Engineer

Рев'ю коду

ꙮ перевірка зміни коду іншою людиною перед злиттям

Що це таке

Рев’ю коду — це етап, на якому зміну, перш ніж вона стане частиною кодової бази, читає хтось інший, ніж її автор. Настанови Google щодо інженерних практик визначають його як процес, у якому код перевіряє хтось, крім автора чи авторів цього коду, а розділ про рев’ю в книжці *Software Engineering at Google* додає, що це часто відбувається до того, як код потрапить у кодову базу. Українською кажуть і код-рев’ю, а конкретна назва здебільшого залежить від інструмента. На GitHub це рев’ю pull request, у Google — рев’ю перед комітом (precommit review), а в ядрі Linux воно відбувається електронною поштою: рецензент, задоволений патчем, пропонує до нього тег Reviewed-by.

Настанови Google перелічують, на що дивиться рецензент: дизайн, функціональність, складність, тести, іменування, коментарі, стиль і документацію. За кожним словом стоїть питання. Чи поводиться код так, як, імовірно, задумав автор, і чи добра ця поведінка для його користувачів? Чи можна зробити його простішим? Чи має він правильні й добре продумані автоматичні тести? Чи оновив автор документацію, яка його описує?

Механіка всюди приблизно однакова. Автор надсилає diff з описом, рецензенти коментують окремі рядки, автор відповідає або завантажує нову версію, і цикл повторюється, доки рецензент не схвалить зміну. У Google таке схвалення називають LGTM, це скорочення від англійського *looks good to me*, а книжка описує його як біт дозволу: за замовчуванням потрібен лише один LGTM, і разом зі схваленням власника коду та перевіркою readability він дає змогу закомітити зміну. GitHub завершує кожне рев’ю одним із трьох рішень. Comment — це загальний відгук без схвалення чи запиту на зміни, Approve означає, що зміни готові до злиття, а Request changes позначає відгук, який автор має врахувати перед злиттям. Рецензувати й коментувати

може будь-хто з доступом на читання, а адміністратори репозиторію можуть вимагати схвалень, перш ніж pull request зіллють. Настанови Google називають кожну зміну CL, тому одна з їхніх сторінок має назву Small CLs.

Застосуйте поняття до рішення

Застосуйте «Що таке рев'ю коду?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Одна команда веде ідею від ескізу до продукту, у який ваші клієнти вже можуть увійти, а не перекидає роботу між трьома підрядниками.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Збільшила дохід GIS-застосунку в 10 разів — 10x зростання виторгу завдяки картографічному продукту

Збільшила дохід GIS-застосунку в 10 разів Зв'язатися з нами →

Чому це важливо

Книжка називає шість переваг добре налагодженого рев'ю. Воно перевіряє, що код правильний, гарантує, що інші інженери зрозуміють зміну, підтримує одноманітність кодової бази, розвиває в команді спільне володіння кодом, поширює знання й залишає історичний запис. Правильність — найочевидніша з них, і автори посилаються на дослідження в IBM, опубліковане 1986 року, яке показало, що вади, знайдені раніше, виправляти швидше. Книжка додає важливу умову: час, вкладений у рев'ю, окуповувався лише доти, доки процес лишався легким. Статичний аналіз і автоматичні тести сьогодні беруть на себе значну частину роботи з правильності, і, за словами книжки, це зменшило потребу покладатися тут на людей.

А от розуміння інструмент не забезпечить. Рецензент може зробити те, чого навіть найкращий інженер не зробить для власної роботи: прочитати її без авторового погляду, і рев'ю часто стає першою перевіркою того, чи зрозуміла зміна ще комусь. Це важливо, бо код читають набагато більше разів, ніж пишуть. Культурні переваги менш помітні. Рев'ю нагадує інженерові, що код належить не йому особисто, а спільній справі, дає експертам змогу передавати знання про свою ділянку, а оскільки кожна зміна зберігає своє рев'ю, будь-хто згодом може з'ясувати, коли з'явився певний патерн, і прочитати рев'ю, яке його принесло.

