Vol.2021-SE-207 No.23
2021/3/2

コーディング規約違反解決までのソースコード特徴量の分析

南 雄太[1]　福元 春輝[1]　伊原 彰紀[1,a)]

概要：ソースコードの可読性と保守性を向上するために，ソフトウェア開発プロジェクトはコーディング規約を定め，静的解析ツールを用いて規約違反している実装を検出している．しかし，静的解析ツールが規約違反として検知するソースコード断片（警告箇所）の中には，開発者にとって修正の必要がない警告（false-positive）も多く，静的解析ツールの導入に消極的なプロジェクトが存在する．従来研究では，false-positive を削減するために警告の優先順位付けが行われているが，将来的に修正される警告であったとしても false-positive と定義されることが多い．ソフトウェア開発では，早急に解決する必要のない，後日に解決する警告箇所も多く存在する．本論文は，警告解決の端緒となるプログラムの特徴を理解するために，規約違反を検知してから解決されるまでのソースコードの進化過程を追跡し，解決が必要となるソースコードの変更を分析する．ケーススタディとして airflow プロジェクトを対象に分析した結果，規約違反が初めて開発者に認識されたソースコードの特徴と，規約違反を解決したソースコードの特徴には統計的に有意な差を確認し，特にソースコードの累積総変更行数と累積ファイル変更回数が規約違反を解決する時期と関係があることを確認した．

An Analysis of Source Code Feature Evolution until Solving Coding Standard Violation

1. はじめに

ソフトウェア開発プロジェクトではソースコードの可読性と保守性を向上するために，開発者がソースコード実装において守るべきルールとしてコーディング規約を制定する．コーディング規約には，Java コーディング標準 *1，PEP8 *2など，ソースコード実装の記述方式に関する規約や，MISRA *3，CERT コーディングスタンダード *4などのソフトウェアのセキュリティに関する規約がある．特に，オープンソースソフトウェア（OSS）開発では，不特定多数の開発者が貢献するためコーディング規約に遵守し実装することが求められる．コーディング規約に遵守しているか否かを容易に確認するために，多くのプロジェクトでは静的解析ツールを使用している．頻繁に使用される静的解析ツールには，Java 言語のための Checkstyle *5，Python 言語のための Pylint*6，C 言語のための CX-Checker [1] などがある．

静的解析ツールはプロジェクトの実装ルールを開発者間で共有するために有用である一方で，ツールが検出した規約違反を解決せずにコミットする開発者が多い [2] [3] [4]．このような開発者によって解決されない警告は false-positive に分類される．しかし，違反が改善されないソースコードがコードレビューにおいて検証者が指摘することも多く [5] [6]，プロジェクトに参加する開発者間で合意する実装ルールを確立することは容易ではない．

従来研究では，false-positive に分類される警告を削減するために，解決すべき規約違反を優先順位付けする手法 [7] や，規約違反解決の履歴に基づくコーディング規約改定手法 [8] が提案されている．ただし，従来研究では将来的に解決される規約違反も false-positive に分類されていることが多い．ソフトウェア開発では，規約違反が検出された時点では優先的に解決されず，後日に解決される規約違反も多く存在する．

1 和歌山大学
Wakayama University，Wakayama 640-8510，Japan
a) ihara@sys.wakayama-u.ac.jp

*1 https://www.oracle.com/technetwork/java/codeconventions-150003.pdf
*2 https://www.python.org/dev/peps/pep-0008/.
*3 http://caxapa.ru/thumbs/468328/misra-c-2004.pdf
*4 https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards
*5 https://checkstyle.sourceforge.io/.
*6 https://pypi.org/project/pylint/.

ⓒ 2021 Information Processing Society of Japan

本論文では，規約違反解決の端緒となるソースコードの特徴を理解するために，規約違反検知から開発者による規約違反解決までのソースコードの進化過程を追跡し，規約違反を解決する必要があると判断するソースコードの特徴を分析する．具体的には，規約違反が存在するコミットと規約違反が解決されたコミットの特徴量に差があるか否かを調査する．ケーススタディとして，Python 言語を主に使用する airflow プロジェクト[*7]のソースコードを対象に，プロジェクトが使用する静的解析ツール Pylint を使用して，836 件の規約違反がそれぞれ解決するに至るソースコードの特徴を分析する．本研究は，規約違反を解決するに至るソースコードの特徴を明らかにすることで，将来的に解決される規約違反の早期解決に繋がることを期待する．

