるソースコードの差分を取り，挿入行を抽出する．差分は行単位の追加・削除で表される．そのうち，追加または変更された行を挿入行とする．

ステップ 3：更新順位表の作成

修正前のリビジョンにおけるソースコード中の全ての行に対して，その内容とコミット日時を取得する．全ての行をコミット日時の新しい順に並び替えて更新順位表を作成する．このとき，同じコミットにおいて追加または変更された行についてはそれらの行の中で再度ランダムに並び替えを行う．

ステップ 4：挿入行の更新順における順位取得

次に，各挿入行の更新順における順位を取得する．順位表を基にそれぞれの挿入行の更新順の順位を取得する．得られた順位は評価に用いるためソースコードの総行数とともにファイルに出力する．以上で修正コミットに対する処理を終了する．

全ての修正コミットにおける挿入行の更新順の順位を取得した後，評価を行う．評価は各挿入行に対して更新順のカバレッジを基にして行う．カバレッジは以下の式で算出される．

$$\text{カバレッジ} = \frac{\text{更新順の順位}}{\text{ソースコードの総行数}}$$

カバレッジは 0 より大きく 1 以下であるような値を取る．値が 0 に近づくにつれ更新順で優先して選択されることを示し，その挿入行に対しては更新順を用いた挿入候補の選択が適しているといえる．全ての挿入行のカバレッジを算出したのち，それらを 0.1 ごとに区切った 10 区間内に分類し，区間ごとの挿入行の数をカウントする．例えばカバレッジが 0.23 である場合，0.2 から 0.3 の区間のカウントを 1 増やす．このようにして全ての挿入行のカバレッジにおいてカウントを行う．最終的なカウント数から，更新順で全ての挿入候補のうちある割合まで選択したときにどれくらいの挿入行を見つけることができるかを判断することができる．具体的な得られた数値の評価は 5. 章の考察で行う．

さらに，本研究の基準である更新順と横山らの手法 [7] における基準である類似度順の比較を行う．同じ調査対象に対して横山らの手法のプログラムを実行し，挿入行を抽出して類似度順のカバレッジを取得する．得られた出力ファイルから更新順の出力ファイルと挿入行の対応付けを行う．対応付けできた挿入行から更新順と類似度順のそれぞれのカバレッジが 0.1 以下，0.2 以下であった挿入行の集合を基に表を作成する．得られた表について 5. 章において考察を行う．

4.3　調 査 対 象

調査対象はバージョン管理システム Subversion で管理された 4 つのオープンソースソフトウェアであり，横山らの研究 [7] における調査対象と同じである．調査対象オープンソースソフト

表 1　調査対象のオープンソースソフトウェア

ソフトウェア	言語	開始リビジョン（日付）	終了リビジョン（日付）
Apache httpd	C	76,295	1,722,377
		(2004/11/19)	(2015/12/31)
CBMC	C++	1	6,211
		(2011/05/08)	(2015/12/30)
JabRef	Java	1	3,718
		(2003/10/15)	(2011/11/11)
jEdit	Java	1	24,280
		(2006/07/01)	(2015/12/31)

表 2　各ソフトウェアにおける挿入行の数

ソフトウェア	挿入行の数	更新順が取得できた挿入行の数
Apache httpd	26,679	7,807
CBMC	4,890	1,642
JabRef	12,260	4,116
jEdit	21,161	7,672

