

# アホのムード

よくないレビューの例と  
レビューで折れない  
メンタルづくり



ACCESS 技術書典同好会

著：tonionagauzzi

# アイのムチ

— よくないレビューの例とレビューで折れないメンタルづくり —

[著] ACCESS 技術書典同好会

技術書典 12 (2022 年冬) 新刊

2022 年 1 月 22 日 ver 1.0

## ■免責

本書は情報の提供のみを目的としています。

本書の内容を実行・適用・運用したことで何が起きようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

## ■商標

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。

また本書では、™、®、©などのマークは省略しています。

# まえがき



「こんにちは、アメです！ 本書をお読みいただきありがとうございます！」



「みなさん、普段レビューしてますか？」



「見る側の方も、見てもらう側の方も、両方の方もいますよね」



「マリアです。プロジェクトマネージャーをしています」



「私のチームでは、ソフトウェアのソースコードレビューをしています」



「それも長くやってると……色々あるよね、アメさん？」



「そうなんです」



「レビューって、人によって気にする点や出すときの心理が違いますよね」



「そんなレビューに楽しく前向きに取り組むため、私たちのチームで起きた出来事を紹介します」

## 登場人物

### エマちゃん



- 睡眠は 1 日 9 時間必要なエンジニア 1 年生
- 映画とお笑いと甘いものを食べるのが大好き
- 6 歳上のお姉さんがいる

### アメさん



- ゆるふわだけど効率重視な 4 年目のエンジニア
- こだわりは少ないほう
- 片付けが苦手なので部屋の物を増やさない

### ムチさん



- 使うものにはとことんこだわりたい寡黙な 6 年目のエンジニア
- 意外とスポーツマン
- 夏は決まってスキーバダイビングに行く

### マリアさん



- 5 年前にエンジニアから転向したプロジェクトマネージャー
- スプレッドシートより Excel
- Slack のバッヂはすぐ消したい派

# 目次

<b>まえがき</b>	<b>i</b>
登場人物 . . . . .	ii
<b>第1章 よくないレビューの例</b>	<b>1</b>
1.1 攻撃的 . . . . .	1
1.2 試験形式 . . . . .	6
1.3 突き放し . . . . .	9
1.4 持久戦 . . . . .	12
1.5 一貫性がない . . . . .	14
1.6 独裁の場 . . . . .	17
1.7 論点がズれている . . . . .	19
1.8 一方通行 . . . . .	21
1.9 よいレビューとは . . . . .	24
<b>第2章 レビューで折れないメンタルづくり</b>	<b>25</b>
2.1 心理的安全性を高める . . . . .	25
2.2 完璧や正解を求めすぎない . . . . .	33
2.3 戦いをせず協力を求める . . . . .	37
<b>あとがき</b>	<b>42</b>

# 第1章

## よくないレビューの例

本章では、よくないレビューの例を取り上げます。

ここでいうよくないレビューとは、人間関係を悪くしたり、不必要的悩む時間を生んだり、レビューを依頼した人のメンタルやモチベーションを不調にするようなレビューのことです。

### 1.1

#### 攻撃的

エマちゃんは、チームに配属されて1か月の新人さんです。

勉強期間も終わって徐々に商用アプリ開発を任されるようになり、一生懸命やってフルリクエストを出しました。



「実装終わったのでレビューお願ひします！」



「アメさん、見てあげれる？」



「ちょっと忙しくて……。ムチさんに見てもらって」



「わかりました！」



「アメさんがよかったなあ……。ムチさんちょっと怖いんだよな……」

こういうシーン、みなさんのプロジェクトでも心あたりがあるかもしれません。なぜ、エマちゃんはアメさんにはお願いしやすくて、ムチさんにはお願いしにくいのでしょうか。

おや、そういうている間にムチさんからレビューコメントが返ってきたようです。

▼ @ Muchi commented

NGです。レビュアーやテスターに伝わる説明を書く努力をしてください。

▼ @ Muchi commented

冗長コードも多すぎます。既存の全体設計を事前に見てますか？

▼ @ Muchi commented

ここはif-elseの判定が仕様と逆ですが、わざと破壊しようとしていますか？

翌日、エマちゃんの姿が見あたりません。



「あれっ、エマちゃん今日どうしたの？」



「午前休って連絡きてましたよ」



「あ、ホントだ。体が重いって言っているのが気になるね。心当たりは？」



「昨日、ムチさんのレビューを受けて、遅くまで修正していたようです」



「なるほど。何かあったのかな。レビュー記録を見てみよう」



「うーん、厳しい指摘が多いみたいだなあ。スケジュールの心配もあるし、ムチさんに話を聞いてみよう」

エマちゃんが午前休を取ってしまったのは、甘えやメンタルが弱いからでしょうか。そう思われては気の毒ですよね。エマちゃんは自分なりによいものを作ろうと一生懸命取り組んでいたはずです。

マリアさんはムチさんを呼んで、コメントの真意を聞いてみました。



「ムチさん、エマちゃんの実装だけど、何か大きな問題がある？」



「いえ、そこまでは」



「レビューの指摘が結構重たいみたいだけど」



「あれは、少し勉強不足を感じたので、自分で考えて成長してほしいと期待しました」

ムチさんの言い分もよくわかります。甘いだけではやっていけないのがプロの世界ですからね。ですが、エマちゃんはやる気を削がれ、望まない理由で有休まで消費してしまいました。これはエマちゃんと会社の双方にとって、非常にもったいないことですよね。

では、このレビューのどこが問題だったのでしょうか。本書では、このようなレビューを攻撃的なレビューと呼びます。

### ■攻撃的なレビューとは

- ・レビュー者が辛口審査員のように実装者を問い合わせる
- ・実装者のコードを攻撃的に批判する

ムチさんは、エマちゃんの努力不足、考えが足りない、さらには悪意があって間違えたと批判するコメントをしています。

このようなコメントは、実装者のメンタルを壊し、チームに鬱憤をため、やる気の低下を引き起こします。



「とはいっても、ムチさんは元々そういうスタイルの人で、悪気はないから注意するのもなあ……。そうだ、アメさんにも見てもらおう」



「アメさん、このレビュー指摘だけど、アメさんからもフォローしてくれない？」



「はーい」



「ふむふむ、なるほど。エマちゃんはこれに悩んでたのか……。私ならこう書くかな」

▼ @Ame commented

ご対応ありがとうございます。

実装箇所は合ってますが、いくつかお願いしたいことをMust/Wantで伝えますね。

▼ @Ame commented

Must: if-elseを仕様通りに直す

▼ @Ame commented

Want: 類似実装の共通化、誤りを未然に防ぐための単体テスト作成

▼ @Ame commented

説明文はフォーマットを使うとよいと思いますよ。

わからないところは過去のPRを参考にしてみてください。



「おはようございます～。今朝はすみませんでした～」



「おはよう。大丈夫？ ムチさんの指摘がちょっと怖かった？」



「あっ、そうじゃないんです。でも、直しててなんか辛くなっちゃいました……」



「そういうときは、抱え込まず相談してね。私も最初、ムチさんにあんな感じでしごかれたのよね」



「えっ、そうなんですか！ 意外です」



「私のほうが後から入ったのもあるけど、あの、新人だから厳しいんじゃなくて、誰に対してもそうなのよね」



「でも、ムチさんはきっとこう伝えたかったと思うよ」

それからアメさんは、「私ならこう書く」と考えたコメント通りエマちゃんに伝えました。それを聞いて、エマちゃんはやる気を取り戻しました。

アメさんのコメントは審査ではなく、実装者に寄り添うような提案をしていますね。よかつた部分は認め、改善点は必須と任意を分けてヒントを出すに留め、また、実装者自身が考えて学ぶ余地を残しています。