続く 2 章では，静的解析ツールとその課題，関連研究について述べる．3 章では，事前分析した結果について述べ，4 章では，規約違反の追跡方法および追跡した規約違反を用いたソースコード特徴量の分析を行う．5 章では，分析結果および考察を述べる．6 章で制約について述べる．7 章でまとめを行う．

2. 静的解析ツールを用いたコーディング規約違反の検出

2.1 静的解析ツール

OSS 開発では，様々な実装スタイルを持つ不特定多数の開発者がプロジェクトに貢献するため，プロジェクトは独自のコーディング規約を制定している．検証者が目視で，規約違反を検出することはコストがかかるため，コーディング規約を違反するソースコード断片を自動的に検出する静的解析ツールが使用される．静的解析ツールは，字句解析や抽象構文解析を用いることで，プログラムを実行せずに，コーディングルール，制御フロー，ソースコード複雑度などを分析し，ソースコード上の欠陥を検出するツールである．静的解析ツールには，Java 言語を対象とする Checkstyle や Python 言語を対象とする Pylint などがある．検証対象となるソースコードは，静的解析ツールが規約違反を検出しているか否か，また規約違反しているソースコードを開発者が解決するか否か，によって 4 種類に分類できる．

- **true-negative（検知不要）**: 静的解析ツールが規約違反を検出せず，開発者が改善することもないソースコード．
- **false-negative（未検知）**: 静的解析ツールが規約違反を検出していないが，開発者が改善するソースコード．
- **false-positive（誤検知）**: 静的解析ツールが規約違反を検出したにもかかわらず，開発者が改善しないソースコード．
- **true-positive（正検知）**: 静的解析ツールが規約違反を検出し，開発者が改善するソースコード．

多くのプロジェクトでは，false-negative を削減するために，コーディング規約違反の検出基準を低く設定している．検出基準を低くすることで，開発者が規約違反ではないと判断している実装にも規約違反が検出され，false-positive に分類されるソースコードが増加する [9]．しかし，新規の開発者ほど静的解析ツールが出力した警告を誤検知であると疑うことは少ない [4] [10]．従来研究では，静的解析ツールの改善に向けた研究が多数行われている．

2.2 従来研究

2.2.1 静的解析ツールの警告への対応に関する研究

Panichella らは，開発者は，静的解析ツールが出力する警告のうち，特定のカテゴリの規約違反を解決することを明らかにした．コードレビューを行う前後で，静的解析ツールが検出した規約違反の多くを開発者は解決しないが，パッケージの呼び出し，正規表現，型解決などの特定のカテゴリの規約違反は解決している [11]．調査結果から，静的解析ツールが出力する警告には false-positive が多く含まれるため，静的解析ツールが検出する規約違反を制御することで，false-positive と true-positive を区別する研究が行われている．

2.2.2 静的解析ツールが出力する警告の制御

Aman らは，静的解析ツールが検出する規約違反に優先順位付けを行うことで，false-positive を削減することを可能とした [7]．検出された規約違反を対象に，生存時間分析を行うことで，長期間解決されない規約違反を特定し，削除することで，出力された 259 種類の警告を 30 種類に削減することを成功している．また，渥美らは，開発者が過去に修正する必要がないと判断した規約違反を false-positive とし，削除することで，解決すべき規約違反であるか否かを判断するコストを削減している [8]．しかし，一度 false-positive と判断された規約違反が，将来的に true-positive となり，解決されることも多いため，問題のなかった実装が不具合へと変化する要因があると考えられる．

3. 事前分析

本論文では，false-positive と判断されたソースコードが改善される端緒をつかみ，警告箇所の解決時期を見積もることができると考える．そこで，事前分析として false-positive と判断された規約違反を含むソースコードと，true-positive であると判断された規約違反を含むソースコードを比較し，ソースコード特徴量の違いを調査する．ソースコード特徴量には，表 1 不具合予測をはじめとするソースコード品質評価に使用されるメトリクスを使用する [12]．

[*7] airflow: `http://hogehoge`

表 1: 規約違反を含むソースコード特徴量と規約違反を解決したソースコード特徴量の統計的有意差と効果量
（有意水準 5%以上，かつ効果量 |0.03| 以上のメトリクスに (*) を付している．）

