ソーシャルコーディングにおけるソースコード中の If 文自動検証システムの開発

上田 裕己[1] 伊原 彰紀[2] 石尾 隆[1] 桂川 大輝[1] 森田 純恵[3] 菊池 慎司[4] 松本 健一[1]

概要：オープンソースソフトウェア (OSS) 開発をはじめ，GitHub 上で不特定多数の開発者と協調作業する場合，開発者によってコーディングスタイルが異なるため，提案されたソースコードがプロジェクトが方針とするソースコードの実装ルールに遵守していないことが多い．実装ルールに沿ったソースコードへ修正するために，検証者と呼ばれる複数の開発者がソースコードを検証し，修正を提案を行うが，その作業には膨大なコストを必要とする．本論文では，(1) 過去のソースコード変更履歴からプロジェクトに暗黙的な実装ルールを検出し，(2) 実装ルールに基づいたソースコードの修正例を自動的に提案するシステム REVAD(Review Adviser) bot を開発し，提案されたソースコードを検証する前にソースコードの修正を促すことによって，検証作業コストを削減することを狙いとする．REVAD bot を評価するために，OSS プロジェクトのコア開発者にシステムの評価を依頼し，当該 OSS プロジェクトにおいて提案されたソースコードに対して，REVAD bot が提示する修正案のうち 40%(2/5) がプロジェクトに有用な修正案であることを確認した．

1. はじめに

オープンソースソフトウェア (OSS)，企業のソフトウェア開発において，ソフトウェアを構成するソースコードを GitHub で広く公開するプロジェクトが増えている．その理由の一つは，OSS 開発特有の開発方式であるバザール方式の「目玉の数さえ十分であれば，どんなバグも深刻ではない」という譬え話にもあるように，開発者らの共同作業によって意図しない不具合がソフトウェア製品に混入しても，OSS 開発に貢献する不特定多数の開発者の誰かが不具合を発見し，修正するため，迅速な修正を期待できることがあげられる．

ソースコードをクラウド上で共有することで，不特定多数の他者とのプログラム実装の協調作業を支援する Web サービス「GitHub」が OSS をはじめとするソフトウェア開発のデファクトスタンダードな開発形態として普及し，そのソフトウェア開発形態は「ソーシャルコーディング」と呼ばれている．今日では，多くの OSS プロジェクトや企業のソフトウェアプロジェクトが GitHub 上で世界中にソースコードを公開している．GitHub は，リポジトリと呼ばれるソースコードを構成管理するデータサーバを SNS 機能によって誰でも取得可能にすることで，他の開発者が実装したソースコードの容易な再利用を可能にした．また，GitHub では，ブランチ機能により他の開発者の作業に影響せずに，機能追加や不具合修正のためのソースコード変更を実装することができ，誰でも Pull Request 機能によってブランチ上の変更内容をソースコードの管理者に提案することができる [7].

GitHub が実現するソーシャルコーディングは，リポジトリからソースコードを取得した誰もが，ソフトウェアの不具合修正，新機能追加の提案を実現する．しかしその一方で，ソフトウェア開発プロジェクトが掲げる実装ルールを十分に理解していない開発者による提案も多く，プロジェクトの管理者は，実装ルール（例えば，コーディングスタイルなど）に基づくソースコードの検証をより重要視することが求められる [2]．ソースコードを実装した開発者（パッチ開発者）とは異なる複数人の開発者（検証者）がソースコードの検証を行い，必要に応じてパッチ開発者による再修正を幾度か繰り返す．このソースコードの保守作業にかかる工数は，ソフトウェア開発全体の約 50%を占める [8].

プロジェクトは，保守作業のコストを軽減するための実装ルールとして PEP8 [13]，CERT C，MISRA C のようなコーディングガイドラインを利用することが多い．このようなコーディングガイドラインには，プログラミング言語

