Engineer

Kodgranskning

~~ბა~~ någon annans granskning av en ändring innan den slås ihop

Vad det är

En kodgranskning är steget där någon annan än författaren läser en ändring innan den blir en del av kodbasen. Googles vägledning för företagets utvecklingspraxis definierar det som en process där någon annan än den eller de som har skrivit en bit kod undersöker den, och kapitlet om ämnet i *Software Engineering at Google* tillägger att det ofta sker innan koden förs in i kodbasen. Många svenska utvecklare säger också code review, och i övrigt beror namnet mest på verktyget. På GitHub är det en granskning av en pull request, hos Google en precommit-granskning, och i Linuxkärnan sker den via e-post, där en granskare som är nöjd med en patch erbjuder en Reviewed-by-tagg för den.

Googles vägledning räknar upp vad en granskare tittar på: design, funktionalitet, komplexitet, tester, namngivning, kommentarer, stil och dokumentation. Bakom varje ord finns en fråga. Beter sig koden som författaren troligen avsåg, och är det beteendet bra för användarna? Kunde den vara enklare? Har den korrekta och väl utformade automatiska tester? Har författaren uppdaterat dokumentationen som beskriver den?

Mekaniken är i stort sett densamma överallt. Författaren skickar en diff med en beskrivning, granskarna kommenterar enskilda rader, författaren svarar eller laddar upp en ny version, och rundan upprepas tills en granskare godkänner ändringen. Hos Google kallas godkännandet LGTM, en förkortning av *looks good to me*, och boken beskriver det som en behörighetsbit: som standard krävs bara ett LGTM, och tillsammans med godkännande från en kodägare och en readability-kontroll ger det rätt att committa ändringen. GitHub avslutar varje granskning med ett av tre beslut. Comment ger allmän återkoppling utan att godkänna eller begära ändringar, Approve signalerar att ändringarna är redo att slås ihop, och Request changes markerar återkoppling som författaren bör ta hand om före sammanslagningen. Alla med läsbehörighet kan granska och

kommentera, och administratörer för ett repo kan kräva godkännanden innan en pull request slås ihop. Googles vägledning kallar varje ändring för en CL, och därför heter en av sidorna Small CLs.

Använd begreppet i ett beslut

Använd ”Vad är kodgranskning?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Ett team som tar en idé från en skiss till något dina kunder kan logga in i, i stället för att lämna den vidare mellan tre leverantörer.

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. Ledde GIS-kartappen till 10 gånger högre intäkter — 10x intäktstillväxt, med kartprodukten i spetsen

Ledde GIS-kartappen till 10 gånger högre intäkter Hör av er →

Varför det spelar roll

Boken räknar upp sex fördelar med en välskött kodgranskning. Den kontrollerar att koden är korrekt, ser till att andra utvecklare förstår ändringen, håller kodbasen konsekvent, främjar gemensamt ägarskap i teamet, sprider kunskap och lämnar efter sig en historik. Korrekthet är den uppenbara, och författarna hänvisar till en studie vid IBM, publicerad 1986, som fann att fel som upptäcktes tidigare tog kortare tid att rätta. Boken lägger till villkoret som avgör: tiden som lades på granskning lönade sig bara så länge processen förblev lätt. Statisk analys och automatiska tester bär i dag mycket av korrekthetsarbetet, och enligt boken har det minskat behovet av att låta människor stå för det.

Förståelse är den fördel som inga verktyg tar över. En granskare kan göra det som inte ens den bästa utvecklaren kan göra med sitt eget arbete, nämligen läsa det utan författarens perspektiv, och en granskning är ofta det första provet på om en ändring är begriplig för någon annan. Det spelar roll eftersom kod läses många fler gånger än den skrivs. De kulturella fördelarna är tystare. Granskningen påminner utvecklaren om att koden inte är ens egen utan en del av ett gemensamt åtagande, den låter experter föra vidare vad de kan om ett område, och eftersom varje ändring behåller sin granskning kan vem som helst senare ta reda på när ett mönster kom in och läsa granskningen som förde in det.

