# ソフトウェア開発履歴の改変例の分析に向けて

林 晋平　佐伯 元司

ソフトウェア構成管理において，開発者はしばしば，ソフトウェア開発履歴に記録される変更の理解性や利用性の向上を目的として，履歴の改変を行う．本稿では，履歴をより適した形に改変するための支援手法や自動化手法の開発を目指し，履歴改変の実例を収集・分析する試みについて述べる．

## 1　はじめに

ソフトウェア構成管理において，開発者はソフトウェアに対して行った変更をリポジトリに蓄え，他の開発者と共有する．蓄えられた変更の記録はコミットと呼ばれ，版管理システムによってこれらが管理されている．コミットは開発者間で共有され，さまざまに利用されるため，適切に構成することが望ましい．例えば，我々は以下の理由によりコミットの粒度を適切に保ち，意図が混在した変更を構成しないことが望ましいとまとめている [3].

**再利用**　混在した変更の一部を他のブランチに適用する際には，その一部の抽出が必要であるため．

**破棄**　混在した変更の一部を棄却する際には，同様にその一部を抽出する必要がある．

**理解**　コミットログに複数の意味が記述されると，それらと差分のどの部分が対応しているかが不明確になってしまう．

複数の意図が混在した変更は，もつれた変更（tangled change） [4] と呼ばれ，変更の理解において悪影響を及ぼしたり，コミットを分析するソフトウェアリポジトリマイニング手法の結果に悪影響を及ぼしたりすることが知られている [5]．また，その検出・解消手法が提案され，既存履歴に対する調査が行われている．こういった試みを促進するためには，履歴改変の実例の収集が重要である．

本稿では，版管理システム Git を用いて開発されたプロジェクトの開発履歴から履歴の改変の実例を収集し，分析を行う試みについて述べる．

## 2　履歴の改変

本稿では，文献 [2] で扱っている履歴リファクタリングに従った履歴の改変を扱う．すなわち，変更列の書き換えであって，一連の変更の前後における管理対象のソフトウェア成果物の内容に書き換えの影響が及ばないようなものを扱う．こういった履歴の改変例は，履歴の変更内容を保った改変として有用と考える．

前述した履歴の改変は，版管理システム Git が管理するオブジェクトの状態としては，図 1 に示すような場合として扱える．図において，各円はコミットオブジェクトを，四角形はコミットが指すデータオブジェクトを，コミットオブジェクト間に張られた矢印は派生元のコミットへの参照を表す．ここでは，共通のコミット $a$ から，2 つの変更列，$a \to b \to c \to d$ と $a \to e \to f$ が派生している．ここで，コミット $d$ と $f$ は，いずれも共通のファイルツリーオブジェクトを指しており，これは，コミット $d$ と $f$ が保持している成果物の内容が等価であることを意味する．こ

---

* Towards Analyzing Rewriting of Revision History. This is an unrefereed paper. Copyrights belong to the Author(s).

Shinpei Hayashi, Saeki Motoshi, 東京工業大学大学院情報理工学研究科計算工学専攻, Dept. of Computer Science, Tokyo Institute of Technology.

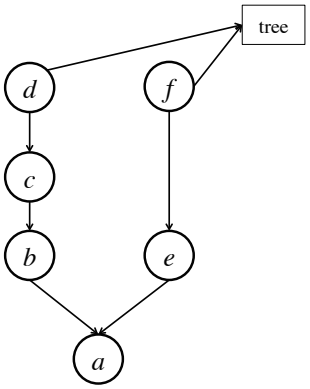


**図 1　対応する部分変更列の例**

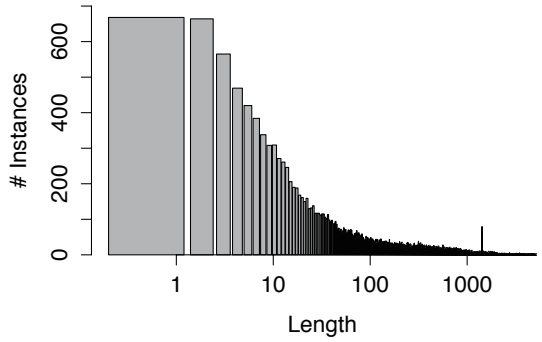


**図 2　分析結果**

れらの変更列は，共通の派生元コミットから同一の成果物の内容へと遷移する，すなわち，同一の変更内容を表しているため，一方を書き換えて他方を得た実例である可能性がある †1．

