Engineer

Code-Review

ᔓᔑ Codeänderungen vor der Übernahme von anderen prüfen lassen

Was es ist

Ein Code-Review ist der Schritt, bei dem jemand anderes als der Verfasser eine Änderung liest, bevor sie Teil der Codebasis wird. Googles Leitfaden zu seinen Entwicklungspraktiken definiert es als Prüfung von Code durch andere Personen als diejenigen, die ihn geschrieben haben. Das entsprechende Kapitel in *Software Engineering at Google* ergänzt, dass dies häufig vor der Aufnahme in die Codebasis geschieht. Die meisten Teams sprechen im Plural von Code-Reviews. Die Namen wechseln mit dem Werkzeug. Bei GitHub heißt es Pull Request Review, bei Google Precommit Review. Im Linux-Kernel geschieht es per E-Mail. Ein zufriedener Prüfer bietet dort einen Reviewed-by-Tag für den Patch an.

Googles Leitfaden nennt die Prüfpunkte: Entwurf, Funktionalität, Komplexität, Tests, Benennung, Kommentare, Stil und Dokumentation. Hinter jedem Wort steht eine Frage. Tut der Code das vom Verfasser Beabsichtigte, und ist das gut für seine Nutzer? Ginge es einfacher? Gibt es automatisierte Tests, und sind sie korrekt und sinnvoll entworfen? Wurde die dazugehörige Dokumentation aktualisiert?

Der Ablauf ist überall ähnlich. Der Verfasser schickt einen Diff mit Beschreibung, Prüfer kommentieren einzelne Zeilen, der Verfasser antwortet oder lädt eine neue Version hoch. Das wiederholt sich, bis ein Prüfer die Änderung annimmt. Bei Google heißt diese Annahme LGTM, kurz für *looks good to me* („sieht für mich gut aus“). Das Buch beschreibt sie als Berechtigungsbit: Standardmäßig ist nur ein LGTM nötig. Zusammen mit der Freigabe eines Codeverantwortlichen und einer Lesbarkeitsprüfung erlaubt es, die Änderung zu committen. GitHub beendet jedes Review mit einer von drei Entscheidungen. Comment gibt Rückmeldung, ohne zuzustimmen oder Änderungen zu verlangen. Approve bedeutet, dass die Änderung unverändert zusammengeführt werden kann. Request changes kennzeichnet Punkte, die der Verfasser zuerst bearbeiten muss. Alle mit Lesezugriff auf das Repository können prüfen und kommentieren. Die

Administratoren können eine Freigabe zur Voraussetzung für das Zusammenführen machen. Googles Leitfaden nennt jede Änderung CL. Deshalb heißt eine seiner Seiten Small CLs.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist ein Code-Review?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Ein Team führt eine Idee von der Skizze bis zu einem Produkt, bei dem Ihre Kunden sich anmelden können, statt sie zwischen drei Dienstleistern weiterzureichen.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird.
GIS-Kartenprodukt zu 10-fachem Umsatz geführt — 10x Umsatzwachstum durch die Leitung des Kartenprodukts

GIS-Kartenprodukt zu 10-fachem Umsatz geführt Kontakt aufnehmen →

Warum es wichtig ist

Das Buch nennt sechs Vorteile eines gut geführten Reviews. Es prüft die Korrektheit des Codes, stellt das Verständnis der Änderung durch andere Entwickler sicher, setzt Einheitlichkeit in der Codebasis durch, fördert gemeinsame Verantwortung, verbreitet Wissen und hinterlässt einen historischen Nachweis. Korrektheit ist der offensichtliche Nutzen. Die Autoren zitieren eine 1986 veröffentlichte IBM-Studie, nach der früher entdeckte Defekte weniger Reparaturzeit brauchten. Das Buch ergänzt die entscheidende Bedingung: Der Zeitaufwand für Reviews lohnte sich nur, solange das Verfahren schlank blieb. Statische Analyse und automatisierte Tests übernehmen inzwischen viel dieser Korrektheitsarbeit. Laut Buch muss man sich dafür deshalb weniger auf Menschen verlassen.

Verständnis ist der Vorteil, den kein Werkzeug ersetzt. Auch ein sehr guter Entwickler kann die eigene Arbeit nicht wie ein Fremder lesen, ein Prüfer dagegen schon. Ein Review ist oft die erste Prüfung durch jemand anderen, ob eine Änderung sinnvoll ist. Das zählt, weil Code viel häufiger gelesen als geschrieben wird. Die kulturellen Vorteile fallen weniger auf. Ein Review erinnert Entwickler daran, dass ihnen der Code nicht gehört, sondern Teil einer gemeinsamen Unternehmung ist. Experten können ihr Wissen über einen Bereich weitergeben. Weil jede Änderung ihr Review behält, kann später jeder nachsehen, wann ein Muster eingeführt wurde, und das zugehörige Review lesen.

