Engineer

Refaktorering

🛠 at ændre kodens struktur uden at ændre, hvad den gør

Hvad det er

Refaktorering ændrer kodens struktur, mens dens observerbare adfærd bevares. Man kan for eksempel flytte beregningen af en ordretotal til en navngiven funktion og kontrollere, at kvitteringen stadig viser det samme beløb. Koden bliver lettere at forstå og nemmere at ændre ved næste opgave. De samme input giver stadig de samme svar og sideeffekter. Udsagnsordet er at refaktorere, og en enkelt sådan ændring kaldes også en refaktorering. På engelsk hedder det refactoring.

Martin Fowler definerer begrebet i sin bog Refactoring og gentog definitionen på sit websted i 2004. Som navneord er en refaktorering en ændring af softwarens indre struktur, der gør den lettere at forstå og billigere at ændre, uden at dens observerbare adfærd ændres. Som udsagnsord betyder at refaktorere at omstrukturere software ved at anvende en række af den slags ændringer, igen uden at ændre den observerbare adfærd. Begge halvdele tæller. Formålet er en billigere næste ændring, og betingelsen er, at ingen uden for koden kan se, at der er sket noget.

Fowler trækker grænsen skarpt: »Refactoring isn’t another word for cleaning up code,« skriver han på refactoring.com. Ordet betegner én bestemt teknik. Han bruger omstrukturering (restructuring) som det bredere ord for at omorganisere kode med alle midler, også nogle, der ikke bevarer adfærden ved hvert skridt. Det, der gør refaktorering til en teknik, er størrelsen. Hvert skridt er en lille transformation, der ikke gør meget i sig selv og sjældent går galt, og systemet holdes fuldt fungerende efter hvert eneste. En lang række af dem kan alligevel samlet blive til en stor ændring af strukturen.

Skridtene har navne. Kataloget på refactoring.com, der hører til bogens anden udgave, opregner dem sammen med de ældre navne, de afløser: Extract Function, tidligere kaldt Extract Method, og dens modstykke Inline Function; Inline Variable; Rename Variable; Move Function; Split Phase og snesevis flere. Et fælles navn lader en udvikler sige med to ord, hvad en ændring var, og lader den, der gennemgår koden, holde den op mod en kendt opskrift.

Brug begrebet i en beslutning

Brug »Hvad er refaktorering?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Ét team, der tager en idé fra skitse til noget, jeres kunder kan logge ind på, i stedet for at kaste den mellem tre leverandører.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har ført en GIS-kortapp til 10x omsætningsvækst — 10x omsætningsvækst, drevet af kortproduktet

Har ført en GIS-kortapp til 10x omsætningsvækst Kontakt os →

Hvorfor det betyder noget

Software, der får succes, bliver ved med at ændre sig. Ny funktionalitet og rettelser lægges oven på hinanden, og hver af dem gør typisk den næste lidt sværere, indtil nyt arbejde, som Fowler beskriver det, næsten går i stå. Refaktorering er modvægten: den omformer koden, så hver tilføjelse ikke bringer kompleksitet med sig, som den ikke har brug for. Udbyttet måles i prisen på den næste ændring, ikke i, hvordan koden ser ud.

Derfor behandler Fowler det som en del af almindelig programmering og ikke som en opgave, der står i en projektplan. Han nævner tre øjeblikke. Før han tilføjer ny funktionalitet, tjekker han, om koden har en form, der gør den let at tilføje, og hvis ikke, refaktorerer han først, hvilket han som regel finder hurtigere end at tvinge funktionaliteten ind. Når funktionaliteten virker, refaktorerer han resultatet til en form, som en, der vender tilbage om et par uger, kan følge uden at gruble. Og når han skal læse ukendt kode for at forstå den, refaktorerer han, så den næste læser ikke kæmper på samme sted. Det første af de tre har sit eget navn, forberedende refaktorering (preparatory refactoring), og Fowler tilskriver Kent Beck reglen bag det: ved hver ændring skal man først omforme koden, indtil ændringen er let, hvilket kan være den svære del, og først derefter foretage den.