エマちゃんは1年目で1つ1つの達成を自信に変えている段階なので、それを後退させるフィードバックはチーム力の後退にも繋がります。

気持ちのよいフィードバックとは、必ずしも全員一緒ではないのですが、コミュニケーションを重ねる中で、適切なバランスを見つけていきましょう。

## 1.2 試験形式



「実装終わったけど今日はアメさん不在だし、期限が近いからムチさんに出さなきゃ」



「今度こそ何も言われなければいいな！」

▼@ Muchi commented

NGです。このAPIが例外を吐いたときの考慮がされていません。

▼@ Muchi commented

このようなtypoが目立ち、インデントも不揃いです。全体的に不合格。



「ぐすん……不合格なんて……。コンビニスイーツちゃんで元気になろう……」

ある日の朝会の後、マリアさんはエマちゃんを呼びました。



「エマちゃんも入って1ヶ月経ったね。何か困ってことある？」



「私はどうすれば1人前になれるでしょうか……」



「というと？」



「こないだムチさんにレビューで不合格と言われちゃって、早く合格して皆さんのお役に立ちたいんです」



「うへん、不合格って言い方は厳しいなあ……。またムチさんに真意を聞いてみようかな」

ところがムチさんは席にいなかったので、マリアさんはアメさんのところに行き、経緯を説明しました。



「ということがあってね、アメさんはどう思う？」



「エマちゃんは既に私たちのメンバーですし、私は不合格とは思ってないですよ」



「エマちゃんは私からフォローしますね」



「いつも悪いね、アメさん」

### ■試験形式のレビューとは

- ・レビューの最後に「合格」「不合格」みたいな用語が飛び交う
- ・タイプミスや改善点を見つけて指摘することを、失敗の通達と捉えている

これは知識が浅い人やニューフェイスがいると起こりがちですが、エマちゃん、ムチさん、アメさん、マリアさんの4人は、立場こそ違えど「ソフトウェアを完成させ、ユーザーを満足させる」という目的は一緒なはずです。

その中で試験のようなレビューをすると、実装者はユーザーではなくレビューの満足度を高くすることに注力したり、試験に合格することが目的と錯覚します。

そして、試験に落ちそう、もしくは落ちたと感じてやる気を損ないます。協調的であるべきメンバーの和が、いつの間にか乱れてしまうのです。



「エマちゃん、それおいしい？」



「あ、これですか！」



「この前落ち込んでるときコンビニで見かけた新作のしらたまあんぱんです！」



「なんかホッとする味なんですよ～！」



「いいね。私も食べてみようかな。ところで、この前ムチさんにレビュー出してた期限の近いあれ、どうなった？」



「時間がなかったので、ムチさんに続きを引き取ってもらいました……」



「えーん、全然できなかったの思い出したら悲しくて、もう1個食べたりになりました！  
買ってきます～」



「待って待って！ あれね、実は私も途中まで見てコメントしかけてたの。これ見て」

▼ @Ame commented

気になったところにコメントしますね。

▼ @Ame commented

1. このAPIが例外を起こしたときの考慮

▼ @Ame commented

2. typoやインデントずれ

▼ @Ame commented

これらを直していただければ、残りはLGTMです。



「LGTM……よかった、全然できてないわけじゃなかったんだ……」



「間違えたところは、次から気をつけよう」

それから、エマちゃんのしらたまあんぱんブームも終わり、体重を増やさずにすみました。

アメさんがエマちゃんに見せたコメントは、評価などの不必要的な言及を避け、そのとき必要な指摘だけを伝えるものでした。そして、やはりよかった部分は認めています。

何かを書いて校正を頼むのは試験ではなく、同じ目的を持つ仲間同士の協力作業なんです。

## 1.3

## 突き放し

それから1か月経ち、エマちゃんは困ったことを何でもアメさんに相談し、順調にスキルアップしていました。

そして、エマちゃんは少し大きな実装を任せられました。この間勉強したばかりのアーキテクチャーやデザインパターンを駆使し、わからないところはアメさんに質問しながら、なんとか仕上げました。



「アメさん、色々とありがとうございました！」



「いえいえ。早くムチさんからも Approve をもらえるといいね！」



「今回はアメさんの言う通りに作ったし、ムチさんも納得してくれるはず……！」

エマちゃんはルンルン気分で PR を作り、退社しました。

水曜日だし、ウェンズデーを活用して前から見たかった映画を見たり、ケーキバイキングに行ったりして、オフをエンジョイしました。

翌日、出社するとムチさんの指摘が来ていました。

▼ @ Muchi commented

この判定はビジネスロジックなので、サービスクラスでやる仕事じゃないです。  
もっとドメインを意識して書き直して下さい。

▼ @ Muchi commented

doTask()に2つ以上の責務があり関数名も適当すぎます。



「ぐす……アメさんに色々聞いて頑張ったのに書き直しだなんて……」



「エマちゃん、Approve もらえた？」



「アメさーん！（泣）」



「ど、どうしたの！？」

アメさんは泣くエマちゃんをなだめました。

後ほど、マリアさんがアメさんに声をかけました。



「エマちゃんと 1 on 1 をしたけど、ちょっと自信を失ってるようだね」



「はい、私がレビューして OK だったんですが、ムチさんにとっては NG だったよう  
で……」



「レビューのチェックポイントは 2 人とも同じなんだよね？ なんで分かれたの？」



「私はそこまで関心の分離を徹底しようと思ってなかったんですが、ムチさんは徹底し  
たかったようです」



「それに、エマちゃんは何も考えてないわけじゃなく、エマちゃんなりの考えでこうい  
う記述してるんだなって感じて、尊重してあげたかったです」



「だから、今は OK を出して、高度な実装は追々学んでくれるといいな～と思ったの  
が、私側の感覚ですかね」



「なるほどね。アメさんから見て、ムチさんのやり方はどう思う？」



「厳しへに見てくれるのはありがたいんですが、頑張って書いたのに突き放すような言  
い方はよくないかなって思いますね」

### ■突き放すレビューとは

- ・「やり直し！」のように、実装者が拒絶と感じる指摘をする
- ・具体的な修正指示が含まれていない

どんな実装にも、実装者にとってはベストの実装だったり、実装者なりによく考えていることが多いので、否定だけはよくありません。具体的な修正指示や例を出してあげる方が親切です。

また、答えをそのまま教えればよいとも限りません。キーワードや例のみ示し、実装者が自分で考え、答えはネットや本で見つけるほうが、スキルアップやモチベーションに繋がることがあります。



