Engineer

品質ゲート

コードを次の段階へ進める前に合格が必要な、一連の検査

どのようなものか

品質ゲートは、変更を次の段階へ進める前に満たす必要がある条件の集合で、最終的な結果は合格か不合格のどちらかです。次の段階には、コミット、メインブランチへのマージ、リリース、デプロイなどがあります。条件にできるのは、リンターの指摘がないこと、テストの合格、カバレッジの数値、ビルドの完了など、機械が判定できるものです。

要点は、ゲートという言葉にあります。検査は結果を報告しますが、ゲートは先へ進めるかどうかを決め、その次の手順を制御する仕組みが決定を強制します。ログへ四十件の警告を出すリンターは、検査です。同じリンターでも、指摘がなくなるまでマージボタンを無効に保つよう組み込めば、ゲートの一部になります。

品質ゲートを機能名として使うSonarQube Cloudは、解析中にコードを評価するための条件の集合と説明しています。各条件は、メトリクス、比較演算子、エラーとする値を組み合わせます。たとえば「信頼性の問題が0より多い」が一つの条件で、どれか一つでも満たすとゲートは不合格になります。条件の適用先は、新しいコードか、コード全体です。

コードのホスティングサービスも、同じ考え方をマージの流れに組み込んでいます。

- **GitHub**には、ブランチ保護の設定「Require status checks before merging」（マージ前にステータスチェックを必須とする）があります。保護されたブランチへ変更を入れる前に、必須の検査が成功、スキップ、または中立のステータスを報告する必要があります。

- **GitLab**には、「Pipelines must succeed」（パイプラインの成功を必須とする）というマージ検査があります。選択すると、パイプラインが失敗している間はマージリクエストをマージできず、パイプラインが一つもない場合にもマージできません。
- **開発者自身のマシン**では、pre-commitが、コミット時やプッシュ時に動くGitフックとして検査を組み込み、Taskなどのタスクランナーが、検査一式を一つの名前で呼べるようにします。

概念を判断に生かす

「品質ゲートとは？」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。 変更を検証し、リリースし、元に戻せる仕組みを整え、金曜の深夜ではなく火曜の午後にお客様へ届けられるようにします。

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。 GitHub ActionsでGIS SaaSのデプロイを自動化 — デプロイ、データ、報告を手作業から解放

GitHub ActionsでGIS SaaSのデプロイを自動化 お問い合わせ →

なぜ重要なのか

文書の中にしかない規則は、例外のたびに交渉の対象になります。ビルドを失敗させる規則なら、書いた本人が、その日の午後は省略したいと思っていても、全員が従います。ゲートは、合意した規則を実行可能にするため、規則を守れるかどうかが、記憶や気分に左右されなくなります。

人によるレビューの役割も変わります。機械が、書式、未使用のインポート、リンク切れ、秘密情報の漏えいを既に検出していれば、レビュアーは、機械には判断できない意図や設計へ注意を向けられます。書き手が誰でも、求められる水準は同じです。

新しいコードに条件を適用する考え方によって、古いコードベースにもゲートを導入できます。SonarQube Cloudの既定のゲートであるSonar wayは、既存コードの修復ではなく、新しいコードを対象とします。条件には、新しいコードのテストカバレッジが80.0%以上であることや、重複が3.0%以下であることなどがあります。プルリクエストでは、新しいコードに関する条件だけを使います。既存の問題が何千件あっても、先にすべてを解消せず、ゲートを有効にして、新しい変更の一つ一つに水準を求められます。

使い続けてもらえるかどうかを決めるのは、速度です。Taskは、タスクの依存先を並行して実行するため、独立した検査を多数組み合わせたゲートでも、順番ではなく同時に動かせます。検査をコストで分けることも役立ちます。軽い方の

半分をコミットのたびに実行し、コードがマシンを離れる前に、全体を実行します。

このサイトにも、独自のゲートである `task check` があり、ホストされたビルドサービスではなく、サイトを書くマシン上で実行します。すべての診断をエラーとし、サイトジェネレーターを `--panicOnWarning` 付きで動かすため、非推奨機能の通知だけでもビルドが止まります。 `# noqa` や `stylelint-disable` のような行内の抑制も禁止し、失敗を黙らせるのではなく、原因を直します。pre-commitフックは、コミットごとにファイル単位の検査を実行し、pre-pushフックは、秒ではなく分単位の時間がかかる `task check` 全体を実行します。未コミットの変更がない作業ツリーで検査に合格すると、そのツリーに合格の印を付け、デプロイスクリプトは、印のないツリーを拒否します。唯一の迂回手段は、目立つ警告を表示する環境変数です。緊急時にゲートを省く人も、その事実を明示しなければなりません。

