Engineer

Teknisk gjeld

merkostnaden som en rask nødløsning i dag gir senere

Hva det er

Teknisk gjeld er den fremtidige prisen for en snarvei i et systems design eller kode: et valg som sparer tid i dag og gjør senere endringer tregere, mer risikable eller umulige, helt til snarveien er fjernet igjen. På engelsk heter det technical debt, ofte forkortet til tech debt. Ordet er en metafor, og det ble valgt fordi snarveien oppfører seg som et lån. Den kjøper noe nå og tar betalt for det senere, så lenge den blir liggende.

Idéen stammer fra Ward Cunninghams erfaringsrapport på konferansen OOPSLA i 1992 om WyCash, et system for porteføljeforvaltning skrevet i Smalltalk. Et lite team hadde oversikt over rundt fire megabyte kildekode, og de modne delene av programmet var blitt revidert eller skrevet om mange ganger. Det er denne konsolideringen, argumenterer rapporten, som gjorde en jevn, gradvis vekst mulig, og mangelen på den er den alvorlige fallgruven: kode som virker og gjør kunden fornøyd, men som aldri er blitt brakt i orden. Cunningham sa det på én linje: «Shipping first time code is like going into debt.» Selve rapporten snakker bare om gjeld og om gjeldsanalogien sin. Senere arbeider bruker det fulle navnet, og en artikkel fra Google fra 2015 kaller teknisk gjeld en metafor som Ward Cunningham introduserte i 1992.

Forskere som møttes på Dagstuhl i april 2016 for å bli enige om hva begrepet dekker, skrev en arbeidsdefinisjon: en samling design- eller implementasjonskonstruksjoner som er beleilige på kort sikt, men som skaper en teknisk kontekst som kan gjøre fremtidige endringer dyrere eller umulige. De avgrenser virkningen til kvaliteter inne i systemet, først og fremst hvor lett det er å vedlikeholde og videreutvikle det. Det skiller gjeld fra en funksjon som ingen bygde: prisen for gjelden faller på dem som endrer systemet, og når brukerne bare gjennom dem. Den samme rapporten beholder det finansielle ordforrådet. Hovedstolen er det snarveien sparte, eller hva en bedre løsning ville koste i dag. Rentene finnes i to slag: løpende renter (recurring interest), som betales i form

av lavere tempo, følgefeil og tapt kvalitet, og voksende renter (accruing interest), den ekstra prisen for hver ny bit programvare som bygges oppå koden som ikke er helt riktig. Gjeld bor mest i kode, men også i dokumentasjon, tester og kjente feil.

Bruk begrepet i en beslutning

Bruk «Hva er teknisk gjeld?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Formen på et system tegnet opp på papiret først, slik at delene passer sammen og kostnaden er kjent før noen skriver kode.

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. Én infrastrukturplan for 14 avdelinger — 14 forskningsavdelinger på én felles infrastrukturplan

Én infrastrukturplan for 14 avdelinger Ta kontakt →

Hvorfor det er viktig

Gjeld som tas opp bevisst, kan være en god beslutning. Cunninghams rapport sier at litt gjeld setter fart på utviklingen, så lenge den raskt betales tilbake med en omskriving, og Google-artikkelen er enig i at det ofte finnes gode strategiske grunner til å låne. Det begge advarer mot, er gjeld som blir liggende. I rapporten teller hvert minutt som brukes på kode som ikke er helt riktig, som renter, og hele utviklingsorganisasjoner kan gå i stå under vekten av en implementasjon som ingen konsoliderte. Før det skjer, blir programmet noe ingen enkeltperson kan overskue, noe som fører til programmerere som kjenner hvert sitt hjørne, og til slutt til et produkt som ikke kan skifte retning.

Problemet er at gjeld ofte ikke er et bevisst valg. Dagstuhl-rapporten påpeker at den ofte oppstår utilsiktet og overrasker utviklerne, og derfor oppdages den mest under vedlikehold, lenge etter beslutningen som skapte den. Årsakene er hverdagslige: tidspress, en nøkkelperson som ikke er tilgjengelig, eller for lite kunnskap om en teknisk egenskap på det tidspunktet. Google-artikkelen legger til at skjult gjeld er den farlige typen, fordi den vokser i det stille.