「もし私だったら、こういうふうに書きますね」

#### ▼ @Ame commented

doTask()の設計から実装までありがとうございます。  
今回もコメントをMust/Wantで伝えますね。

#### ▼ @Ame commented

Must: この判定をビジネスロジックのクラスに移す

#### ▼ @Ame commented

Must: doTask()に2つ以上の責務があるので、ダウンロードと開く処理とに分ける

#### ▼ @Ame commented

Want: doTask()を4つに分ける

#### ▼ @Ame commented

4つに分けるのは私も考えただけで未確認なので、よい実装があれば教えてほしいです！  
過去の似たような実装や、同じアーキテクチャやデザインパターンで作られているオープンソースはたくさんあると思うので、迷ったら調べてみてください！

アメさんが心がけたのは、エマちゃんが自分で考えて進めたことを尊重しつつ、必須事項に関しては具体的な修正指示、そうでないものは「気になる」「調べてみよう」と思ってくれそうなコメントをすることです。

ヒントから答えを見つけ、それをアメさんに教えることでエマちゃんにとって知識や考え方を深めつつ、誰かの役に立ったと自信まで深めるチャンスなのです。

どうすればエンジニアがやる気を引き出せるかを、アメさんは普段から考えているようです。

## 1.4 持久戦

さて、レビューを受けるのはエマちゃんだけではありません。

マリアさんとアメさんは、今後のプロジェクト計画についてプロジェクト責任者やステークホルダーたちと対面でのレビューを受けました。しかし、その会議は予定終了時刻を2時間もオーバーした上、いくつかの議論は行われませんでした。



「遅くまでお疲れ様」



「あの方々とやると、いつも予定時間を超えますよね～」



「ホントにね、やんなってくる」



「それに最初は段取りよかったですけど、途中から同じ話の繰り返しでした」



「終わりのほうは、向こうも疲れてたしね」



「しかも結論がいくつか出てないと思うので、別の日に再レビュー設定することになります」



「最初の1時間で切り上げてもそんなに変わんなかったよね」



「決めた予定だから最後までやりたかったんでしょうね」



「今頃向こうは、やった！って満足感だけ残ってるんじゃない？」

実はこのレビュー、向こうも満足感ではなく、マリアさんと同じようなことを感じ、愚痴っていたのでした。

つまり、誰か特定の人の振る舞いがよくなかったわけではなく、お互いにこのまま続けると非効率とわかっているながら、引っ込みがつかなかったのです。

### ■持久戦のレビューとは

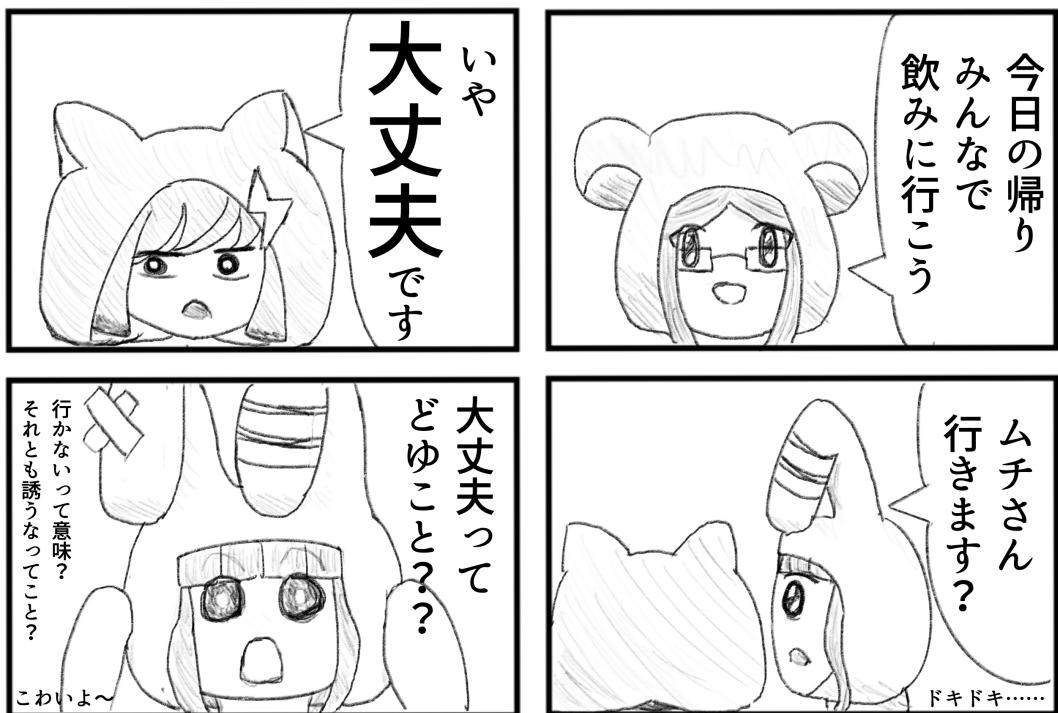
- ・やたらと長く、予定を超過する
- ・最初はやる気があるっても、時間が経つと疲れる
- ・実装者もレビューも、時間が経つにつれ不注意になるが、切り上げられない
- ・体力を吸われ、レビューの精度も低くなる
- ・充実感を伴って終わり、本質的な問題に気づかない

とはいっても、対面での議論を途中で切り上げるのは、思い切りの要ることです。スケジュール調整も必要なので容易ではありません。

最初に「この場ではこれだけ話す!」「議題が逸れたらチャットツールやチケットなど、非同期な方法で続きを議論する」など、あらかじめ合意しておくことが重要でしょう。

## 飲みのお誘い★

絵: tonionagauzzi



## 1.5

### 一貫性がない

エマちゃんたちの会社は、360度評価が採用されています。今日はムチさんがメンバーからの評価フィードバックを受ける日です。



「ムチさんのプロジェクト貢献度、コードの質のよさ、ドキュメントの更新頻度は高い点数だったよ」



「はい」



「よい点は、ベースの知識の豊富や、解決力、実装の早さなどだって。私もその通りだと思う」



「はい」



「でもね、改善点として、円滑なコミュニケーションがややしづらい、他の人との交流が少なく話しかけづらいって書かれてたのが気になるね」



「なるほど」



「改善点について、何か思うことは？」



「特に……ないです」



「（ムチさん、毎回こうなんだよな～。本心が読めない……）」

席に戻ったムチさんは、さっさと帰り支度を始めました。

実はムチさん、帰りに書店でインテリア雑誌を立ち読みし、よい本なら買うのが日課です。

今日も自室の観葉植物を替えたくて新刊を読んでいますが、どうにも落ち着きません。360度評価をきっかけに、過去数ヶ月のことが気になり始めたようです。



「今まで通りやってるはずなのに、新人が配属されてからなんかペースが狂うなあ……」



「アメさんが人によって態度やレビューの許可基準を変えるから、僕がうまくフォローしてみるつもりなんだけど」



「このままレビュー品質が担保されなくなる前に、PMに警告しとこうかな」



「……じゃ、リスクはバグとかのことじゃないんだね」



「ええ。アメさんをレビュアーとして教育すべきだと思います」



「（ムチさんが、気になることを自分から話してくれたのは初めてだなあ）」



「（でも、エマちゃんのフォローとか頑張ってくれてるアメさんに、何か足りない点があるんだろうか……）」



「アメさんのどのあたりが足りてないの？」



「レビューが行き当たりばったりなところかと」



「行き当たりばったり……か。私にはよくわかんないけど事実としてムチさんはそう感じてるってことだね？」



「はい」



「（もしかして、この前の OK と NG が 2 人の間で分かれた話が何か関係してるのが  
なあ）」



「わかった、アメさんにも意見聞いてみるね」



「よろしくお願ひします」

マリアさんは、レビューに一貫性がないのではと感じました。

一貫性のないレビューとは、次のような状態を指します。

.....

