Engineer

Kvalitetsgrind

kontroller som kod måste klara innan den går vidare

Vad det är

En kvalitetsgrind är en uppsättning villkor som en ändring måste uppfylla innan den får gå vidare till nästa steg, med ett enda resultat i slutet: godkänt eller underkänt. Steget kan vara en commit, en merge till huvudgrenen, en release eller en driftsättning. Villkoren kan vara vad som helst som en maskin kan avgöra: en lintkörning utan anmärkningar, tester som går igenom, en siffra för testtäckning, ett bygge som blir klart.

Det bärande ordet är grind. En kontroll rapporterar, en grind beslutar, och beslutet upprätthålls av det som styr nästa steg. En linter som skriver fyrtio varningar till en logg är en kontroll. Samma linter, inkopplad så att merge-knappen förblir avstängd tills utdata är rena, är en del av en grind.

SonarQube Cloud, där begreppet är en namngiven funktion, beskriver en kvalitetsgrind som en uppsättning villkor som koden mäts mot under analysen. Varje villkor parar ett mätvärde med en jämförelseoperator och ett felvärde, så ”reliability issues greater than 0” är ett villkor, och om något villkor uppfylls fallerar grinden. Ett villkor gäller antingen ny kod eller koden som helhet.

Plattformarna för kodvärd bygger in samma idé i sitt flöde för merge:

- **GitHub** har en inställning för grenskydd, ”Require status checks before merging”. En obligatorisk kontroll måste rapportera statusen lyckad, överhoppad eller neutral innan ändringar når den skyddade grenen.
- **GitLab** har en merge-kontroll som heter ”Pipelines must succeed”. När den är vald kan en merge request inte slås ihop medan dess pipeline fallerar, och inte heller när det saknas pipeline helt.

- **På utvecklarens egen dator** installerar pre-commit kontroller som git-hookar som körs vid commit eller push, och en uppgiftskörare som Task ger hela uppsättningen ett namn.

Använd begreppet i ett beslut

Använd ”Vad är en kvalitetsgrind?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Testade, släppta och möjliga att backa, så att ändringar når kunderna en tisdagseftermiddag i stället för sent en fredagskväll.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Automatiserade GIS-driftsättning i GitHub Actions — driftsättning, data och rapportering, helt utan manuellt arbete

Automatiserade GIS-driftsättning i GitHub Actions Hör av er →

Varför det spelar roll

En standard som bara finns i ett dokument förhandlas bort ett undantag i taget. En standard som får bygget att fallera följs av alla, även av den som skrev den, den eftermiddag då den personen helst hade hoppat över den. En grind gör överenskomna regler körbara, så att det inte längre hänger på minne och humör om de hålls.

Grinden ändrar också vad mänsklig granskning är till för. När en maskin redan har fångat formatering, oanvända importer, trasiga länkar och läckta hemligheter kan granskaren lägga sin uppmärksamhet på avsikt och design, som en maskin inte kan bedöma. Ribban är dessutom densamma för varje författare.

Idén om ny kod gör att en grind går att införa i en gammal kodbas. Sonar way, standardgrinden i SonarQube Cloud, inriktar sig på ny kod snarare än på att åtgärda gammal kod, med villkor som minst 80,0 % testtäckning och högst 3,0 % duplicering i ny kod. På en pull request används bara villkoren om ny kod. Ett projekt med tusentals befintliga anmärkningar kan slå på grinden och hålla varje ny ändring till ribban utan att först betala av hela eftersläpningen.

Farten avgör om folk låter grinden vara påslagen. Task kör en uppgifts beroenden samtidigt, så en grind som är byggd av många oberoende kontroller kan köra dem sida vid sida i stället för en i taget. Att dela upp kontrollerna efter kostnad hjälper också: den billiga halvan vid varje commit, hela uppsättningen innan koden lämnar maskinen.

Den här webbplatsen har en egen grind, `task check`, som körs på den maskin där webbplatsen skrivs snarare än på en byggtjänst i molnet. Varje diagnos räknas som ett fel, webbplatsgeneratorn körs med `--panicOnWarning` så att även en utfasningsvarning stoppar bygget, och undantag direkt i koden, som `# noqa` eller `stylelint-disable`, är förbjudna, så att ett fel rättas vid sin orsak i stället för att tystas. En hook före commit kör kontrollerna på filnivå vid varje commit, och en hook före push kör hela `task check`, som tar minuter snarare än sekunder. En grön körning från ett rent träd stämplar det trädet, och skriptet för driftsättning vägrar ta emot ett träd utan stämpel. Den enda vägen förbi är en miljövariabel som skriver ut en högljudd varning, så den som hoppar över grinden i ett nödläge måste säga det.

