

コードレビューのススメ

2021/12/03 TomoakiTANAKA

自己紹介

田中智章

- 富岡市出身
- 群馬工業高等専門学校を卒業後、他県の大学院を経て就職
- SE としてキャリアを始め、現在は事業会社勤務
- 主に Web、時々アプリ開発 (Rails や Flutter)
- ウィスキーと LEGO が好き



会社紹介

株式会社エバーセンス

Vision

- 家族を幸せにすることで、笑顔溢れる社会をつくる。

Products

- 妊婦さんへの情報提供をする
「ninaru」
- 子育てに必要な情報や機能満載の
育児アプリ 「ninaru baby」



本日の勉強会について

本日の勉強会について

そもそもなぜコードレビューが必要なのか？からはじめ

レビュー「される側」・「する側」それぞれで、

どういった点に気をつけるとより良いコードレビューになるかを説明します。

また、かんたんに僕（及び周辺）で実践している内容も紹介します。

明日からのコードレビューに何か活かせるものをお話できればと思っています。

目次

目次

コードレビューに関して、下記の内容をお話します。

1. 目的
2. 心得（される側）
3. 心得（する側）
4. 事例紹介
5. まとめ

コードレビューの目的

Q.

コードレビューの目的は一体なんでしょうか？
わざわざ時間をとってやる意味は何でしょうか？

コードレビューの目的

コード（≒ プロダクト）の品質を高めるため

- 仕様を満たしているかの検証
- コードでレベルで不具合がないかの検証
- その他
 - 拡張性 / セキュリティ / パフォーマンス / トランザクション、etc

メンバーの技術力を底上げするため

- 良いコードの認識を揃える
- フィードバックを通じた育成

Q.

コードレビューを実りあるものにするには？

A.

レビューされる側 / する側 双方に心構えや準備が必要（個人的には、される側の準備が最も大切）

勉強会資料 コードレビューのススメ

The screenshot shows a mobile browser interface. At the top, the status bar displays the time (22:12), battery level (9%), signal strength, and other icons. Below the status bar is the address bar with the URL 'qiita.com' and the search query '「コードレビュー」の検...'. The main header features the 'Qiita' logo and navigation links for 'ユーザ登録' (User Registration) and 'ログイン' (Login). A search bar contains the text 'コードレビュー' with a magnifying glass icon. Below the search bar are three filter buttons: 'すべて 5736' (All 5736), 'ストック済み' (Stocked), and '関連順 ▾' (Related). The first search result is a post by user 'nnnobuo' titled 'コードレビューの基本'. It includes a profile picture, the date (2021/08/22), and a summary: '>レビューとは? コードレビューとは、ソフトウェア開発活動の中の一つで、色々な目的ながります。もつとも一般的な目的は、コード ...'. The post has 0 LGTM reviews. The second result is a post by user 'teradonburi' titled 'コードレビュー虎の巻'. It includes a profile picture, the date (2019/11/19), and a summary: '的な初期フェーズの要求仕様や、クリティカルな決定の基礎になる仕様、使用頻度が高いモジュールなどを重点的にレビューします。以下に書く項目はレビューに負担をかけないようにする ...'. The post has 1262 LGTM reviews and 1 comment. The third result is a post by user 'shane' titled 'コードレビューの指針'. It includes a profile picture, the date (2021/05/23), and a summary: 'ことはないかなどをチェックできると良いです。上記には掲げていませんが知識の共有や教育といったことを目的とすることもあるかと思います。ちなみにgoogleのコードレビュー ...'. The post has 0 LGTM reviews.

世の中の情報の多くはレビューする側（こういった視点でコードを書いてください）というものが中心

この勉強会では、「レビューされる側」・「する側」双方の視点で、注意している点をお話しします。

レビューの心得（される側）

レビューの心得（される側）

1. 何はともあれセルフレビュー
2. 何をレビューしてもらいたいのか明確に
3. フィードバックは素直に受け入れよう
4. フィードバックは人格否定ではないと理解する
5. 対応しないフィードバックがあれば、理由を述べる