### ■一貫性のないレビューとは

- ・複数のレビュー者がおり、人それぞれレビューポイントや評価基準が大きく違う
  - ・同じレビューでも一貫性がない。たとえば、グローバル変数がある日は禁止し、別の日は許可する
    - ・前と同じ問題が見つかっても、前と同じ指摘が行われない。言うのを躊躇う。同じ指摘は2度と行われない
- .....

これらは人数が増えれば起こりがちです。

ここで間違えないでほしいのが、「評価基準が大きく違う」と書いたように、多少は違ってよいのです。人間の目によるチェック漏れをゼロにはできませんし、レビュー者のスキルや経験に応じてレビューポイントが違うのも当然です。

ですが、一貫性を損ないすぎている場合、実装者が理不尽さを感じ、鬱憤を溜めたりやる気の低下を引き起こしかねません。

また、このようなレビューはチームの士気だけではなく、品質低下につながるリスクもあります。



「記録を見てみると、アメさん自身の OK/NG の基準は日によって少しはバラつくけど、そんなに気にならないなあ」



「でも、アメさんとムチさんの間では、OK/NG の基準がだいぶ違ってるみたいだ」

マリアさんはどちらの基準がよいか悩みましたが、ムチさんの厳しい指摘によってコードの品質が一定レベルに保たれていると感じたので、ムチさんの基準が望ましいと考えました。

## 1.6

## 独裁の場



「アメさん、ちょっといいかな」



「はい、なんでしょう」



「ムチさんから、アメさんもムチさんと同じように厳しくレビューしてほしいと言われてるんだけど」



「ほお、そうなんですか」



「アメさんはどう思う？」



「確かに、レビュー基準は合わせたいですよね」



「どんなふうに合わせる？」



「例えば、Google 社の公開する Code Review Developer Guide をみんなで読み合わせたりとか」



「なるほど、じゃ、アメさんもムチさんに賛成なんだね」



「あ、でもムチさんには合わせられないです」



「誰かを絶対正しいみたいにすると、独裁的なレビューになってよくないと思うんです」



「独裁的なレビュー？」

■独裁的なレビューとは

- ・誰か1人が権力を握り、レビューを支配している
- ・権力者に承認されることがレビューの目的となる

誰かが権力者になると、権力者なしで意思決定できないチームになったり、開発者のキャリアアップを妨げ、チームの人数が増えても成果も上がらなくなる傾向があります。

アメさんは、ムチさんに支配されるのを恐れているわけではありません。ムチさんに依存せず、各メンバーが自立して考え、挑戦し、改善を続けていける環境を望んでいるようです。



「なるほどね、じゃ続きはアメさんとムチさんで直接相談してもらえるかな？」



「はい、そうします！」

## 1.7

## 論点がズれている



「ということで、ムチさん、よろしくお願ひします！」



「はい」



「まず、私たちのレビューのやり方で、認識が合っている部分を明確にしません？」



「私は重要じゃないチェックに時間をかけなくてよいと思ってます。それはムチさんも一緒になんじゃないかと思います」



「重要じゃないとは具体的に？」



「例えば命名規則とか、プラグインバージョン、スペルミス、マジックナンバーとかです」



「そんなの人力で見てちゃダメだね」

#### ■論点がズれているレビューとは

- ・ソフトウェアの出荷には重要でない、コーディングの細かい部分ばかり言及される
- ・自動レビューで貰える箇所を、手動で頑張ってチェックする

もちろんその洗い出し自体は大事ですが、そこばかり見すぎていると重要なものを見逃したり、時間を予想以上に費やしてしまいます。

これは几帳面な性格的人がいると起こりがちです。人が見るレビューでは、ソフトウェアの品質に満足しているか、メンテナンス性、複数のやり方から最適な方法の選択といった、人でなければ判断できないことに集中しましょう。



「それ以外で、ムチさんと私が認識が違ってる部分って、どこだと思いますか？」



「アメさんは新人に優しすぎたり、以前 NG だったものも OK で返すことがあるよね」



「う～ん……確かに状況に応じてスピードを上げたり、反対に性能や品質を優先したりはします」



「では、その状況判断の根拠は何？」



「えーと……正直、直感ですね」



「それがよくないよね。レビューは責任持ってやってくれなきゃ」



「すみません、わかりました。今後はムチさんの指示通りにレビューします」



「お願いします」



「（またなんかペース狂う……）」



「アメさん、ムチさん、大変だよ、市場トラブルが起きた！」

## 1.8 一方通行

マリアさんはプロジェクトメンバー全員を集め、今大規模な障害で全ユーザーがアプリを使えなくなっていること、至急修正が必要なことを伝えました。

エマちゃん、アメさん、ムチさんのチームは、大掛かりの修正をタスク分けしてスピーディーにこなしましたが、先ほどのレビューの話をしたばかりで時間がなく、レビューはムチさんがすべて引き受けました。

▼ @ Muchi commented

@Ame

このユーティリティクラスを作った意図が伝わりません。

▼ @ Ame commented

今後他のモジュールが呼ぶ可能性があるので作りました！

▼ @ Muchi commented

それなら必要に応じてリファクタリングすればよいと思います。

こんなイケないことするくらいなら、今はコンパクトな参照関係にすべきです。

▼ @ Ame commented

わかりました、修正します！



「なんか、アメさんがムチさんにしごかれる流れになってるけど……チーム大丈夫なのかな」

▼ @ Muchi commented

@Ema

UIの変更が含まれてませんが、スケジュールはUI含め本日までです。

レビュー都合もあるのでボヤボヤせず早めに出してください。



「ひいん、人の心配してる場合じゃなかったあ……」



「でも、ムチさんのコードって誰がレビューしてるんだろ……」

エマちゃんはメインブランチのマージログを確認しました。すると、ムチさんはレビューなしでメインブランチに入れしていました。



「アメさん！ ムチさんのコードって誰もレビューしなくていいんですか？」



「う～ん、したほうがいいよね……でも今そんな余裕ないし、ムチさん一番先輩だから  
まあ大丈夫かなって」



「なるほど、そういうものなんですね」



「あっ、ごめんねエマちゃん！ そういうものではなく、見過ごしてただけでこれは本  
当はダメなパターンなの」



「えっ？」



「一方通行のアンチパターンってよく呼んでるけど、余裕がなくてすっかり忘れてた  
わ。思い出させてくれてありがとう！」

### ■一方通行のレビューとは

- ・レビュアーのコードを誰もレビューしない
- ・上下関係があり、若手が先輩のコードをレビューしない
- ・年齢や経験の高い人が自分のコードを unreviewed のままマージする