[1] 奈良先端科学技術大学院大学情報科学研究科
[2] 和歌山大学システム工学部
[3] 富士通ゼネラル
[4] 富士通研究所

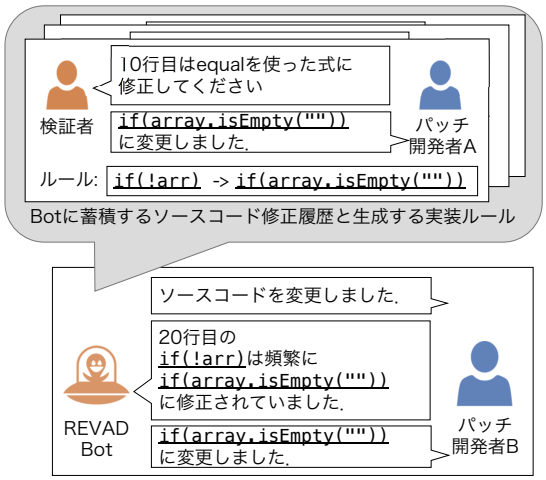

図 1: 提案システム REVAD bot の利用概要

に応じて可読性の高いソースコードの実装を実現するための方法が記されている．また，Python 言語によって記述されたソースコードの可読性を向上するためのコーディングチェックツール pylint [11] が存在し，Pull Request として変更したソースコードを提出する前に実装ルールの違反を検出することができる．しかし，開発者はツールが検出した違反を解決せずに提案を提出することもある．また，技術の進化に伴って実装ルールを改定したとしても普及させることは容易ではない．

本論文では，パッチ開発者が提案するソースコード検証のコスト削減に向けて，GitHub に提出される Pull Request に対して検証結果を出力するボットシステム REVAD (REView ADviser) bot を開発する．REVAD bot は，GitHub の Pull Request 機能を使ってパッチ開発者が提出した変更済みのソースコードを解析し，過去の修正履歴に基づき，当該ソースコードの修正案を提示する．REVAD bot の出力例を図 1 に示す．

REVAD bot は，2 種類の機能によって構成される．(1) GitHub に蓄積されているソースコード修正履歴から修正ルールを作る機能（**ルール生成機能**）．(2) ボット開発フレームワークである Probot を使い，Pull Request 機能によって投稿されたソースコードを受け取り，ルールに基づいた修正案を GitHub に出力する機能（**入出力機能**）．本論文では，ソフトウェア開発において最も頻繁に利用されるプログラムの要素の一つである `if` 文の修正に着目し，`if` 文の修正ルール生成と修正提案を行う [5], [6]．

REVAD bot の出力結果を評価するために，OSS プロジェクトの 1 つである Open Service Catalog Maneger [*1] を対象に修正案を生成し，プロジェクトのコア開発者へのインタビューを行う．

本論文の構成を以下に示す．続く 3 章では，開発した REVAD bot についての概要を説明する．4 章では，REVAD bot の実装内容を記述する．5 章では，評価結果についての議論する．6 章では，本論文の成果と今後の展望についてまとめる．

[*1] https://github.com/servicecatalog/development

2. ソフトウェア開発における実装ルール

コードレビューでは実装ルールに基づいてソースコードの修正やリファクタリングが行われている [3], [10]．実装ルールへの違反を自動的に検出，修正するために多くのツールが提案されており，Smit らが提案した *CheckStyle* や Thenault らによって提案された *Pylint* などの自動検出ツールは，記述されたソースコードがプログラミング言語に基づくコーディング規約に違反しているか否かを検出する [9], [11]．さらに，Allamanis らはコーディング規約に従った修正を行うツールを開発している [1]．これらのツールを利用することで Pull Request による投稿前に，言語仕様に基づくコーディング規約に従ったソースコードへ修正することができる．

しかし，開発者は，従来研究で提案されたツールが検出した違反を解決せずに提案を提出することもあるため最終的に開発者による検証作業が必要となる．また，技術の進化に伴って実装ルールを改定したとしても，それらルールを普及させることは容易ではない．本論文で提案する REVAD bot は，GitHub 上でプルリクエストを通して提出されたソースコードを検証する手法であり，検証するためのルールはプロジェクトの判断で変更することが可能である．

3. REVAD bot の概要

本論文で開発する REVAD bot は，ソースコードの修正履歴から頻繁に出現する `if` 文の修正パターンを抽出する従来手法を用いて，暗黙的な `if` 文の実装ルールを抽出する [12]．REVAD bot は GitHub における Pull Request 機能を通して投稿されるソースコード変更に対し，ルールに合致する `if` 文の変更が行われているか否かを自動判定し，パッチ投稿者に修正提案を行う．REVAD bot は以下の手順で GitHub の Pull Request を通した提案に対して修正案を提示する：