Reviews kosten auch Zeit, langsame Reviews noch mehr. Googles Leitfaden zur Geschwindigkeit sagt, dass langsame Reviews das ganze Team verlangsamen. Entwickler beginnen das Verfahren abzulehnen, und die Codequalität leidet: Es entsteht Druck, schwächere Änderungen durchzulassen, und Aufräumarbeiten werden entmutigt. Die meisten Beschwerden über zu strenge Prüfer verschwinden laut Leitfaden, wenn dieselben Prüfer einfach schneller antworten. Sein Freigabemaßstab ist pragmatisch: Verbessert eine Änderung den Zustand des Codes im System eindeutig, sollten Prüfer eher zustimmen, selbst wenn sie nicht perfekt ist. Der Leitfaden nennt das seinen obersten Grundsatz und begründet ihn in einem Satz: „Es gibt keinen ‚perfekten' Code, nur besseren Code."

Wie es in der Praxis aussieht

Betrachten Sie einen Shop, dessen Preise in ganzen Pence gespeichert werden. Ein Verfasser ergänzt eine Frühlingsaktion und reicht diese Änderung zum Review ein:

```
--- 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)
```

Der Prüfer hinterlässt vier Kommentare:

```
shop/pricing.py, Zeile 9
  Die anderen Gutscheine und ihre Rabattsätze stehen in VOUCHERS in
  shop/vouchers.py. Könnte auch SPRING dorthin? Dann ist die nächste Aktion
  eine Datenänderung statt einer Änderung an dieser Funktion.

shop/pricing.py, Zeile 10
  Überall sonst in diesem Modul sind Preise in ganzen Pence gespeichert.
  Multiplizieren mit 0.9 kann Bruchteile eines Pennys ergeben. Die anderen
  Rabatte verwenden round_money() in shop/money.py.

shop/pricing.py, Zeile 11
  Nit: Die Versandkosten sind vom Rabatt ausgenommen. Falls das Absicht
  ist, würde ein kurzer Kommentar dazu dem nächsten Leser helfen.

Allgemein
```

```
Es gibt keinen Test für einen Warenkorb mit Gutschein. Könnten Sie
einen für SPRING ergänzen und einen, der zeigt, dass ein unbekannter
Code nichts ändert?
```

Jeder Kommentar betrifft den Code statt die Person und nennt den Grund. Die ersten beiden und der letzte tragen kein Label und sollen deshalb bearbeitet werden. Der dritte beginnt mit Nit:. Googles Leitfaden verwendet das für einen kleinen Punkt, den der Verfasser beheben sollte, der aber wenig ändert. Weitere Labels sind Optional oder Consider für eine gute, nicht verpflichtende Idee und FYI für einen Gedanken, zu dem in dieser Änderung keine Handlung erwartet wird. Ohne Labels könnten Verfasser jeden Kommentar als verpflichtend lesen, warnt der Leitfaden. Bei GitHub endet diese Runde mit Request changes. Der Verfasser verschiebt SPRING in die Tabelle, nutzt für den Rabatt die Hilfsfunktion zum Runden, ergänzt Kommentar und beide Tests und bittet um eine erneute Prüfung. Die zweite Runde endet mit Approve, bei Google mit LGTM. Der Leitfaden verlangt eine Antwort auf eine Reviewanfrage innerhalb höchstens eines Arbeitstags. Dafür soll keine konzentrierte Aufgabe unterbrochen, sondern eine natürliche Pause abgewartet werden.

Kein Kommentar betrifft Formatierung, Importreihenfolge oder bestandene Tests. Das sollte eine Maschine bereits vor dem Öffnen des Diffs geklärt haben. Das Buch beschreibt Presubmit-Prüfungen, die bei der ersten Revieweinreichung laufen und eine Änderung ablehnen können, bevor sie jemand liest. Der Kernel verlangt von Verfassern, vor der Einreichung seinen Stilprüfer `scripts/checkpatch.pl` auszuführen. Die für ein Repository entwickelte Prüfengine aus der seit 2025 laufenden Tätigkeit treibt die Idee weit: 268 registrierte Commitprüfungen, 277 Lintregeln und 15 eigene ESLint-Regeln sowie 146 Tests der Schutzprüfungen selbst. Diese prüfen etwa die Rücknehmbarkeit einer Migration oder das Vorhandensein eines Übersetzungsschlüssels in beiden Gebietsschemata. Jeder als Fehlerkorrektur markierte Commit muss einen Test enthalten, der ohne ihn scheitert. Die Prüfungen bildeten mechanische Regeln ab, sodass die Reviewenden mehr Zeit für den Entwurf aufwenden konnten. Sie nennt auch den Preis: Committen ist langsam, und es ist äußerst frustrierend, eine schlecht geschriebene Schutzprüfung umgehen zu müssen.