現実には，履歴書き換えは開発者のローカルリポジトリや少人数の開発者のみ注目する派生リポジトリで完結するかもしれない．その場合，書き換え元のコミット（図 1 では，$d$ と $f$ のうち導入時刻が古い方）の取得が困難となる．開発者のローカルリポジトリ上の振る舞いを監視したり [6]，リポジトリの継続的なマイニング [1] が必要になるかもしれない．

## 3　公開リポジトリに対する予備調査

現実の版管理リポジトリからどれだけ前述の変更列の対が得られるかを確かめるため，既存の公開リポジトリからの抽出を試みた．German らが提供している Linux カーネルの開発リポジトリのデータベース [1] に記載されているリポジトリ群のうち取得可能な 432 リポジトリを用意した．データベースはリポジトリ群における各コミットの指すファイルツリーオブジェクトのハッシュ値を保持しているため，同一リポジトリ内で同一のハッシュ値を持つ，異なるコミットオブジェクトを対象とした．簡単化のため，3 つ以上のコミットオブジェクトが同一のファイルツリーを指している場合は除外した．データベースからは，36290 個のコミットの対が得られ，ここから，該当コミットのオブジェクトが公開リポジトリから取得できなかった 16847 対を除いた 19443 対を対象とした．得られたコミットオブジェクトの対に対して，共通の派生元コミットを特定し，それに至るまでの部分変更列の長さを計測した．

得られた部分変更列の長さの分析結果を図 2 に示す．図の横軸は変更列の長さ（対数スケール），縦軸は観測された実例数を表しており，長さ 10000 以上の変更列については省略している．列が長くなるほど観測数が少なくなっている．有益な履歴改変例は，開発者が把握可能でかつ短かすぎないものに含まれると考えるが，公開リポジトリ上にも十分にそれらが含まれていることがわかる．今後，実際の部分履歴に含まれるコミットの内容や部分履歴間におけるコミットの対応関係を調査することにより，より有用なデータを構築していきたい．

## 4　むすび

本稿では，版管理システム Git を用いて開発されたプロジェクトの開発履歴から，履歴の改変の実例を収集し，その分析を行う試みについて述べた．

**謝辞** 本論文に対して様々なコメントを頂いた，東京工業大学の松田淳平氏に感謝する．

†1　もちろん，履歴の書き換え以外にも変更内容が共通となるコミットが生じる原因はある．

### 参 考 文 献

[1] German, D. M., Adams, B., and Hassan, A. E.: A Dataset of the Activity of the Git Super-repository of Linux in 2012, *Proc. IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, pp. 470–473.

[2] Hayashi, S., Omori, T., Zenmyo, T., Maruyama, K., and Saeki, M.: Refactoring Edit History of Source Code, *Proceedings of the 28th IEEE International Conference on Software Maintenance*, 2012, pp. 617–620.

[3] Hayashi, S. and Saeki, M.: Recording Finer-grained Software Evolution with IDE: An Annotation-based Approach, *Proc. Joint ERCIM Workshop on Software Evolution and International Workshop on Principles of Software Evolution (IWPSE-EVOL 2010)*, 2010, pp. 8–12.

[4] Herzig, K. and Zeller, A.: The Impact of Tangled Code Changes, *Proc. 10th International Workshop on Mining Software Repositories (MSR 2013)*, 2013, pp. 121–130.

[5] Nguyen, H. A., Nguyen, A. T., and Nguyen, T.: Filtering Noise in Mixed-Purpose Fixing Commits to Improve Defect Prediction and Localization, *Proc. 24th IEEE International Symposium on Software Reliability Engineering (ISSRE 2013)*, 2013, pp. 138–147.

[6] 松田淳平, 林晋平, 佐伯元司: 分散型版管理リポジトリでの作業履歴記録ツールの試作, 電子情報通信学会技術研究報告, Vol. 115, No. 153(2015), pp. 45–50.