# TECHNISCHE SCHULDEN IN SOFTWAREENTWICKLUNGSPROJEKTEN: WIE DAS PROJEKTTEAM MIT DEN SCHULDEN UMGEHEN KANN

In Projekten kommt es immer wieder vor, dass technische Schulden aufgenommen werden. Auch bei technischen Schulden fallen in der Regel Zinsen an und das Projektteam muss darüber nachdenken, wie diese Schulden zurückgezahlt werden können. Es gibt jedoch auch Fälle, in denen gar keine Zinsen anfallen und die Schulden deshalb nicht getilgt werden müssen. Dieser Artikel beschreibt die Metapher der technischen Schulden, wodurch diese entstehen und wie man die Schulden kontrollieren kann.

der autor

Philipp Leinius
(E-Mail: philipp.leinius@opitz-consulting.com)
arbeitet als Consultant bei der OPITZ Consulting Gummersbach GmbH. Er verfügt über mehrjährige Erfahrung in der Durchführung von Softwareprojekten. Zu seinen Schwerpunkten zählt die Realisierung von Individualsoftware im Web-Umfeld und im Client-Server-Bereich.

## Was bedeutet die Metapher der technischen Schulden?

Für das Verständnis der Metapher der technischen Schulden ist die Definition des Begriffs „Schulden" wichtig: „Schulden ist der umgangssprachliche und auch zivil- und handelsrechtlich oft verwendete Begriff für Verbindlichkeiten, also Rückzahlungsverpflichtungen von natürlichen oder juristischen Personen gegenüber Dritten, die bereits eine Gegenleistung erbracht haben" (vgl. [Wik]). Ein typisches Beispiel für Schulden ist der Immobilienkredit. Der Schuldner erhält von einem Gläubiger (üblicherweise einer Bank) einen Geldbetrag, um ein Haus zu kaufen. Der Schuldner zahlt von nun an den geschuldeten Geldbetrag in regelmäßigen Abständen an den Gläubiger zurück (Tilgung). Neben dem Tilgungsbetrag fallen für den Restschuldbetrag Zinsen zu einem vereinbarten Zinssatz an.

### Technische Schulden

Ward Cunningham gilt als Erfinder der Metapher der technischen Schulden und prägte den Begriff (vgl. [Cun]). Bei der Softwareentwicklung nimmt ein Projektteam technische Schulden auf, indem es eine „einfache" Implementierung einer „sauberen" Lösung vorzieht. Der kurzfristige Vorteil dieser schnellen Lösung führt zu der Aufnahme von technischen Schulden (vgl. [Fow09-a]).

Durch die technischen Schulden fallen in der Regel Zinsen an. Dies macht sich in der Verlangsamung des Entwicklungsprozesses bemerkbar. Die Zinsen können entweder für jede Erweiterung der Software in Kauf genommen werden oder die technischen Schulden können durch das Redesign der Software zurückgezahlt werden.

### Ein Beispiel

Im Rahmen der Realisierung eines Softwareentwicklungsprojekts befindet sich ein Team kurz vor einem Auslieferungstermin. Um den Release-Termin noch mit vollem funktionalem Umfang einhalten zu können, entscheidet das Projektteam in Abstimmung mit dem Auftraggeber, an einer Stelle der Anwendung eine „einfache" Lösung zu wählen. Das Projektteam hat technische Schulden aufgenommen.

In der folgenden Iteration sollen weitere Anforderungen für den Bereich mit der suboptimalen Lösung umgesetzt werden. Die Realisierung der Anpassungen wird länger dauern und der Entwicklungsprozess wird langsamer sein. Vor der Umsetzung der neuen Anforderungen steht das Team aber vor der Wahl:

- Zum einen könnte der Programmcode mit dem suboptimalen Design überarbeitet werden. Die technischen Schulden wären damit getilgt.
- Zum anderen könnten die Anforderungen auf Grundlage der technischen Schulden umgesetzt und somit Zinsen gezahlt werden.

Letzteres Vorgehen würde die suboptimale Lösung nicht verbessern und es bestünde die Gefahr, dass sich das Design noch mehr verschlechtert. Der Zeitaufwand für die Integration neuer Anforderungen würde mehr und mehr steigen. Das Team müsste Zinseszinsen zahlen.