1. 何はともあれセルフレビュー

Bad

エンジニア：

- 「よし、コード作成完了。一通り動いたのでレビューに出そう」

レビュー：

- 「デバックコメントも残ってるなあ。print 文もあるし...このコードは、レビュー以前の問題では？」

1. 何はともあれセルフレビュー

Good

エンジニア

- 「よし、コードつくれた。まずは自分の PR を確認しよう」
- 「あ、デバックコメント残っているな。消しておこう」
- 「一通りセルフチェックはしたので、レビューに出そう」

レビュー：

- (一通り見て) 「特にパットみ変なところはないな。詳細見よう」

1. 何はともあれセルフレビュー

一通り動いて嬉しいのはよくわかりますが、その時点でのクオリティが高いことは稀です

デバックコード残りやタイプの発見、より良い変数名やメソッド化などの検討などなど。

一旦立ち止まって、自分のコード確認をぜひに！！

2. 何をレビューしてもらいたいのか明確に

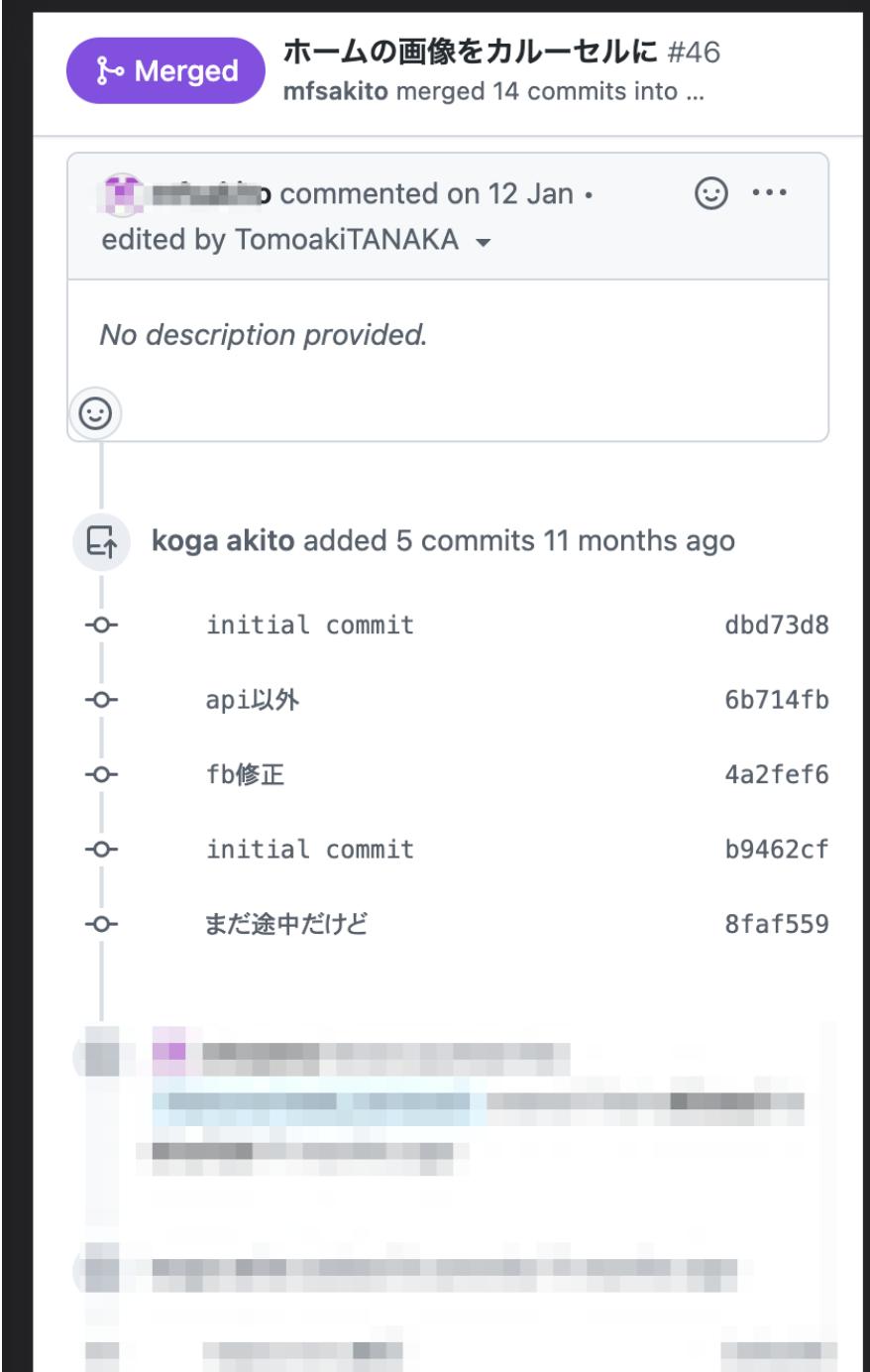
Bad

エンジニア：

- 「よし、コード作成完了。先輩、レビューお願いします 😊」

レビュー：

- 「うーん、突っ込みは沢山あるけど...どんなフィードバックを期待してるのかなあ？」



2. 何をレビューしてもらいたいのか明確に

Good

エンジニア

- 「よし、コード作成完了。概要やレビューしてもらいたい点を PR に書いて、っと。先輩、レビューお願いします 😊」

レビュワー：

- 「OK。そしたら ●● という観点メインでレビューするね」

A screenshot of a GitHub pull request page. At the top, there's a purple button labeled 'Merged' and a message in Japanese: 'ホームの画像をカルーセルに #46 mfsakito merged 14 commits into ...'