Engineer

Contrôle qualité obligatoire

vérifications à réussir avant de passer à l'étape suivante

Ce que c'est

Un contrôle qualité obligatoire est un ensemble de conditions qu'une modification doit remplir avant de passer à l'étape suivante, avec un résultat final unique : réussite ou échec. Cette étape peut être un commit, une fusion dans la branche principale, une publication ou un déploiement. Les conditions peuvent porter sur tout ce qu'une machine sait décider : une exécution de lint sans diagnostic, des tests réussis, un taux de couverture ou une construction qui aboutit.

Le mot décisif est obligatoire. Une vérification signale ; un contrôle obligatoire décide, et cette décision est appliquée par ce qui commande l'étape suivante. Un linter qui imprime quarante avertissements dans un journal est une vérification. Le même linter, relié au bouton de fusion pour que celui-ci reste désactivé tant que sa sortie contient des diagnostics, fait partie d'un contrôle obligatoire.

SonarQube Cloud, où le terme désigne une fonctionnalité nommée, décrit un contrôle qualité obligatoire comme un ensemble de conditions auxquelles le code est comparé pendant l'analyse. Chaque condition associe une métrique, un opérateur de comparaison et une valeur d'erreur : « nombre de problèmes de fiabilité supérieur à 0 » est ainsi une condition, et si l'une des conditions est remplie, le contrôle échoue. Une condition s'applique soit au nouveau code, soit à l'ensemble du code.

Les plateformes d'hébergement intègrent la même idée dans leur processus de fusion :

- **GitHub** possède un paramètre de protection de branche, « Require status checks before merging », qui impose les vérifications d'état avant fusion. Une vérification obligatoire doit signaler un état réussi, ignoré ou neutre avant que les modifications atteignent la branche protégée.
- **GitLab** possède une vérification de fusion appelée « Pipelines must succeed », qui exige la réussite des pipelines. Lorsqu'elle est sélectionnée, une demande de fusion ne peut aboutir si son pipeline échoue, ni même si aucun pipeline n'existe.
- **Sur la machine de développement**, pre-commit installe des vérifications sous forme de hooks Git exécutés au commit ou au push, et un exécuteur de tâches comme Task donne un nom unique à l'ensemble.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'un contrôle qualité obligatoire ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. Des changements testés, publiés et réversibles, pour livrer aux clients un mardi après-midi plutôt qu'un vendredi soir tard.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. Déploiement SaaS GIS automatisé : GitHub Actions — déploiement, données et reporting automatisés

Déploiement SaaS GIS automatisé : GitHub Actions Contactez-nous →

Pourquoi c'est important

Une règle qui n'existe que dans un document se négocie exception après exception. Une règle qui fait échouer la construction est suivie par tout le monde, y compris par la personne qui l'a écrite, même l'après-midi où elle préférerait s'en dispenser. Un contrôle obligatoire rend les règles convenues exécutables : leur respect cesse de dépendre de la mémoire et de l'humeur.

Il change aussi le rôle de la revue humaine. Lorsque la machine a déjà détecté les défauts de mise en forme, les imports inutilisés, les liens cassés et les secrets divulgués, la personne chargée de la revue peut consacrer son attention à l'intention et à la conception, que la machine ne sait pas juger. Le niveau d'exigence est également le même pour chaque auteur.

L'approche par nouveau code rend un contrôle applicable à une base de code ancienne. Sonar way, le contrôle par défaut de SonarQube Cloud, se concentre sur le nouveau code plutôt que sur la correction de l'ancien, avec des conditions telles qu'au moins 80,0 % de couverture des tests et au plus 3,0 % de duplication dans le nouveau code. Pour une pull request, seules les conditions relatives au

nouveau code sont utilisées. Un projet ayant des milliers de problèmes existants peut activer ce contrôle et soumettre chaque nouvelle modification à ses exigences sans d'abord résorber tout le passif.

La vitesse détermine si les gens laissent le contrôle activé. Task exécute les dépendances d'une tâche simultanément : un contrôle construit à partir de nombreuses vérifications indépendantes peut donc les exécuter en parallèle plutôt que l'une après l'autre. Les séparer selon leur coût aide aussi : la moitié peu coûteuse à chaque commit, l'ensemble avant que le code quitte la machine.

Ce site possède son propre contrôle obligatoire : `task check` , exécuté sur la machine où le site est écrit plutôt que sur un service de construction hébergé. Chaque diagnostic compte comme une erreur, le générateur du site s'exécute avec `--panicOnWarning` pour qu'un simple avis de dépréciation arrête la construction, et les suppressions de diagnostics directement dans le code telles que `# noqa` ou `stylelint-disable` sont interdites : la cause d'un échec doit être corrigée plutôt que masquée. Un hook pre-commit exécute les vérifications par fichier à chaque commit, et un hook pre-push exécute le `task check` complet, qui prend des minutes plutôt que des secondes. Une exécution réussie depuis une arborescence propre y appose une marque de validation, et le script de déploiement refuse une arborescence sans cette marque. Son unique dérogation est une variable d'environnement qui affiche un avertissement très visible : quiconque saute le contrôle en urgence doit donc le signaler.