Refaktorering holder også risikoen nede på en måde, som arbejde på ny funktionalitet ikke kan. Fowler giver endnu en af Becks metaforer videre, de to hatte. Med refaktoreringshatten på bevarer hver ændring adfærden, og testene forbliver grønne hele vejen. Når man tilføjer funktionalitet, kommer der nye tests til, og eksisterende tests går i stykker, hvilket er mere stressende og mere risikabelt. En udvikler, der ved, hvilken hat der sidder på hovedet, ved også, hvad en fejlende test betyder: under en refaktorering betyder den, at det sidste skridt var forkert.

Det er også sådan, det meste af den tekniske gæld bliver betalt tilbage. Googles artikel om skjult gæld i maskinlæringssystemer nævner refaktorering af kode først blandt måderne at afdrage gæld på. Værktøjer hjælper: mange sprog har nu editorer, der automatiserer almindelige refaktoreringer. Fowler regner sådanne værktøjer for værdifulde, men ikke nødvendige, og hvor de mangler, holder han sig til små skridt og hyppige tests.

Sådan ser det ud i praksis

Tag en funktion, skrevet til denne artikel, der lægger en ordre sammen og udskriver en kvittering samme sted:

```
function printReceipt(order) {
  let total = 0;
  for (const line of order.lines) {
    total += line.price * line.quantity;
  }

  // print the receipt lines
  console.log(`Order ${order.id}`);
  console.log(`Total: ${total}`);
}
```

Et nyt krav kommer ind: den samme total skal stå på en faktura. I stedet for at kopiere løkken refaktorerer udvikleren først. Første skridt er Extract Function: løkken flyttes ind i sin egen funktion, opkaldt efter det, den beregner, og testene køres. Andet skridt trækker udskrivningen ud på samme måde, så et funktionsnavn erstatter kommentaren, og testene køres igen. Tredje skridt er Inline Variable: variablen `total` bruges nu kun ét sted, så kaldet sættes ind i stedet for den. Efter hvert skridt gør programmet præcis det samme som før og kunne committes for sig:

```
function printReceipt(order) {
  printLines(order.id, orderTotal(order));
}

function orderTotal(order) {
  let total = 0;
  for (const line of order.lines) {
    total += line.price * line.quantity;
  }
  return total;
}

function printLines(id, total) {
  console.log(`Order ${id}`);
  console.log(`Total: ${total}`);
}
```

Først nu kommer totalen på fakturaen til, og det kræver én linje i fakturakoden, der kalder `orderTotal`. Intet i de tre refaktoreringsskridt krævede en ny test, for intet observerbart ændrede sig; det er de tests, der allerede dækkede kvitteringen, som beviser det.

Infrastrukturkode bliver også refaktoreret, og der er indsatsen højere, fordi navnene henviser til kørende ressourcer. Terraforms dokumentation advarer om, at en ændret adresse på en ressource i konfigurationen som standard er en instruks om at slette den eksisterende ressource og oprette en ny. Omdøber man `aws_instance.a` til `aws_instance.web` for læsbarhedens skyld, bliver en server altså erstattet. Siden Terraform v1.1 registrerer en `moved` -blok i stedet den gamle adresse, og den næste plan opdaterer adressen uden at slette noget:

```
resource "aws_instance" "web" {
  # (configuration unchanged)
}

moved {
  from = aws_instance.a
  to   = aws_instance.web
}
```

I tidligere versioner henviser dokumentationen til kommandoen `terraform state mv` til den samme flytning. Uanset hvad er refaktoreringen først sikker, når planen er læst og viser en flytning og ikke en sletning og en oprettelse.

Det er lettere at holde den slags ændringer adskilt fra arbejde på ny funktionalitet, når historikken siger, hvad der er hvad. Commit-standarden, som sidens porteføljepost om at holde fire repositories på én historikstandard beskriver, lader hver commit-besked begynde med et konventionelt præfiks, der angiver ændringens art, og en hook på commit-beskeden håndhæver det. Porteføljeposten, der handler om arbejde udført i juli 2026, tæller 26 refaktoreringer i historikken for det største repository ved siden af 156 funktions-commits og 133 rettelser. Den siger også ærligt, at hooken tjekker formatet og ikke indholdet. En commit, der er mærket som refaktorering, skal stadig gennem sine tests og en kodegennemgang vise, at den har ladet adfærden være i fred.

