Engineer

Technische Schulden

spätere Mehrkosten einer schnellen Abkürzung von heute

Was es ist

Technische Schulden sind die künftigen Kosten einer Abkürzung im Entwurf oder Code eines Systems: Eine Entscheidung spart heute Zeit und macht spätere Änderungen langsamer, riskanter oder unmöglich, bis die Abkürzung beseitigt wird. Teams verwenden oft die Kurzform tech debt. Das Wort ist eine Metapher. Es wurde gewählt, weil sich eine solche Abkürzung wie ein Kredit verhält: Sie verschafft jetzt einen Vorteil und stellt ihn später in Rechnung, solange sie bestehen bleibt.

Die Idee stammt aus Ward Cunninghams Erfahrungsbericht auf der OOPSLA 1992 über WyCash, ein in Smalltalk geschriebenes System zur Portfolioverwaltung. Ein kleines Team behielt einen allgemeinen Überblick über ungefähr vier Megabyte Quelltext. Die ausgereiften Teile des Programms waren vielfach überarbeitet oder neu geschrieben worden. Erst diese Konsolidierung, so der Bericht, ermöglichte stetiges schrittweises Wachstum. Ihr Fehlen ist die ernste Falle: Code funktioniert und stellt den Kunden zufrieden, wurde aber nie in Form gebracht. Cunningham fasste es in einem Satz zusammen: „Code in seiner ersten Fassung auszuliefern ist wie Schulden aufzunehmen.“ Der Bericht selbst spricht nur von Schulden und seiner Schuldenanalogie. Spätere Arbeiten verwenden den aus zwei Wörtern bestehenden Namen; eine Google-Veröffentlichung von 2015 schreibt die Metapher Ward Cunningham zu und datiert sie auf 1992.

Forschende, die sich im April 2016 in Dagstuhl trafen, um den Umfang des Begriffs abzugrenzen, formulierten eine Arbeitsdefinition. Darin sind Schulden eine Menge von Entwurfs- oder Implementierungskonstrukten, die kurzfristig Aufwand sparen, das System aber in einem Zustand hinterlassen, der spätere Änderungen verteuert oder teilweise unmöglich macht. Ihre Auswirkungen werden auf innere Systemeigenschaften begrenzt, vor allem Wartbarkeit und Weiterentwickelbarkeit. Das unterscheidet Schulden von einer nie entwickelten

Funktion: Ihre Kosten treffen diejenigen, die das System ändern, und erreichen Nutzer erst über sie. Derselbe Bericht bleibt beim Finanzvokabular. Der Kapitalbetrag entspricht der ursprünglichen Ersparnis oder den heutigen Kosten einer besseren Lösung. Zinsen gibt es in zwei Formen: wiederkehrende Zinsen als geringere Entwicklungsgeschwindigkeit, verursachte Defekte und Qualitätsverlust sowie auflaufende Zinsen als Zusatzkosten jeder neuen Software, die auf dem nicht ganz passenden Code aufbaut. Schulden stecken überwiegend im Code, aber auch in Dokumentation, Tests und bekannten Defekten.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was sind technische Schulden?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Die Struktur eines Systems wird zuerst auf dem Papier durchdacht, damit die Teile zusammenpassen und die Kosten feststehen, bevor die erste Zeile Code entsteht.

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. Ein Infrastrukturplan für 14 Abteilungen — 14 Forschungsabteilungen mit einem gemeinsamen Infrastrukturplan

Ein Infrastrukturplan für 14 Abteilungen Kontakt aufnehmen →

Warum es wichtig ist

Bewusst eingegangene Schulden können eine gute Entscheidung sein. Cunninghams Bericht sagt, dass ein kleiner Betrag das Team beschleunigt, sofern er bald durch Umschreiben des Codes getilgt wird. Auch der Google-Beitrag hält einen Kredit oft für eine strategisch sinnvolle Entscheidung. Beide warnen vor Schulden, die bestehen bleiben. Im Bericht werden die Zinsen mit Zeit bezahlt, mit jeder an nicht ganz passendem Code verlorenen Minute. Eine nie konsolidierte Implementierung kann so schwer werden, dass ganze Entwicklungsorganisationen zum Stillstand kommen. Schon vorher wird das Programm zu etwas, das kein Einzelner mehr beherrscht. Daraus entstehen Programmierer, die jeweils nur eine Ecke kennen, und schließlich ein Produkt, das seine Richtung nicht ändern kann.

Das Problem ist, dass Schulden oft nicht bewusst gewählt werden. Der Dagstuhl-Bericht beobachtet, dass sie häufig unbeabsichtigt entstehen und Entwickler überraschen. Deshalb werden sie meist erst bei der Wartung entdeckt, lange nach der verursachenden Entscheidung. Die Ursachen sind alltäglich:

Termindruck, eine fehlende Schlüsselperson oder zu wenig Wissen über eine technische Eigenschaft zum damaligen Zeitpunkt. Der Google-Beitrag ergänzt, dass verborgene Schulden gefährlich sind, weil sie sich unbemerkt aufschaukeln.

