# レガシーソフトウェア保守プロセスにおける開発者によるコードクローン認識についての観察

山科 隆伸 上野 秀剛 伏田 享平 亀井 靖高 名倉 正剛 川口 真司 飯田 元
国立大学法人 奈良先端科学技術大学院大学 情報科学研究科
{takanobu-y, hideta-u, kyohei-f, yasuta-k, nag, kawaguti}@is.naist.jp iida@itc.naist.jp

## 1. はじめに

一般的に，大規模なレガシーソフトウェアは，長年にわたって機能の追加・変更や，不具合修正が行われており，コードクローン（以後，クローンと略記する）を含むモジュールの割合が大きいことが知られている．ソフトウェアの一つのクローンに変更を加える時には，他の全てのクローンにも同様の変更を行わなければならないことが多く，保守コストが増大する原因となる．この際に，変更し忘れや見逃しが生じた場合には，不具合が混入し，ソフトウェアの信頼性が低下する原因にもなる[1]．そのため，レガシーソフトウェア保守開発プロセスでは，クローンを認識して保守を行うことが特に重要である．しかし，クローンに関する多くの研究は，クローンの検出手法や検出率の向上に関して行われている[2]．我々の知る限りでは，検出されたクローンが保守プロセスでどのように役立っているのか，利用されているのかなど，実際の保守現場における開発者の行動を分析し評価した研究はほとんどない．

本論文では，まず，レガシーソフトウェア保守を行う開発者が，クローンをどの程度意識してプログラムを修正しているのか，またクローンを検出した場合にどのように修正を行っているのかを実験により明らかにする．その上で，修正時の開発者の行動を分析し問題点を明らかにする．そして，クローン検出手法を実際の開発現場に適用するために必要な要件を定義する．

## 2. 実験

### 2.1 実験概要

まず，クローンを含むファイルを開発者がどの程度意識して修正しているかを定量的に計測するために，ソースコードリポジトリの分析を行った．次に，実際の開発者 3 人の保守作業を観察した．そして，その 3 人を含む開発者 8 人に対し，インタビューを行った．

調査したプロジェクトは，10 年前に開発された商用アプリケーションのライブラリを修正するための保守開発プロジェクトである．開発対象のソースコードは，コード行数は合計 160 万行，4400 ファイルで構成されており，C および C++言語で記述され，CVS を利用してバージョン管理されている．全ソースコード行数に占めるクローンの割合は，クローン検出ツール CCFinderX[3]で計測したところ，21%であった．開発メンバー8 人のプロジェクト経験は，2 年未満 2 人，2 年～8 年未満 4 人，8 年以上 2 人である．

**実験 1：クローンを含むファイルの修正状況の分析**

最初に，CVSリポジトリに管理されているソースコードを対象として，[4]において提案されている手順に従い，5分以内に同じ開発者が同じコメントでコミットしている処理（以降，トランザクションと呼ぶ）を抽出した．

次に，クローンを含むファイルを修正しているトランザクションと，同一のクローンを含む複数のファイルを修正しているトランザクションが，全トランザクションに占める割合 $R_c$, $R_s$ を，それぞれ以下の式から求めた．

$$R_c = C_C / C_{all} \qquad R_s = C_S / C_{all}$$

（$C_C$：クローンを含むファイルを修正しているトランザクションの数，$C_S$：同一のクローンを含む複数のファイルを修正しているトランザクションの数，$C_{all}$：全トランザクション数）

$R_c$に比べ$R_s$が低いほど，同一クローンを含むファイル群を同時に修正しておらず，変更し忘れや見逃しが発生した可能性があることを示す．

図1に$R_c$と$R_s$の計算例を示す．この例において，クローンを含むファイルを修正しているトランザクション数$C_C$は，$T_1$, $T_3$, $T_4$, $T_5$, $T_6$の5回である．同一のクローンを含む複数のファイルを修正しているトランザクション数$C_S$は，$T_4$, $T_5$の2回である．全トランザクション数 $C_{all}$ は $T_1$～$T_6$ の6 回であるので，$R_c$=5/6，$R_s$=2/6と求められる．

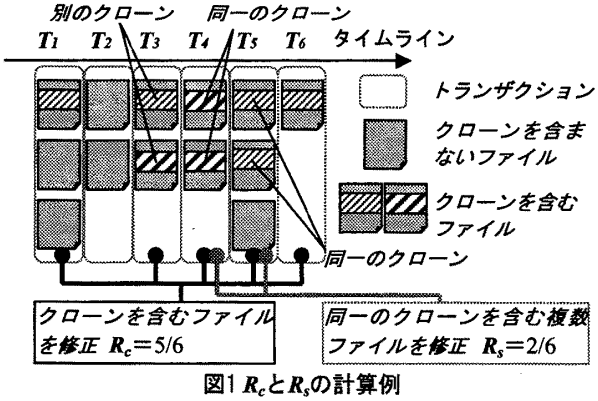


図1 $R_c$と$R_s$の計算例

**実験 2：保守作業の観察とインタビュー**

次に，保守開発者が不具合の修正を行うときのコード修正手順を観察した．開発者に通常の保守と同じ操作を行わせるため，被験者 3 人には，実験に関する説明を全く行わず，観察中であることも説明せずに，席の後ろより観察した．観察した操作は，不具合発見，修正，コンパイルである．観察終了後，観察中に気になった点や不明な点を，被験者それぞれに 30 分間インタビューを行った．またこれとは別に，プロジェクトの全開発メンバー8 人それぞれに対し，クローンについての知識，検出する方法，修正手順について，30 分間インタビューを行った．