1. パッチ開発者がソースコードを変更し，Pull Request として GitHub 上のプロジェクトに投稿する．
2. パッチ開発者が投稿したコードで `if` 文が変更されていた時，`if` 文に対する修正方法があるかを過去の Pull Request の履歴から確認する．修正方法がある場合はどのように修正すればよいか，ソースコード修正前後を示した**コード修正例**をフィードバックとして開発者に提供する．パッチ開発者は提供されたコード修正例を参考に，再度ソースコードを修正・再投稿する．
3. パッチ開発者による REVAD bot のフィードバックに基づいたソースコード修正後，検証者がソースコー

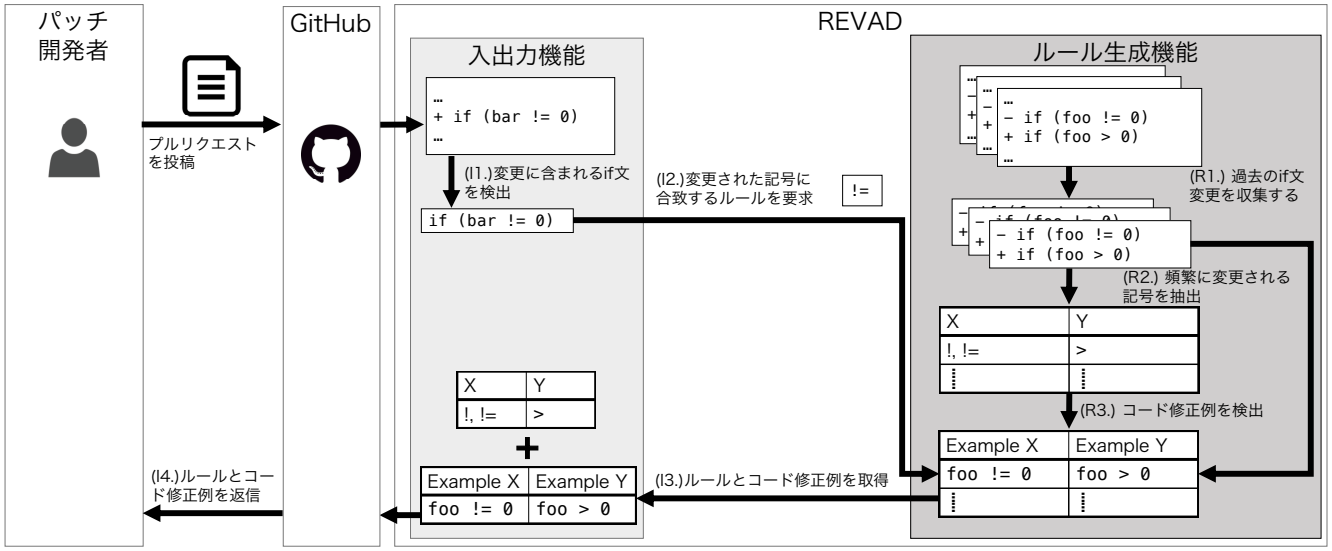

図 2: GitHub 上での REVAD bot の動作概要

ド変更内容を手作業で確認する．検証者によって問題が見つかった場合は再度ソースコードの修正・投稿をパッチ開発者に要求する．

4. パッチ開発者が再度修正を行ったソースコードを投稿した時，REVAD bot も再度検証とフィードバックを行う．パッチ開発者は REVAD bot と検証者から Pull Request の受理または拒否が決定するまで修正を投稿を繰り返す．
5. パッチ開発者による修正完了後，変更内容はバージョン管理システムによってプロジェクトに統合される．

REVAD bot は，将来のレビューで行われる修正を予測し，レビュー担当者によるコード検証前にパッチ開発者へ修正提案を提供する．本アプローチを使用することで，検証者が提出されたコード変更の検証に費やす時間を削減できることを期待する．

4. REVAD bot の構成

REVAD bot は過去の検証者によるフィードバックから暗黙的な実装ルールを生成し，GitHub の Pull Request を投稿したパッチ開発者に対して，ルールに基づいた修正提案を提供する．本章では，REVAD bot による，暗黙的な実装ルールの検出方法と修正提案の検出方法について示す．

