Engineer

Teknisk gæld

den merpris, en hurtig nødløsning i dag koster senere

Hvad det er

Teknisk gæld er den fremtidige pris for en genvej i et systems design eller kode: et valg, der sparer tid i dag og gør senere ændringer langsommere, mere risikable eller umulige, indtil genvejen er fjernet igen. På engelsk hedder det technical debt, ofte forkortet til tech debt. Ordet er en metafor, og det blev valgt, fordi genvejen opfører sig som et lån. Den køber noget nu og tager betaling for det senere, så længe den bliver liggende.

Idéen stammer fra Ward Cunninghams erfaringsrapport på konferencen OOPSLA i 1992 om WyCash, et system til porteføljestyring skrevet i Smalltalk. Et lille team havde overblik over omkring fire megabyte kildekode, og programmets modne dele var blevet revideret eller omskrevet mange gange. Det er den konsolidering, argumenterer rapporten, der gjorde en jævn, gradvis vækst mulig, og fraværet af den er den alvorlige faldgrube: kode, der virker og gør kunden glad, men aldrig er blevet bragt i form. Cunningham sagde det på én linje: »Shipping first time code is like going into debt.« Selve rapporten taler kun om gæld og om sin gældsanalogi. Senere arbejder bruger det fulde navn, og en artikel fra Google fra 2015 kalder teknisk gæld en metafor, som Ward Cunningham introducerede i 1992.

Forskere, der mødtes på Dagstuhl i april 2016 for at blive enige om, hvad begrebet dækker, skrev en arbejdsdefinition: en samling design- eller implementeringskonstruktioner, der er bekvemme på kort sigt, men skaber en teknisk kontekst, som kan gøre fremtidige ændringer dyrere eller umulige. De afgrænser virkningen til kvaliteter inde i systemet, først og fremmest hvor let det er at vedligeholde og videreudvikle. Det skiller gæld fra en funktion, som ingen byggede: gældens pris falder på dem, der ændrer systemet, og når kun brugerne gennem dem. Den samme rapport beholder det finansielle ordforråd. Hovedstolen er det, genvejen sparede, eller hvad en bedre løsning ville koste i dag. Renterne findes i to slags: løbende renter (recurring interest), der betales

som lavere hastighed, afledte fejl og tabt kvalitet, og voksende renter (accruing interest), den ekstra pris for hvert nyt stykke software, der bygges oven på koden, som ikke er helt rigtig. Gæld bor mest i kode, men også i dokumentation, tests og kendte fejl.

Brug begrebet i en beslutning

Brug »Hvad er teknisk gæld?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Systemets form lagt fast på papir først, så delene passer sammen, og prisen er kendt, før nogen skriver kode.

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. Én infrastrukturplan til 14 afdelinger — 14 forskningsafdelinger på én infrastrukturplan

Én infrastrukturplan til 14 afdelinger Kontakt os →

Hvorfor det betyder noget

Gæld, der optages bevidst, kan være en god beslutning. Cunninghams rapport siger, at lidt gæld sætter fart på udviklingen, så længe den hurtigt betales tilbage med en omskrivning, og Google-artiklen er enig i, at der ofte er gode strategiske grunde til at låne. Det, begge advarer imod, er gæld, der bliver liggende. I rapporten tæller hvert minut, der bruges på kode, som ikke er helt rigtig, som renter, og hele udviklingsorganisationer kan gå i stå under vægten af en implementering, som ingen konsoliderede. Før det sker, bliver programmet noget, som ingen enkelt person kan overskue, hvilket fører til programmører, der hver kender ét hjørne, og til sidst til et produkt, der ikke kan skifte retning.

Problemet er, at gæld ofte ikke er et bevidst valg. Dagstuhl-rapporten bemærker, at den ofte opstår utilsigtet og overrasker udviklerne, og derfor opdages den mest under vedligeholdelse, længe efter den beslutning, der skabte den. Årsagerne er almindelige: tidspres, en nøgleperson, der ikke er til rådighed, eller for lidt viden om en teknisk egenskab på det tidspunkt. Google-artiklen tilføjer, at skjult gæld er den farlige slags, fordi den vokser i det stille.

