コーディング規約改定によるコードレビュー中の軽微な変更の分析

上田 裕己[1,a)]　伊原 彰紀[1,b)]　平尾 俊貴[1,c)]　石尾 隆[1,d)]　松本 健一[1,e)]

概要：コードレビューにおいて検証者は保守・運用作業を容易にするために，ソースコードに改行や変数名の変更などの軽微な変更 (以下 Small Change) を指示することがある．一部のソフトウェア開発プロジェクトでは，Small Change の発生を防ぐためにコーディング規約を設けている．本論文では，OSS のコーディング規約の改定前後における Small Change の発生頻度の変化を Qt プロジェクト対象に分析した．分析の結果，コーディング規約を改定後，コードレビュー前のパッチにはスペースやタブの変更の発生頻度が減少した．しかし，コードレビュー後の変更からは記号の変更などの発生頻度が上昇したことを確認した．

1. はじめに

コードレビューは，開発者が機能追加，欠陥修正のために投稿したソースコードの変更に対して，検証者が欠陥の混入有無などを確認する作業である．コードレビューは，ソフトウェア開発にかかる工数の約 50%を占めており [1]，その要因の一つは，一度の検証で全ての欠陥を解決することが困難であり，何度も検証する必要があるためである．

本論文は，ソースコードの動作に直接影響を与えることはないが将来の保守作業を容易にするための改行，空白の追加などの軽微な変更（Small Change）に着目する．Small Change は，開発者が容易に修正可能な問題であることが多く，コードレビュー依頼（以下，投稿）前に開発者が自身で Small Change を実施しておくことが期待される．

一部のソフトウェア開発プロジェクトは，コードレビューにおける Small Change の発生を未然に防ぐために，コーディング規約を定めている．しかし，オープンソースソフトウェア (OSS) 開発のような不特定多数の開発者が参加するプロジェクトでは，開発者がコーディング規約に違反したソースコードを提出することが多く，コードレビューによる Small Change の特定が不可欠となっている [2]．

本論文では，コーディング規約と Small Change の関係を調査するために，コーディング規約改定前後での Small Change の発生頻度の違いを分析する．Qt プロジェクトでは，2012 年 3 月にそれ以前に比べ空白や命名規則などを対象に詳細なコーディング規約が定められている．

事前分析として，コードレビュー管理システム Gerrit の Qt プロジェクトに投稿されたソースコードについて，投稿後にコードレビューを通して再修正が行われたソースコードの変更に含む Small Change を目視で調査した結果判明した 5 種類の Small Change の検出を自動化し，全体のデータから Small Change の発生頻度を調査した．

2. Small Change の事前分析

コードレビュープロセスにおいて，改変前と改変後のソースコードの差分ファイルをパッチ (Patch) と呼び，Patch を作成した開発者をパッチ作成者と呼ぶ [3], [4]．昨今の OSS 開発ではコードレビューの一連の作業を管理するためにコードレビュー管理システムが利用されている [5]．分析対象である Qt プロジェクトでは，Gerrit Code Review を利用している．

(1) パッチ作成者は，パッチを Gerrit Code Review へ投稿する．投稿されたパッチを $Patch_1$ とする．
(2) 検証者は，$Patch_1$ の品質を評価して，パッチ作成者にフィードバックする．もし $Patch_1$ に修正すべき問題が存在する場合，再修正を要求する．
(3) パッチの再修正が必要な場合，パッチ作成者は $Patch_1$ を修正し，$Patch_2$ として投稿する．n 回目に再修正されたパッチを，$Patch_n$ とする．
(4) プロジェクトはパッチ作成者による再修正が完了した $Patch_n$ を，プロジェクトのリポジトリへ統合する．

Small Chagne の種類を明らかにするために Qt プロジェクトに投稿された 69,325 件のパッチセット [6] からランダ

[1] 奈良先端科学技術大学院大学，情報科学研究科
[a)] ueda.yuki.un7@is.naist.jp
[b)] akinori-i@is.naist.jp
[c)] hirao.toshiki.ho7@is.naist.jp
[d)] ishio@is.naist.jp
[e)] matumoto@is.naist.jp

