コミット間の依存関係を考慮した部分コミット履歴の再構成支援に向けて

舟木 亮介[1,a)]　林 晋平[1,b)]　佐伯 元司[1,c)]

概要：本稿では，バージョン管理システムにおける開発者の部分コミット履歴の再構成を支援する手法およびそのツールを提案する．コミット間の依存関係をテキストレベルやビルドレベルで事前に解析し，コミットの取り込みの失敗を防ぐ．これによりバージョン管理システムの高度な知識や試行錯誤を開発者に要求する再構成のプロセスの改善を試みる．

1. はじめに

ソフトウェアの開発や保守の工程において，Git[1] などのバージョン管理システムが広く活用されている．バージョン管理を行うことによって，開発者が成果物のリリース管理をしたり，リポジトリをフォークして独自に機能を追加するなどの拡張が容易になる．

バージョン管理システムを利用する上で，開発者はしばしば，ブランチをまたいでコミットを移動させる [2]．具体的には，リリース管理において，リリースする機能を分割して製品のバリエーションを作る際に，新たなブランチを作成して必要なコミットだけを取り込む．また，独自拡張においては，フォーク元のリポジトリの更新を追従するために，独自拡張部分と競合しないコミットだけを取り込む．このような履歴の再構築を行うために，開発者は取り込みが成功するコミットを試行錯誤して探す必要がある．

ある OSS をフォークして独自拡張している開発者を考える．開発者は，フォーク元でリリースされた機能の一部を，自らが独自拡張を行っているプロジェクトに取り込みたいとする．開発者はまず，フォーク元のリリース履歴から，自らが取り込みたいコミットを特定しようとするが，コミットメッセージのみを元に特定することは困難である．多くのプロジェクトはイシューベースで機能の追加やバグ修正を行っている [3]．コミットメッセージを見るよりも，イシューを参照した方が変更内容を把握するのに役立つはずである．そこで開発者はまずイシューを参照してから，取り込みたい機能に関連するイシューに紐付いたコミットを，取り込み対象のコミットとする．また，開発者は，取り込みを成功させるために，取り込み対象のコミットに依存するコミットを再帰的に探し，取り込み対象に加えていく必要もある．これらの作業は繁雑でミスが起きやすいため，ツールによる支援が求められる．

本稿の目的は，部分コミット履歴の再構成における開発者の負担を減らすことである．開発者は煩雑でミスの起きやすいタスクを繰り返し要求されるため，この負担をツールにより支援する．ツールが達成すべき具体的な課題は，取り込みたいコミットに関連するコミットを特定し，取り込みの失敗を事前に防ぐことである．

2. 提案手法

本手法は，Git で管理された Java プロジェクトが対象である．ツールの入力は，再構成対象のコミット履歴，及び注目するイシュー番号である．出力は，注目するイシュー番号に紐付いたコミットと，それに関連するコミットを含んだ部分コミット履歴である．ツールのプロセスは，抽出，依存解析，再構成の 3 段階からなる．抽出では，取り込むイシュー番号から，予め特定しておいたイシューとコミットの対応関係にも基づき，取り込むコミット集合を特定する．次の依存解析では，テキストレベルとビルドレベルで，取り込みたいコミットに関連するコミットを取得する．最後の再構成では，新たにブランチを作成し，前段階で得られたコミットの集合をオリジナルの履歴順に逐次取り込んでいく．ここで作成されたブランチが，部分コミット履歴の再構成結果となる．

提案手法では，テキストおよびビルドの 2 種類の依存関係を扱う．あるコミット A で変更された箇所と，同じ範囲

[1] 東京工業大学
Tokyo Institute of Technology
[a)] rfunaki@se.cs.titech.ac.jp
[b)] hayashi@se.cs.titech.ac.jp
[c)] saeki@se.cs.titech.ac.jp