. Below this, there's a comment section with a placeholder for a user's profile picture and the text 'commented on 12 Jan · edited'. The main content area is divided into sections: '概要' (Summary), '技術的な実装詳細' (Technical Implementation Details), '確認項目' (Checklist), and 'レビューしてほしい点' (Review Points). Each section contains a bulleted list of items.

概要

- homeの特集をいくつも表示させたい
- ので、カルーセル式にする。

技術的な実装詳細

- APIは特集のリストを返すようになり、
- それにあわせて特集をリストに格納
- リストからカルーセルにするパッケージに渡し、表示させる

確認項目

- iOS実機を確認した
- android実機を確認した

レビューしてほしい点

- コードチェック
- 設計の議論
- 文言確認

2. 何をレビューしてもらいたいのか明確に

コードレビューする側は、仕事の背景を全部理解しているわけではありません

コードの書き方なのか、動作チェックなのか、パフォーマンスやセキュリティなのか、でコードチェックの観点が変わります

どんな観点でチェックしてほしいか、自分がどんなコードを書いたのか、簡単に PR に書きましょう

3. フィードバックは素直に受け入れよう

3. フィードバックは素直に受け入れよう

Bad

レビュー：

- 「このプロジェクトでは ●● というアーキテクチャなので、このコードを xxx という風に直してください」

エンジニア：

- 「動いているし、前の PJ では指摘なかったし無視でいいや」

レビュー：

- 「指摘事項が修正されていないなあ。フィードバックしても無駄なのかな...」

3. フィードバックは素直に受け入れよう

Good

レビュー：

- 「このプロジェクトでは ●● というアーキテクチャなので、このコードを xxx という風に直してください」

エンジニア：

- 「承知しました！（前の PJ では指摘なかったけど、この PJ では指摘あるんだな、ふむふむ）」

3. フィードバックは素直に受け入れよう

フィードバックを 真摯に受け止めましょう

特に初学者や参加したての PJ では、まず話をききましょう。

あるいは、勝手に無視せずきちんと議論しましょう

(ピックアップした事象がちょっと悪いです)