Metaforen giver også udviklere og ledere et fælles sprog. Dagstuhl-rapporten kalder den en stærk kommunikationsmekanisme, og den vision, deltagerne beskriver, omfatter at styre gæld lige så omhyggeligt som fejl og nye funktioner, med en måde at oversætte mellem det, udviklere bekymrer sig om, og det, ledere beslutter. Et studie, der blev præsenteret på seminaret, viser, hvad det kan betyde i et team. Graziela Tonin, Alfredo Goldman og Carolyn Seaman bad studerende i teams med i alt 60 personer om at identificere og følge deres gæld

eksplicit. En synlig liste lod teamene forhandle tid til refaktorering med deres kunde, fungerede som hukommelse for projektets umodne dele og viste, om kodekvaliteten blev bedre. De fleste teammedlemmer sagde, at de kommunikerede mere, tænkte grundigere over, om en gæld virkelig var nødvendig, og refaktorerede oftere. Fordi beslutninger om at optage gæld blev drøftet, tilføjer forfatterne, endte nogle teams med færre eksperter, som ingen andre kunne træde til for.

Sådan ser det ud i praksis

Gæld bliver håndterbar, når den skrives ned der, hvor teamet planlægger sit arbejde. En post i et register, skrevet til denne artikel, kan være så kort som denne:

```
TD-14       Fakturatotaler beregnes tre steder
Artefakter: billing/web, billing/export, reports/monthly
Årsag:      en fast lanceringsdato; det var hurtigere at kopiere reglen
            end at flytte den ind i et fælles modul
Hovedstol:  omkring 2 dage at flytte reglen ind i ét modul, med tests
Renter:     hver ændring af skattereglerne laves og testes tre gange, og
            én af de seneste fem ændringer overså en af kopierne
Næste:      indfri med næste ændring af skattereglerne; eksporten af
            e-fakturaer, der er planlagt i dette kvartal, må ikke blive
            kopi nummer fire
Ejer:       faktureringsteamet
```

Felterne følger modellen fra Dagstuhl-rapporten: en post knyttet til konkrete artefakter, med en årsag, en hovedstol og renter. Det er rentefeltet, der gør posten nyttig, for det forvandler en fornemmelse af, at faktureringskoden er rodet, til en pris, som nogen kan veje. Ændres skattereglerne fire gange om året, og koster hver ændring en ekstra dags arbejde og test, er de to dages hovedstol tjent hjem inden for et halvt år. Den planlagte eksport er voksende renter i Dagstuhl-forstand: hvert nyt stykke, der bygges på den kopierede regel, gør det dyrere at rette den senere. Var koden derimod frosset, og ventede ingen at røre den igen, ville den samme post være et argument for at lade gælden ligge.

Gæld handler ikke kun om kode. Sidens porteføljepost om at forny interne processer og spare 8.000 timer dokumenterer grundlæggerens tidligere arbejde hos Mappitall. Den spørger læseren, om teknisk gæld bremser dem, og beskriver den slags, der hober sig op omkring, hvordan arbejdet foregår: en softwarearkitektur, der var vokset ved påbygninger og ikke ved design, systemer,

der virkede, men ikke effektivt, og en planlægning, der lod folk enten vente eller drukne i arbejde. Intet af det var akut, og netop derfor havde ingen rørt ved det. At lægge arkitekturen, systemerne og planlægningen om sparede omkring 8.000 timer i arbejdet fra 2015 til 2022, og porteføljeposten lægger vægt på, at ændringerne blev forankret i teamets måde at arbejde på, for procesforbedringer, der ikke gøres permanente, glider tilbage til den gamle måde.

Hvor det går galt

Alt bliver kaldt teknisk gæld. Når udtrykket først er populært, får enhver irritation mærkatet, fra et gammelt bibliotek over en manglende funktion til en kollegas stil. Dagstuhl-rapporten nævner den risiko direkte og beder om skarpe definitioner som svar. Brug kun begrebet om genveje, hvis pris lander på fremtidige ændringer, registrér fejl som fejl og ønsker som funktioner, og afvis en post i registret, der ikke kan nævne sin årsag og sine renter.