Die Metapher gibt Entwicklern und Führungskräften auch eine gemeinsame Sprache. Der Dagstuhl-Bericht nennt sie einen starken Kommunikationsmechanismus. Zur Vision seiner Teilnehmenden gehört, Schulden genauso sorgfältig zu verwalten wie Defekte und neue Funktionen und zwischen den Sorgen der Entwickler und den Entscheidungen der Führungskräfte übersetzen zu können. Eine auf dem Seminar vorgestellte Studie zeigt, was das im Team bedeuten kann. Graziela Tonin, Alfredo Goldman und Carolyn Seaman baten studentische Teams mit insgesamt 60 Personen, ihre Schulden ausdrücklich zu erfassen und zu verfolgen. Eine sichtbare Liste ermöglichte es den Teams, mit dem Kunden Zeit für Refactoring auszuhandeln, hielt fest, welche Projektteile noch unausgereift waren, und zeigte, ob sich die Codequalität verbesserte. Die meisten Teammitglieder berichteten von mehr Kommunikation, gründlicherem Abwägen, ob eine Schuld wirklich nötig war, und häufigerem Refactoring. Weil Entscheidungen über neue Schulden gemeinsam besprochen wurden, gab es in einigen Teams laut den Autoren schließlich weniger Experten, die niemand sonst vertreten konnte.

Wie es in der Praxis aussieht

Schulden werden handhabbar, wenn sie dort festgehalten werden, wo das Team seine Arbeit plant. Ein für diesen Artikel formulierter Registereintrag kann so kurz sein:

```
TD-14       Rechnungssummen werden an drei Stellen berechnet
Artefakte:  billing/web, billing/export, reports/monthly
Ursache:    fester Starttermin; die Regel zu kopieren war schneller,
            als sie in ein gemeinsames Modul zu verlagern
Kapital:    etwa 2 Tage, um die Regel in ein Modul zu verlagern, mit Tests
Zinsen:     jede Steuerregeländerung wird dreimal umgesetzt und getestet;
            bei einer der letzten fünf Änderungen blieb eine Kopie unverändert
Nächstes:   bei nächster Steuerregeländerung tilgen; der für dieses Quartal
            geplante E-Rechnungsexport darf keine vierte Kopie werden
Zuständig:  Abrechnungsteam
```

Die Felder folgen dem Modell des Dagstuhl-Berichts: ein mit konkreten Artefakten verknüpfter Posten mit Ursache, Kapitalbetrag und Zinsen. Gerade die erfassten Zinsen machen den Eintrag nützlich. Sie verwandeln das Gefühl, der

Abrechnungscode sei unordentlich, in abwägbare Kosten. Ändern sich die Steuerregeln viermal im Jahr und kostet jede Änderung einen zusätzlichen Tag Arbeit und Tests, sind die zwei Tage Kapitalbetrag innerhalb von sechs Monaten getilgt. Der geplante Export entspricht auflaufenden Zinsen im Dagstuhl-Sinn: Jedes neue Stück auf der duplizierten Regel erhöht die späteren Bereinigungskosten. Wäre der Code dagegen eingefroren und erwartete niemand mehr eine Änderung, würde derselbe Eintrag begründen, die Schulden bestehen zu lassen.

Schulden sind nicht nur eine Frage des Codes. Der Portfolioeintrag dieser Website über überarbeitete interne Prozesse und 8.000 eingesparte Stunden dokumentiert die frühere Arbeit der Gründerin bei Mappitall. Er fragt, ob technische Schulden den Leser bremsen, und beschreibt die Art, die sich in den Arbeitsabläufen ansammelt: Softwarearchitektur, die durch Anhäufen statt durch Entwurf gewachsen war, funktionierende, aber ineffiziente Systeme und eine Planung, die Menschen entweder warten ließ oder überlastete. Nichts davon brannte akut; deshalb war es liegen geblieben. Die Überarbeitung von Architektur, Systemen und Planung sparte während der Arbeit von 2015 bis 2022 ungefähr 8.000 Stunden. Die Seite führt einen Teil davon darauf zurück, die Änderungen dauerhaft in den Arbeitsabläufen des Teams zu verankern: Werden Verbesserungen nicht dauerhaft verankert, setzen sich nach und nach wieder die alten Arbeitsweisen durch.