Wo es schiefgeht

Die Änderung ist zu groß zum Lesen. Googles Leitfaden nennt 100 Zeilen als meist vernünftige Änderungsgröße und 1.000 als meist zu groß. Die Entscheidung bleibt dem Prüfer. 200 Zeilen in einer Datei können in Ordnung

sein, dieselben 200 über 50 Dateien verteilt dagegen nicht. Kleine Änderungen werden schneller und gründlicher geprüft und führen seltener Fehler ein. Laut Buch dürfen Prüfer eine Änderung zu Recht als zu groß für ein einzelnes Review zurückweisen. Teilen Sie die Arbeit: Bauen Sie kleine Änderungen aufeinander auf, schicken Sie Dateigruppen an verschiedene Prüfer, trennen Sie Refactorings von Funktionsänderungen und Fehlerkorrekturen und halten Sie die Tests jeder Änderung in derselben Änderung.

Das Review bleibt tagelang liegen. Jeder Wartetag verzögert die darauf aufbauende Arbeit. Ein Prüfer, der alle paar Tage große Änderungswünsche schickt, lässt das Verfahren strenger wirken, als es ist. Antworten Sie innerhalb eines Arbeitstags. Das Buch erwartet Rückmeldung innerhalb von 24 Arbeitsstunden. Muss das vollständige Review warten, soll der Prüfer zumindest bestätigen, dass er die Änderung gesehen hat. Schicken Sie alle Kommentare auf einmal statt stückweise. Wenig ärgert Verfasser mehr als eine neue Runde von Rückmeldungen zu anderen Punkten, nachdem die vorige erledigt wurde.

Geschmack wird als Anforderung dargestellt. Wer eine andere Schleife oder einen anderen Namen bevorzugt und deswegen die Änderung blockiert, gewöhnt Verfasser daran, den Code nicht mehr zu verbessern. Googles Leitfaden setzt eine Rangfolge: Fakten und Daten zählen mehr als Meinung oder persönlicher Geschmack. Bei Stilfragen entscheidet der Stilleitfaden. Sind mehrere Ansätze gleichwertig, wählt der Verfasser. Kennzeichnen Sie eine Vorliebe als Nit oder Optional, damit sie von einer Anforderung unterscheidbar ist.

Der Kommentar betrifft die Person. Das Negativbeispiel des Leitfadens fragt, warum der Verfasser Threads verwendete, obwohl Nebenläufigkeit offensichtlich keinen Nutzen bringe. Die gute Fassung sagt, dass das Nebenläufigkeitsmodell Komplexität ohne einen für den Prüfer erkennbaren Leistungsvorteil ergänzt und der Code deshalb mit einem Thread arbeiten sollte. Kommentieren Sie den Code, nie den Entwickler, und begründen Sie den Wunsch.

Menschen prüfen, was eine Maschine könnte. Prüfer, die sich auf Abstände und Importreihenfolge konzentrieren, übersehen die Entwurfsfrage, die nur ein Mensch beantworten kann. Übertragen Sie jede formulierbare Regel in einen Linter, Formatter oder eine Presubmit-Prüfung. Überlassen Sie den Prüfern die Ermessensentscheidungen.

Die Antwort bleibt im Reviewwerkzeug. Ein Prüfer fragt nach dem Zweck eines Codeblocks, der Verfasser erklärt ihn im Thread, der Prüfer stimmt zu. Die nächste Person liest den Code ohne die Erklärung. Laut Googles Leitfaden sollte eine solche Frage meist zu klarer geschriebenem Code oder wenigstens einem Codekommentar führen. Der Kernel sagt Verfassern dasselbe: Ändert die Frage eines Prüfers keinen Code, sollte die Antwort in der Regel in einem Kommentar oder im Änderungsprotokoll landen.

Weiterführende Lektüre

- Einführung — der Einstieg in Googles Code-Review-Leitfaden: Prüfpunkte, Prüferauswahl und wann Paarprogrammierung als Review zählt.
- Der Maßstab für Code-Reviews — der oberste Grundsatz, Mentoring durch Reviews und die Lösung von Konflikten zwischen Verfasser und Prüfer.
- Kleine CLs — warum kleine Änderungen bessere Reviews erhalten, wann sie zu groß sind und vier Möglichkeiten der Aufteilung.
- Code Review — Tom Manshrecks und Caitlin Sadowskis Kapitel in *Software Engineering at Google*: Reviewvorteile, LGTM, Verantwortlichkeit und bewährte Praktiken.
- Pull Request Reviews — GitHubs Referenz zu Prüfung, Zustimmung, Änderungswünschen und verpflichtenden Freigaben vor einer Zusammenführung.
- Patches einreichen: der wesentliche Leitfaden für Code im Kernel — wie der Linux-Kernel per E-Mail prüft, was ein Reviewed-by-Tag aussagt und wie Verfasser Prüfern antworten sollten.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist ein Code-Review?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/code-review/