Рев'ю також забирає час, а повільне рев'ю — ще більше. Сторінка настанов Google про швидкість каже, що коли рев'ю повільні, падає швидкість усієї команди, розробники починають опиратися процесу, а стан коду

погіршується, бо з’являється тиск пропускати слабші зміни, а охоти прибирати в коді меншає. Більшість скарг на надто суворого рецензента, додають настанови, зникає, коли той самий рецензент просто відповідає швидше. Їхній стандарт схвалення прагматичний: рецензентам варто схвалювати зміну, щойно вона безперечно покращує загальний стан коду системи, навіть якщо вона не досконала. Настанови називають це головним принципом серед своїх правил і вміщують обґрунтування в один рядок: «ідеального коду не існує — існує лише кращий код».

Як це виглядає на практиці

Уявіть інтернет-магазин, де ціни записані в цілих пенсах. Автор додає весняну акцію й надсилає на рев’ю таку зміну:

```
--- 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)
```

Рецензент залишає чотири коментарі:

```
shop/pricing.py, рядок 9
  Інші промокоди та їхні знижки лежать у VOUCHERS у
  shop/vouchers.py. Може, SPRING теж туди? Тоді наступна
  акція буде зміною даних, а не цієї функції.

shop/pricing.py, рядок 10
  Скрізь у цьому модулі ціни в цілих пенсах, а множення
  на 0.9 може залишити частку пенса. Інші знижки проходять
  через round_money() у shop/money.py.

shop/pricing.py, рядок 11
  Nit: доставка не входить у знижку. Якщо це навмисно,
  короткий коментар про це допоможе наступному читачеві.

Загалом
  Бракує тесту з промокодом у кошику. Додай, будь ласка,
  один для SPRING і один, який показує, що невідомий код
  ціну не змінює.
```

Кожен коментар стосується коду, а не людини, і кожен пояснює причину. Перший, другий і останній не мають позначки, тож їх очікують виконати; третій починається з Nit:, яким настанови Google позначають щось дрібне, що автор технічно мав би зробити, але що мало на що вплине. Інші позначки в настановах — Optional, або Consider, для доброї ідеї, яка не є обов'язковою, і FYI для думки, на яку автор не мусить реагувати в цій зміні. Без позначок, попереджають настанови, автори можуть сприймати кожен коментар як обов'язковий. На GitHub цей раунд завершується рішенням Request changes. Автор переносить SPRING у таблицю, пропускає знижку через функцію округлення, додає коментар і обидва тести та просить подивитися ще раз; другий раунд завершується рішенням Approve, а в Google — LGTM. Настанови просять рецензентів відповідати на запит про рев'ю щонайбільше протягом одного робочого дня, але не переривати заради цього зосереджену роботу, а дочекатися природної перерви.

Коментарі не стосуються ні форматування, ні порядку імпортів, ні того, чи проходять тести, бо про це мала сказати машина ще до того, як рецензент відкрив diff. Книжка описує presubmit-перевірки, які запускаються, щойно зміну вперше надіслано на рев'ю, і можуть відхилити її ще до того, як хтось її прочитає; ядро Linux просить авторів перед надсиланням запустити його перевірку стилю, `scripts/checkpatch.pl` . Механізм перевірок, побудований для репозиторію, робота 2025 року, доводить цю ідею далеко: 268 зареєстрованих перевірок коміту, 277 правил лінтингу і 15 власних правил ESLint, а також 146 тестів самих перевірок. Перевірки стежать, зокрема, за тим, що міграцію можна відкотити і що ключ перекладу існує в обох локалях, а будь-який коміт із префіксом fix: мусить містити тест, який без цього виправлення падає. Перевірки закодували механічні правила, тож рецензенти могли приділяти більше часу проєктуванню. Вона ж називає й ціну: комітити повільно, а погано написана перевірка, яку доводиться обходити, справді доводить до сказу.

Де виникають проблеми

Зміна завелика, щоб її прочитати. Настанови Google вважають 100 рядків зазвичай розумним розміром зміни, а 1 000 рядків — зазвичай завеликим, хоча остаточне рішення лишають рецензентові, і зауважують, що 200 рядків в одному файлі можуть бути цілком прийнятними, тоді як ті самі 200 рядків, розкидані по 50 файлах, — ні. Невеликі зміни переглядають швидше й ретельніше, і вони рідше вносять помилки; книжка додає, що рецензенти мають повне право відхилити зміну як завелику для одного рев'ю.