Kodgranskning kostar också tid, och långsam granskning kostar mer. Googles vägledning om snabbhet säger att när granskningarna går långsamt sjunker hela teamets tempo, utvecklarna börjar protestera mot processen och kodhälsan blir lidande, eftersom det uppstår press att släppa igenom sämre ändringar och den som vill städa upp i koden avskräcks. De flesta klagomål på att en granskare är för

sträng försvinner enligt vägledningen när samma granskare helt enkelt svarar snabbare. Standarden för att godkänna är pragmatisk: granskare bör hellre godkänna en ändring när den helt klart förbättrar systemets övergripande kodhälsa, även om den inte är perfekt. Vägledningen kallar det den överordnade principen bland sina regler och sammanfattar skälet på en rad: ”det finns ingen perfekt kod – det finns bara bättre kod”.

Så ser det ut i praktiken

Tänk dig en webbutik där priserna anges i hela ören. En författare lägger till en vårkampanj och skickar den här ändringen på granskning:

```
--- 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)
```

Granskaren lämnar fyra kommentarer:

```
shop/pricing.py, rad 9
  De andra rabattkoderna och deras satser ligger i VOUCHERS i
  shop/vouchers.py. Kan SPRING flyttas dit också? Då blir nästa
  kampanj en ändring i data, inte i den här funktionen.

shop/pricing.py, rad 10
  Överallt annars i modulen är priserna hela ören, och att
  multiplicera med 0.9 kan ge en bråkdel av ett öre. De andra
  rabatterna går genom round_money() i shop/money.py.

shop/pricing.py, rad 11
  Nit: frakten ingår inte i rabatten. Om det är avsiktligt
  skulle en kort kommentar om det hjälpa nästa läsare.

Allmänt
  Det saknas ett test med en rabattkod i varukorgen. Kan du
  lägga till ett för SPRING och ett som visar att en okänd kod
  lämnar priset oförändrat?
```

Varje kommentar handlar om koden och inte om personen, och varje kommentar säger varför. Den första, den andra och den sista saknar etikett och förväntas därför åtgärdas; den tredje börjar med Nit:, som Googles vägledning använder för något litet som författaren tekniskt sett borde göra men som inte påverkar särskilt mycket. Vägledningens andra etiketter är Optional, eller Consider, för en bra idé som inte är ett krav, och FYI för en tanke som författaren inte förväntas agera på i den här ändringen. Utan etiketter, varnar den, kan författare läsa varje kommentar som ett krav. På GitHub slutar den här rundan med Request changes. Författaren flyttar SPRING till tabellen, skickar rabatten genom avrundningsfunktionen, lägger till kommentaren och båda testerna och ber om en ny titt; andra rundan slutar med Approve, eller hos Google med LGTM. Vägledningen ber granskare svara på en granskningsförfrågan inom högst en arbetsdag, och att inte avbryta en koncentrerad uppgift för att göra det utan vänta på ett naturligt avbrott.

Ingen av kommentarerna handlar om formatering, importordning eller om testerna går igenom, för det borde en maskin ha sagt innan granskaren öppnade diffen. Boken beskriver presubmits som körs när en ändring först skickas på granskning och som kan avvisa den innan någon läser den; Linuxkärnan ber författare att köra kärnans stilkontroll, `scripts/checkpatch.pl`, innan de skickar in. Kontrollmotorn som byggdes för ett repo, från 2025, driver den idén långt: 268 registrerade commit-kontroller, 277 lintregler och 15 egna ESLint-regler, plus 146 tester av själva kontrollerna. Kontrollerna ser bland annat till att en migrering går att rulla tillbaka och att en översättningsnyckel finns i båda språkversionerna, och varje commit med prefixet fix: måste innehålla ett test som fallerar utan ändringen. Kontrollerna kodade in mekaniska regler, så granskarna kunde ägna mer tid åt design. Sidan säger också vad det kostar: det går långsamt att committa, och en dåligt skriven kontroll är verkligen enerverande att ta sig runt.