実際の活用例

サイトのクレジットページは、ゲートに含まれるツールを挙げています。HTML用のvnu、Stylelint、テンプレートごとに一ページを調べるaxe-core、Lighthouse、リンク用のlychee、Prettier、markdownlint、ESLint、Ruff、basedpyright、yamllint、ShellCheck、codespell、gitleaksです。Taskが、それぞれの処理を一つのタスク名で実行し、pre-commitは、ファイル単位の検査を担う半分を、コミットごとに改めて実行します。同じページは、ゲートの外に置く検査も明記しています。Puppeteerで動くブラウザー検査のうち、表示領域からのはみ出しと視覚的な差分の二つで、人が必要に応じて実行します。

あるポートフォリオのページは、一人のプロジェクトにはレビュアーがまったくいない、という状況から始まります。規則に違反したら、誰かが気付くのを待つのではなく、コミットを失敗させるため、規則を実行可能にする必要がありました。記録された2026年8月の実装段階での解決策は、警告という区分を設けない、22本のリンターによるコミットゲートと、既製のツールでは判断できない、プロジェクト固有の検査でした。たとえば、デプロイするツリーに非公開のパスやホスト名がないことや、三言語の文章が文体の規則を守ることです。既にある違反も、受け入れず、その作業の中で直します。ページは、代償を明確に述べています。コミットは遅くなり、このプロジェクトのレビュアーは、ゲートだけです。

ゲートの中で食い違いが起きることもあります。別のページは、コミットフックと完全なゲートがあり、文書にはフックがゲートを実行すると書かれている一方、二つのファイルで検査の一覧を手作業で管理していたプロジェクトを説明し

ます。両方の一覧を、実際に呼び出す検査までたどって比較するテストにより、完全なゲートでしか動かない検査が五つ見つかりました。五つとも、文章を読む検査でした。コードを守る検査はすべて自動実行され、文章を守る検査は、誰かが思い出したときにしか動いていませんでした。五つをフックに移し、そのうち二つを、フックの実行時間に収まる速さにしました。一覧の違いを検出するため、比較テストを残しました。

問題が起きるところ

ゲートの失敗の多くは、合格と表示しながら問題を通してしまう形で起きます。

合格表示の裏で、警告が積み上がります。 エラーのときだけ失敗するゲートでは、警告が増え続け、やがてログが長すぎて読めなくなり、本当の問題が二百一件目の警告として現れます。警告という区分をなくします。不合格にする価値がある規則なら不合格にし、そうでなければ無効にします。

スキップした検査が、合格として扱われます。 GitHubでは、スキップのステータスでも必須の検査を満たし、`if:` 条件によってスキップしたジョブも成功を報告します。失敗したジョブに依存するジョブもスキップされ、マージを止めない場合があります。GitHubは、`always()` を `needs` と組み合わせるよう勧めています。常に実行され、必要とするジョブがすべて成功したときだけ合格するジョブを、一つ必須にします。

```
# .github/workflows/check.yml（抜粋）
jobs:
    gate:
        if: always()
        needs: [lint, test, build]
        runs-on: ubuntu-latest
        steps:
            - run: |
                  for result in ${{ join(needs.*.result, ' ') }}; do
                      test "$result" = success || exit 1
                  done
```

逆の落とし穴もあります。パスやブランチのフィルターによって必須のワークフローをスキップすると、検査が保留のままになり、すべてのマージを止めます。スキップされ得るワークフローを、必須にしないようにします。

二つの一覧が食い違います。 フックが実行する検査とサーバーの検査が違い、文書には一致すると書かれています。検査の定義を一つにして、どの場所からも呼び出すか、一覧を比較し、違っていたら失敗するテストを維持します。

ゲートが遅くなり、人が迂回します。 フックを省略するフラグはすぐに使えます。また、GitHubのブランチ保護は、既定ではリポジトリー管理者に適用されません。「Do not allow bypassing the above settings」（上記設定の迂回を許可しない）が、それを変えます。どこに時間がかかるかを測り、独立した検査を並行して実行し、コミット時に動かす半分を小さく保ち、本当に遅い検査は、文書化した手動の手順へ移します。

何も検査しなかったのに、合格します。 一つのファイルにも一致しないglobや、ディレクトリー全体を除外するパターンを指定されたリンターは、エラーなく終了します。その終了結果は、問題のないコードを検査した場合と同じに見えます。ゲート自体をテストします。各検査に、間違いがあると分かっているファイルを渡し、確実に不合格になるかを確認します。

参考資料

- 品質ゲート入門 — 条件、既定のSonar way、新しいコードを対象とする考え方です。
- 保護されたブランチについて — 必須のステータスチェックと、迂回できる人です。
- 必須のステータスチェックのトラブルシューティング — 合格とみなすステータスと、スキップしたワークフローやジョブの動作です。
- 自動マージ — GitLabの「Pipelines must succeed」検査と、スキップしたパイプラインの扱いです。
- pre-commit — 一度定義した検査を、コミット時とプッシュ時のGitフックとして組み込む仕組みです。
- Taskの依存関係 — タスクの依存先を並行して実行し、コマンドを順番に実行する仕組みです。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「品質ゲートとは？」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/quality-gate/