À quoi cela ressemble en pratique

La page des crédits du site énumère les outils de ce contrôle : vnu pour HTML, Stylelint, axe-core sur une page par modèle, Lighthouse, lychee pour les liens, Prettier, markdownlint, ESLint, Ruff, basedpyright, yamllint, ShellCheck, codespell et gitleaks. Task les exécute sous un nom unique par tâche, et pre-commit réexécute la partie portant sur les fichiers à chaque commit. La page nomme aussi ce qui reste hors du contrôle : deux vérifications de navigateur pilotées par Puppeteer, le débordement de la fenêtre et la régression visuelle, qu'une personne lance à la demande.

Un projet mené par une seule personne n'a aucun relecteur : c'est le point de départ d'une réalisation du portfolio. Les règles devaient être exécutables pour qu'en enfreindre une fasse échouer le commit au lieu d'attendre que quelqu'un s'en aperçoive. À l'étape d'implémentation d'août 2026 décrite dans la documentation, la réponse était un contrôle au commit comprenant 22 linters

sans niveau d'avertissement, complété par des vérifications propres au projet sur lesquelles aucun outil standard ne se prononce, par exemple l'absence de chemin privé ou de nom d'hôte privé dans l'arborescence déployée et le respect des règles de ton dans les trois langues de cette étape. Un échec déjà présent est corrigé dans la même passe plutôt qu'accepté. La page annonce clairement le coût : créer un commit est lent, et ce contrôle est le seul relecteur du projet.

Un contrôle peut aussi se contredire. Une autre réalisation décrit un projet doté d'un hook de commit et d'un contrôle complet, d'un document affirmant que le hook exécutait ce contrôle et de deux listes de vérifications maintenues à la main dans deux fichiers. Un test qui résout chaque liste en vérifications réellement appelées a révélé cinq vérifications exécutées uniquement dans le contrôle complet : c'étaient précisément celles qui lisaient les textes. Chaque vérification protégeant le code s'exécutait automatiquement ; chaque vérification protégeant l'écriture n'était lancée que lorsque quelqu'un y pensait. Les cinq ont été intégrées au hook, deux d'entre elles ont été suffisamment accélérées pour tenir dans son budget de temps, et le test de comparaison a été conservé pour détecter les différences entre les listes.

Où les problèmes surviennent

La plupart des contrôles échouent en laissant passer des défauts tout en restant au vert.

Les avertissements s'accumulent derrière une réussite. Un contrôle qui n'échoue que sur les erreurs laisse les avertissements s'accumuler jusqu'à rendre le journal trop long à lire ; un vrai problème arrive alors en deux cent unième position. Supprimez le niveau d'avertissement : soit une règle mérite de faire échouer le contrôle, soit elle est désactivée.

Une vérification ignorée compte comme réussie. Sur GitHub, un état ignoré satisfait une vérification obligatoire, et une tâche ignorée par sa condition `if:` signale une réussite. Une tâche dépendant d'une tâche en échec est aussi ignorée et peut ne pas bloquer la fusion. GitHub conseille d'utiliser `always()` avec `needs` : exigez une tâche qui s'exécute toujours et ne réussit que si toutes ses dépendances ont réussi.

```
# .github/workflows/check.yml (excerpt)
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
```

Le piège inverse existe aussi : un workflow obligatoire ignoré par un filtre de chemin ou de branche laisse ses vérifications en attente et bloque toutes les fusions. Évitez de rendre obligatoires des workflows qui peuvent être ignorés.

Deux listes divergent. Le hook exécute un ensemble, le serveur un autre, et un document prétend qu'ils concordent. Donnez une définition unique aux vérifications et appelez-la partout, ou conservez un test qui compare les listes et échoue si elles diffèrent.

Le contrôle ralentit et les gens le contournent. Une option permettant de sauter les hooks est à portée de main, et sur GitHub la protection de branche ne s'applique pas par défaut aux administrateurs du dépôt ; « Do not allow bypassing the above settings » est le paramètre qui interdit cette dérogation. Mesurez où passe le temps, exécutez les vérifications indépendantes simultanément, limitez la partie exécutée au commit et déplacez une vérification réellement lente vers une étape manuelle documentée.

Une vérification réussit parce qu'elle n'a rien vérifié. Un linter visant un motif glob qui ne correspond à aucun fichier, ou un motif d'exclusion qui engloutit tout un répertoire, se termine sans erreur : cette sortie propre ressemble à du code propre. Testez le contrôle lui-même : fournissez à chaque vérification un fichier dont vous savez qu'il est incorrect et attendez-vous à un échec.

Pour aller plus loin

- Introduction aux contrôles qualité obligatoires — les conditions, le contrôle Sonar way par défaut et l'accent sur le nouveau code.
- À propos des branches protégées — les vérifications d'état obligatoires et les personnes autorisées à les contourner.
- Résoudre les problèmes de vérifications d'état obligatoires — les états qui comptent comme réussis et le comportement des workflows et tâches ignorés.

- Fusion automatique — la vérification « Pipelines must succeed » de GitLab et son traitement des pipelines ignorés.
- pre-commit — des vérifications déclarées une seule fois et installées comme hooks Git aux étapes de commit et de push.
- Dépendances de Task — comment Task exécute simultanément les dépendances d'une tâche et ses commandes dans l'ordre.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le parcours de déploiement, les risques opérationnels et les personnes d'astreinte. Nous pouvons définir un changement avec des vérifications, une procédure de reprise et une responsabilité claire.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'un contrôle qualité obligatoire ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

welcome@engineer.company +45 50 10 61 84 Copenhague, Danemark

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/quality-gate/