これは年功序列や経験差がハッキリしたチームで起こりがちです。

実は経験が浅い人こそ、他の人にはない観点で問題に気づくことがあります。みんなのチーム  
でも若手が思わぬ改善点、たとえば可読性の低さやブラックボックス化の懸念などを見つけてくれ  
ることはよくありますよね。

ペテランの書くコードにも意外な問題が潜んでいるものです。厳しく述べると、長くいる人は成  
長してスキルが優れているというより、単に**仕事に慣れただけ**ということもあります。

そういう互いの立場を理解して、足りないところを埋め合うほうが、チーム内の雰囲気も品質  
もよくなるのです。



「だからエマちゃん、もし手が空いてたら私やムチさんのコードをレビューしてくれない？」



「えっ、私なんかがいいんですか？」



「エマちゃんしか気づけないこともたくさんあると思うの、だからお願ひ。ムチさんは私から説明しておくわ」



「わかりました、やります！」

そして、はじめてレビューーとなったエマちゃんでした。



「なぜエマさんがレビューをしているんですか？ 誰の指示ですか？」



「（あっ……説明する前に見つかっちゃった）」



「私です。エマちゃんの成長と一方通行レビューにならないようにお願いします！」



「（また独断で……。新人に教える前に自分が学んで欲しいんだけど……）」



「（ま、新人に細かい実装のこと教える手間が減るのならいいか）」

エマちゃんは最初ほとんど Approve でしたが、やがて小さな部分に目が届くようになりました。そして、アメさんとも技術的な踏み込んだ話ができるようになりました。ムチさんはまだ怖いようですが……。

一方、アメさんとムチさんの間も変化しつつあります。ムチさんはアメさんに厳しくはじめ、アメさんはムチさんのやり方に合わせるのを最初拒んでいたのに、今では一切反論せず同意しています。

アメさんには一体どんな考えがあるのでしょうか。次章に続きます。

## 1.9 よいレビューとは

よくないレビュー集はここまでです。では、よいレビューとは一体何を満たすレビューでしょうか。筆者は、以下の要素を持つものと考えます。

1つめは、建設的なフィードバック、健全で活発な議論があることです。コメントを書く人は、自分の考えを明確にしつつ、自分が正しいとは主張しないようにします。状況に応じていろんな代替案や回避策が存在することを認め、修正の強要を控え、どう思う？などと問いかけます。もし実装者から反対意見が出た場合は、本やインターネット、先人の知恵を見て、2人の間でよい着地点を探しましょう。それが難しそうなら、第三者に客観的な意見を求めるのがよいでしょう。

2つめは、レビュー者が実装者に共感していることです。いかなる方法でも、実装者が実装にかけた時間や労力を称賛し、強い口調ではなく親切で控えめな口調で接することが大切です。たくさんコメントした場合は、フルリクエスト以外の場所、たとえば対面やオンライン会議、チャットやメールなどで積極的に連絡を取り、誤解が生じることを防ぎます。そうすればきっとつらい思いをする人は減り、レビューの雰囲気や会話の方向性がとてもポジティブなものとなるでしょう。

3つめは、すべての人が対等であることです。職種や職歴、スキルレベル、入社時期、サラリーなどに影響されず、全員が同じ基準、同じ目線で、かつお互い足りないものを埋める場所としてレビューを活用します。本章で取り上げたような、スキルテストや評価の場として利用することは不適切です。

## 第2章

# レビューで折れないメンタルづくり

### 2.1

#### 心理的安全性を高める

エマちゃんたちのチームは、紆余曲折ありながらもなんとかトラブルを収束させました。



「みんなお疲れ様。遅くまで対応してくれてありがとう」



「今後こういうことを未然に防ぐため、ふりかえりを開かないとね」



「ふりかえり……？」



「エマちゃん、やったことなかった？」



「そういえば、最初は週1回やってたけど、忙しくなってから忘れてたね」



「ふりかえりは、アジャイル開発では必須と言われるチーム改善のための会議で、KPT や YWT などの方法で定期的に行うものよ」



「どんな手法にも共通しているのは、チーム全員がよりよいやり方のアイデアを見つけ、提案することだよ」



「私のような新人でも発言していいんですか？」



「もちろんよ。レビューと同じでプロジェクトの問題にも、入ったばかりの方方が気づきやすいし」



「あと、そういう人の不安を取り除く会もあるの」



「だから、感じたことはそのまま言ってね」



「はい、わかりました！」



「でも、ふりかえり中に特定の人を非難するとか、盛り上がって関係ない話に行くのは NG だよ」



「そうですね。多くの問題は人ではなく、チームのルールやプロセスにありますし、決めた時間どおりに進行するのも大事ですよね」

そして、さっそくふりかえり KPT 方式で行われました。

エマちゃんは、忙しいのが過ぎ去ってよく疲れそうとか、技術的に成長したなどを Keep で挙げつつ、レビューに時間がかかってしまうことを Problem に書きました。

Try を書く時間では、レビュアーと実装者が直接話して疑問点を解決する、ペアプログラミングをするなどのアイデアが挙がりましたが、エマちゃんはどうもスッキリしません。ふりかえりが終わってから、アメさんに相談しました。



「すみません、個人的に聞きたいんですが……」



「なあに？」



「今回、アメさんもだいぶムチさんの指摘を受けてたじゃないですか」



「だねえ。久しぶりでなんか懐かしい気分だった♪」



「つらくならなかったんですか？」



「ん～、そういうのはなかったかなあ。細かく見てくれたし、助かります！ みたいな気持ちで」



「すごいです。私だとなんか怖くて……」



「あっ、エマちゃん、最初ムチさん怖いって言ってたもんね。今もなの？」



「はい……あまり印象変わってないです。ムチさんと直接話したりペアプロするの、気が重いです」



「そっかあ。ふりかえりの Try ジャ実は解決しないのね」



「じゃ、エマちゃんは、ムチさんにもっと優しく接してほしい？」



「うへん……そのほうが安心ですが、アメさんや他の人も厳しく言われてるんだから、私だけって難しいのかなあって思います……」



「それに、友達になってほしいわけでもないし、むしろちゃんと言ってくれたほうが成長できる気もして……」



「なんだか、うまく言葉にできないんです」



「そうねえ……たとえば、きつく言われても気にしなくなる方法があるならどう？」



「あっ、鋼のメンタルってやつですか！」



「ちょっと違うわね（笑）」



「きつく言われたときって、責められたとか、評価が落ちたって風に捉えてない？」



「はい、そう思っちゃうことが多いです」



「もし、そうならないって確信できれば、さっきエマちゃんが自分で言ったように、成長のためとか、細かく見てくれて助かったとかって思えない？」



「そうかも……」



「エマちゃんが必要なのは、心理的安全性なのかもしれないわね」



「心理的安全性、ですか？」



「詳しくは、マリアさんに教えてもらって。その人、実はこういう話好きだから。知らなかったでしょ？」



「はい！ マリアさんに聞いてみます！」

心理的安全性とは、恐怖や不安を感じずに自分の意見を安心して伝えられる状態を指します。つまり、エマちゃんは今、心理的安全性が低い状態なのです。

心理的安全性が低い状態で感じる不安は、大きく分けると4種類あります。

### ■無知だと思われる不安

誰かに質問や相談をするとき、「そんなことも知らないのか」と思われるんじゃないかと感じる状態です。もし言わなくても、査定の低評価に繋がるのではないかと、一度感じた不安は勝手に膨らんでいきますよね。