Metaforen gir også utviklere og ledere et felles språk. Dagstuhl-rapporten kaller den en sterk kommunikasjonsmekanisme, og visjonen deltakerne beskriver, omfatter å styre gjeld like nøye som feil og nye funksjoner, med en måte å oversette mellom det utviklere bekymrer seg for, og det ledere bestemmer. En studie som ble lagt fram på seminaret, viser hva det kan bety i et team. Graziela Tonin, Alfredo Goldman og Carolyn Seaman ba studenter i team med til sammen 60 personer om å identifisere og følge gjelden sin eksplisitt. En synlig liste lot

teamene forhandle om tid til refaktorering med kunden sin, fungerte som hukommelse for de umodne delene av prosjektet og viste om kodekvaliteten ble bedre. De fleste teammedlemmene sa at de kommuniserte mer, tenkte grundigere over om en gjeld virkelig var nødvendig, og refaktorerte oftere. Fordi beslutninger om å ta opp gjeld ble diskutert, legger forfatterne til, endte noen team opp med færre eksperter som ingen andre kunne tre inn for.

Slik ser det ut i praksis

Gjeld blir håndterbar når den skrives ned der teamet planlegger arbeidet sitt. En oppføring i et register, skrevet for denne artikkelen, kan være så kort som denne:

```
TD-14       Fakturatotaler beregnes tre steder
Artefakter: billing/web, billing/export, reports/monthly
Årsak:      en fast lanseringsdato; det var raskere å kopiere regelen
            enn å flytte den inn i en felles modul
Hovedstol:  rundt 2 dager å flytte regelen inn i én modul, med tester
Renter:     hver endring i skattereglene gjøres og testes tre ganger, og
            én av de siste fem endringene glemte en av kopiene
Neste:      nedbetal med neste endring i skattereglene; eksporten av
            e-fakturaer som er planlagt dette kvartalet, må ikke bli
            kopi nummer fire
Eier:       faktureringsteamet
```

Feltene følger modellen fra Dagstuhl-rapporten: en oppføring knyttet til konkrete artefakter, med en årsak, en hovedstol og renter. Det er rentefeltet som gjør oppføringen nyttig, for det gjør en følelse av at faktureringskoden er rotete, om til en pris som noen kan veie. Endres skattereglene fire ganger i året, og koster hver endring en ekstra dags arbeid og testing, er de to dagene med hovedstol tjent inn i løpet av et halvt år. Den planlagte eksporten er voksende renter i Dagstuhl-forstand: hver ny bit som bygges på den kopierte regelen, gjør det dyrere å rette den senere. Var koden derimot frosset, og ventet ingen å røre den igjen, ville den samme oppføringen være et argument for å la gjelden ligge.

Gjeld handler ikke bare om kode. Porteføljeoppføringen her på nettstedet om å fornye interne prosesser og spare 8 000 timer dokumenterer grunnleggerens tidligere arbeid hos Mappitall. Den spør leseren om teknisk gjeld bremser dem, og beskriver den typen som hoper seg opp rundt hvordan arbeidet gjøres: en programvarearkitektur som hadde vokst lag på lag og ikke etter design, systemer som fungerte, men ikke effektivt, og planlegging som enten lot folk vente eller druknet dem i arbeid. Ingenting av det sto i brann, og det var nettopp derfor det

hadde fått ligge. Å legge om arkitekturen, systemene og planleggingen sparte omtrent 8 000 timer i arbeidet fra 2015 til 2022, og porteføljeoppføringen legger vekt på at endringene ble bygget inn i hvordan teamet jobbet, for prosessforbedringer som ikke gjøres permanente, forfaller tilbake til gamle vaner.