Wo es schiefgeht

Alles heißt technische Schulden. Sobald der Ausdruck verbreitet ist, wird jede Unannehmlichkeit so bezeichnet: eine alte Bibliothek, eine fehlende Funktion oder der Stil eines Kollegen. Der Dagstuhl-Bericht nennt dieses Risiko ausdrücklich und fordert als Antwort klare Definitionen. Den Begriff für Abkürzungen reservieren, deren Kosten bei künftigen Änderungen anfallen. Defekte als Defekte und Wünsche als Funktionen erfassen; einen Registereintrag ohne benennbare Ursache und Zinsen ablehnen.

Die Schulden existieren nur in den Köpfen. Nicht festgehaltene Schulden bezahlt, wer den Code als Nächstes anfasst, meist ohne zu wissen, warum die Arbeit so lange dauert. Ein Register neben dem Backlog führen und vor Änderungen an einem Systemteil prüfen, so wie die studentischen Teams in Tonins Studie ihre Liste verwendeten. Ein Posten wird für diejenigen, die über Zeit entscheiden, erst sichtbar, wenn er dort steht, wo sie nachsehen.

Kapital wird getilgt, wo keine Zinsen anfallen. Ein Modul aufzuräumen, das niemand ändert, fühlt sich produktiv an und bringt nichts. Gleichzeitig verursacht der wöchentlich geänderte Code weiter Kosten. Zinsen laufen nur dort auf, wo Menschen arbeiten. Das Register deshalb danach ordnen, wie häufig sich jeder Teil ändert und was demnächst darauf aufgebaut werden soll. Die teuren Posten zuerst tilgen.

Die Tilgung wartet auf eine große Neuentwicklung. Jahrelang liegen gebliebene Schulden werden oft mit einem Projekt zum Ersatz des ganzen Systems beantwortet. Dieses nimmt eigene neue Schulden auf, während das alte System weiter Kosten verursacht. Cunningham empfahl die zeitnahe Tilgung. Der Google-Beitrag nennt die üblichen Mittel: Refactoring, stärkere Unit-Tests, Entfernen toten Codes, weniger Abhängigkeiten, enger gefasste APIs und bessere Dokumentation. Nichts davon ergänzt eine Funktion. Ziel ist, die nächste Änderung billiger zu machen. Solche Arbeiten als Teil des Alltags einplanen, nahe an der Änderung, die sie benötigt.

Experimente bleiben in Produktion. Eine Alternative als bedingten Zweig im Produktionscode auszuprobieren ist je Experiment billig. Der Google-Beitrag beschreibt, wie sich solche Zweige ansammeln, bis sämtliche Kombinationen schwer oder gar nicht mehr testbar sind. Er nennt Knight Capital, dessen System laut dem Beitrag in 45 Minuten 465 Millionen US-Dollar verlor, offenbar weil veraltete experimentelle Codepfade unerwartet reagierten. Jeden experimentellen Zweig und jedes Flag nach einem Zeitplan erneut prüfen, nicht mehr Genutztes löschen und das Entfernen von Code genauso belohnen wie das Hinzufügen.

Die Punktzahl eines Werkzeugs wird mit den Schulden gleichgesetzt. Statische Analyse kann duplizierte Zeilen und lange Funktionen zählen; ein Dashboard lässt die Zahl wie einen Kontostand wirken. Der Google-Beitrag sagt ausdrücklich, dass die Metapher keine präzise Kennzahl liefert, die sich über die Zeit verfolgen ließe. Dass ein Team weiter schnell vorankommt, belegt außerdem keine geringen Schulden. In Dagstuhl berichtete Forrest Shull, dass kein einzelner Regelsatz zur Schuldenerkennung über verschiedene Projekte hinweg durchgängig mit Symptomen wie steigenden Änderungskosten und sinkender Entwicklungsgeschwindigkeit korrelierte. Die Befunde eines Werkzeugs mit der eigenen Änderungs- und Defekthistorie vergleichen, kostenvorhersagende Regeln behalten und die übrigen verwerfen. Ein besserer Test sind die Fragen des Beitrags selbst, darunter, wie schnell ein neues Teammitglied eingearbeitet werden kann.

Weiterführende Lektüre

- The WyCash Portfolio Management System — Ward Cunninghams Erfahrungsbericht von der OOPSLA ‘92, in dem die Schuldenanalogie erstmals erscheint.
- Managing Technical Debt in Software Engineering — der Bericht des Dagstuhl-Seminars 16162 mit Arbeitsdefinition, Kapital und Zinsen sowie der Studie zu ausdrücklich geführten Schuldenlisten.
- Hidden Technical Debt in Machine Learning Systems — der Beitrag von Sculley und Kollegen bei Google von 2015 darüber, wie Schulden entstehen, getilgt werden und sich der Messung entziehen.
- Refactoring — Martin Fowlers Website zu kleinen, verhaltenserhaltenden Änderungen, die der Google-Beitrag unter den Tilgungsmöglichkeiten zuerst nennt.

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 sind technische Schulden?“ 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/technical-debt/