その結果、わからないことを気軽に相談や質問できなくなり、大きなミスや孤立に繋がるかもしれませんのです。

### ■無能だと思われる不安

何かをやったとき、「この程度なのか」と思われるんじゃないかと感じる状態です。

この不安はあらゆるネガティブな判断を引き起こします。ミスをしても報告せず隠蔽する、ミスを素直に認めない、本質から離れた努力をして有能に見せようとする、などです。

その結果、大きなトラブルや損失に繋がる可能性がありますし、発覚が後になればなるほど大ごとになってしまうのです。

### ■邪魔だと思われる不安

発言や指摘をするとき、他人の足を引っ張って嫌われるのではないかと感じる状態です。

その結果、発言や指摘が少なくなり、アイデアの提案や他人のミスのフォローなどができなくなります。それにより、チームの一部の人の考え方ばかりが適用され、それ以外の人にとって居心地の悪い状態になったり、チームの改善機会が失われてしまいます。

### ■ネガティブだと思われる不安

発言や指摘をするとき、マイナスなことは言ってはいけないんじゃないのか、せっかくのよい雰囲気を壊すのではないかと感じたり、それを繰り返すことでネガティブで後ろ向きな人だと思われるのを恐れる状態です。

最終的にチームをよくできる話でも、少しでもマイナスな要素が含まれていると発言や指摘をできないので、チームの改善機会が失われたり、本来持っている問題解決能力を発揮できず終わってしまいます。



「同じ状況でも感じ方は人それぞれだけど、心理的安全性が低いと、こんなにたくさん の問題があるんだよ」



「ひえー！ 大きなミスやトラブルなんて起こしたくないです！」



「それはみんな同じだよ。だから、エマちゃんが今感じている不安は、早く無くさないと後が大変だよ」



「そうですね……どうすればムチさんともうまくやっていけますか？」



「ムチさんは悪意があってそう言ってるんじゃなく、エマちゃんに期待して言ってくれると思うけどね」



「それはわかるんですが……どうすればその、信頼性？ が高くなるんでしょう」



「心理的安全性ね！」

よく誤解されますが、やわらかい言葉を選んで使う、はっきり言うのを避ける、賛成するなどの優しさによって心理的安全性が高まるわけではありません。

心理的安全性が高いとは、無知・無能・邪魔・ネガティブと思われそうな行動をとっても安全であるとメンバー全員が感じている状態です。お互いに踏み込んだ話し合いをしたり、助けを求めたり、時には厳しいことを言ったり、建設的な反対意見を出せる状態でもあります。

また、そうなることがゴールではなく、組織として高いパフォーマンスを出すことが目的で、そのための土台と考えるのがよいです。

心理的安全性を上げるために、私たちができるのは以下のようなことです。

### ■心理的安全性を高めるための意識づくり

- ・失敗が起こることを受け入れる
- ・人や過去を責めず、プロセスやこれから成功する方法を考える
- ・うまくいかないのは学びや成長の機会だととらえる
- ・指摘は一定レベルのことができてるから受けられるもの、土台には好評価と承認があるととらえる
  - ・相手に好奇心を持ち、質問や意見を積極的にする
  - ・予想と異なることに好奇心や関心を持つ



「どれも私に足りてないことかもです……」



「エマちゃん、入社してから今まで許してもらえたかった間違いってある？」



「言われてみると、思いつかないです」



「そうだね。私もない。お客様や偉い人たちとも色々あったけど、1つでも許しても  
らえてなかったらここにいないよね（笑）」



「たしかに……」



「でも不安になるのって、これからが予測しにくいからなんだよね」



「私たちのいるソフトウェア業界は、VUCA と言われる不安定で、不確実で、複雑で、  
曖昧な状態の典型だから」



「その中で予測をして、確実な利益にするには、心理的安全性が高いチームであること  
が大前提なんだよ」



「なるほど、少しずつわかつてきました」



「でも、みなさんに好奇心を持っても、新人なので質問や意見をしづらいことが多いん  
ですが……」



「たしかに、意見しないほうが楽だもんね」



「あ、そういうつもりじゃなく……」



「エマちゃんのことじゃなくて、人は誰でも安全な方法を取りたがるってことだよ」



「黙ってて解雇された人は聞いたことないし、先輩の意見に従えば責任も感じないからね」



「でも、成果を出すチームの多くはミスが少ないチームじゃなくて、ミスを繰り返して成長してきた、報告が多かったチームだと言われているんだよ」



「そうなんですかあ」

人は誰でも、無意識に思い込んだり失敗を犯すものです。また、多くの人にとて、厳格なヒエラルキーは発言しづらさを伴います。

心理的安全性が低いと失敗の指摘が埋もれ、上層部や決定者にはうまくいっているという錯覚が生まれます。その結果、重大な失敗を引き起こすこともあるのです。

## ふたりの本名★

絵：tonionagauzzi



## 2.2

## 完璧や正解を求めすぎない



「アメさんとムチさんって、性格は違ってもうまくやってるようですが、2人とも心理的安全性が高いんですか？」



「私が見た限り、2人ともまだまだだね」



「えっ、そうなんですか」



「ムチさんは知ってるのとおり完璧主義で、的確な意見を出してくれるし、意見することも厭わない人なんだけど」



「一言が多くて、つい人を責めちゃったりね～」



「あはは……」



「あと、必要最小限の会話しかしないけど、それは自分の責任範囲が増える怖れかなって思うんだ」



「安心してもっといろんな事柄や変化に対して考え方を言って欲しいかな」



「アメさんは反対ですよね」



「そうだね。アメさんは謙虚な自信家で、人から多少何か言われたり失敗した程度じゃ気持ちが揺らがないから、積極的に動いたり場をまとめてくれる」



「それでこの前のトラブルのとき、ムチさんに色々言われても飄々とされてたんですね！」



「いや、そこがアメさんの欠点なんだけど」



「えっ、そうなんですか」



「意見が食い違って余裕がないとき、アメさんって自分の意見を簡単に引っ込めちゃうんだよね」



「こだわりが無いのか、議論が面倒なのか、単に詰めが甘いのかはわからないんだけど」



「実は意見を言いづらいんでしょうか」



「アメさんのことだから、それは違うかな。Best じゃなく Better でいいやって満足しちゃう感じかと」



「アメさんくらいのポジションなら、もうちょっと徹底的に問題を洗い出して報告して欲しいかな」



「反論しないのがよいとは限らないんですね」



「建設的な議論はどんどんあってほしいし、安心して鋭い指摘ができるのが本当の優しさだよね」

補足ですが、アメさんが自分の意見を強く出さないのは、実は **Better でよい** という判断をしています。

心の中ではこれが Best だという判断基準を持っていますが、深く追究するほどのこだわりがないのです。Best は趣味プログラミングの時間に全力で追究しており、仕事は「頼まれたものを効率的に作る時間」とドライに考えています。