Så ser det ut i praktiken

Webbplatsens kolofon listar verktygen i grinden: vnu för HTML, Stylelint, axe-core på en sida per mall, Lighthouse, lychee för länkar, Prettier, markdownlint, ESLint, Ruff, basedpyright, yamllint, ShellCheck, codespell och gitleaks. Task kör alltihop under ett namn per jobb, och pre-commit kör den halva som gäller enskilda filer igen vid varje commit. Sidan visar också vad grinden lämnar åt en människa att köra vid behov: två av kontrollerna i webbläsaren som Puppeteer styr, innehåll som sticker ut utanför visningsytan och den visuella regressionsjämförelsen.

Ett enmansprojekt saknar granskare helt, och det är den situation som en sida i portfolion utgår från. Standarderna måste vara körbara, så att ett brott mot en av dem får en commit att fallera i stället för att vänta på att någon ska märka det. I den dokumenterade implementationsfasen från augusti 2026 blev svaret en commit-grind med 22 lintverktyg och utan varningsnivå, plus projektspecifika kontroller som inget färdigt verktyg har någon åsikt om, som att det driftsatta trädet inte innehåller någon privat sökväg eller något privat värdnamn och att texten följde sina tonregler på tre språk. Ett fel som redan fanns rättas i samma svep i stället för att godtas. Sidan anger kostnaden rakt ut: att committa går långsamt, och grinden är den enda granskare som projektet har.

En grind kan också vara oense med sig själv. En annan sida beskriver ett projekt med en hook för commit och en fullständig grind, ett dokument som påstod att hooken körde grinden och två listor med kontroller som underhölls för hand i två filer. Ett test som löser upp båda listorna till de kontroller de faktiskt anropar visade fem kontroller som bara kördes i den fullständiga grinden, och de var precis de som läser prosan. Varje kontroll som skyddade koden kördes automatiskt, och varje kontroll som skyddade texten kördes bara när någon kom

ihåg det. De fem flyttades in i hooken, två av dem gjordes snabba nog att rymmas i dess tidsbudget, och jämförelsetestet fick stanna kvar för att fånga skillnader mellan listorna.

Var det går fel

De flesta grindar sviker genom att släppa igenom saker medan de fortfarande visar grönt.

Varningar hopar sig bakom ett godkänt resultat. En grind som bara fallerar på fel låter varningarna växa tills loggen är för lång att läsa, och sedan kommer ett verkligt problem som varning nummer tvåhundraett. Ta bort varningsnivån: antingen är en regel värd att fallera på, eller så är den avslagen.

En överhoppad kontroll räknas som godkänd. På GitHub uppfyller statusen överhoppad en obligatorisk kontroll, och ett jobb som hoppas över av sitt `if:`-villkor rapporterar lyckat. Ett jobb som beror på ett jobb som fallerat hoppas också över och blockerar kanske inte merge. GitHubs råd är att använda `always()` med `needs`: kräv ett enda jobb som alltid körs och som bara går igenom när allt det beror på har lyckats.

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

Den motsatta fällan finns också: ett obligatoriskt arbetsflöde som hoppas över av ett filter på sökväg eller gren lämnar sina kontroller väntande och blockerar varje merge. Gör bara sådana arbetsflöden obligatoriska som alltid körs.

Två listor glider isär. Hooken kör en uppsättning, servern en annan, och ett dokument påstår att de stämmer överens. Ge kontrollerna en enda definition och anropa den från varje ställe, eller behåll ett test som jämför listorna och fallerar

när de skiljer sig åt.

Grinden blir långsam och folk går runt den. En flagga som hoppar över hookar finns nära till hands, och på GitHub gäller grenskyddet som standard inte för administratörer av ett repo. Inställningen ”Do not allow bypassing the above settings” ändrar det. Mät var tiden går, kör oberoende kontroller samtidigt, håll halvan vid commit liten och flytta en verkligt långsam kontroll till ett dokumenterat manuellt steg.

En kontroll blir grön utan att ha granskat något. En linter som pekar på ett mönster som matchar noll filer, eller ett ignoreringsmönster som sväljer en hel katalog, avslutas utan fel, och en ren avslutning ser likadan ut som ren kod. Testa själva grinden: mata varje kontroll med en fil som är känd för att vara fel och förvänta dig att den fallerar.

Läs mer

- Introduction to quality gates — villkoren, standardgrinden Sonar way och inriktningen på ny kod.
- About protected branches — obligatoriska statuskontroller och vem som får gå förbi dem.
- Troubleshooting required status checks — vilka statusar som räknas som godkända och hur överhoppade arbetsflöden och jobb beter sig.
- Auto-merge — kontrollen ”Pipelines must succeed” i GitLab och hur den behandlar överhoppade pipelines.
- pre-commit — kontroller som deklareras en gång och installeras som git-hookar för stegen commit och push.
- Task dependencies — hur Task kör en uppgifts beroenden samtidigt och dess kommandon i ordning.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är en kvalitetsgrind?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/quality-gate/