変数名	説明	有意差	効果量
accum_total_loc	過去 30 日間のソースコードの累積総変更行数	0.000	-0.209 (*)
accum_add_loc	過去 30 日間のソースコードの累積追加行数	0.000	-0.196 (*)
accum_delete_loc	過去 30 日間のソースコードの累積削除行数	0.000	-0.229 (*)
changed_total_loc	変更によるソースコードの総変更行数	0.000	-0.602 (*)
changed_add_loc	変更によるソースコードの追加行数	0.000	-0.615 (*)
changed_delete_loc	変更によるソースコードの削除行数	0.000	-0.547 (*)
accum_file_changed_num	過去 30 日間の累積ファイル変更回数	0.000	-0.372 (*)
accum_changed_file_num	過去 30 日間の累積変更ファイル数	0.000	-0.406 (*)
accum_changed_file_kind	過去 30 日間の累積変更ファイルの種類数	0.000	-0.363 (*)
accum_commit_num	過去 30 日間の累積コミット数	0.000	-0.406 (*)
changed_file_num	変更による変更ファイル数	0.317	0.042 (–)
impr_warn_num	変更前に存在する全警告数（後に解決される警告箇所が対象）	0.000	0.602(*)
all_warn_num	変更前に存在する全警告数（全警告箇所が対象）	0.000	-0.550(*)
impr_warn_num_per_file	変更前に存在する変更ファイル内の警告数（後に解決される警告箇所が対象）	0.412	0.034(–)
all_warn_num_per_file	変更前に存在する変更ファイル内の警告数（全警告箇所が対象）	0.377	0.037(–)

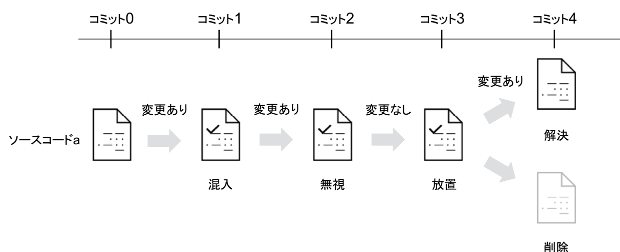

図 1: 規約違反を含むソースコードの状態変化

3.1 規約違反の分類

静的解析ツールが検出する規約違反は，開発者が true-positive であると判断して解決することもあるが，機能削除のために規約違反を含むソースコードを削除することで解決されることもある．本論文は，検出された規約違反の解決が，意図した解決であるか否かを区別するために，規約違反を含むソースコード行を 5 つの状態に区別する．図 1 は，規約違反の混入から，解決または削除までの例を示す．事前分析では，最初に無視された規約違反を含むソースコード行と，規約違反が解決されたソースコード行を比較することにより，5 つの状態に区別する．表 1 は，各状態の説明を示す．

3.2 対象データセット

本論文では，静的解析ツールを使用しているリポジトリを分析対象とする．静的解析ツールを使用するリポジトリの選定方法は，リポジトリ内の設定ファイルの有無によって判断する．本分析では，Python を主に使用するプロジェクトで，かつ静的解析ツール Pylint を使用するプロジェクトを対象とする．したがって，設定ファイルである pylintrc ファイルを所持するリポジトリを選択する．2021 年 2 月時点で，スター数上位 100 件に含まれる，Apache が開発する airflow リポジトリを対象とする．airflow リポジトリは，2019 年 6 月 10 日に Pylint の使用開始が確認されたため，2019 年 6 月 10 日から 2019 年 11 月 21 日までの 958 コミットを分析対象とする．

3.3 ソースコード特徴量の比較

表 1 に示すソースコード特徴量を用いて，開発者が最初に無視した規約違反を含むソースコードと，当該規約違反を解決したソースコードの 2 群間の統計的有意差と効果量を計測する．有意差の計測には，マン・ホイットニーの U 検定（有意水準 5%）を使用し，効果量の測定には，Cliff's Delta を使用する．

表 1 は，958 コミットに含まれる規約違反を含むプログラム行 377 件を対象にソースコード特徴量を計測をした結果を示す．15 個の特徴量のうち 12 個は統計的に有意な差があり（有意水準 5%），効果量が十分に高い値（|0.03| 以上）を示していることを確認した．したがって，規約違反の混入から解決までにソースコード特徴量が変化し，解決するに至るまでに開発者が警告を解決する閾値が存在すると考える．本論文では，ソースコード特徴量の変化過程を，規約違反の無視から解決まで，追跡することで，規約違反解決の端緒となる特徴量を明らかにする．