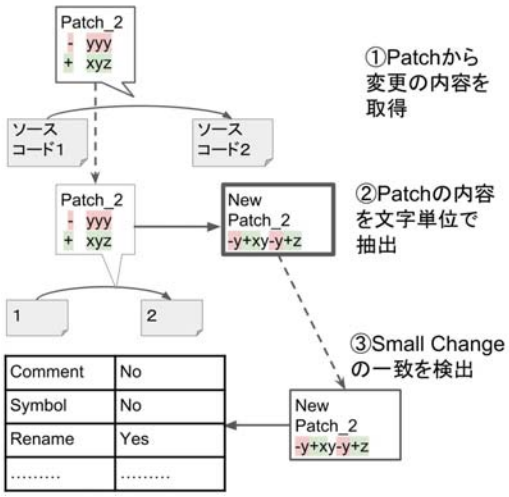

図 1　Small Change の検出手順

ムに 382 件（信頼区間 95%，信頼度 ±5%）を抽出し，検証者の具体的な変更方法の指示内容とその変更から各パッチセットに対して Small Change を含むか目視で検出する．事前分析の結果検出した Small Change を表 1 に示す．

3. Small Change の検出

事前分析によって見つけた 5 種類の Small Change を，Gerrit の Qt プロジェクトに投稿された 69,325 件のパッチセットを対象に自動的に検出する．検出手順を図 1 に示す．

(1) **Patch から行単位の変更の取得．** Patch に記載されている行単位の変更の差分を取得する．

(2) **文字単位の変更の検出．** google-diff-match-patch[*1] を利用して，行単位での変更の差分を文字単位の変更の差分に変換する．

(3) **Small Change の検出．** 行単位の差分または文字単位の差分が各 Small Change に定めた条件にマッチするか否かを確認し，各 Patch に対して，各 Small Change の有無を検出する．

5 種類の Small Change の内容と検出のルールを示す．

NewLine: 改行のみ追加，削除された Patch を検出する．改行前後の文字が変更された場合は検出しない．

Renamed: 英数字文字列のみ追加，削除を検出する．ただし，変数が関数に変わるような変更はプログラムの振る舞いが変わる可能性の高いため，前後の記号が変更された場合は検出しない．

SpaceOrTab: スペース，タブの追加，削除を検出する．空白前後に文字の変更があれば検出しない．変更行に対して正規表現`^(\s|\t)+$`に一致を検出する．

Symbol: 記号のみの追加，削除を検出する．変更行に対して正規表現`^([!-/:-@[-‘{-~])+$`の一致を検出する．

UpperLower: 英字文字列の大文字，小文字を変換した変更を検出する．

[*1] https://code.google.com/p/google-diff-match-patch/

4. ケーススタディ

3 章の検出ルールを適用して，投稿されたパッチに発生する Small Change の発生頻度を調査する．

4.1 データセット

本論文では，Qt プロジェクトで最もパッチセット数の多いサブプロジェクト t/ tbase を対象とする． tbase はパッチセット数 19,393，開発者数 492 人，開発期間は 2008 年から 2013 年のプロジェクトである．

2013 年までに Gerrit Code Review へ投稿された 19,393 件のパッチセットを対象に分析した結果，4433 件 (約 4.3%) が Small Change を含んでいた．また，2011 年以前は 2011 年以後に比べて，Small Change が多い．

2011 年以後に Small Change を含むパッチセットの割合が減少した要因の一つとして，Qt プロジェクトでは，2012 年 3 月にコーディング規約[*2]が改定され，パッチ開発者がコーディング規約を注視して開発したため，Small Change の指摘頻度が低くなったことが考えられる．表 2 は，改定前後のコーディング規約に各 Small Change が記載されているか否か確認，また改定前後の Small Change の発生頻度を示した．コーディング規約の改定前後における Small Change の発生割合を分析するために，最初に投稿された $Patch_1$ と， 2 回目以降に投稿された $Patch_{2-n}$ に分類し，プロジェクトに投稿された各 $Patch$ に含まれる SmallChange の割合を分析する．

