Technische Schulden erkennen, bewerten und abbauen

Technical Debt sichtbar machen und als Engineering-Risiko systematisch steuern

01 INTRO

Technische Schulden entstehen nicht nur durch schlechten Code. Sie können in Architektur, Tests, Abhängigkeiten, Daten, Infrastruktur und Entwicklungsprozessen liegen. Manche Schulden werden bewusst aufgenommen, andere entstehen schleichend und werden erst sichtbar, wenn Änderungen immer länger dauern oder Fehler immer schwerer beherrschbar werden.

Der Kurs vermittelt einen systematischen Umgang mit technischen Schulden. Die Teilnehmer lernen sie zu identifizieren, nachvollziehbar zu dokumentieren, ihre Auswirkungen zu bewerten und geeignete Maßnahmen zum kontrollierten Abbau zu entwickeln. Dabei wird bewusst zwischen messbaren technischen Befunden und wirtschaftlichen Priorisierungsentscheidungen unterschieden.

DAUER

3 Tage

FORMAT

Präsenz · Live Online · Inhouse

VORKENNTNISSE

Mehrjährige Erfahrung in Softwareentwicklung, Architektur oder technischer Qualitätssicherung ist hilfreich

02 TRAININGSPROGRAMM

Inhalte

Was sind technische Schulden?

- Schuldenmetapher
- Tilgung
- Zinsen
- bewusste Schulden
- unbeabsichtigte Schulden
- Abgrenzung zu schlechtem Code
- Abgrenzung zum Fehler
- Kontext

Wo technische Schulden entstehen

- Code
- Architektur
- Tests
- Abhängigkeiten
- Infrastruktur
- Daten
- Dokumentation
- Prozesse

Ursachen verstehen

- Zeitdruck
- fehlendes Wissen
- veränderte Anforderungen
- veraltete Technologie
- Übergangslösungen
- fehlende Ownership
- Architekturerosion
- bewusst eingegangene Trade-offs

Schulden im Code erkennen

- Code Smells
- Komplexität
- Duplikation
- Kopplung
- große Komponenten
- Änderungsverteilung
- statische Analyse
- Grenzen von Metriken

Architektur- und Systemschulden erkennen

- Abhängigkeitszyklen
- Boundary Erosion
- Shared Database
- veraltete Schnittstellen
- Deployment-Kopplung
- Skalierungsprobleme
- fehlende Observability
- Architekturdrift

Test- und Qualitätsschulden

- fehlende Tests
- fragile Tests
- lange Regression
- Flaky Tests
- manuelle Engpässe
- unklare Coverage
- fehlende Testbarkeit
- falsches Sicherheitsgefühl

Schulden dokumentieren

- Debt Register
- Fundstelle
- Ursache
- Auswirkung
- betroffene Komponenten
- Risiken
- mögliche Maßnahmen
- Nachvollziehbarkeit

Auswirkungen bewerten

- Änderungsaufwand
- Fehlerwahrscheinlichkeit
- Liefergeschwindigkeit
- Betrieb
- Security
- Wissensrisiko
- Geschäftsauswirkung
- Cost of Delay

Schulden priorisieren

- Risiko
- Häufigkeit von Änderungen
- strategische Bedeutung
- Zinswirkung
- Aufwand
- Abhängigkeiten
- Quick Wins
- bewusste Nichtbehebung

Technische Schulden abbauen

- Refactoring
- Upgrade
- Modularisierung
- Testautomatisierung
- Dependency Replacement
- Migration
- Replatforming
- Rewrite als letzte Option

Refactoring als kontrollierter Schuldenabbau

- konkretes Ziel
- Verhalten erhalten
- Sicherheitsnetz
- kleine Transformationen
- messbarer Ausgangszustand
- schrittweise Verbesserung
- keine versteckten Feature-Änderungen
- Abschlusskriterien

Schulden im Entwicklungsprozess steuern

- Backlog
- Definition of Done
- Code Reviews
- Architekturentscheidungen
- Boy Scout Rule
- regelmäßige Reviews
- Ownership
- kontinuierlicher Abbau

