Engineer

Kvalitetsport

et sett kontroller kode må bestå før den går videre

Hva det er

En kvalitetsport er et sett krav en endring må oppfylle før den får gå videre til neste trinn, med ett resultat til slutt: bestått eller feilet. Trinnet kan være en commit, en merge inn i hovedgrenen, en utgivelse eller en produksjonssetting. Kravene kan være alt en maskin kan avgjøre: en ren kjøring av en linter, tester som består, et tall for testdekning, et bygg som fullføres.

Forskjellen ligger i hvem som bestemmer. En sjekk rapporterer; en kvalitetsport bestemmer, og beslutningen håndheves av det som styrer neste steg. En linter som skriver førti advarsler inn i en logg, er en sjekk. Den samme linteren, koblet slik at merge-knappen er deaktivert til utdataene er rene, er en del av en kvalitetsport.

SonarQube Cloud, der begrepet er en egen funksjon med navn, beskriver en kvalitetsport (quality gate) som et sett betingelser koden måles mot under analysen. Hver betingelse kobler en metrikk med en sammenligningsoperator og en feilverdi, så «reliability issues greater than 0» er en betingelse, og hvis en hvilken som helst betingelse slår til, feiler kvalitetsporten. En betingelse gjelder enten ny kode eller koden som helhet.

Vertsplattformene bygger den samme tanken inn i flyten for merge:

- **GitHub** har en innstilling for beskyttede grener, «Require status checks before merging». En påkrevd sjekk må rapportere status som vellykket, hoppet over eller nøytral før endringer når den beskyttede grenen.
- **GitLab** har en merge-sjekk som heter «Pipelines must succeed». Når den er valgt, kan en merge request ikke slås sammen mens pipelinen feiler, og heller ikke når det ikke finnes noen pipeline i det hele tatt.

- **På utviklerens egen maskin** installerer pre-commit sjekker som Git-hooks som kjører ved commit eller ved push, og en oppgavekjører som Task gir hele settet ett navn.

Bruk begrepet i en beslutning

Bruk «Hva er en kvalitetsport?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Testet, utgitt og mulig å reversere, slik at endringer når kundene en tirsdag ettermiddag i stedet for sent en fredagskveld.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Automatiserte GIS-SaaS-utrulling i GitHub Actions — produksjonssetting, data og rapportering, alt uten manuelle grep

Automatiserte GIS-SaaS-utrulling i GitHub Actions Ta kontakt →

Hvorfor det er viktig

En standard som bare lever i et dokument, blir forhandlet bort ett unntak om gangen. En standard som får bygget til å feile, følges av alle, også av den som skrev den, den ettermiddagen vedkommende helst ville hoppet over den. En kvalitetsport gjør avtalte regler kjørbare, slik at det å holde dem slutter å avhenge av hukommelse og humør.

Den endrer også hva menneskelig gjennomgang er til for. Når en maskin allerede har fanget formatering, ubrukte importer, døde lenker og hemmeligheter som har lekket, kan den som gjennomgår, bruke oppmerksomheten på hensikt og design, som en maskin ikke kan vurdere. Lista er dessuten den samme for hver forfatter.

Tanken om ny kode gjør det mulig å ta i bruk en kvalitetsport på en gammel kodebase. Sonar way, den innebygde kvalitetsporten i SonarQube Cloud, retter oppmerksomheten mot ny kode i stedet for å rydde opp i gammel, med betingelser som minst 80,0 % testdekning og høyst 3,0 % duplisering i ny kode. På en pull request brukes bare betingelsene om ny kode. Et prosjekt med tusenvis av eksisterende funn kan slå kvalitetsporten på og holde hver ny endring til lista uten først å betale ned etterslepet.

Hastighet avgjør om folk lar kvalitetsporten stå på. Task kjører avhengighetene til en oppgave samtidig, så en kvalitetsport bygd av mange uavhengige sjekker kan kjøre dem side om side i stedet for den ene etter den andre. Det hjelper også å dele sjekkene etter kostnad: den billige halvdelen ved hver commit, hele settet før koden forlater maskinen.

Dette nettstedet har sin egen kvalitetsport: `task check`, som kjøres på maskinen nettstedet skrives på, og ikke på en byggetjeneste hos noen andre. Hver diagnostikk teller som en feil, nettstedsgeneratoren kjører med `--panicOnWarning`, slik at selv en melding om utfasing stopper bygget, og undertrykking i koden, som `# noqa` eller `stylelint-disable`, er forbudt, så en feil rettes der den oppstår, i stedet for å bli dysset ned. En pre-commit-hook kjører sjekkene på filnivå ved hver commit, og en pre-push-hook kjører hele `task check`, som tar minutter og ikke sekunder. En grønn kjøring fra et rent tre stempler det treet, og skriptet for produksjonssetting nekter et tre uten stempel. Den eneste overstyringen er en miljøvariabel som skriver ut en tydelig advarsel, så den som hopper over kvalitetsporten i en nødsituasjon, må si det høyt.