[分析 1（$Patch_1$）] コーディング規約の改定後にパッチ作成者がプロジェクトに対してどのような Small Change のパッチを投稿するのか調査する．

[分析 2（$Patch_{2-n}$）] コーディング規約の改定後に検証者が開発者に対してどのような Small Change を提案するのか調査する．

4.2 分析 1: パッチ作成者によって投稿された $Patch_1$ に含まれる Small Change

[アプローチ]

本分析では，コーディング規約改定前後の両方の時期に参加するパッチ作成者が改定後のコーディング規約に従っていないプロジェクトに対して Small Change を投稿しているか否かを分析する．

[結果] 表 3 は，コーディング規約改定前後に $Patch_1$ に含まれた Small Change の発生頻度の中央値とマン・ホイットニーの U 検定の結果を示す．また図 3 に分布を示す．

改定後にコーディング規約に記述された SpaceOrTab は減少しており，改定前後で有意な差を確認した．また，

[*2] https://www.ics.com/designpatterns/book/style.html

表 1　5 種類の Small Change の概要と変更例

変更	概要	変更前例	変更後例
NewLine	改行の追加または削除	int foo;int bar;	int foo; int bar;
Renamed	変数，関数名の変更	int foo;	int bar;
SpaceOrTab	スペースまたはタブの追加，削除	print(”HelloWorld”);	print(”Hello␣World”);
Symbol	記号の追加，削除	int foo	int foo**;**
UpperLower	大文字から小文字またはその逆への変更	int foo;	int FOO;

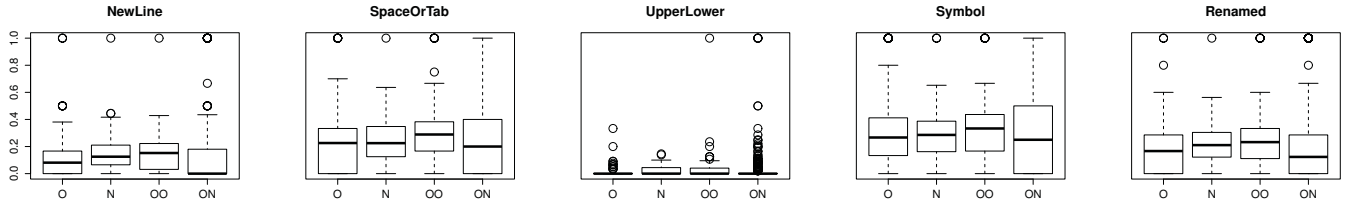

図 2　$Patch_1$ における Small Change の発生割合
（共通の開発者：改定前 O，改定後 N，特有の開発者：改定前 OO，改定後 ON）

表 2　各 Small Change のコーディング規約による定義状況

	定義の有無		発生割合	
Small Change	改定前	改定後	改定前	改定後
NewLine	未定義	**定義**	0.20	0.09
Renamed	**定義**	**定義**	0.31	0.13
SpaceOrTab	未定義	**定義**	0.35	0.15
Symbol	未定義	**定義**	0.33	0.14
UpperLower	**定義**	**定義**	0.04	0.03

表 3　コーディング規約改定前後の $Patch_1$ に含まれる Small Change の発生割合（p 値は 0.05 未満）

	共通の開発者		特有の開発者	
変更	改定前	改定後	改定前	改定後
NewLine	0.08	0.12	0.15	0.00
Renamed	0.17	0.21	0.23	0.12
SpaceOrTab	0.23	0.22	0.29	0.20
Symbol	0.27	0.29	0.33	0.25
UpperLower	0.00	0.00	0.00	0.00