4. 分析手法

4.1 概要

規約違反が無視されるソースコードの中には，開発者の勘や経験，その他のソースコードの変更などに伴い，開発者が違反を優先的に解決すべきと判断することがある．本論文では，警告解決の端緒となるプログラムの特徴を理解

ⓒ 2021 Information Processing Society of Japan

表 2: 規約違反を含むソースコード行の状態

状態	定義
混入	前のリビジョンでは検出されなかった規約違反が，次のリビジョンで新たに検出された場合.
放置	前のリビジョンに含まれていた規約違反が，次のリビジョンで規約違反するソースコードに変更を加えたが規約違反が解決されなかった場合.
無視	前のリビジョンに含まれていた規約違反が，次のリビジョンで規約違反するソースコードに変更を加えなかった場合.
解決	前のリビジョンに含まれていた規約違反が，次のリビジョンで規約違反するソースコードに変更を加え，規約違反が解決された場合.
削除	前のリビジョンに含まれていた規約違反が，次のリビジョンで規約違反するソースコードを削除した場合.

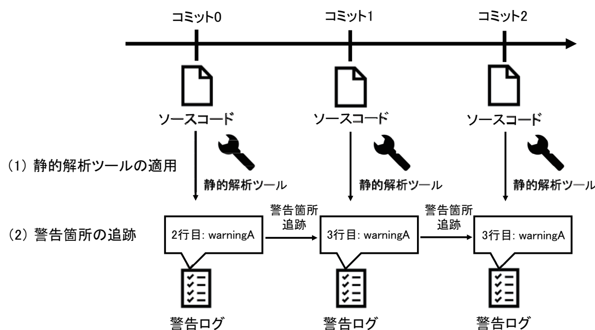

図 2: 追跡方法の概略図

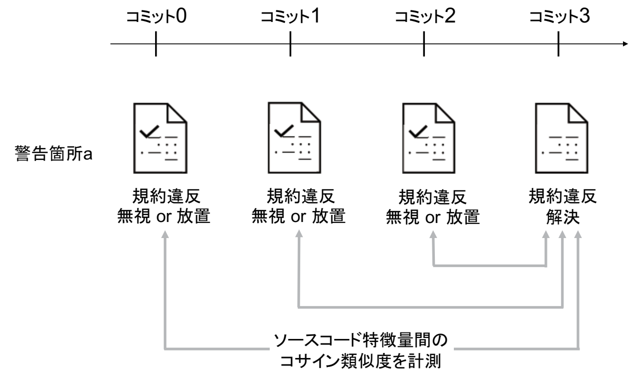

図 3: 分析 2 の計測手法の概略図

するために，規約違反を検知してから解決されるまでのソースコードの進化過程を追跡し，解決が必要となるソースコードの変更を分析する．具体的には，規約違反を含むソースコード行を対象に，開発者が無視または放置したコミットのソースコード特徴量と，解決したコミットのソースコード特徴量の類似度を算出し，違反が解決するまでの特徴量の変化を分析する．

分析: 無視または放置したソースコード特徴量と解決したソースコード特徴量の類似度の時系列分析

4.2 規約違反ソースコード行の追跡方法

規約違反を含むソースコードの特徴量を計測するために，規約違反を含んでいるソースコード行の特定と，特定したソースコード行に規約違反の混入から解決までを追跡する．図 2 は追跡方法の概略図を示す．規約違反を含むソースコード行を特定するために，静的解析ツールを導入する全てのコミットにおいて静的解析ツールを実行する．実行結果から，規約違反が検出されたファイルパスと規約違反を含む行番号を取得する．ソースコードの変更によって，規約違反が含まれる行番号が変化する場合があるため，コミット間の規約違反するソースコード行の追跡には icdiff [*8]を用いる．icdiff は，ソースコードの変更に伴う行番号の変化を追跡するツールである．行番号の変化に対処しつつ，検出された規約違反を含むソースコード行の状態（放置，無視，解決，削除）を判断する．