4. フィードバックは人格否定ではないと理解する

4. フィードバックは人格否定ではないと理解する

Bad

レビュー：

- 「A 機能、B 機能、C 機能、それぞれ xxx という観点で修正をお願いします。」

エンジニア：

- 「ああ、また沢山指摘をもらってしまった。（何だか自分のスキルが足りない点を否定されているようだ...）」

4. フィードバックは人格否定ではないと理解する

Good

レビュー：

- 「A 機能、B 機能、C 機能、それぞれ xxx という観点で修正をお願いします。」

エンジニア：

- 「ああ、また沢山指摘をもらってしまった。（とはいえ、コードの品質が高くなるので、しっかり修正していこう）」

4. フィードバックは人格否定ではないと理解する

フィードバックは、人でなく、「モノ・コト」に向かっている

確かに、指摘が多いと凹むこともあるが、自分とコードは切り離して考えると良い

なお、フィードバックする側に関しては後述

5. 対応しないフィードバックがあれば、理由を述べる

5. 対応しないフィードバックがあれば、理由を述べる

Bad

レビュー：

- ・ 「A 機能、B 機能、C 機能、それぞれ xxx という観点で修正をお願いします。」

エンジニア：

- ・ 「（C 機能は後で手直しするから、一旦おいておこう）。A 機能と B 機能の修正をしました！」

レビュー：

- ・ 「????C 機能はどうなったんだ？」

5. 対応しないフィードバックがあれば、理由を述べる

Good

レビュー：

- 「A 機能、B 機能、C 機能、それぞれ xxx という観点で修正をお願いします。」

エンジニア：

- 「A 機能と B 機能の修正をしました！C 機能は、別ブランチとまとめて修正するので今回は保留しました。」

レビュー：

- 「OK。LGTM。」

5. 対応しないフィードバックがあれば、理由を述べる

指摘事項に対して、何もしない場合でも、きちんと理由を説明する

理由を説明したら、やはり実装が必要なこともある

そもそも指摘に対して真摯に向き合わないと、フィードバックされなくなる可能性も...

レビューの心得（する側）

1. info / want / must などで指摘のレベルを明確に
2. どんな観点でレビューしているか明確にする
3. 論理的にフィードバックする
4. 人格否定はしない
5. 自分のことを棚にあげて、コードに向き合う

1. info / want / must などで指摘のレベルを明確に

Bad

```
def get_user_name
  return 'tanaka'
end
```

rubyでは「get」をつけないので、外したほうがいいです

1. info / want / must などで指摘のレベルを明確に

Good

```
def get_user_name
  return 'tanaka'
end
```

[must]
rubyでは「get」をつけないので、外したほうがいいです
また、returnもつけないのがデフォです

1. **info / want / must** などで指摘のレベルを明確に

指摘には、色々なレベル感がある。たいてい、自分の思っている用に解釈してもらえない

コミュニケーションミスが起きる前提で、指摘にはラベルをつけて指摘を明確化するとよいです

2. どんな観点でレビューしているか明確にする

2. どんな観点でレビューしているか明確にする

bad

```
# メンバーを全件取得し、部署名とともに表示する
member = Member.all
member.each do |m|
  pp "#{m.name}, #{m.department.name}"
end
```

```
member -> members
N+1が発生しているので修正してください
```

2. どんな観点でレビューしているか明確にする

Good

```
# メンバーを全件取得し、部署名とともに表示する
member = Member.all
member.each do |m|
  pp "#{m.name}, #{m.department.name}"
end
```

member -> members

- ・命名の観点でおかしいです。
- ・複数のオブジェクトに対して、変数名が単数になっています

N+1が発生しています

- ・データ量が増えるとSQL発行数が増えパフォーマンス的に望ましくないです
- ・departmentの先読みを実施してください

2. どんな観点でレビューしているか明確にする

往々にして、指摘事項を同じレベルで捉えてしまいがち

命名なのか、アーキテクチャなのか、パフォーマンスなのか、どういった観点での指摘かを明確にし、相手がどういった点で抜け漏れが出やすいか伝えてあげる

3. 論理的にフィードバックする

Bad

```
<body>
  ...
  <script src="./main.js"></script>
</body>
```

jsの呼び出しは、`async` or `defer` を使ってください

3. 論理的にフィードバックする

Good

```
<body>
  ...
  <script src="./main.js"></script>
</body>
```

jsの呼び出しを同期的に行うと、レンダリングが進まなくなってしまう可能性があります。
今は1つのファイルだけですが、コピペで同期的に呼び出すコードが増える可能性があります。

動作に問題ないはずなので、jsの呼び出しは、`async` or `defer` を使ってください

3. 論理的にフィードバックする

フィードバックの理由を説明しましょう（特に初学者向け）

あれしろ、これしろ、だけのフィードバックは危険

自分がやったほうが早いといった危うい感情を持ったり、応用が効かない、本質の理解が甘いメンバーを生み出す可能性があります

4. 人格否定はしない

Bad

```
const citySelect = document.querySelector("#city1");
```

Selectは動詞でつかうことが多い単語です。
前回も指摘しているので、注意してください。

4. 人格否定はしない

Good

```
const citySelect = document.querySelector("#city1");
```

Selectは動詞でつかうことが多い単語ですね。
変数名は名詞としましょう。

4. 人格否定はしない

ダメ絶対、パワハラ

あからさまなことは少ないとと思うが、気づかずやってしまうこともあるので、今一度振り返ると良い

良いコードがかけなかった相手ではなく、かけていないという問題や、コードそのものに关心を向けましょう

なお、仕事へのスタンスや態度にフィードバックは、コードレビューのようなオープンな場でなく、1on1などプライベートな場で

5. 自分のことを棚にあげて、コードに向き合う

Bad

```
firebaseMessaging.getToken().then((String token) {  
    print("firebase token: $token");  
});
```

