伝わるコードレビューのために

2018/03/09

Aiming
黒木 慎介

自己紹介

- Aiming大阪スタジオ（ここ）勤務
- 新規タイトルのチームでwebエンジニアのリーダー
- RailsでAPIサーバー書いたり、k8sでサーバー構成書いたり、…

いきなりですが宣伝

- Rails Developers Meetup 2018に出ます
- 3/24,25
  - 私がしゃべるのは25日
- 東京の会場は埋まっちゃってるけど大阪の中継会場（ここ）は両日まだ席あります！！
- 私はRailsエンジニアの採用～教育についての話をします
  - 今日の発表はその内容から一部先出しです

今日話すこと

- コードレビューするときに気をつけている事
  - 「具体的にどう困るか」に落として言う
  - カッとなって本人のところに行く
  - 習熟を意識する
  - 褒める

「具体的にどう困るか」に落として言う

- コードレビューで開発のスピードを落としてしまう要因→やりとりの回数
- PRを提出したメンバーは次の作業に着手する
  - ので、常時PRを監視しているわけではなく、やりとりをした分だけのタイムラグが積み上がる
  - また、PRの修正と次に始めた作業を行き来するとスイッチングコストがかかる

やりとりの回数を減らしたい

- では、どのようなやり取りが回数を重ねてしまうのか？
  - 書き方などの宗教戦争
    - チームの規約を決めてRubocop（とか）に働いてもらいましょう
  - 合意形成までが長い
    - 今日はこっちの話
    - 昔見たコードレビューの話をします

昔見たコードレビュー（1）

「これなんで実装の片方しかテストしてないの」
「ロジック変わらないんで」
「でも実装は別々に書いたんでしょ」
「不安に思ったところはカバーできてるんで」
「いやでも他の人が修正して壊れるかもしれないよね」
「んーまあ僕は不安じゃないですけどじゃあ書きますわ」

昔見たコードレビュー(1)

- たぶん最初から具体的にどこがどう不安なのかを指摘できていれば、1-2往復くらいで済んだ

昔見たコードレビュー(2)

「ざっと見た感じHogeパターンを使ったほうが良さそう」

「大げさじゃないですか」

「どのへんが？」

「パターン使わなくても十分見通し良いと思うんで」

「うーん、でもこれSRP違反だよね ごめん見た目の問題じゃなかった」

「じゃあまあもう少し責務が明確になるようにします」

昔見たコードレビュー(2)

- 最初の指摘がふわっとしすぎ
  - 議論によって問題点が明白になるプロセスは必ずしも否定すべきものでもない
  - でもSRP違反を一発目に言えてればもっと速かったよね

昔見たコードレビューたち

- 類例多数
- 積み重なる不毛感
- 自分がレビューするときは一発で合意に到れるように、何がどう困るのかをちゃんと言うようにしようという決意

自分の最近のレビュコメ

[IMO]

メソッド名と同じ変数名は、rubyの仕様上「ここはメソッド呼び出しじゃないよね？違うよね？」っていう不安が高まってくるので（実際書き方をミスると簡単に無限再起が起きる）できればやめてほしいなあと
自分がよく使うのはresult スコープ4行だし多少ニュアンスが抜けるのは許容される気がする

自分の最近のレビュコメ

[SHOULD]

これを全部[redacted]にしていると、[redacted]判定の処理に正しく情報を渡せているかどうかがテストできてなく、たとえば間違って何か違う値を渡す処理を書いてしまっていても検出できませんよね
テストケースを増やさずにパラメータとかを変えるだけで検出できるテストに変えられるはずなのでやってみてほしいです

自分の最近のレビュコメ

- まあ割と終始こんな感じ
  - コード例を叩きつけたりもする
- 結構時間はかかってますが、やりとりはほぼ1往復に納まってます
  - 他の条件（メンバーとかチームとか）もあるけどね

応用：習熟度を見て判断

- レビュイーのスキルによっては「もうこれについては丁寧に説明する必要はなさそう」と判断できる場合もある
- 丁寧なレビュコメは時間かかるので、そう判断できるときには省略もする

カッとなって本人のところに行く

- そもそも文字のやり取りって結構つらくないですか
  - 情報量が少ない
  - 同じ言葉でも棘が立って見えることがある
- ならば、文字のやり取りをやめればよい
  - 本人のところに行って直接話しましょう
  - なんだったらペアプロしましょう

直接話すと

- 嬉しいこと
  - 反応が速い
  - コミュニケーションの手段が広がる
    - 図を書いて説明してもいい
    - その場でコードを書いて見せてもいい
  - 相手の反応を見られる
    - どこで思考が詰まるか、快・不快になるかを観察できる
    - それに合わせた話し方ができる
- 悲しいこと
  - 記録に残らない
    - あとで話したことをコメントに残しても良い

習熟を意識する

- 人間が1つの事に対して行える思考の量には限界がある
  - 私はこれをMPと呼んでます
- 同じことを考えたり意識するのにかかるMPは、繰り返すことにより低減していく
- そうして余剰するようになるMPで新たに追加で考えることができるようになる
- でもこの低減はすぐには起きない
  - 繰り返すことが必要

習熟を意識する

- 相手の最大MP以上のことを教えても頭に入らないし、やってもらおうとしてもできない
- 低減が十分に進んでいない場合、前に言ったことをまた考えられず、同じミスをしてくる事がある
- そういう状態なんだと思って、**辛抱強くやるのが大事**
- （学ぶ側としては素振りが大事みたいな話し）

褒める

- 良くないことの指摘だけしてると雰囲気が悪くなってくる場合がある
- 開発のテンション維持は大事
- フィードバックは正負両方有効
  - 良い行動は強化したい
- 前言った事ができるようになってたらそれはちゃんと言いたい

今日話したこと

- コードレビューするときに気をつけている事
  - 「具体的にどう困るか」に落として言う
  - カッとなって本人のところに行く
  - 習熟を意識する
  - 褒める