本論文では，規約違反ソースコード行を含むソースコードファイルが関係するコミットを対象に，表 1 において効果量の大きい 12 個の特徴量を計測し，分析を行う．

[*8] icdiff: https://www.jefftk.com/icdiff

4.3 分析: 無視または放置したソースコード特徴量と解決したソースコード特徴量の類似度の時系列分析

分析では，コミットの経過に伴うソースコード特徴量と規約違反解決に相関があることを明らかにするために，開発者が規約を無視または放置したコミットのソースコード特徴量と，解決したコミットのソースコード特徴量の類似度を算出し，違反が解決するまでの特徴量の変化を分析する．図 3 は，ソースコード行単位でそれぞれ時系列の追跡方法の概略図を示す．類似度は，規約違反を解決したソースコード行のソースコード特徴量を基準に計測を順次行う．類似度を計測する際に，特徴量によって，値の取りうる範囲や単位が異なるため，特徴量は z 得点による標準化を行い，算出したソースコード特徴量毎の z 得点をベクトル化し，ベクトル間の類似度を計測する．

5. 分析結果

図 5 は，分析対象とする規約違反を含む全てのソースコード行をそれぞれ追跡し，規約違反が解決したコミットを基準に規約違反を無視したコミットとの類似度の分布を箱ひげ図で示した結果を示す．横軸は追跡した時間軸（コミット）を示し，縦軸はコサイン類似度を示す．最も左が規約違反を解決したコミットであり，右にいくほど，規約違反が解決したコミットから 1 コミットずつ遠ざかっていく．規約違反が解決したコミットを基準にしているため，最も左の類似度は 1 となる．

図 5 に示す結果から，規約違反を解決したコミットから遠ざかるほど，ソースコード特徴量の類似度は低い値を示し，ソースコード特徴量の類似度は徐々に増加すること

ⓒ 2021 Information Processing Society of Japan

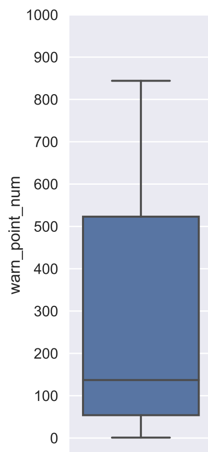

図 4: 警告解決までのコミット数

を確認した．図 5 に示す結果には，規約違反混入後すぐに解決されるソースコード行と，解決に長期間かかるソースコード行が含まれるため，本論文では規約違反解決にかかるコミット数に基づき，四分位範囲で 4 つのグループを作成し，それぞれのグループで類似度の変遷を確認する．図 4 は，規約違反を含むソースコード行が解決までにかかるコミット数の分布を示す．

- 短期修正: 1 コミット以上 54 コミット以下（121 の警告箇所）
- 中短期修正: 55 コミット以上 137 コミット以下（229 の警告箇所）
- 中長期修正: 138 コミット以上 523 コミット以下（244 の警告箇所）
- 長期修正: 524 コミット以上 844 コミット以下（242 の警告箇所）

図 6 は，4 つのグループそれぞれの類似度の分布を箱ひげ図で示す．上から順に，短期修正，中短期修正，中長期修正，長期修正のグループの結果を示し，横軸は追跡した時間軸（コミット）を示し，縦軸はコサイン類似度を示す．短期修正，中短期修正のグループでは，最初に規約違反を無視した時点からソースコード特徴量の類似度は高く，中長期修正と長期修正のグループでは，無視したタイミングでは類似度は低く，徐々に類似度が高くなることがわかった．特に，中長期修正と長期修正では，類似度が 0.8 以上に達するのは，解決から約 120 コミット前（中長期グループ: 122 コミット前，長期グループ: 118 コミット前）である．次の節では，規約違反を含むソースコードが解決されるまでのソースコード特徴量の分析を述べる．

5.1 ソースコード特徴量と規約違反解決の関係

図 5 における類似度の変遷の理由を明らかにするため，各ソースコード特徴量と類似度の関係性について分析する．図 7 の上から一つ目の図は，図 5 と同じ図であり，その下は規約違反を解決したコミットと無視したコミット間の一部のソースコード特徴量（ソースコードの累積総変更行数，変更によるソースコードの追加行数，累積ファイル変更回数，累積コミット数，累積変更ファイル数，変更前に存在する変更ファイル内の警告数）の統計的有意差をそれぞれ時系列順に示す．横軸は追跡した時間軸（コミット）を示し，縦軸は各コミット時における検定結果（P 値）を示す．赤線は，0.05 を示し，折れ線グラフが赤線より下の期間は，規約違反が解決したコミットと規約違反を無視したコミットとのソースコード特徴量に統計的有意な差があることを示す．