Розбивайте роботу: складайте невеликі зміни одна на одну, розсилайте групи файлів різним рецензентам, тримайте рефакторинг окремо від нових функцій і виправлень помилок, а тести кожної зміни — в тій самій зміні.

Рев'ю лежить днями. Кожен день, поки зміна чекає, відсуває роботу, що на ній будується, а рецензент, який відповідає раз на кілька днів, та ще й великими вимогами, робить процес суворішим на вигляд, ніж він є. Відповідайте протягом одного робочого дня; книжка очікує відгуку протягом 24 робочих годин, а якщо повне рев'ю мусить зачекати, рецензентові варто сказати, що зміну він бачив. Надсилайте всі коментарі разом, а не частинами, бо мало що дратує автора більше, ніж новий раунд не пов'язаних зауважень після того, як попередні вже враховано.

Смак видають за вимогу. Рецензент, який віддає перевагу іншому циклу чи іншій назві й блокує через це зміну, привчає авторів більше не покращувати код. Настанови Google встановлюють порядок: технічні факти й дані важать більше, ніж думки та особисті вподобання, у питаннях стилю абсолютним авторитетом є посібник зі стилю, а там, де кілька підходів однаково слушні, перевагу має вибір автора. Позначайте вподобання як Nit або Optional, щоб автор міг відрізнити його від вимоги.

Коментар стосується людини. У прикладі поганого коментаря з настанов рецензент питає, навіщо автор використав потоки, коли від конкурентності очевидно немає користі; добрий варіант каже, що модель конкурентності додає складності без виграшу в продуктивності, який бачив би рецензент, тож код краще зробити однопотоковим. Коментуйте код, а не розробника, і пояснюйте, чим обґрунтоване прохання.

Люди перевіряють те, що могла б машина. Рецензенти, які витрачають увагу на відступи й порядок імпортів, пропускають питання проєктування, на яке може відповісти лише людина. Перенесіть кожне правило, яке можна записати, у лінтер, форматер або presubmit-перевірку, а рецензентові залиште судження.

Відповідь лишається в інструменті рев'ю. Рецензент питає, для чого потрібен блок коду, автор пояснює це в обговоренні, рецензент схвалює, і наступна людина, яка читатиме код, пояснення так і не побачить. Настанови Google кажуть, що таке питання зазвичай має привести до того, що код перепишуть зрозуміліше, або принаймні до коментаря в коді. Ядро Linux

каже авторам те саме: запитання на рев'ю, які не ведуть до зміни коду, майже напевно мають обернутися коментарем у коді або записом у журналі змін.

Що почитати далі

- Introduction — вступна сторінка настанов Google щодо рев'ю коду: на що дивляться рецензенти, як їх обирати і коли парне програмування зараховують як рев'ю.
- The Standard of Code Review — головний принцип, наставництво через рев'ю і розв'язання суперечок між автором і рецензентом.
- Small CLs — чому невеликі зміни отримують кращі рев'ю, коли зміна завелика, і чотири способи її розбити.
- Code Review — розділ Tom Manshreck і Caitlin Sadowski у *Software Engineering at Google*: переваги рев'ю, LGTM і власники коду, найкращі практики.
- Pull request reviews — довідка GitHub про рев'ю, схвалення й запит на зміни та про обов'язкові схвалення перед злиттям.
- Submitting patches: the essential guide to getting your code into the kernel — як ядро Linux рецензує зміни через електронну пошту, що засвідчує тег Reviewed-by і як авторам відповідати рецензентам.

Словник

Обирайте за фактичними умовами

Опишіть робочий процес, наявні інтерфейси та користувачів, яким потрібна зміна. Ми можемо визначити перевірюваний результат і спосіб передачі, який дасть вашій команді змогу підтримувати його.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке рев'ю коду?» стосується вашої організації.
Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/code-review/