Werkzeuge und Metriken sinnvoll einsetzen

- statische Analyse
- Komplexität
- Duplikation
- Coverage
- Abhängigkeitsanalyse
- Hotspot-Analyse
- Repository Mining
- Metrik versus Realität

Praktische Analyse und Abbauplanung

- System oder Codebasis untersuchen
- Schulden identifizieren
- kategorisieren
- dokumentieren
- Auswirkungen bewerten
- Maßnahmen priorisieren
- Abbauplan entwickeln
- Erfolgskriterien festlegen

03 DURCHFÜHRUNG

Rahmen und Organisation

Dauer	3 Tage
Durchführung	Präsenz oder Live Online, vorzugsweise als Inhouse-Schulung
Schwerpunkt	Inhouse-Schulungen für Unternehmen und Teams
Sprache	Deutsch, englische Fachbegriffe und Dokumentation
Unterlagen	Kursunterlagen, Beispiele und Übungsaufgaben
Technik	Eigener Rechner für die praktischen Übungen

> **Inhouse-Schulungen**
> Eine eigene Codebasis kann als Grundlage für Analyse, Debt Register und Abbauplanung dienen.

04 ZIELGRUPPE

Für wen ist der Kurs gedacht?

Senior-Entwickler, Softwarearchitekten, technische Leads, Quality Engineers und Verantwortliche für gewachsene Softwaresysteme.

> **Voraussetzungen**
> Praktische Erfahrung mit Softwareentwicklung, Architektur oder technischer Qualitätssicherung.

05 LERNZIELE

Was Sie nach dem Kurs können

Nach dem Kurs können die Teilnehmer:

- unterschiedliche Arten technischer Schulden unterscheiden
- technische Schulden auf Code-, Architektur-, Test- und Infrastrukturebene systematisch erkennen
- Befunde nachvollziehbar in einem Debt Register dokumentieren

- technische Auswirkungen und wirtschaftliche Relevanz getrennt bewerten
- Schulden risikoorientiert priorisieren
- zwischen Refactoring, Modernisierung, Migration und Rewrite unterscheiden
- Refactoring als kontrollierte verhaltenserhaltende Maßnahme einsetzen
- einen realistischen Plan zum schrittweisen Schuldenabbau entwickeln
- Mechanismen etablieren, die neue technische Schulden sichtbar und steuerbar machen

06 FAQ

Häufige Fragen

Kann technische Schuld objektiv gemessen werden?

Nur teilweise. Metriken können Symptome wie Komplexität, Kopplung oder fehlende Tests sichtbar machen. Ob daraus relevante technische Schuld entsteht, hängt jedoch vom Nutzungskontext, der Änderungsfrequenz und den Auswirkungen auf das System ab.

Sind technische Schulden grundsätzlich schlecht?

Nein. Eine bewusst eingegangene technische Abkürzung kann wirtschaftlich sinnvoll sein. Problematisch wird sie insbesondere dann, wenn sie unsichtbar bleibt, ihre Folgen nicht bewertet werden oder eine ursprünglich temporäre Lösung dauerhaft bestehen bleibt.

Ist Refactoring gleichbedeutend mit dem Abbau technischer Schulden?

Nein. Refactoring ist eine wichtige Methode für bestimmte Arten von Code- und Designschulden. Architektur-, Infrastruktur-, Daten- oder Abhängigkeitsschulden können andere Maßnahmen erfordern.

Wann ist ein Rewrite sinnvoll?

Wenn die bestehende Lösung ihre Anforderungen mit vertretbarem Aufwand nicht mehr erfüllen kann und eine schrittweise Modernisierung ungünstiger ist. Ein Rewrite sollte Ergebnis einer Analyse sein und nicht die spontane Reaktion auf unangenehmen Legacy Code.

Wie verhindert man, dass technische Schulden nach dem Abbau erneut entstehen?

Vollständig verhindern lassen sie sich nicht. Entscheidend sind Sichtbarkeit, technische Qualitätskriterien, Reviews, automatisierte Prüfungen, klare Architekturentscheidungen und regelmäßige Neubewertung.

UC IT Service · Ulrich Cuber · kontakt@uc-it.de
https://www.uc-it.de/coaching/technische-schulden/