ソースコードの累積総変更行数と累積ファイル変更回数は，解決される約 120 コミット前から統計的有意な差がないコミットが増えている．つまり，ソースコードの変更が収束したのちに規約違反しているソースコードを解決していることが示唆される．

累積コミット数や累積変更ファイル数は，全てのコミットにおいて規約違反が解決したコミットと規約違反を無視したコミットとのソースコード特徴量に統計的有意な差があるため，規約違反を解決するソースコードの特定に有用であるとは考えにくい．また，変更によるソースコードの追加行数は，統計的有意差がある時期とない時期が繰り返されているため，規約違反解決を理解するためのソースコード特徴量として利用できないと考えられる．

変更前に存在する変更ファイル内の警告数は，約 120 コミット前から統計的有意な差がないコミットが増えているため，開発者は規約違反をまとめて解決していると考えられる．このメトリクスは，後に解決される警告箇所を対象としているため，規約違反解決を理解するためのメトリクスとして利用することは困難である．

6. 制約

本論文では，設定ファイルが存在するリポジトリを静的解析ツールを使用しているリポジトリを対象としているが，Pylint を使用せずに貢献していることも考えられる．

本論文では，airflow リポジトリを対象として分析を行なった．静的解析ツールは，Pylint 以外にも多く存在し，また他のリポジトリを対象とすることで，異なる結果になることも考えられる．今後は，異なる静的解析ツール，異なるリポジトリを対象に同様の分析を行う．

7. おわりに

本論文では，静的解析ツールによる規約違反解決の端緒となるソースコードの特徴を理解することを目的とし，規約違反を無視したソースコード行と解決したソースコード行のソースコード特徴量の違いを分析した．分析の結果，開発者が規約違反を無視するソースコード行と解決するソースコード行では，ソースコード特徴量に統計的有意な

