ソースコード編集履歴の不吉な臭いの検出に向けて

星野 大樹[1]　林 晋平[1]　佐伯 元司[1]

概要：本稿では，ソースコード編集履歴における不吉な臭いとその検出法について議論する．ソースコード編集履歴の理解性や利用性を向上させるために編集履歴をリファクタリングする手法があるが，どのような履歴にどのようなリファクタリングが必要なのかは明確でない．現在我々が行っているソフトウェア開発プロジェクトにおいて蓄積された編集履歴の分析を行い，リファクタリングが必要となる履歴の特徴を「履歴の臭い」とし，臭いを検出する試みについて述べる．

1. はじめに

ソースコードの編集の記録 (編集履歴) はソフトウェア開発の中で利用されており，開発の効率化や理解に役立っている．例えば開発者はコードエディタが保持している編集履歴を利用し，これまでの編集操作を取り消す Undo 操作や，Undo した操作を適用し直す Redo 操作によりさまざまな試行錯誤を行ったり，編集内容の確認を行ったりしている．また，版管理リポジトリが蓄えるソースコードの改版の情報も，開発者が行った編集結果に基づくものであり，編集履歴を反映している．そこで，編集履歴の理解や利用がより容易となるよう，履歴全体が表している効果を変えないまま履歴の内容を再構成する，編集履歴のリファクタリング手法 [1] が提案されている．

しかし，既存手法では具体的にどのような編集履歴をどのようにリファクタリングすべきであるかが明確でないという点が課題として残っている．この課題に対し本稿では，実際の開発プロジェクトにおいて蓄積された編集履歴を分析することで，編集履歴をリファクタリングすべき兆候を発見し，これを「履歴の臭い」として定義する．また，臭いの検出のためのメトリクスについても議論する．

2. 編集履歴の分析

開発者 4 名，開発期間 2 ヶ月である Android アプリケーションの開発プロジェクトにおいて，各開発者の開発環境に編集履歴収集ツールである OperationRecorder [3] を組み込み，2 ヶ月間の編集履歴を収集した．プログラムの最終的なステップ数は約 3000 ステップであった．また，このプロジェクトは Git を用いて版管理されており，総コミット数は 119 であった．

はじめに，119 のコミット各々について，コミットメッセージとコミット内容に意味的な食い違いが存在しないか調査した．ここで意味的な食い違いが存在した場合，前回コミットが行われた後から今回のコミットが行われるまでの編集履歴にどのような特徴があるか分析し，編集履歴の臭いやその検出メトリクスの設定に活用した．ここで編集履歴とは，特定のソースコードに対する連続する字句の追加や削除，あるいは置換が行われた際に，その編集時刻，編集先ファイル，編集開始オフセット，編集文字列が記録されたデータセットの集合である．

実際に意味的な食い違いがあった例を**図 1** に示す．この例では，「通知を出さないよう条件を追加した」というコミットメッセージに対して，1:46:31–1:49:52 内の連続した時間内で，通知を出さないよう DropboxOperator.java において if 文が追加されている．その後約 4 分の時間を隔て，1:53:35–2:02:07 内の連続した時間内で MainActivity.java, PagerActivity.java, strings.xml に対してコメントアウトの消去や改行の消去，インデントの修正が行われている．後者の編集はプログラムの振る舞いに影響しないが，コミットメッセージの内容とは無関係であり，前者の条件追加の編集と同時にコミットされてしまっている．これにより，本来行われるべき編集は前者の DropboxOperator.java に対する 12 行だけであるにも関わらず，差分は 170 行以上に渡っており，その理解性を低下させている．すなわち，編集操作群に複数の意図が混在している場合，編集意図ごとに編集操作群を切り分けるリファクタリングが必要であり，単一意図の編集操作群ごとに版管理リポジトリへ反映させるべきだと考える．

3. 臭いの候補

前章で挙げた開発プロジェクトの分析や，関連研究 [2]