図 2 に REVAD による修正提案システム概要を示す．REVAD bot は**ルール生成機能**と**入出力機能**によって構成する．ルール生成機能は修正パターンを収集し，過去の Pull Request で行われた検証者のフィードバックによるコード修正例を検出する．入出力機能はパッチ開発者が投稿した Pull Request に対してルール生成機能で生成されたルールから適切なものを選びフィードバックとしてパッチ開発者に提案を行う．

4.1 ルール生成機能

ルール生成機能は暗黙的な実装ルールとルールに基づいたルールを生成する．プロジェクト管理者は，Pull Request が提出される前にルール生成機能を実行し，ルールとコード修正例を生成する．ルール生成機能は図 2 に示した手順 (R1.) から (R3.) を実行する．

(R1.)過去の Pull Request に含まれる `if` 文の変更を収集する．

(R2.)各 `if` 文変更から変更された記号を抽出する (e.g. “`!`” または “`!=`”)．本分析では，頻繁に変更される記号を構文解析によって抽出する．

(R3.)各実装ルールに対してコード修正例を検出する．また，ルールに当てはまる最新の Pull Request の修正からコード修正例を抽出する．(e.g. “`!`” と “`!=`” を “`>`” に変更するコード修正例として “`foo != 0`” を “`foo > 0`” を検出する)

最後に，ルール生成機能によって暗黙的な実装ルールとルールに基づいたコード修正例を出力する．

また，頻繁に出現しない実装ルールを取り除くために，ルール生成機能では代表的なデータマイニング手法であるアソシエーションルールマイニングを利用する [4], [14]．

アソシエーションルールマイニング手法は複数存在する要素の組み合わせから，関連性の高い要素同士を実装ルールとして抽出する手法である．アソシエーションルールマイニング手法を利用して，条件部，結論部をそれぞれ **X**，**Y** とし，“X という記号が存在するときに Y という記号が存在する” というルールとそのルールが発生する割合 (**Confidence 値**) を抽出する．本稿ではアソシエーションルールマイニングを R のパッケージである `arules` を利用し，**Confidence** 値が 0.01 以上のルールを抽出する．

4.2 入出力機能

入出力機能は GitHub を通して Pull Request が投稿されたときに，ルール生成機能で生成した実装ルールを利用し，ルールに合致する `if` 文の変更を検出する．

Pull Request によるコード変更をリアルタイムで抽出するために GitHub Apps のフレームワークである Probot [*2]を利用して REVAD bot を実装した．GitHub Apps [*3]は GitHub 上のプロジェクトで発生したイベント取得やアクションを行うアプリケーションであり，REVAD bot は GitHub プロジェクトに投稿された Pull Request のソースコードを取得し，パッチ開発者へ返信を行う．

入出力機能は図 2 に示した手順 (I1.) to (I4.) を実行する．

(I1.)Pull Request を利用して投稿されたソースコードに含まれる `if` 文の変更を検出する．(e.g. “`if (bar != 0)`”)

(I2.)`if` 文の条件式で変更された記号に合致する実装ルールをルール生成機能に要求する (e.g. “`!=`”)

(I3.)ルールとルールに合致するコード修正例をルール生成機能から取得する．

(I4.)Pull Request を通してパッチ開発者にルールとコード修正例を修正提案として提供する．

4.3 動作デモ

図 3，図 4，図 5 に GitHub 上で REVAD bot を動作させた際のスクリーンショットを示す．まず， 図 3 ではパッチ開発者が Java ファイル “`sample.java`” の “`if (mId.equals(""))`” を “`if (mId == null || mId == "")`” に変更した Pull Request を投稿している．次に，図 4 で REVAD bot は 3 つの実装ルール (e.g. “`==`” に対して “`.`” を追加する) とそれに基づくコード修正例を提供した．この時，複数のルールが同じコード修正例を持っているため，それらが同時に提案されることがある．最後に，図 5 では，パッチ開発者がコード修正例に基づいて `if` 文を修正している．

5. 評価

本章では，提案したルールと REVAD bot の有効性を評価するケーススタディについて述べる．評価では，OSS プロジェクトのコア開発者へインタビューを行う．

5.1 対象プロジェクト

本評価では企業が開発するオープンソースプロジェクトである Open Service Catalog Manager (OSCM) [*4]を対象プロジェクトとする．OSCM は SaaS マーケットプレイスから企業用 IaaS ストアなど幅広い用途に対応し，Amazon Web Services（AWS）や OpenStack のような IaaS プロバイダへの連携プラグインを提供する．また，他のプラットフォームでの利用を可能とするために OSCM のソースコードをオープンソースとして公開し，コミニティを設立している．