でも、本人に聞いてもそんなことは言いません。「もっとよい実装を心がけます！」と、あたりがわりなく爽やかに言うでしょう。

皆さんのチームにも細かい性格的人がいる一方、そのような人もいるかもしれません。過去に心理的安全性が低い組織にいた経験からそうなっている可能性もあるので、スキルを持つ人が Best に挑戦したいと思えることも大事なのです。



「ムチさんの一言多いのとかもアメさんは全く気にしないし、私もムチさんはそういう人だって慣れてたから、今までうまくやれてだけど……」



「新しく来た人もそれで居心地がいいとは限らないってことだね」



「なるほど……チームのバランスがあるんですね」



「バランス。いい表現だね。どこまで完璧を求めるかも、このバランスでいくよって決めて、それを全員が納得してるのが大事だと思う」



「ちなみに、この話ってアメさんやムチさんにも直接言うんですか？」



「もちろん、2人にはそれぞれ 1on1 でよいことと改善点をオープンに伝えるよ。私が感じた課題を隠してたら、それこそ心理的安全性に問題があるってことだからね」



「でも、なんだかんだこの2人で互いに足りないものを埋めあってきたんだよね」



「2人とも詰めきれてない部分があったら、そこはエマちゃんの出番だからね」



「はい！ 私もお役に立ちたいです、いや、立ちます！」



「いいね！ 期待してるよ！」

私たちはみな、誰に何を言われたとしても、最終的には自分の行動を自分で選びます。チームはそんな個々の集合で成り立っていることを今一度思い出しましょう。

1つの改善点をどこまで追究すればよいかは、明確な答えがありません。したがって、個人の裁量による判断の差がもっとも表れやすいところです。

完璧さや正解を求ることは大事です。求めない人とは一緒にやりたくないと思った読者の方も少なからずいるでしょう。

しかし、全員がやりすぎると時間を取られすぎたり心身を壊したりし、成果が出ないチームに

陥ってしまいます。

やる人とやらない人の差が激しくても、不平不満が出やすくなったり、アメさんとムチさんのように調和が取れていても新しく入る人が不安に感じてしまうでしょう。

そうならないため、目的に応じたトレードオフライダーを設定することが大切です。

### ■トレードオフライダーとは

アジャイル開発におけるインセプションデッキの一項目として定められている、以下の4つに優先度を設定する手法。

- ・予算：予算内に収める
- ・時間：期日を守る
- ・品質：品質を高く、欠陥を少なくする
- ・スコープ：必要な機能を全部揃える

不足の事態が発生したとき、これらのどれを守ってどれを諦めるか、チームが判断する指標を前もって作っておきます。

たとえば機能が追加されると、予算とスコープを同時に守ることは不可能ですよね。そこで、予算派とスコープ派に分かれてしまっては円滑に進みません。

チームの指標とはいえ、チーム内で決めてしまうのではなく、必ず責任者や依頼元と合意して設定しましょう。

また、必ずこの4項目である必要はなく、プロジェクト状況に応じて項目を追加したり、1つの項目を分割したりしてもよいです。

レビューや組織運営でも、こうすれば絶対正解とか、全部が完璧にできるような落としどころはなかなかありません。トレードオフライダーは、レビューにおいても共通認識や、意見が分かれた場合の判断材料にも使えます。