Gælden findes kun i folks hoveder. Gæld, som ingen har skrevet ned, betales af den, der rører koden næste gang, som regel uden at vide, hvorfor arbejdet tager så lang tid. Hold et register ved siden af backloggen, og tjek det, før en del af systemet ændres, sådan som de studerendes teams i Tonin-studiet brugte deres liste. En post bliver først synlig for dem, der beslutter, hvad tiden bruges på, når den står der, hvor de kigger.

Hovedstolen betales af, hvor der ikke løber renter. At rydde op i et modul, som ingen ændrer, føles produktivt og giver intet igen, mens koden, der ændres hver uge, bliver ved med at koste. Renter løber kun på, hvor folk arbejder, så sortér registret efter, hvor ofte hver del ændres, og efter, hvad der snart skal bygges oven på den, og betal de dyre poster først.

Afviklingen venter på én stor omskrivning. Gæld, der har ligget i årevis, bliver tit besvaret med et projekt, der skal erstatte hele systemet, og som optager ny gæld for egen regning, mens det gamle system bliver ved med at koste. Cunninghams råd var at betale hurtigt tilbage, og Google-artiklen opregner de sædvanlige måder at afdrage på: refaktorering af kode, bedre unit tests, sletning af død kode, færre afhængigheder, strammere API'er og bedre dokumentation. Ingen af dem tilføjer en funktion; pointen er at gøre den næste ændring billigere. Planlæg dem som en del af det almindelige arbejde, tæt på den ændring, der har brug for dem.

Eksperimenter bliver i produktion. At afprøve et alternativ som en betinget gren i produktionskoden er billigt for hvert enkelt eksperiment, og Google-artiklen beskriver, hvordan sådanne grene hober sig op, indtil det bliver svært eller umuligt at teste alle kombinationer. Den nævner Knight Capital, hvis system ifølge artiklen tabte 465 millioner dollars på 45 minutter, tilsyneladende fordi forældede eksperimentelle kodestier opførte sig uventet. Gennemgå hver eksperimentel gren og hvert flag efter en fast plan, slet det, der ikke længere bruges, og beløn sletning af kode lige så meget som tilføjelse.

Et værktøjs score bliver taget for gælden. Statisk analyse kan tælle duplikerede linjer og lange funktioner, og et dashboard får tallet til at ligne en saldo. Google-artiklen siger ligeud, at metaforen ikke giver nogen stringent målestok at følge over tid, og at et team, der stadig bevæger sig hurtigt, ikke er bevis for lav gæld. På Dagstuhl fortalte Forrest Shull, at intet enkelt sæt regler til at finde gæld hang konsekvent sammen med dens symptomer, såsom stigende ændringsomkostninger og faldende hastighed, på tværs af forskellige projekter. Hold et værktøjs fund op mod jeres egen historik af ændringer og fejl, behold de regler, der forudsiger omkostninger, og drop resten. Artiklens egne spørgsmål er en bedre prøve, blandt andet hvor hurtigt et nyt medlem af teamet kan blive sat ind i tingene.

Læs mere

- The WyCash Portfolio Management System — Ward Cunninghams erfaringsrapport fra OOPSLA ‘92, hvor gældsanalogien optræder første gang.
- Managing Technical Debt in Software Engineering — rapporten fra Dagstuhl-seminar 16162 med arbejdsdefinitionen, hovedstol og renter og studiet af eksplicitte gældslister.
- Hidden Technical Debt in Machine Learning Systems — artiklen fra 2015 af Sculley og kolleger hos Google om, hvordan gæld hober sig op, hvordan den afdrages, og hvorfor den er svær at måle.
- Refactoring — Martin Fowlers side om de små, adfærdsbevarende ændringer, som Google-artiklen nævner først blandt måderne at afdrage gæld på.

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 teknisk gæld?« 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/technical-debt/