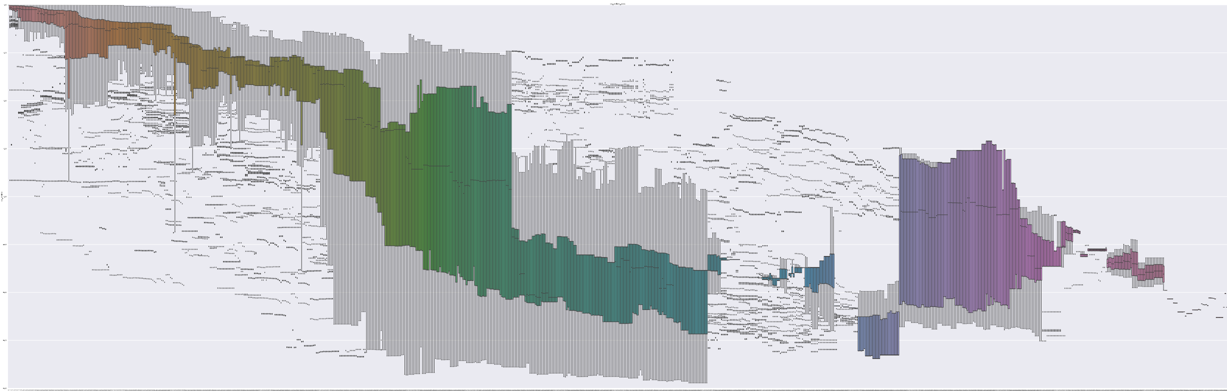

図 5: 規約違反が解決したコミットを基準に規約違反を無視したコミットとの類似度の分布

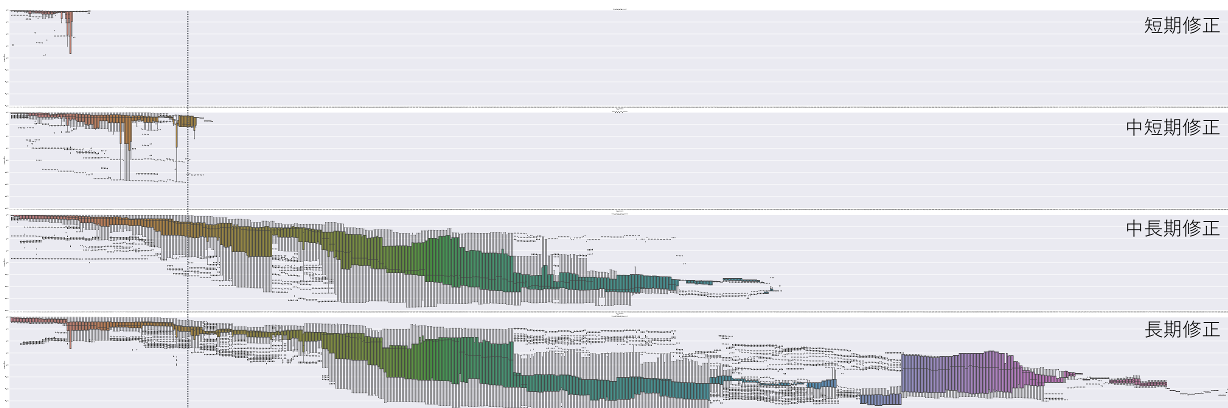

図 6: 規約違反が解決したコミットを基準に規約違反を無視したコミットとの類似度の分布（解決までの期間別）

差があることを確認した．また，ソースコード特徴量は，解決される約 120 コミット前から統計的優位な差がなくなるため，ソースコードの変更が収束したのちに規約違反しているソースコードを解決していることが示唆される．特に，ソースコードの累積総変更行数と累積ファイル変更回数が，開発者が規約違反を解決するコミットの検討に有用であると考えられる．

今後は，規約違反を解決する判断基準として，ソースコード特徴量の閾値を定めることによって，将来的に解決される規約違反が早期に解決されるべき規約違反であるかを判断する手法を提案する．ただし，規約違反解決の端緒となるソースコード特徴量にはばらつきがあるため，ソースコードの内容，規約違反の種類に着目する必要があると考える．

謝辞 本研究は JSPS 科研費 JP18KT0013 の助成を受けたものです．

参考文献

[1] 大須賀俊憲，小林隆志，渥美紀寿，間瀬順一，山本晋一郎，鈴村延保，阿草清滋：CX-Chekcer: 柔軟にカスタマイズ可能な C 言語プログラムのコーディングチェッカ，情報処理学会論文誌， Vol. 53, No. 2, pp. 590–600 (2012).

[2] Rigby, P. C. and Storey, M.-A.: Understanding broadcast based peer review on open source software projects, *In Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*, pp. 541–550 (2011).

[3] Bacchelli, A. and Bird, C.: Expectations, outcomes, and challenges of modern code review, *In Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*, pp. 712–721 (2013).

[4] Johnson, B., Song, Y. and Murphy-Hill, E.: Why Don't Software Developers Use Static Analysis Tools to Find Bugs?, *In Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*, pp. 672–681 (2013).

[5] Bosu, A. and Carver, J. C.: Impactof developer reputation on code review outcomes in oss projects: an empirical investigation, *In Proceedings of the 8th ACM/IEEE international symposium on empirical software engineering and measurement (ESEM'14)*, pp. 33–42 (2014).

[6] Rigby, P. C. and Bird, C.: Convergent contemporary software peer review practices, *In Proceedings of the 9th Joint Meeting on Foundations of Software Engineering (FSE'13)*, pp. 202–212 (2013).

[7] Aman, H., Amasaki, S., Yokogawa, T. and Kawahara, M.: A Survival Analysis-Based Prioritization of Code Checker Warning: A Case Study Using PMD, *Proceedings of the 14th International Conference on Big Data, Cloud Computing, and Data Science Engineering (BCD'19)*, Vol. 844, pp. 69–83 (2020).