表 4　コーディング規約改定前後の $Patch_{2-n}$ に含まれる Small Change の発生割合 (共通の開発者のみ p 値が 0.05 未満)

	共通の開発者		特有の開発者	
Small Change	改定前	改定後	改定前	改定後
NewLine	0.00	0.09	0.03	0.00
Renamed	0.06	0.14	0.06	0.07
SpaceOrTab	0.16	0.18	0.10	0.16
Symbol	0.13	0.16	0.09	0.10
UpperLower	0.00	0.00	0.00	0.00

UpperLower の発生割合に改定前後で有意な差を確認した．

4.3　分析 2: $Patch_{2-n}$ に含まれる Small Change

[アプローチ] 分析 1 と同様に，(1) コーディング規約改定前後の両方に参加するパッチ作成者に検証者が改定されたコーディング規約に従った指摘をしているか否か分析する．(2) コーディング規約の改定によって，Small Change を指摘されたコードも増加しているか否かを分析する．

[結果] 表 4 は，コーディング規約改定前後に $Patch_{2-n}$ に含まれた Small Change の発生割合の中央値を示す．また図 3 に分布を示す．

コーディング規約の改定後に記述された Symbol の発生割合は増加しており，改定前後で有意な差を確認した，コーディング規約の改定後に Renamed の発生割合は増加しており，改定前後で有意な差を確認した．コーディング規約の改定後の SpaceOrTab の発生割合は t-base で増加し改定前後で有意な差を確認した．

5.　考察

Qt プロジェクトのコーディング規約の改定後に新たに定義された SpaceOrTab，Symbol について考察する．**SpaceOrTab** は $Patch_1$ において改定後に発生割合が低くなっている．スペースやタブの変更は比較的行いやすいため，1 度のパッチ投稿で解決されていることが予想できる．**Symbol** は $Patch_{2-n}$ において発生頻度が増加した．検証者が改定後に定義された Symbol の項目に注意を払ってコードレビューしたため，指摘回数が多くなったことが考えられる．**Renamed** は $Patch_{2-n}$ において発生割合が増加した．コーディング規約の改定により，検証者が改定前よりも命名規則を意識したことが予想される．

いくつかの Small Change はコーディング規約の改定により，検証者が指摘するようになったと考えられる．コーディング規約の改定はパッチ作成者が参照するだけでなく，検証者が正確なコードレビューを実施するためのガイドラインとして役割を果たしていると考えられる．

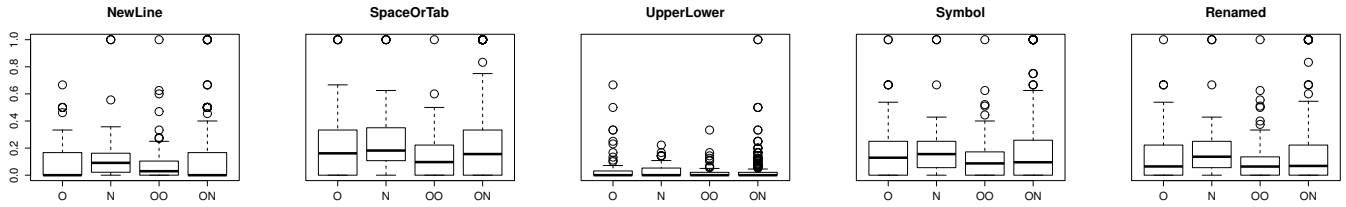

図 3　$Patch_{2-n}$ の投稿における Small Change の発生割合

6. 妥当性の脅威

本論文で使用した検出ルールは，1 行単位での変更を分析しているため，Small Change をソースコード全体が持つ意味を考慮して検出できない．1 行の変更だけでなく，変更されたソースコード全体の意味を考慮することで，正確な Small Change の検出を行うことが今後の課題である．