[1] 東京工業大学大学院情報理工学研究科計算工学専攻

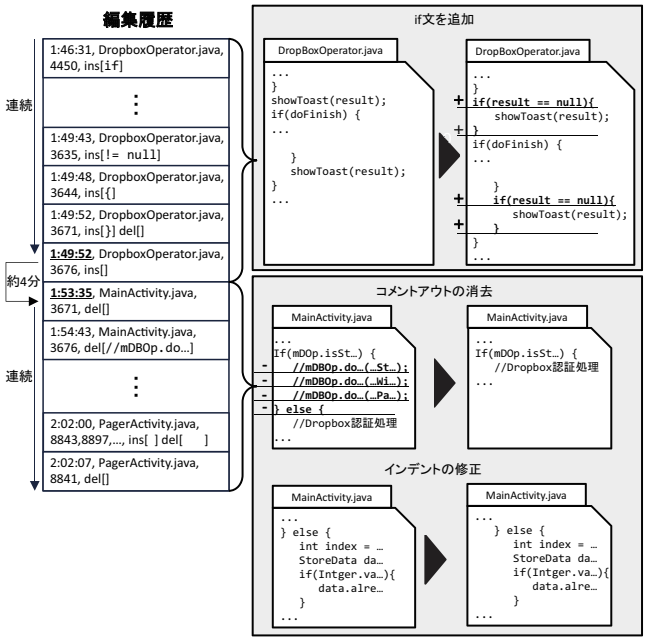

図 1 意味に食い違いのある例

の調査などにより，臭いの候補を 3 つ選出した．

Tangled Changes：図 1 の例のように，編集操作群に複数の編集意図が混在している状況は，版管理リポジトリへの反映後に差分の理解性を低下させるため，臭いの候補となる．この臭いは，編集意図ごとに編集操作群を分けるよう編集履歴リファクタリングを行うことで解消できる．

Shotgun Surgery：編集履歴上に同一意図の編集操作が点在している状況は，版管理リポジトリへの反映後に差分の理解性を低下させるため，臭いの候補となる．この臭いは，点在した編集操作をまとめるよう編集履歴リファクタリングを行うことで解消できる．

Scattered Logging：Logging の編集操作が編集履歴に点在している状況は，各 Logging の Undo が労力となるため，臭いの候補となる．この臭いは，点在した Logging をまとめるよう編集履歴リファクタリングを行うことで解消できる．

4. 検出に向けて

図 1 の例では，編集意図の違いを示す特徴として，コミットメッセージに関わる編集とその他の編集の間に存在した約 4 分の時間的距離や，編集先ファイルの違いが見受けられた．

こうした開発プロジェクトの分析結果や，コミットされた内容を意図毎に分割する既存手法 [4] の調査から，編集操作間の関連度を測るメトリクスを 6 つ設定した．

File Distance：編集先オフセットが近いほど関連度が高い

Package Distance：編集先ファイルの距離が近いほど関連度が高い

Call Graph：編集先メソッドのコールグラフ上での距離が近いほど関連度が高い

Change Coupling：編集時間軸上で頻繁に共起する編集操作は関連度が高い

Data Dependency：同一の変数を扱う編集操作は関連度が高い

Time Interval：編集時間軸上の距離が近いほど関連度が高い

これらにより各編集操作の編集意図が近いかどうか判別し，Tangled Changes や Shotgun Surgery の検出に利用する．

編集履歴という細粒度な情報を扱う本手法では，Herzig らの手法 [4] のように，ソースコードの修正内容だけでなく，編集操作間に存在する時間的距離を扱える．Time Interval のメトリクスを用いることが可能であり，図 1 で述べた約 4 分の時間的距離を関連度の低さに結びつけることができる．また，こういった時間的距離については Omori ら [5] も既に述べているが，他のメトリクスと組み合わせることにより，より有益な指標となると考える．

5. 今後の課題

今後の課題として，更なる臭いの候補の選出や，各検出メトリクスに対する重み付け，また，編集履歴リファクタリングを実現する既存の Eclipse プラグインである Historef [1] の拡張として臭いの自動検出を実装し，精度の評価を行うことが挙げられる．

参考文献

[1] Hayashi, S., Omori, T., Zenmyo, T., Maruyama, K. and Saeki, M.: Refactoring Edit History of Source Code, *Proc. ICSM*, pp. 617–620 (2010).

[2] Hayashi, S. and Saeki, M.: Recording Finer-Grained Software Evolution with IDE: An Annotation-Based Approach, *Proc. IWPSE-EVOL*, pp. 8–12 (2010).

[3] Omori, T. and Maruyama, K.: An Editing-operation Replayer with Highlights Supporting Investigation of Program Modifications, *Proc. IWPSE-EVOL*, pp. 101–105 (2011).

[4] Herzig, K. and Zeller, A.: The Impact of Tangled Code Changes, *Proc. MSR*, pp. 121–130 (2013).

[5] Omori, T. and Maruyama, K.: Identifying Stagnation Periods in Software Evolution by Replaying Editing Operations, *Proc. APSEC*, pp. 389–396 (2009).