OSCM プロジェクトはソースコードを GitHub で公開し，24 人の開発者によって実装され，ソースコード変更の検証は Pull Request を利用して行っている．このとき，ソースコードの実装，レビュー，リファクタリングは，開発者が経験に基づいて定義したコーディングルールに基づいて行われている．

本ケーススタディでは，OSCM プロジェクトから 2017 年 12 月 14 日以前の Pull Request によるソースコード変更を 187 件抽出し，ルール生成の対象とした．

表 1: 生成したルール例 (Confidence 値によりソート)

X	Y	Confidence (# of X ∧ Y / # of X)	Case
`*`	`.`	1.00 (2 / 2)	1
`!=`	`==`	0.24 (6 / 25)	3
`!=`	`.`	0.24 (6 / 25)	3
`==`	`.`	0.22 (7 / 32)	5
`==`	`||`	0.16 (5 / 32)	5
`==`	`==`	0.16 (5 / 32)	5
`!`	`.`	0.15 (7 / 46)	3
`!`	`==`	0.13 (6 / 45)	3
`!=`	`&&`	0.12 (3 / 25)	3
`!`	`&&`	0.11 (5 / 46)	3
`!`	`!`	0.09 (4 / 46)	3
`||`	`||`	0.08 (1 / 12)	2
`!=`	`!`	0.08 (2 / 25)	3
`!=`	`>`	0.04 (1 / 25)	3
`!`	`>`	0.02 (1 / 46)	3
`.`	`!=`	0.01 (2 / 157)	4

表 2: コード修正例

Case	X	Example URL
1	`*`	`https://github.com/revad-ueda/bot-test/pull/100`
2	`||`	`https://github.com/revad-ueda/bot-test/pull/101`
3	`!`, `!=`	`https://github.com/revad-ueda/bot-test/pull/95`
4	`.`	`https://github.com/revad-ueda/bot-test/pull/97`
5	`==`	`https://github.com/revad-ueda/bot-test/pull/98`

5.2 生成ルールとコード修正例

4 章で述べた REVAD bot のアプローチに基づき，187 件の Pull Request に含まれるコード修正から 16 件のルールを抽出した．また，出現頻度の低いルールを除外するために，0.01 以上の *Confidence* 値をもつルールのみを生成した．*Confidence* の閾値を変更することによって，生成ルール数は変更可能である．表 1 に生成されたルールを示す．

- **X** は投稿時点での `if` 文に含まれている記号を示す (e.g. “`>`”).

[*2] `https://probot.github.io/`
[*3] `https://developer.github.com/apps/`
[*4] OSCM: `https://openservicecatalogmanager.org/`

- **Y** は変更後の `if` 文に含まれている記号を示す (e.g. “=”).
- **Confidence** は記号 **X** が出現する履歴の中で記号 **Y** も出現する履歴の割合を示す
- **Case** は実装ルールが合致するコード修正例が含まれる表 2 に示すタイプを示す.

16 件の実装ルールを 5 つのコード修正例に分けることができた．そのため，同じコード修正例を持つルールを表 1 に示す **Case** として定義した. 評価インタビューでは，OSCM プロジェクトで行われた Pull Request でのコード変更を収集した．次に，Pull Request を通じてコード変更を修正するルールとコード修正例を示し，評価を得た．表 2 に，OSCM プロジェクトのコア開発者へ提供した Pull Request を示す．また，Case1-5 に各コード修正例を示す．Case 3 は 2 つの記号 “`!`” と “`!=`” を含むため Case 3-(1), Case3-(2) に分ける．本調査では，各コード修正例に対して OSCM プロジェクトのコア開発者にインタビューを行い，生成したルールとコード修正例に対して有用性を評価する.

5.3 生成したルール評価

表 2 にインタビューで評価を行ったコード修正例の URL を示す．コア開発者は Case 1 と Case 3,Case 4 のコード修正例を利用可能でないという評価した．一方で，Case 2 と Case 5 はコア開発者に利用可能であると評価を受けた．この結果から，REVAD bot が提案した修正提案を開発者が精査することで，提案した 40 パーセントのコード修正例をルールとして提供可能であると示した.