全ての規約項目を網羅的に検出することが難しいため，正規表現を用いて検出した．今後は，構文解析で網羅的に Small Change を検出できる手法を提案する．

本論文では，大規模 OSS プロジェクトである Qt プロジェクトのみを使用した．今後，他の OSS プロジェクトを対象に分析することで，より汎用性の高い分析結果を得ることができると考えられる．

7. 関連研究

Smit ら [7] は Subversion で管理される Java のプロジェクトのコーディング規則違反の検出に CheckStyle を利用しマジックナンバーや括弧の有無が問題となっていたことを確認した．現在，ソースコードをコーディング規約に定義された形式に変換するツールは多く存在し，命名規則の違反等を検出している．Allamanis[8] らはコーディング規約を修正するツールを作成している．本論文もツールとして Small Change の修正に応用が可能である．

Czerwonka ら [9] は，コードレビューで議論する内容で，15%はパッチの機能的問題に関する内容であることを確認した．一方で，検証者は Small Change に関しても議論することがある [10]．従来研究では，Small Change について検証者が議論に費やすレビュー時間，また，Small Change の優先度について分析されていない．

8. まとめと今後の展望

分析 1，2 の結果より，いくつかの Small Change はコーディング規約の改定によって $Patch_1$，$Patch_{2-n}$ で発生割合が変化したことを確認した．コーディング規約の改定が検証者の正確なコードレビューを実施するためのガイドラインとしての役割を果たすことが考えられる．今後は，Small Change を指摘されたパッチ作成者及び指摘する検証者の成長の分析を進めることで，Small Change が発生する理由などを明らかにする．

謝辞　本研究は，JSPS 科研費（JP17J09333，17H00731），及び，テレコム先端技術研究支援センター SCAT 研究の助成を受けたものです．

参考文献

[1] Alberts, D. S.: The Economics of Software Quality Assurance, *Proceedings National Computer Conference and Exposition*, ACM, pp. 433–442 (1976).

[2] Kochhar, P. S., Bissyande, T. F., Lo, D. and Jiang, L.: An Empirical Study of Adoption of Software Testing in Open Source Projects, *Proceedings of International Conference on Quality Software*, pp. 103–112 (2013).

[3] Thongtanunam, P., McIntosh, S., Hassan, A. E. and Iida, H.: Revisiting Code Ownership and its Relationship with Software Quality in the Scope of Modern Code Review, *Proceedings of the 38th International Conference on Software Engineering*, pp. 1039–1050 (2016).

[4] Thongtanunam, P., McIntosh, S., Hassan, A. E. and Iida, H.: Review Participation in Modern Code Review: An Empirical Study of the Android, Qt, and OpenStack Projects, *Empirical Software Engineering*, Vol. 22, No. 2, pp. 768–817 (2017).

[5] Rigby, P. C. and Bird, C.: Convergent Contemporary Software Peer Review Practices, *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*, pp. 202–212 (2013).

[6] Thongtanunam, P., McIntosh, S., Hassan, A. E. and Iida, H.: Investigating Code Review Practices in Defective Files: An Empirical Study of the Qt System, *Proceedings of the 12th Working Conference on Mining Software Repositories*, pp. 168–179 (2015).

[7] Smit, M., Gergel, B., Hoover, H. J. and Stroulia, E.: Code convention adherence in evolving software, *Proceedings of the 27th IEEE International Conference on Software Maintenance*, IEEE, pp. 504–507 (2011).

[8] Allamanis, M., Barr, E. T., Bird, C. and Sutton, C.: Learning natural coding conventions, *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 281–293 (2014).

[9] Czerwonka, J., Greiler, M. and Tilford, J.: Code Reviews Do Not Find Bugs: How the Current Code Review Best Practice Slows Us Down, *Proceedings of the 37th International Conference on Software Engineering*, pp. 27–28 (2015).

[10] Rigby, P. C. and Storey, M.-A.: Understanding Broadcast Based Peer Review on Open Source Software Projects, *Proceedings of the 33rd International Conference on Software Engineering*, pp. 541–550 (2011).