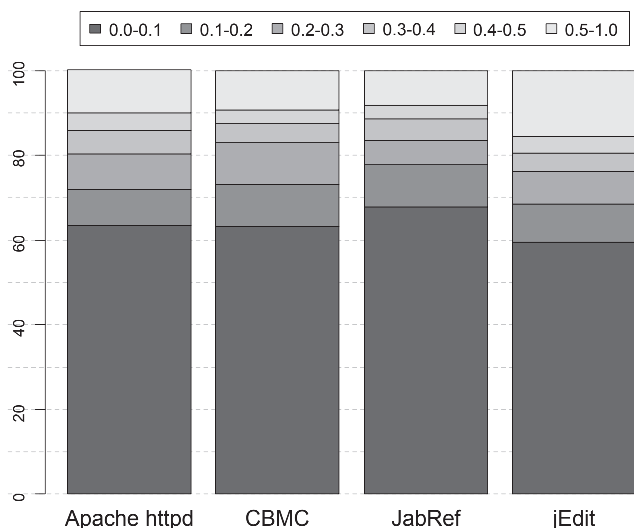

図 4　更新順のカバレッジの分布

ウェアの表は表 1 である．横山らは調査する挿入行を調査対象のオープンソースソフトウェアのリポジトリに対して特定のリビジョン範囲から視認で選択していた．それに対して，本研究ではリビジョンの範囲を初期リビジョンから 2015/12/31 までのリビジョンと定め，全ての挿入行を調査する．なお，JabRef に関しては現在 Git のみのバージョン管理となっているため，Subversion でバージョン管理されていたリビジョンまでを範囲とする．

調査対象のオープンソースソフトウェアは C 系の言語と Java からそれぞれ 2 つずつ採用している．これにより，結果が言語に依存するかどうかを見ることができる．

5. 結果と考察

本章では，調査で得られた結果を述べ，考察を行う．

更新順の調査

各ソフトウェアにおける挿入行の数を表 2 にまとめる．挿入行の数は 4.2 節の調査手順で述べた挿入行の抽出において抽出した行の数である．更新順が取得できた挿入行の数は挿入行のうち修正前のリビジョンのソースコード中に挿入行が存在するものの数である．表 2 の更新順が取得できた挿入行のそれぞれにおいて取得したカバレッジの分布をまとめたものが図 4 である．図が複雑になるのを防ぐためにカバレッジが 0.5 以上の範囲は 1 つにまとめている．図 4 からそれぞれの調査対象のオープンソースソフトウェアにおいて挿入候補全体の 10 分の 1（カバレッジが 0.1 以下）に約 60%の挿入行が集中しているということが分かる．JabRef においては 70%近くの挿入行が集中している．また，挿入候補の半分を見ることで約 90%の挿入行を発見することができる．つまり，更新順で順位が全体の半分より下位の挿入候補に挿入行があるのは 10 回中 1 回しかないということである．調査結果より更新順が有用であることが確認できた．

調査対象のオープンソースソフトウェアは C 系の言語で記述されたソフトウェアと Java で記述されたソフトウェアからそれぞれ 2 つずつ選択した．Apache httpd，CBMC は C 系の言語で記述されたソフトウェア，JabRef，jEdit は Java で記述されたソフトウェアである．挿入行が上位に集中する割合でソフトウェアの大小を表すと，JabRef，CBMC，Apache httpd，jEdit の順番になる．そのため，本研究の調査対象においては，更新順が言語に依存していないと判断できる．

更新順と類似度順の比較

更新順と類似度順の上位の関係を表 3 に表す．各ソフトウェアに対して条件を満たす挿入行の数を記載している．挿入行の数は，更新順と類似度順の出力ファイルから対応付けができた挿入行の数である．カバレッジが 0.1 以下，0.2 以下とはすなわち基準の上位 10%，20%で見つかった挿入行の集合である．両方の列は更新順と類似度順の両方で上位であった挿入行の集合である．更新順のみの列は更新順では上位であり類似度順では上位でない挿入行の集合である．類似度順のみの列は類似度順では上位であり更新順では上位でない挿入行の集合である．表 3 の更新順のみの列の値から，更新順では優先的に選択されるが，類似度順では優先的に選択されない挿入行が存在することを確認できた．

また，更新順と類似度順の上位 10%を組み合わせることで挿入行の約 70%を見つけることができ，上位 20%を組み合わせることで挿入行の約 80%を見つけることができる．片方の基準のみで上位である挿入行が全体に対して無視できない割合で存在するため，互いの基準が相手の基準の欠点を補っていることが分かる．よって，更新順と類似度順から交互に挿入行を選択するといった手法を再利用に基づく自動プログラム修正手法に実装することで，修正にかかる時間を短縮できる可能性がある．しかし，更新順と類似度順を組み合わせる方法にはデメリットがある．それはそれぞれの基準によって挿入候補を順位付けする処理を行う必要があり，それぞれの基準を単独で使用するより時間を要してしまうことである．基準を組み合わせて再利用に基づく自動プログラム修正手法に実装をするときは，それぞれの順位付けの処理を並列で行うなど，デメリットを考慮した実装が必要になる．