5.4 REVAD bot 評価

本分析では OSCM プロジェクトのコア開発者へインタビューを行い，REVAD bot の有用性を評価する．インタビューは質問のリストと[*5]に加えてメールでのやり取りによって行う．質問内容は，REVAD bot の出力　(生成したルール，コード修正例，Confidence 値) がレビューコスト削減へ貢献するか否かをを 5 段階 (1:全く同意できる，5:非常に同意できる) の評価である．その結果，生成したルールはレビューコスト削減へ貢献しない (評価:2) と評価された一方，コード修正例や Confidence 値はどちらも貢献する (評価:4) と評価を受けた.

追加の質問として，実装ルールが古くなると感じるか否かについて質問を行った．その結果，「新規の開発技術や，フレームワーク，IDE の導入によってコーディングルールの変更が必要である一方で，コーディングルールの継続的な管理や指導，実装は困難である」という解答を得た．インタビュー結果から，今後必要な機能として，出力内容の改善や，開発のトレンドや状況に応じたコーディングルールの自動的な改善を行う必要がある.

[*5] https://goo.gl/forms/H7tYSifvWzemrDTo1

Listing 1: ETAG_WILDCARD を参照するために*を . に置き換えるコード修正例

```
- if (tag != null && !"*".equals(tag))
+ if (match != null && !CommonParams.
    ETAG_WILDCARD.equals(match))
```

Listing 2: Null チェックを行うために||を追加するコード修正例

```
- if (StringUtils.isBlank(id) || "
    PLATFORM_OPERATOR".equals(id))
+ if (id == null || id.equals("") || "
    PLATFORM_OPERATOR".equals(id))
```

Case 3- (1) 条件文を追加するために&&を追加するコード修正例

```
- if (field.checked != value)
+ if (field.checked != value && !field.
    disabled)
```

Case 3- (2) listdp の長さが 0 以上であることを確認するために>を追加するコード修正例

```
- if (listdp != null)
+ if (listdp != null && listdp.size() > 0)
```

Case 3: Changes from ! or !=

Case 4: keyPath が空の文字列でないことを確認するために!=を追加するコード修正例

```
- if (StringUtils.isNotEmpty(keyPath))
+ if (keyPath != null && !"".equals(keyPath))
```

Case 5: mId が空の文字列でないことを確認するために . と ,, ||と==を追加するコード修正例

```
- if (configBean == null)
+ if (mId == null || mId.equals("") ||
    configBean == null)
```

6. まとめと今後の展望

本稿は，パッチ開発者個々の実装ルールを検証するコストの削減を実現する REVAD bot を開発した．REVAD bot はプロジェクトから暗黙的な実装ルールを検出し，パッチ開発者が提出した `if` 文変更に対して，ルールに基づいた修正提案を自動的に行った.

REVAD bot が提供するルールとコード修正例の有用性を評価するため，ケーススタディとして，Open Service Catalog Manager プロジェクトに対して，REVAD bot を試行し，修正提案内容に関してプロジェクトのコア開発者にインタビューを行った．その結果，REVAD bot が提供したコード修正例のうち，40 パーセント (2/5) をコア

開発者がOSSプロジェクトで利用可能であると評価した．REVAD botによる提案を開発者が精査することで，不特定多数のパッチ開発者へ暗黙的なルールを提供することが期待できる．

本REVAD botは，プロジェクト固有の暗黙的な実装ルールを発見し，ルールに基づいて問題を検出を実現する．プログラミング言語の仕様に基づいた問題を検出する既存ツールと併用することよって問題検出範囲の拡大を可能にする．

今後の研究として開発者への一方的な提案だけでなく，開発者と相互に議論や提案を行うチャットボットを目指す．具体的には，

- 開発者への質問，また回答を反映した実装ルールへ追加，編集
- 複数プロジェクト間または開発者間での，実装ルール共有と相互的な改善

を行う機能を実装することで，プロジェクトだけでなく個人単位でも会話や相談を行いながら開発作業を補助するチャットボットを提供する．

謝辞

本研究の一部は，JSPS 科研費 JP18H03221, JP17H00731，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．また，本研究を進めるにあたり，Goebel Lorenz様にインタビューへの回答や今後の改善のためのアドバイスをしていただきました．ここに感謝の意を表します．

参考文献

[1] 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 (FSE'14)*, pp. 281–293 (2014).