## 2.3

### 戦いをせず協力を求める



「さっき通路でストラップ拾ったんですけど、どなたのですか？」



「あ、僕の」



「ムチさんですか！ どうぞ」



「ありがとう」



「これ、あの人気ゲームのですよね。推しなんですか？」



「まあ、そうですね」



「（考えてみたら、アメさんとはたくさん話したけど、ムチさんとはほとんど話していないなあ）」



「（意外と打ち解けられるのかも。今度困ったらペアプロをお願いしてみようかな！）」

▼ @ Muchi commented

なぜこんなイケてない実装をしてるんですか？

普通じゃ考えられません。

納得のいくよう説明求めます。



「えーん、えーん」



「エマちゃん！？ その泣き方は……またレビューできつい言われ方したの？」



「あ、大丈夫です！ 儀式です」



「ムチさんとは直接ペアプロで話そうと思います！」



「まあ！ あのムチさんとペアプロするなんて！ エマちゃん成長したね！」



「そんなことないです、でも学ぶことがホントに多いですから」



「どういうふうに考え方が変わったの？ 興味あるんだけど！」



「えへへ……内緒ですっ！」

実は、エマちゃんは先日の心理的安全性の話、そしてマリアさんとの会話を経て、どんな人とも安心して話すにはどうすればよいかを繰り返し考えていました。

社会人経験の長いお姉さんに相談したり、SNS や動画サイトで臆せず人と接している人を見て、参考にしながら以下のルールを決めたのです。

---

### ■言葉の隅々を受け止めない

イケてない、普通じゃ考えられないなどの一言多い部分は読み返さず、さらっと流す。

考え方や言葉選びのセンスは誰しも異なるし、時間をかけて悩んでも、正確なニュアンスを読んで納得できるわけではない。

大事なのは、言葉のどこに改善や成長につながるヒントが隠れているか、である。

---

■必要以上に自己の正当性を主張しない

自分の考えの軸は伝えつつ、相手との違いは対立ではなく発見や均しの機会と捉える。

コメントを細かく書く派の人と、コメントなしで理解できるコードを目指す人は相容れないが、双方がいると両端の外れ値は検出できる。

コードレビューを戦いではなく協力作業、レビューは味方であると常に意識する。

とはいっても、人は常に感情に左右されます。だからエマちゃんは儀式をしていたのです。

ネガティブに感じたから正直に泣いて発散し、短期間で思いきり受け止めたあと、すぐに今からできる Best を考えて動き出したのでした。

これが、アメさんとも違うエマちゃん流のやり方。みんなの協力作業を充実するために見つけた方法なのです。



「ペアプロありがとうございます！ ムチさん！」



「はい。どうも」



「次の画面ですが、既存の MessageList クラスをコピーして実装すればできそうですよね」



「う～ん……あのクラスは肥大化してて古いから、参考にしてほしくないな」



「どうすればよさそうですか？」



「View の一部を共通実装にして、依存関係は外から DI で入れるのはどうだろう」



「なるほど……やったことないので、リファクタリング込みだと、5 日ほどかかると思います。どうすれば短縮できそうですか？」



「前ブックマークした記事送るから、同じように書けば多分早いと思う」



「勉強したければ、この本も持ってっていいよ」



「ホントですか！ ありがとうございます」



「事前に相談できてよかったです！ 前だと相談せずやっちゃってました」



「そうですね」



「では、明後日にはできたところまでコミットするので、またレビューかペアプロをお願いします！」



「はい」

ムチさんの冷淡な感じは相変わらずですが、エマちゃんがオープンに技術的な相談をしてくれたり、ムチさんとしてもレビューの負荷や手戻りが減って嬉しいのでした。



「エマちゃん、すごく積極的に相談したりイキイキしてますよね」



「そうだね、ムチさんとあんなに上手くやる新人は初めてだよね」



「でも、ムチさんは昔から愛のムチだから」



「期待しない人には何も言わないし、期待してる人なら話せばちゃんと教えてくれるんだよね」



「エマちゃんに愛が伝わってよかったです！」



「そういえばアメさん、来月から新しいプロジェクトだね。未経験なのによく手を挙げたね」



「はい、前からやりたかったことですし、今までではちょっと失敗を恐れてましたけど……」



「エマちゃん見てたら、私も初心を思い出して、知らないことはどんどんオープンに質問して、好奇心 MAX でいこうかなって思えたんです」



「そっか。いい機会なんじゃない？ この業界、ずっと同じ技術だけじゃやっていけないもんね」



「それもありますけど、できないことができるようになったり、新しいものを発見する瞬間の喜びって、忘れちゃもったいないですよね」



「レビューって、そんなチャンスをたくさんくれる機会だなってあらためて気づきました」



「だから、次のチームはどんな人たちかわからないけど、とにかくレビューを受けるのはめっちゃ楽しみですね！」



「その突き抜けた考え方はさすがだね」



「いえいえ。マリアさんやムチさんから色々学べたおかげです！」



「じゃ、頑張って。次の新人さん！」



「はい！」

# あとがき

tonionagauzzi（トニオナガウツィ）です。自著のあとがきって、実は昔から少し憧れていました。延々と自分語りをしても聞いてもらえそうな場所なんて、最近あまりないですからね。

私は、ソフトウェアエンジニアとしては珍しいかもしれません、メンタルヘルスへの関心が強いです。それは、自分自身がメンタル不調から立ち直った経験をしたのがきっかけでした。

私は得意なこと、苦手なことが極端に分かれるタイプで、社会人になってから、周囲の人とのいろんな違いを軋轢に感じました。今いる環境に精いっぱい適合しようと、普通に見られようと頑張ったつもりですが、何をやっても印象は変わり者でした。

そして何週間も食事が喉を通らないような不調の最中、当時 84 歳の祖母から「私なんか今まで何も悩まずパッパラパーで生きてきたで」と励されました。その言葉が背中を押してくれました。私ももう少しパッパラパーで生きていいんじゃないかな、と思ったのです。

周囲の人のすごいところは自分なりに取り入れたり、時には頼ったりしながら、自分の目の前にあることを思いきり楽しもう。その一方、自分の軸となる考えは大事に持とう、とも思いました。人生は有限で、一度しかない中、こうして 1 つの職業を選んでいます。後で振り返って「本当によかったです」と思える日々にするためには、人に言われたから……ではなく、自分の行動をちゃんと自分で選ぶことが大事だと思いました。

そして、今の会社に来てから、自社フレームワークで！ みたいな縛りもなく、技術面での融通はだいぶ効く環境になりましたし、エンジニアの興味やモチベーションを尊重してもらっています。今までの経験を本にしたいと思ったとき、会社のバックアップを受けて書くこともできました。嬉しい限りですね。でも、本著の例のような厳しいレビューもときどき飛び交います（笑）。

私は社会に出てからそこそこ長く、先述のような経験もしてきたので、アメさんのように飄々として何事もチャンスだ！ と考えられるポジションにいますが、ときどきエマちゃんみたいな新人さんが来て、やはりアメさんみたいに悩みを聞く役になります。

そんな経緯があり、この本の元である 2019 年 9 月 11 日に公開した Qiita の記事 [クソレビューを防ごう（レビューのアンチパターン集）](#) \*1 を書いたのです。

当時の開発チームで、電子レビューだと誤解やタイムラグが発生するからと、対面式の集合レビューを行なっていました。Qiita の記事は、その環境をもっとよくしたい、レビューのモヤモヤに寄り添いたい、といった気持ちで書きました。一応弁解しておくと、クソレビューというやや過激な表現をしたのも、より多くの人に興味を持ってもらうためでした。

もうひとつ弁解しておくと、その記事や本著で紹介したアンチパターンは、全部がそのチームで起きた話ではありません。前職時代含めて、10 年以上いろんなチームを見てきて当事者として感じたものと、見聞きしたさまざまな体験談をベースにしています。

本著は、レビューをする側とされる側、そしてレビューに慣れている方と慣れてない方の双方に

---

\*1 <https://qiita.com/tonionagauzzi/items/96587b0848ec782010ad>

---

読んで欲しい内容です。でも、誤解しないで頂きたいのは、こうすればいいよ！ みたいな成功必勝パターンを紹介したいわけではありません。レビューという機会の活かし方について、少しだけ見つめ直すきっかけになって欲しいのです。

本書にはモデルケースはおらず、アメさん、ムチさんともに改善できる部分をたくさん持っていますし、マリアさんにもコントロールできない要素はたくさんあります。

実は、最初1章はムチさんがアンチパターンをすべてやらかし、アメさんが正解パターンを見せていく計画で書き始めました。しかし、すぐに頓挫しました。すべての開発チームに通ずる正解パターン、いわゆる銀の弾丸なんて私は知らないからです。

それならば、苦痛を感じやすいレビュー事例に対して私ならこうするという話や、著者がリラックスしてレビューやチーム開発に臨んでいる方法を紹介しようとしました。レビューを苦痛に感じている人に寄り添えるような本ができるといいなと思って書いていたら、現在の形になりました。

それをこれから皆さんにレビューしていただくような感覚です。これから多くのフィードバックを得られそうと心躍る気持ちでいます。

レビューや開発のベストプラクティスは、チームの目指す方向性や、そこにいる人たちによって作り上げていくものだと思います。正解がないとはいえ、安心して悩みを吐き出せたり、個々人が自分の軸を大事にしながらも協調して活躍できたり、不安にならないような開発現場が魅力的なのは確かですよね。

本著としては、もっとさまざまなことを書き入れたかった気持ちがありますが、はじめてのテックブック以外のゆるい感じの本なので、こうして送り出せること自体とても嬉しいと感じます。イラストでご協力いただいた toki さん、表紙デザインの numatami さん、校正の fjt.sh さん、出版を支援してくださった技術書典同好会と株式会社 ACCESS の皆さん、そして毎晩のように執筆することを理解して色々と支えてくれた私の家族に、この場を借りて心よりお礼を申し上げます。

## ♣ 著者紹介

**Vitantonio Nagauzzi (@tonionagauzzi)**

Mobile App Engineer, "You decide you're happy or not."

- <https://about.me/knagauchi>
- 株式会社 ACCESS

## ♣ 他刊一覧

- 『ACCESS テックブック』(技術書典 7)
- 『ACCESS テックブック 2』(技術書典 12)

# アイのムチ

よくないレビューの例とレビューで折れないメンタルづくり

---

2022年1月22日 ver 1.0 (技術書典 12)

著 者 ACCESS 技術書典同好会

デザイン numatami

イラスト toki

発行所 株式会社 ACCESS

発行者 tonionagauzzi

連絡先 vitantonionagauzzi@gmail.com, techbookfest12-gr@access-company.com

<http://about.me/knagauchi>

@tonionagauzzi (<https://twitter.com/tonionagauzzi>)

---

© 2022 ACCESS 技術書典同好会