表 3　更新順と類似度順の上位の関係

ソフトウェア	挿入行の数	カバレッジが 0.1 以下				カバレッジが 0.2 以下			
		両方	更新順のみ	類似度順のみ	計	両方	更新順のみ	類似度順のみ	計
Apache httpd	4,739	1,715	1,079	734	3,528（74.6%）	2,186	1,083	609	3,878（81.8%）
CBMC	1,209	483	280	204	967（80.0%）	610	271	174	1,055（87.2%）
JabRef	1,675	569	387	267	1,223（73.0%）	778	397	186	1,361（81.3%）
jEdit	5,996	1,577	1,779	693	4,049（67.5%）	2,008	1,932	641	4,582（76.4%）

6. 妥当性への脅威

本章では本研究における妥当性への脅威について述べる．

本研究では，修正が行われたコミットを表す修正コミットという用語を，コミットログに特定の修正に関するキーワードを含むコミット，と定義している．そのため，キーワードを含まないが修正を行ったコミットを対象にすることができていない．また，キーワードを含むが修正を行っていないコミットを対象にしてしまっている可能性が考えられる．

本研究では，修正コミットにおける挿入行を対象に調査している．しかし，修正の際には修正とは無関係の行を同時にコミットすることが考えられる．そのような場合，挿入行に修正とは無関係な行が含まれる可能性がある．

本研究では，調査対象として 4 つのオープンソースソフトウェアを調査した．調査対象を変更すると調査の結果が変わる可能性がある．プログラムは C 系の言語から 2 つ，Java から 2 つ選択して対象としている．それ以外の言語のプログラムでは調査を行っていない．そのような言語を対象とした自動プログラム修正手法に更新順を実装する際には再度調査を行うべきである．

7. あとがき

本研究では，再利用に基づく自動プログラム修正における挿入候補の選択手法として，更新順という基準を用いることを提案した．更新順の有用性を調べるために，オープンソースソフトウェアで実際に行われたプログラム修正に対して調査を行った．調査の結果，更新順が挿入候補から行を選択する手法として有用であることがわかった．さらに，横山らの手法の基準である類似度順でも同様の対象に調査を行うことで，類似度順と更新順の関係性を調べた．その結果，類似度順では優先的に選択されないが更新順だと優先的に選択されるような挿入行の存在が確認できた．

今後の課題は，挿入候補を選択する基準として，更新順単独または更新順と類似度順を組み合わせた基準を実際に再利用に基づく自動プログラム修正手法に実装して，プログラム修正に要する時間の短縮できるかどうか調べることである．

謝辞

本研究は，日本学術振興会科学研究費補助金基盤研究（S）（課題番号：25220003），および文部科学省研究費補助金若手研究（A）（課題番号：24680002）の助成を得て行われた．

文献

[1] J.A. Jones, M.J. Harrold, and J. Stasko, “Visualization of test information to assist fault localization,” ICSE ’02, pp.467–477, 2002.
[2] X. Wang, S.C. Cheung, W.K. Chan, and Z. Zhang, “Taming coincidental correctness: Coverage refinement with context patterns to improve fault localization,” ICSE ’09, pp.45–55, 2009.
[3] 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, 2012.
[4] 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, 2014.
[5] D. Kim, J. Nam, J. Song, and S. Kim, “Automatic patch generation learned from human-written patches,” ICSE ’13, pp.802–811, 2013.
[6] H.D.T. Nguyen, D. Qi, A. Roychoudhury, and S. Chandra, “Semfix: Program repair via semantic analysis,” ICSE ’13, pp.772–781, 2013.
[7] 横山晴樹，大田崇史，堀田圭佑，肥後芳樹，岡野浩三，楠本真二，“再利用に基づく自動バグ修正における再利用候補の絞込に向けた調査，” 電子情報通信学会技術研究報告，pp.047–052，2015.