[8] 渥美紀寿，桑原寛明：静的検査ツールにおける警告箇所の版間追跡による確認コスト削減手法，研究報告ソフト

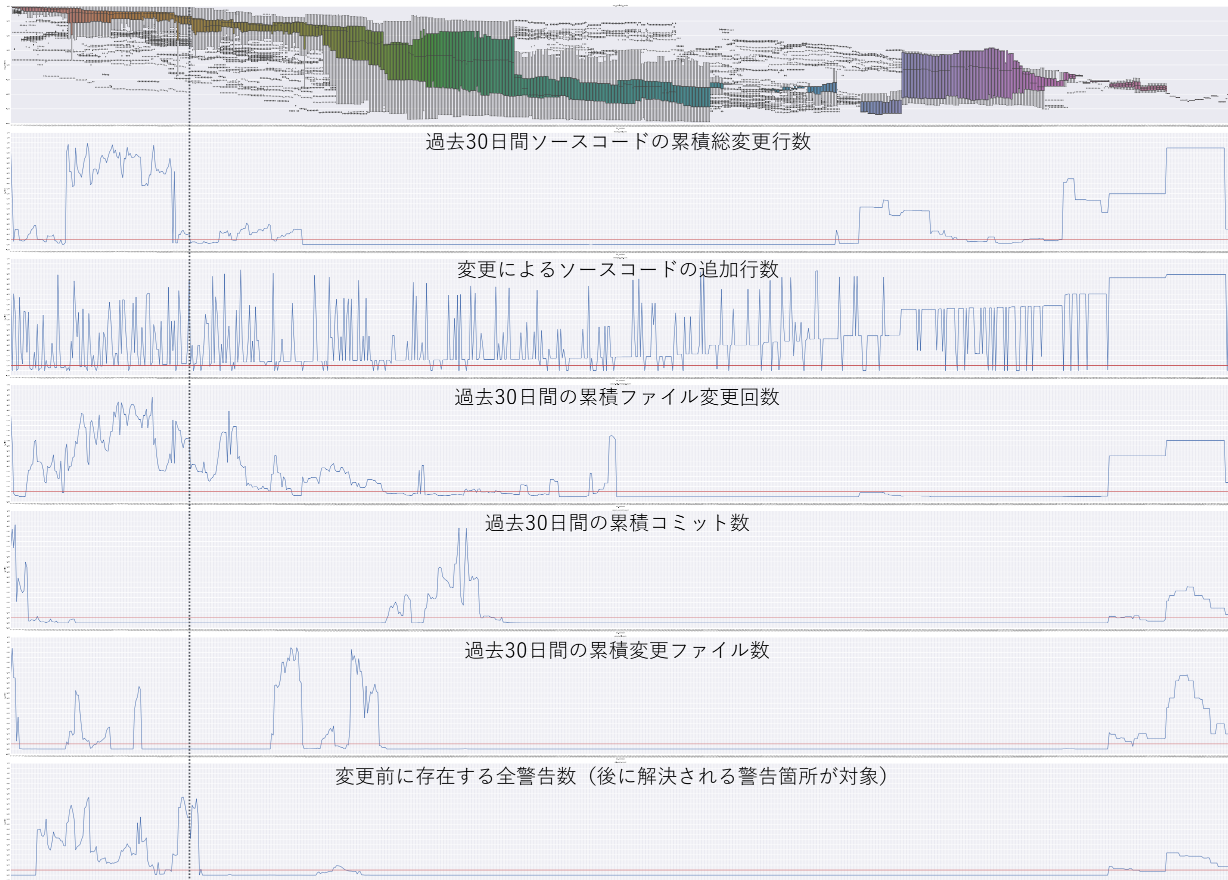

図 7: 規約違反を解決したコミットと無視したコミット間の一部のソースコード特徴量の統計的有意差

ウェア工学 (SE'15)，Vol. 44, pp. 1–8 (2015).

[9] Marcilio, D., Monteiro, E., Canedo, E., Luz, W. and Pinto, G.: Are Static Analysis Violations Really Fixed? A Closer Look at Realistic Usage of SonarQube, *In Proceedings of the IEEE/ACM 27th International Conference on Program Comprehension (ICPC'19)*, Vol. 53, pp. 672–681 (2019).

[10] Imtiaz, N., Rahman, A., Farhana, E. and Williams, L.: Challenges with Responding to Static Analysis Tool Alerts, *In Proceedings of the IEEE/ACM 16th International Conference on Mining Software Repositories (MSR'19)*, pp. 245–249 (2019).

[11] Panichella, S., Arnaoudova, V., Penta, M. D. and Antoniol, G.: Would Static Analysis Tools Help Developers with Code Reviews?, *In Proceedings of the 22th International Conference on Software Analysis, Evolution and Reengineering (SANER'15)*, pp. 161–170 (2015).

[12] 畑秀明，水野修，菊野亨：不具合予測に関するメトリクスについての研究論文の系統的レビュー，コンピュータソフトウェア，Vol. 29, No. 1, pp. 106–117 (2012).