Var det går fel

Ändringen är för stor för att läsas. Googles vägledning kallar 100 rader för en vanligtvis rimlig storlek på en ändring och 1 000 rader för vanligtvis för stort, men lämnar bedömningen åt granskaren och påpekar att 200 rader i en fil kan vara rimligt medan samma 200 rader utspridda över 50 filer inte vore det. Små ändringar granskas snabbare och grundligare och för mer sällan in fel; boken tillägger att granskare med rätta kan avvisa en ändring som för stor för en enda granskning. Dela upp arbetet: stapla små ändringar på varandra, skicka grupper av filer till olika granskare, håll refaktoreringar åtskilda från nya funktioner och felrättningar, och låt varje ändrings tester ligga i samma ändring.

Granskningen blir liggande i flera dagar. Varje dag som en ändring får vänta försenar det arbete som bygger på den, och en granskare som bara svarar med några dagars mellanrum och med stora krav får processen att kännas hårdare än den är. Svara inom en arbetsdag; bokens förväntan är återkoppling inom 24 arbetstimmar, och om hela granskningen måste vänta bör granskaren säga att ändringen är sedd. Skicka alla kommentarer samtidigt i stället för styckevis, för få saker irriterar en författare mer än en ny runda orelaterad återkoppling efter att den förra har åtgärdats.

Smak framställs som krav. En granskare som föredrar en annan loop eller ett annat namn och blockerar ändringen för det lär författarna att sluta förbättra koden. Googles vägledning bestämmer ordningen: tekniska fakta och data väger tyngre än åsikter och personliga preferenser, stilguiden är den absoluta auktoriteten i stilfrågor, och där flera angreppssätt är lika bra vinner författarens preferens. Märk en preferens som Nit eller Optional, så att författaren kan skilja den från ett krav.

Kommentaren handlar om personen. Vägledningens eget exempel på en dålig kommentar frågar varför författaren använde trådar när det uppenbarligen inte finns någon vinst med samtidighet; den bra versionen säger att samtidighetsmodellen tillför komplexitet utan någon prestandavinst som granskaren kan se, så koden bör vara enkeltrådad. Kommentera koden, aldrig utvecklaren, och förklara resonemanget bakom önskemålet.

Människor kontrollerar det en maskin kunde. Granskare som lägger sin uppmärksamhet på blanksteg och importordning missar designfrågan som bara en människa kan svara på. Flytta varje regel som går att skriva ner till ett lintverktyg, ett formateringsverktyg eller en presubmit-kontroll, och låt granskaren behålla bedömningarna.

Svaret stannar i granskningsverktyget. En granskare frågar vad ett kodblock är till för, författaren förklarar i tråden, granskaren godkänner, och nästa person som läser koden ser aldrig förklaringen. Googles vägledning säger att en sådan fråga oftast bör leda till att koden skrivs om så att den blir tydligare, eller åtminstone till en kommentar i koden. Linuxkärnan säger samma sak till författarna: frågor i en granskning som inte leder till en kodändring bör nästan säkert leda till en kommentar eller en post i ändringsloggen.

Läs mer

- Introduction — startsidan i Googles vägledning för kodgranskning: vad granskare tittar efter, hur man väljer granskare och när parprogrammering räknas som granskning.
- The Standard of Code Review — den överordnade principen, handledning genom granskning och hur konflikter mellan författare och granskare löses.
- Small CLs — varför små ändringar får bättre granskningar, när en ändring är för stor och fyra sätt att dela upp den.
- Code Review — Tom Manshreck och Caitlin Sadowskis kapitel i *Software Engineering at Google*: fördelarna med granskning, LGTM och ägarskap, och god praxis.
- Pull request reviews — GitHubs referens om att granska, godkänna och begära ändringar och om att kräva godkännanden före sammanslagning.
- Submitting patches: the essential guide to getting your code into the kernel — hur Linuxkärnan granskar via e-post, vad en Reviewed-by-tagg intygar och hur författare bör svara granskarna.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är kodgranskning?” 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/code-review/