### 2.2 結果および考察

**(実験 1 の結果に関して)**

表 1 に実験 1 の結果を示す．$R_c$, $R_s$ を算出すると，そ

れぞれ 79.3%, 9.7%であった．この結果より，ソフトウェア保守時にクローンを含むファイルを開発者が修正している割合は高いが，必ずしも同一のクローンを含む他のファイルを修正していないことがわかる．すなわち，(1)ソフトウェア保守時に，開発者はクローンの存在をそれほど意識していないといえる．

**表 1 クローンを含むファイルの修正状況の分析結果**

| 項目名 | 値 |
|---|---|
| 全トランザクション数( $C_{all}$ ) | 1890 |
| クローンを含むファイルを修正しているトランザクション数( $C_C$ ) | 1498<br>$R_c$ = 79.3% |
| 同一のクローンを含む複数のファイルを修正しているトランザクション数( $C_S$ ) | 183<br>$R_s$ = 9.7% |

**(実験 2 の結果に関して)**

観察の結果，開発者はソースコード修正時に修正する部分を特定するための手段として，文字列検索を多用していることがわかった．このとき，(2)プロジェクト経験が短い開発者は，検索する範囲や検索語の指定に必要なスキルが不足しているため，的確に検索できていない場合が多かった．修正作業プロセスを観察すると，(3)経験が短い開発者は，プログラムの修正を行った後に反映すべき範囲を検索し，必要であれば再修正を行っていた．一方，経験が長い開発者は，まず修正による影響範囲を確認した後で，修正方針を決定していた．そのため，影響する全ての部分の整合性が取れる修正方針を決定でき，再修正を発生させず，効率よく修正を行っていた．

またインタビューの結果，(4)プロジェクト経験が短い開発者ほど，ソフトウェア保守時にクローンの存在を意識していないことがわかった．クローンの存在を意識しない理由として，修正部分に対応するクローンが他ファイルに存在するかどうかわからない，存在を確認するためのコストが高い，(5)存在を確認できたとしても，修正すべきか判断できないなどの意見が挙がった．さらに，(6)検索語の指定に関しては，変数名が変更されている場合に特に難しくなるとの意見を得た．

## 3. クローン検出に求められる要件

本章では，2.2 節での分析結果より，我々がクローン検出手法を実際の開発現場に適用するために必要であると考える要件を挙げる．

**要件 1**（クローン検出の動機に関して）

2.2 節 (1), (4) より，クローンの検出に関する意識はそれほど高くない．そのため，開発プロセスに対して特別な手順の追加を必要とせず，クローンを自動的に検出できる必要がある．

**要件 2**（クローン検出方法に関して）

2.2 節 (2) より，クローンを的確に検出するためには，開発経験を要する．経験の度合に伴うパラメータの指定の相違に関わらず，誰でも同じ結果が得られるクローン検出方法が必要である．

また，2.2 節 (6) より，変数名が変更された場合でも，容易に検出できるような方法が必要である．

**要件 3**（検出したクローンの修正に関して）

2.2 節 (5) より，仮にクローンを自動的に検出できたとしても，開発者は修正すべきかどうかを判断できないことがある．そのため，検出結果表示の際には，検出したクローンコードだけではなく，修正すべきかどうか判断できるだけの情報を付加する必要がある．

また，2.2 節 (3) より，修正対象部分に対応するクローンを，開発者がプログラムを実際に修正する前に，修正による影響範囲として開発者に提示する必要がある．

現在，提案されている代表的なクローン検出ツールである CCFinderX[3] や ICCA[5]によってクローン検出を行うためには，ソースコード変更のための編集作業とは別に，クローン検出のためにそれらのツールを実行する手順が必要になる．ユーザの操作に特別な手順を必要とするため，要件 1 を満たさない．さらに，これらのツールは，いずれも検出されたクローンのコードを表示するのみであり，誰が，いつ，どのような状況でクローンを生成したのかというような，修正すべきかどうかを判断するために必要な情報を十分に表示しない．このため，要件 3 を満たさない．

## 4. 結論および今後の研究

本論文では，保守開発者がどのようにクローンを意識しているのかを明らかにするために，保守作業の観察とインタビューを行った．そして，レガシーソフトウェア開発者にクローンを意識した保守開発を行わせる上での問題点を明らかにした．さらにそれらの問題点に基づき，実際の開発現場に適用するために必要な要件を定義した．

今後の研究としては，得られた要件を満たす保守開発支援ツールの検討と実装を行う．そして，ツールをレガシーシステムに適用し，導入前と導入後でクローン修正漏れの割合やクローン増加率の推移を調べ，ツールの有効性を確認する予定である．

## 謝辞

本研究の一部は，科学研究費若手研究(スタートアップ) 18800024 の補助を受けた．また，一部は文部科学省「e-Society 基盤ソフトウェアの総合開発」および「次世代 IT 基盤構築のための研究開発」の成果に基づく．

## 参考文献

[1] Lague, B., Proulx, D., Mayrand, J., Merlo, E. M., and Hudepohl, J., "Assessing the Benefits of Incorporating Function Clone Detection in a Development Process," Proc. of IEEE Int'l Conf. on Software Maintenance, pp. 314-321, 1997.

[2] Bellon, S., Koschke, R., Antoniol, G., Krinke, J., and Merlo, E., "Comparison and Evaluation of Clone Detection Tools, " IEEE Trans. on Software Engineering, vol. 33, no. 9, pp. 577-591, 2007.

[3] http://www.ccfinder.net/ccfinderx.html

[4] Zimmermann, T., Weissgerber, P., Diehl, S., and Zeller, A., "Mining Version Histories to Guide Software Changes, " Proc. of IEEE Int'l Conf. on Software Engineering, pp. 563–572, 2004.

[5] http://sel.ist.osaka-u.ac.jp/icca/index.html