Engineer

コードレビュー

👓 コードベースへ取り込む前に、作者以外が変更を読んで確認する工程

どのようなものか

コードレビューは、変更がコードベースに入る前に、書いた本人以外の人がその変更を読む工程です。Googleの開発実践ガイドは、コードを書いた人以外の誰かがコードを調べることと定義しています。*Software Engineering at Google* の該当章は、それがコードベースに入る前に行われることが多いと補足しています。多くのチームでは、この実践を英語の複数形でcode reviewsと呼びますが、ツールによって名称は変わります。GitHubではプルリクエストレビュー、Googleではコミット前レビューと呼びます。Linuxカーネルではメールで行い、内容に納得したレビュアーがパッチにReviewed-byタグを提供します。

Googleのガイドは、レビュアーの確認対象として、設計、機能、複雑さ、テスト、命名、コメント、スタイル、ドキュメントを挙げています。それぞれの言葉の背後には問いがあります。コードは作者の意図どおりに動き、それは利用者にとってよいものでしょうか。もっと単純にできないでしょうか。自動テストはあり、その内容は正しく、適切に設計されているでしょうか。関連するドキュメントは更新されているでしょうか。

進め方はどこでもおおむね同じです。作者が説明を添えて差分を送り、レビュアーが各行にコメントし、作者が回答するか新しい版を送ります。レビュアーが変更を受け入れるまで、このやり取りを繰り返します。Googleでは承認をLGTMと呼び、*looks good to me（問題なさそうです）* の略です。同書は、これを権限ビットになぞらえています。標準ではLGTMは一つあればよく、コードの所有者による承認と可読性の確認がそろうと、変更をコミットできます。GitHubでは、各レビューを三つの判断のいずれかで終えます。Commentは、承認も変更要求もせずに意見を伝えます。Approveは、そのままマージしてよいという判断です。Request changesは、作者が先に対応する必要のある点を示します。リポジトリを読める人なら誰でもレビューやコメントができ、管理者はマージの条件として承認を必須にできます。Googleのガイドは個々の変更をCLと呼んでいるため、その中にSmall CLsという題のページがあります。

概念を判断に生かす

「コードレビューとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 3社の間で受け渡す代わりに、1つのチームがアイデアのスケッチから、お客様がログインして使える製品まで担当します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 GIS地図アプリの開発を主導し売上を10倍に — 10x 地図製品の開発を主導して実現した売上成長

GIS地図アプリの開発を主導し売上を10倍に お問い合わせ →

なぜ重要なのか

同書は、適切に行うレビューの利点を六つ挙げています。コードの正しさを確認し、ほかのエンジニアが変更を理解できるようにし、コードベース全体の一貫性を保ち、チームでコードを所有する意識を育て、知識を共有し、履歴を残すことです。正しさの確認は分かりやすい利点で、著者らは1986年に公表されたIBMでの研究を引用しています。早い段階で見つかった不具合ほど、修正にかかる時間が短かったという研究です。ただし同書は、重要な条件も添えています。レビューに投じた時間が見合うのは、その手順を軽量に保っている間だけでした。現在では、正しさの確認の多くを静的解析と自動テストが担っており、人に頼る必要性は小さくなったと述べています。

理解できるかどうかの確認は、ツールでは置き換えられない利点です。どれほど優れたエンジニアでも、自分の仕事を初めて見る人と同じようには読めませんが、レビュアーにはそれができます。変更の意味が通るかを作者以外の誰かが確かめる最初の機会が、レビューであることは少なくありません。コードは書く回数よりはるかに多く読まれるため、これは大切です。文化面の効果は、もっと目立ちにくいものです。コードは個人の所有物ではなく共同の仕事の一部だと意識でき、専門家が担当領域の知識を伝えられます。また、すべての変更にレビューが残るため、後から誰でも、ある書き方がいつ導入されたかを調べ、その際のレビューを読むことができます。