Hvor det går galt

Adfærdsændringer sniger sig med i refaktoreringen. En udvikler rydder op i en funktion og retter en fejl i den i samme commit. Når en test fejler, kan ingen se, hvilken af de to der var årsagen, og den, der gennemgår koden, må læse hver linje som en mulig ændring af adfærden. Hav én hat på ad gangen: refaktorér i én commit, ændr adfærden i den næste, og hvis en test fejler under refaktoreringen, så fortryd det sidste skridt i stedet for at lappe videre på det.

En stor omstrukturering bliver kaldt refaktorering. En branch står i ugevis med et build, der ikke virker, mens koden angiveligt »bliver refaktoreret«. Det er omstrukturering i Fowlers forstand, og den bærer netop den risiko, som refaktorering er skabt til at undgå. Del arbejdet op i skridt, der hver især efterlader systemet i fungerende stand og kan merges for sig; hvor det ikke kan lade sig gøre, så kald det en omskrivning og planlæg det som en.

Der er ingen tests, der viser, at intet har ændret sig. Bevaret adfærd er en påstand, og uden tests er den et gæt. Før du rører kode uden tests, så skriv tests, der fastholder, hvad den gør i dag, også hvor det ser forkert ud, og ret mærkværdighederne i separate ændringer bagefter. Fowlers tommelfingerregel fra hans eksempel på forberedende refaktorering gælder her: hver gang trangen melder sig til at køre koden og kigge på outputtet med egne øjne, så skriv en test i stedet.

En omdøbning i infrastrukturkode erstatter ressourcen. Et pænt nyt navn i Terraform ligner en refaktorering i kodegennemgangen, og planen foreslår i al stilhed at slette en database og oprette en tom. Registrér hver omdøbning og

hver flytning ind i eller ud af et modul med en `moved` -blok, og betragt enhver sletning i planen for en ændring, der var tænkt som refaktorering, som et mislykket skridt.

En gammel `moved` -blok bliver slettet. Moved-blokke hober sig op, og at fjerne dem ligner almindelig oprydning. Terraforms dokumentation kalder det en ændring, der bryder kompatibiliteten: enhver konfiguration, der stadig står på den gamle adresse, vil så planlægge at slette det eksisterende objekt i stedet for at flytte det. Behold de historiske blokke, kæd dem sammen, når et objekt omdøbes to gange, og fjern dem kun fra et privat modul, når alle, der bruger det, har kørt apply med den version, der indeholder dem.

Refaktoreringen venter på en særlig fase. Et team, der udskyder refaktoreringen til et særligt forløb senere, bliver ved med at tilføje funktionalitet til kode, der stritter imod, og et forløb, der er planlagt til senere, er let at skære væk, når en deadline flytter sig. Fowlers råd går den anden vej: refaktorér som en del af hver ændring, først for at gøre ændringen let og bagefter for at efterlade resultatet klart, så arbejdet aldrig har brug for sin egen plads i planen.

Læs mere

- Definition Of Refactoring — Martin Fowlers definitioner som navneord og udsagnsord, sådan som han giver dem i sin bog.
- Refactoring — Fowlers websted for bogen, om hvorfor refaktorering hører til den daglige programmering, og med hele kataloget.
- Extract Function — katalogets opslag om en af de mest almindelige refaktoreringer, med dens modstykke og navnet fra første udgave.
- An example of preparatory refactoring — Fowler omformer sin egen kode i små skridt, før han tilføjer ny funktionalitet, og forklarer de to hatte.
- Refactor modules — Terraforms dokumentation om `moved` -blokke, omdøbninger, opdeling af moduler, og hvorfor gamle blokke skal blive stående.



Ordliste

Vælg ud fra jeres faktiske vilkår

Beskriv arbejdsgangen, de eksisterende grænseflader og de brugere, der har brug for ændringen. Vi kan definere en testbar leverance og en overdragelse, der gør teamet i stand til at vedligeholde løsningen.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er refaktorering?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

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

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

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

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/refactoring/