僕もよく間違うので恐縮ですが、タイポしてます
firebase -> firebase

5. 自分のことを棚にあげて、コードに向き合う

Good

```
firebaseMessaging.getToken().then((String token) {  
    print("firebase token: $token");  
});
```

タイプします
firebase -> firebase

5. 自分のことを棚にあげて、コードに向き合う

レビューするときに自分ができる、できないは気にしない

恥ずかしさや、申し訳なさがあれば、次回から気をつければいいだけ

気になった点を指摘してコード品質を高めていきましょう

コードレビュー事例

コードレビュー事例

実際に実施している事例を紹介します

- **セルフレビュー**

- かんたんでもいいので、テストケースを作成。一通り流す。

- **レビュー負担低減**

- リファクタする際に専用の PR を作成する

- **デザイン系**

- CSS や XIB ファイルのレビューは困難
 - 仮組みですすめ、最終デザインチェック

- **その他**

- コードスタイルはコード書く以前に決めておく
 - 設計（アーキテクチャ、DB、ディレクトリ構成など）も事前に

セルフレビュー時にテストケースを利用する

- かんたんにテストケースを作成（あくまで自分用）。
- コード完成後に、順番に実施。NGがでたら、コード修正し、最初から全項目チェック
- 雛形決めておけばさほど時間はからない
- なれたら github のチェックリストなどを使っても良い

C	D	E
		version
v1	9	c
v2	0	抜け漏れ
v3	0	知識
		思い込み
		段取り
		特になし
分類	項目	開発チェック
iOS	記事詳細（妊娠カテ） → footerである h	done
iOS	アーカイブ（妊娠カテ） → footerである h h	done
iOS	クリックして画面遷移すること（上中下の3枚）	done
iOS	コード上でエラーでていないこと	done
iOS	記事詳細 → 参照している\$single_postがちゃんととれていること	done
iOS	画面サイズが変わっても平気なこと（3パターンくらい）	done
iOS	記事詳細（子育てカテ） → footerでない	done
iOS	タグページ	56

目的別の PR をつくる

- リファクタリングや lint 系の適用、フレームワーク versionUp など、PR を完全分離
- 大量の変更検知はほぼ不可能
- やったこと（コマンドなど）をきちんと記載した PR を用意、、その手順をチェックする
- * リファクタリングは、ついででやりがちだが、可能であれば別 PR のほうがレビューしやすいので吉

NAKA was merged on 8 Jul 2020

NAKA was merged on 17 Mar 2020

NAKA was merged on 16 Mar 2020

[master ブランチ] Ruby 2.3 -> 2.6 / RoR 4.2 -

NAKA was merged on 24 Mar 2020

NAKA was merged on 16 Mar 2020

o 4.2.11

NAKA was merged on 16 Mar 2020

デザイン要素が多いコードチェックについて

- エンジニアが色や形状の細部をチェックするのは、職種上難しいという前提で考えている
 - エンジニアが色や形状の議論するのは若干不毛
 - 餅は餅屋
- デザイン（css や xib）は、まず仮組みの状態で開発を行う
 - エンジニアはレスポンシブ対応や、文字が多い場合などのエッジケースを確認
- ある程度開発がまとまつたら、デザインチェックを行いながら、フィードバック対応をする

その他

- コードチェックで揉めがちなことは、先に潰しておきましょう
 - コードスタイル
 - 設計（アーキテクチャ、DB、ディレクトリ構成、etc）
- 弊社では自社開発ということもあり、GoogleSpredsheet で簡単に作成し、その成果物をレビュー&共有しています

まとめ

まとめ

コードレビューに関して「される側」・「する側」の双方の視点で注意したほうが良い点を説明しました

コードレビュー関連で実践している Tips をいくつか紹介しました

開発業務において、何かしらの足しになれば幸いです

ご清聴ありがとうございました

質疑応答

おわり