レビューには時間もかかり、遅いレビューにはさらに大きな負担が伴います。Googleのレビューの速さに関するガイドは、レビューが遅いとチーム全体の進みが遅くなり、開発者が手順に不満を持ち始め、コードの健全性も損なわれると述べています。不十分な変更を通す圧力が生まれ、整理や改善をしようという意欲もそがれるためです。同じレビュアーでも、単に返事を早くすれば、厳しすぎるという苦情の大半は消えるとも述べています。承認の基準は実務的です。変更によってシステムのコードが明らかに健全になるなら、完全でなくても承認する方向で考えます。ガイドはこれをほかの規則に優先する原則とし、その理由を「『完璧な』コードというものはなく、あるのは、よりよいコードだけです」と一文で示しています。

実際の活用例

価格をペニー単位の整数で保持する店を考えます。作者が春のキャンペーンを追加し、次の変更をレビューに送ります。

```
--- a/shop/pricing.py
+++ b/shop/pricing.py
@@ -8,2 +8,4 @@ def order_total(basket):
     subtotal = sum(line.price * line.quantity for line in basket.lines)
+    if basket.voucher == "SPRING":
+        subtotal = subtotal * 0.9
     return subtotal + shipping_for(basket)
```

レビュアーは四つのコメントを残します。

```
shop/pricing.py, 9行目
  ほかのクーポンと割引率は、shop/vouchers.pyの
  VOUCHERSにあります。SPRINGもそちらに入れられますか。
  そうすれば、次のキャンペーンではこの関数ではなく、
  データを変更するだけで済みます。

shop/pricing.py, 10行目
  このモジュールのほかの箇所では、価格はすべて
  ペニー単位の整数です。0.9を掛けると、一ペニー未満の
  端数が生じることがあります。ほかの割引は
  shop/money.pyのround_money()を通しています。

shop/pricing.py, 11行目
  Nit: 送料が割引の対象から外れています。意図どおりなら、
  その旨を短いコメントにすると、次に読む人が助かります。

全体
  買い物かごにクーポンがある場合のテストがありません。
  SPRINGを使うテストと、不明なコードでは何も変わらない
  ことを示すテストを追加できますか。
```

どのコメントも人ではなくコードに向けられ、理由を説明しています。最初の二つと最後の一つにはラベルがないため、対応が求められます。三つ目はNit:で始まります。Googleのガイドでは、作者が直すべきではあるものの、影響の小さい指摘に使います。ほかには、必須ではない良い案を示すOptionalやConsider、この変更では作者の対応を求めない考えを伝えるFYIがあります。ラベルがないと、作者がすべてのコメントを必須事項と受け取るかもしれないと、ガイドは注意しています。GitHubでは、この回のレビューはRequest changesで終わります。作者はSPRINGを表へ移し、割引に丸め処理の補助関数を使い、コメントと

二つのテストを追加して、再確認を依頼します。二回目はApprove、GoogleならLGTMで終わります。ガイドは、レビューの依頼には遅くとも一営業日以内に返事をするよう求めています。ただし、集中している作業を中断するのではなく、自然な区切りまで待って対応します。

書式、インポートの順序、テストの成否についてのコメントがないのは、レビュアーが差分を開く前に機械が知らせるべきことだからです。同書は、変更が最初にレビューへ送られた時点で動き、誰も読む前に却下できる事前チェックを説明しています。カーネルのガイドも、作者に対し、提出前にスタイル検査の `scripts/checkpatch.pl` を実行するよう求めています。2025年からの職務で手がけたリポジトリ用ガードエンジンの構築は、この考えを大きく推し進めた例です。登録済みのコミットチェック268件、リント規則277件、独自のESLint規則15件に加え、ガード自体のテスト146件があります。ガードは、マイグレーションをロールバックできるか、翻訳キーが両方のロケールに存在するかなどを検査します。修正として示すコミットには、その修正がなければ失敗するテストも必須です。機械的な規則を検査として実装したことで、レビュアーは設計により多くの時間を使えるようになりました。一方、代償も明記されています。コミットに時間がかかり、出来の悪いガードを回避しながら作業するのは、いら立ちのもとになります。