Hvor det går galt

Alt blir kalt teknisk gjeld. Når uttrykket først er populært, får enhver irritasjon merkelappen, fra et gammelt bibliotek via en manglende funksjon til en kollegas stil. Dagstuhl-rapporten nevner den risikoen direkte og ber om skarpe definisjoner som svar. Bruk bare begrepet om snarveier der prisen havner på fremtidige endringer, registrer feil som feil og ønsker som funksjoner, og avvis en oppføring i registeret som ikke kan oppgi årsak og renter.

Gjelden finnes bare i hodene til folk. Gjeld som ingen har skrevet ned, betales av den som rører koden neste gang, som regel uten å vite hvorfor arbeidet tar så lang tid. Hold et register ved siden av backloggen, og sjekk det før en del av systemet endres, slik studentteamene i Tonin-studien brukte listen sin. En oppføring blir først synlig for dem som bestemmer hva tiden brukes på, når den står der de ser.

Hovedstolen betales ned der det ikke løper renter. Å rydde i en modul som ingen endrer, føles produktivt og gir ingenting tilbake, mens koden som endres hver uke, fortsetter å koste. Renter løper bare der folk jobber, så sorter registeret etter hvor ofte hver del endres og etter hva som snart skal bygges oppå den, og betal ned de dyre oppføringene først.

Nedbetalingen venter på én stor omskriving. Gjeld som har ligget i årevis, blir ofte møtt med et prosjekt som skal erstatte hele systemet, og som tar opp ny gjeld for egen regning mens det gamle systemet fortsetter å koste. Cunninghams råd var å betale raskt tilbake, og Google-artikkelen lister opp de vanlige måtene å betale ned på: refaktorering av kode, bedre enhetstester, sletting av død kode, færre avhengigheter, strammere API-er og bedre dokumentasjon. Ingen av dem legger til en funksjon; poenget er å gjøre neste endring billigere. Planlegg dem som en del av det vanlige arbeidet, tett på endringen som trenger dem.

Eksperimenter blir værende i produksjon. Å prøve ut et alternativ som en betinget gren i produksjonskoden er billig for hvert enkelt eksperiment, og Google-artikkelen beskriver hvordan slike grener hoper seg opp til det blir vanskelig eller umulig å teste alle kombinasjoner. Den nevner Knight Capital, der systemet ifølge artikkelen tapte 465 millioner dollar på 45 minutter, tilsynelatende fordi foreldede eksperimentelle kodestier oppførte seg uventet. Gå gjennom hver eksperimentelle gren og hvert flagg etter en fast plan, slett det som ikke lenger brukes, og belønn sletting av kode like mye som tillegg.

Et verktøys poengsum blir tatt for gjelden. Statisk analyse kan telle dupliserte linjer og lange funksjoner, og et dashbord får tallet til å se ut som en saldo. Google-artikkelen sier rett ut at metaforen ikke gir noen streng målestokk å følge over tid, og at et team som fortsatt beveger seg raskt, ikke er bevis for lav gjeld. På Dagstuhl fortalte Forrest Shull at ikke noe enkelt sett med regler for å finne gjeld hang konsekvent sammen med symptomene på den, som økende endringskostnader og fallende tempo, på tvers av ulike prosjekter. Hold et verktøys funn opp mot deres egen historikk av endringer og feil, behold reglene som forutsier kostnader, og dropp resten. Artikkelens egne spørsmål er en bedre prøve, blant annet hvor raskt et nytt medlem av teamet kan komme i gang.

Les mer

- The WyCash Portfolio Management System — Ward Cunninghams erfaringsrapport fra OOPSLA ‘92, der gjeldsanalogien dukker opp for første gang.
- Managing Technical Debt in Software Engineering — rapporten fra Dagstuhl-seminar 16162 med arbeidsdefinisjonen, hovedstol og renter og studien av eksplisitte gjeldslister.
- Hidden Technical Debt in Machine Learning Systems — artikkelen fra 2015 av Sculley og kolleger hos Google om hvordan gjeld hoper seg opp, hvordan den betales ned, og hvorfor den er vanskelig å måle.
- Refactoring — Martin Fowlers side om de små, atferdsbevarende endringene som Google-artikkelen nevner først blant måtene å betale ned gjeld på.



Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er teknisk gjeld?» 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/technical-debt/