[2] Bacchelli, A. and Bird, C.: Expectations, Outcomes, and Challenges of Modern Code Review, *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*, pp. 712–721 (2013).

[3] Boogerd, C. and Moonen, L.: Assessing the value of coding standards: An empirical study, *IEEE International Conference on Software Maintenance (ICSM'08)*, pp. 277–286 (2008).

[4] Han, J., Pei, J. and Yin, Y.: Mining Frequent Patterns Without Candidate Generation, *Proceedings of the International Conference on Management of Data (SIGMOD'00)*, pp. 1–12 (2000).

[5] Martinez, M., Duchien, L. and Monperrus, M.: Automatically extracting instances of code change patterns with AST analysis, pp. 388–391 (2013).

[6] Pan, K., Kim, S. and Whitehead, E. J.: Toward an understanding of bug fix patterns, *Empirical Software Engineering*, Vol. 14, No. 3, pp. 286–315 (2009).

[7] Raymond, E. S.: The Cathedral and the Bazaar, *Knowledge, Technology & Policy*, Vol. 12, No. 3, pp. 23–49 (1999).

[8] 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 (ICSE'11)*, pp. 541–550 (2011).

[9] 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 (ICSME'11)*, IEEE, pp. 504–507 (2011).

[10] Tao, Y., Han, D. and Kim, S.: Writing Acceptable Patches: An Empirical Study of Open Source Project Patches, *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME'14)*, pp. 271–280 (2014).

[11] Thenault, S. et al.: Pylint. Code analysis for Python.

[12] Ueda, Y., Ihara, A., Hirao, T., Ishio, T. and Matsumoto, K.: How is IF Statement Fixed Through Code Review? - A case study of Qt project -, *Proceedings of the 8th IEEE International Workshop on Program Debugging (IWPD'17)* (2017).

[13] van Rossum, G., Warsaw, B. and Coghlan, N.: PEP 8: style guide for Python code, *Python. org* (2001).

[14] Zhang, C. and Zhang, S.: *Association rule mining: models and algorithms*, Springer-Verlag (2002).

2 sample.java View

@@ -80,7 +80,7 @@ public void doFilter(ServletRequest request, ServletResponse response,

```
80  VOUserDetails voUserDetails = (VOUserDetails) httpRequest
81          .getSession().getAttribute(Constants.SESS_ATTR_USER);
82
83  -   if (mId == null) {
84          chain.doFilter(request, response);
85          return;
86      }
```

```
80  VOUserDetails voUserDetails = (VOUserDetails) httpRequest
81          .getSession().getAttribute(Constants.SESS_ATTR_USER);
82
83  +   if (mId == null || mId == "") {
84          chain.doFilter(request, response);
85          return;
86      }
```

図 3: 最初に Pull Request を通じて投稿されたソースコード変更

revad bot commented on 14 Dec 2017 • edited by revad-ueda

Your `if` change is `if (mId == null || mId == "")`

This bot suggests change candidates based on past 187 merged changes.

Suggest adding `.` with `==` (confidence: 7 / 32).
Recent fix example.

```
- configBean == null
+ mId == null || mId.equals("") || configBean == null
```

Suggest adding `||` with `==` (confidence: 5 / 32).
Recent fix example.

```
- configBean == null
+ mId == null || mId.equals("") || configBean == null
```

Suggest adding `==` with `==` (confidence: 5 / 32).
Recent fix example.

```
- configBean == null
+ mId == null || mId.equals("") || configBean == null
```

図 4: REVAD Bot によるフィードバック

2 sample.java View

@@ -80,7 +80,7 @@ public void doFilter(ServletRequest request, ServletResponse response,

```
80  VOUserDetails voUserDetails = (VOUserDetails) httpRequest
81          .getSession().getAttribute(Constants.SESS_ATTR_USER);
82
83  -   if (mId == null) {
84          chain.doFilter(request, response);
85          return;
86      }
```

```
80  VOUserDetails voUserDetails = (VOUserDetails) httpRequest
81          .getSession().getAttribute(Constants.SESS_ATTR_USER);
82
83  +   if (mId == null || mId.equals("") || configBean == null) {
84          chain.doFilter(request, response);
85          return;
86      }
```

図 5: フィードバックを基に行われた修正