問題が起きるところ

変更が大きすぎて読めません。 Googleのガイドは、通常、100行は妥当な変更規模で、1,000行は大きすぎるとしていますが、判断はレビュアーに任せています。一つのファイルの200行なら問題なくても、同じ200行が50ファイルに散らばるとそうはいかない場合もあると説明しています。小さな変更は、より速く、より丁寧にレビューされ、不具合を持ち込む可能性も低くなります。同書は、一回でレビューするには大きすぎるという理由で変更を却下するのは妥当だとも述べています。作業を分けます。小さな変更を順に積み重ね、ファイルのまとまりごとに別のレビュアーへ送り、リファクタリングは機能変更や不具合修正と分けます。ただし、それぞれの変更のテストは、その変更に含めます。

レビューが何日も止まります。 変更が一日待つごとに、その上に積み重なる仕事も遅れます。数日おきに大きな変更要求を返すレビュアーは、手順を実際以上に厳しく感じさせます。一営業日以内に返事をします。同書が期待するのは勤務時間で数えて24時間以内のフィードバックであり、すぐにすべてをレビューできない場合でも、変更を見たことは伝えます。コメントは小出しにせず、まとめて送ります。前の指摘をすべて直した後に、関係のない新たな指摘が一式届くことほど、作者をいら立たせるものはそうありません。

好みを必須条件として伝えてしまいます。 別のループや名前を好むという理由で変更を止めると、作者はコードを改善しようとしなくなります。Googleのガイドは優先順位を示しています。意見や個人の好みより事実とデータが優先し、スタイルについてはスタイルガイドが最終的な基準になります。複数の方法が同じように妥当なら、作者が選びます。好みにはNitやOptionalのラベルを付け、必須事項との違いが分かるようにします。

コメントが人に向いてしまいます。 ガイドにある悪い例は、並行処理に明らかな利点がないのに、なぜスレッドを使ったのかと作者を問い詰めます。良い例は、この並行処理のモデルは、レビュアーに確認できる性能上の利点がないまま複雑さを増やすため、コードをシングルスレッドにすべきだと述べます。開発者本人ではなくコードにコメントし、依頼の理由を説明します。

機械でできる確認を人が行います。 空白やインポートの順序に注意を割くレビュアーは、人にしか答えられない設計の問題を見落とします。明文化できる規則はすべて、リンター、フォーマッター、事前チェックに移し、判断が必要な点をレビュアーに任せます。

回答がレビューツールの中に残ります。 レビュアーがあるコードの目的を尋ね、作者がスレッドで説明し、承認されます。しかし、次にコードを読む人はその説明を目にしません。Googleのガイドは、このような質問を受けたら、通常はコードをもっと分かりやすく書き直すか、少なくともコード内のコメントに説明を残すべきだとしています。カーネルのガイドも作者に同じことを伝えています。レビュアーの質問に答えてもコードを変更しない場合、その回答は原則として、コメントか変更履歴に残すべきです。

参考資料

- はじめに — Googleのコードレビューガイドの導入ページです。確認する点、レビュアーの選び方、ペアプログラミングをレビューと見なせる場合を説明します。
- コードレビューの基準 — ほかの規則に優先する原則、レビューを通じた指導、作者とレビュアーの対立を解消する方法を扱います。
- 小さなCL — 小さな変更の方がよいレビューを受けられる理由、大きすぎる変更の目安、変更を分ける四つの方法を説明します。
- コードレビュー — Tom ManshreckとCaitlin Sadowskiによる *Software Engineering at Google* の一章です。レビューの利点、LGTMと所有権、望ましい実践を扱います。
- プルリクエストのレビュー — レビュー、承認、変更要求と、マージ前に承認を必須にする設定についてのGitHubのリファレンスです。

- パッチの提出：コードをカーネルに取り込むための必須ガイド — Linuxカーネルのメールによるレビュー、Reviewed-byタグが表明する内容、レビュアーへの回答方法を説明します。

用語集

実際の制約に合わせて選ぶ

業務の流れ、既存のインターフェース、変更を必要とする利用者をお知らせください。検証可能な納品内容と、お客様のチームが保守を続けられる引き継ぎ内容を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「コードレビューとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/code-review/