Slik ser det ut i praksis

Nettstedets kolofon lister opp verktøyene inne i den: vnu for HTML, Stylelint, axe-core på én side per mal, Lighthouse, lychee for lenker, Prettier, markdownlint, ESLint, Ruff, basedpyright, yamllint, ShellCheck, codespell og gitleaks. Task kjører alt under ett navn per jobb, og pre-commit kjører halvdelen på filnivå på nytt ved hver commit. Siden viser også hva som står utenfor kvalitetsporten: blant nettlesersjekkene som Puppeteer styrer, er det overløp i visningsområdet og den visuelle regresjonstesten, og dem kjører et menneske ved behov.

Et enpersonsprosjekt har ingen som gjennomgår endringene i det hele tatt, og det er situasjonen én side i porteføljen starter fra. Standardene måtte være kjørbare, slik at et brudd på én av dem får en commit til å feile i stedet for å vente på at noen legger merke til det. I den dokumenterte implementeringsfasen fra august 2026 ble svaret en kvalitetsport ved commit med 22 lintere og uten advarselsnivå, pluss prosjektspesifikke sjekker som intet hyllevareverktøy har noen mening om, f.eks. at treet som settes i produksjon, ikke inneholder noen private stier eller vertsnavn, og at teksten fulgte tonereglene på tre språk. En feil som fantes fra før, rettes i samme omgang i stedet for å bli godtatt. Siden sier rett ut hva det koster: det går tregt å committe, og kvalitetsporten er den eneste som gjennomgår endringer i prosjektet.

En kvalitetsport kan også være uenig med seg selv. En annen side beskriver et prosjekt med en commit-hook og en full kvalitetsport, et dokument som sa at hooken kjørte hele kvalitetsporten, og to lister over sjekker som ble holdt ved like for hånd i to filer. En test som løser opp begge listene til sjekkene de faktisk

kaller, viste fem sjekker som bare kjørte i den fulle kvalitetsporten, og det var akkurat de som leser teksten. Hver sjekk som beskyttet kode, kjørte automatisk; hver sjekk som beskyttet teksten, kjørte bare når noen husket det. De fem ble flyttet inn i hooken, to av dem ble gjort raske nok til å holde seg innenfor tidsbudsjettet, og sammenligningstesten ble beholdt for å fange forskjeller mellom listene.

Hvor det går galt

De fleste kvalitetsporter svikter ved å slippe ting gjennom mens de fortsatt viser grønt.

Advarsler hoper seg opp bak en godkjenning. En kvalitetsport som bare feiler på feil, lar advarsler samle seg til loggen er for lang til å lese, og så kommer et ekte problem som advarsel nummer to hundre og én. Fjern advarselsnivået: enten er en regel verdt å feile på, eller så er den slått av.

En sjekk som hoppes over, teller som bestått. På GitHub oppfyller en status som er hoppet over, en påkrevd sjekk, og en jobb som hoppes over på grunn av `if:` -betingelsen sin, rapporterer suksess. En jobb som avhenger av en jobb som feilet, hoppes også over, og blokkerer kanskje ikke merge. GitHubs råd er å bruke `always()` sammen med `needs` : krev én jobb som alltid kjører, og som bare består når alt den trenger, har lyktes.

```
# .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
```

Den motsatte fellen finnes også: en påkrevd arbeidsflyt som hoppes over på grunn av et filter for sti eller gren, lar sjekkene sine stå som ventende og blokkerer hver merge. Unngå å kreve arbeidsflyter som kan hoppes over.

To lister glir fra hverandre. Hooken kjører ett sett, serveren et annet, og et dokument hevder at de stemmer overens. Gi sjekkene én definisjon og kall den fra hvert sted, eller behold en test som sammenligner listene og feiler når de er forskjellige.

Kvalitetsporten blir treg, og folk går rundt den. Et flagg som hopper over hooks, er lett å ta i bruk, og på GitHub gjelder beskyttelse av grener som standard ikke for administratorer av repoet; «Do not allow bypassing the above settings» endrer det. Mål hvor tiden går, kjør uavhengige sjekker samtidig, hold halvdelen som kjører ved commit, liten, og flytt en sjekk som virkelig er treg, til et dokumentert manuelt steg.

En sjekk består fordi den sjekket ingenting. En linter som peker på et mønster som treffer null filer, eller et ignoremønster som sluker en hel mappe, avslutter rent, og en ren avslutning ser lik ut som ren kode. Test selve kvalitetsporten: gi hver sjekk en fil du vet er feil, og forvent at den feiler.

Les mer

- Introduction to quality gates — betingelser, den innebygde kvalitetsporten Sonar way og vekten på ny kode.
- About protected branches — påkrevde statussjekker, og hvem som kan gå forbi dem.
- Troubleshooting required status checks — hvilke statuser som teller som bestått, og hvordan arbeidsflyter og jobber som hoppes over, oppfører seg.
- Auto-merge — sjekken «Pipelines must succeed» i GitLab, og hvordan den behandler pipeliner som hoppes over.
- pre-commit — sjekker som deklareres én gang og installeres som Git-hooks for trinnene commit og push.
- Task dependencies — hvordan Task kjører avhengighetene til en oppgave samtidig og kommandoene i rekkefølge.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er en kvalitetsport?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/quality-gate/