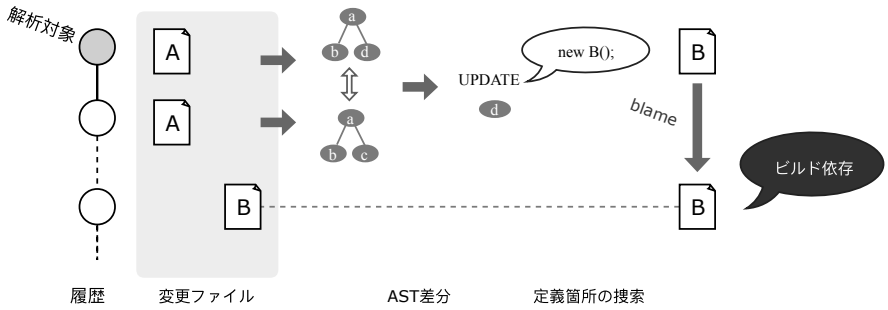

図 1 ビルド依存解析

を変更している A より過去のコミット B が存在したとき，コミット A だけを取り込むことはできない．なぜならバージョン管理システムは，テキストの行単位で変更を認識しテキスト競合を解決しようとするが，必ずしも成功するとは限らない [4][5]．本稿では，コミット A の取り込みの失敗原因となったコミット B を，A の「テキスト依存」と表現する．テキスト依存は，変更範囲が重複するコミットのペアを探索し求める．

また，開発者は取り込み後のスナップショットにおいて，ビルドが成功することを保証したい．そのために，実際にビルドを実行し，失敗するならその原因を特定し逐一やり直す必要がある．テキスト依存コミットをすべて特定し，取り込みに成功したとしても，テキストレベルでの解決ができたに過ぎず，ビルドが成功する保証はない [6]．本稿では，コミット A を取り込む際にコミット B を取り込まなかったことによりビルドに失敗する場合，コミット A はコミット B に「ビルド依存」すると表現する．ビルド依存を全て正確に求めることは，解析に時間的コストがかかる．ツールによる支援では使用性と解析時間を考慮する必要がある．

そこで提案手法では，ビルド依存を求める際に，識別子の定義箇所と使用箇所のペアに着目する．概要を図 1 に示す．まず，注目するコミットで変更された識別子を，直前のコミットとのソースコードの AST 差分から求める．次に，その識別子の定義箇所を探し，該当箇所を変更する過去のコミットを特定する．このコミットをビルド依存とする．もちろん全てのビルド依存を網羅できるわけではないが，ツールが開発中に繰り返し使用されることを考えると，現実的な時間で解析が終わることを重視する方針を取ることにした．

3. 実装

提案手法を自動化するツールを実装中である．抽出は，コミットメッセージに記載されたイシュー番号から，コミットとイシューを対応関係を構築して実現した．テキスト依存解析には，CSlicer[2] を利用した．ビルド依存解析は，AST の構築に Eclipse JDT[7] を用い，AST の差分を GumTree[8] で求めることによって実現した．使用箇所から定義箇所の捜索には，JDT の機能を用いた．再構成でのコミットの取り込みには，Git の cherry-pick コマンドを利用した．

4. 今後の課題

本稿では，解析コストを重視しており，全てのビルド依存を網羅するべきという立場を取らないが，開発者が強く支援を求めるものを精査し，不十分な点があれば実装を改良する必要がある．そのために，現実装を実際のプロジェクトに適用して，結果の分析を進める．

また，本稿では取り上げなかった依存関係の考慮も行いたい．例えば，再構成後にビルドが通ったとしても，テストが通らないような場合，これを「テスト依存」として考えることもできる．開発者にとって，ビルドと同様テストが通る保証をすることは大きなコストである．もちろんツールによる支援が必要だと考える．

参考文献

[1] Git SCM, (online), available from ⟨https://git-scm.com⟩ (accessed 2018-07-25).
[2] Li, Y., Zhu, C., Rubin, J. and Chechik, M.: Semantic Slicing of Software Version Histories, *IEEE Trans. Softw. Eng.*, Vol. 44, No. 2, pp. 182–201 (2018).
[3] Wu, R., Zhang, H., Kim, S. and Cheung, S.: ReLink: Recovering links between bugs and changes, *Proc. FSE*, pp. 15–25 (2011).
[4] Mens, T.: A State-of-the-Art Survey on Software Merging, *IEEE Trans. Softw. Eng.*, Vol. 28, No. 5, pp. 449–462 (2002).
[5] Yuzuki, R., Hata, H. and Matsumoto, K.: How we resolve conflict: an empirical study of method-level conflict resolution, *Proc. SWAN*, pp. 21–24 (2015).
[6] Brun, Y., Holmes, R., Ernst, M. D. and Notkin, D.: Proactive detection of collaboration conflicts, *Proc. FSE*, pp. 168–178 (2011).
[7] Eclipse Java development tools, (online), available from ⟨https://www.eclipse.org/jdt/⟩ (accessed 2018-07-25).
[8] Falleri, J., Morandat, F., Blanc, X., Martinez, M. and Monperrus, M.: Fine-grained and accurate source code differencing, *Proc. ASE*, pp. 313–324 (2014).