## Technische Schuldenaufnahme während der Realisierung

Technische Schulden entstehen unmittelbar dadurch, dass das Projektteam für eine bestimmte Problemlösung eine Designwahl trifft, die sich langfristig als suboptimal herausstellt. Diese Designwahl verursacht den suboptimalen Quellcode im Rahmen der Realisierung der Lösung. Während technische Schulden streng genommen nur vom Entwicklungsteam eingegangen werden können, kann die Ursache für technische Schulden in anderen Bereichen liegen:

- Im Rahmen der Aufwandschätzung wird schon während der Projektdefinition deutlich, dass das fixe Budget des Auftraggebers nicht ausreicht, um den geplanten Projektumfang umzusetzen. Da der Auftraggeber nicht bereit ist, auf einzelne Anforderungen zu verzichten, trifft er gemeinsam mit dem Projektteam die Entscheidung, Abstriche am Design der Software zu machen. Die Ursache für technische Schulden liegt damit vor Beginn des Projekts.
- Bei der Anforderungserhebung werden die Inhalte nicht vollständig definiert. Später stellt sich aufgrund der unvollständigen Anforderungen heraus, dass das Design schlecht gewählt worden ist. Während der Realisierungsphase geht das Projektteam technische Schulden ein. Die Ursache lag in der Anforderungserhebung.
- Technische Schulden können in einem Softwareprojekt auch durch Defizite in der Qualitätssicherung verursacht werden. Fälschlicherweise wird die Abnahme für eine Software erteilt. In der folgenden Iteration trifft das Projektteam auf Grundlage der unvollständigen Informationen Entscheidungen für den Ausbau des bisherigen Designs. Das Design der Anwendung ist falsch gewählt und technische Schulden sind aufgenommen worden.

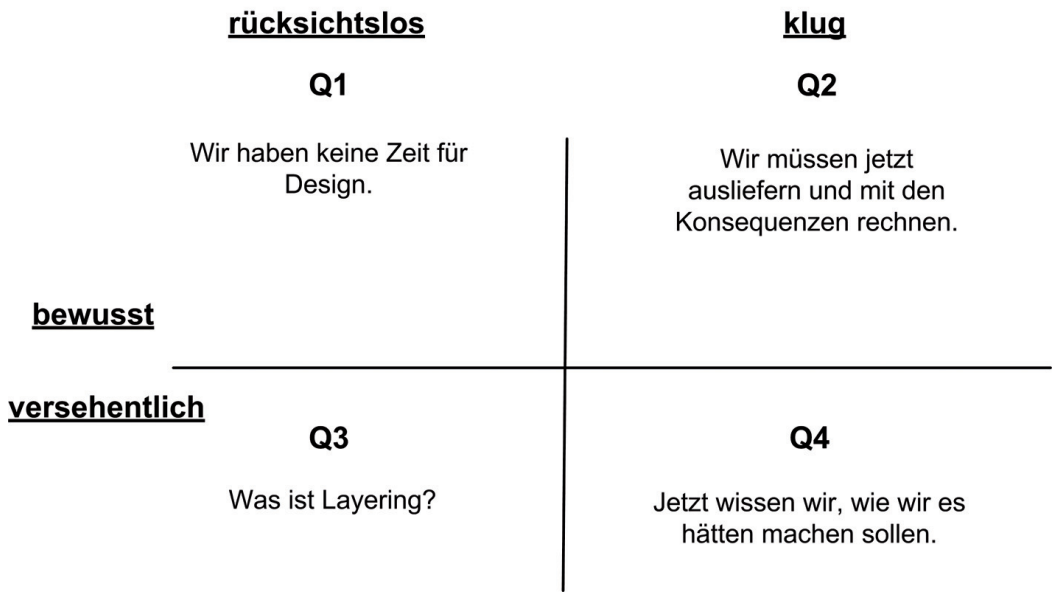


*Abb. 1: Gründe für die Aufnahme von technischen Schulden.*

## Anzeichen von technischen Schulden

Technische Schulden können sich durch verschiedene Aspekte bemerkbar machen. Die Projektbeteiligten sollten aufmerksam werden und die Aspekte kritisch hinterfragen, wenn folgende Sachverhalte auftreten:

- Eine Software ist nur mit großem Aufwand erweiterbar.
- Es gibt *Code Ownership*, d. h. eine Person implementiert einzig und allein einen Programmteil. Erweiterungen führt nur diese Person durch, andere trauen sich nicht an den Code.
- An den Programmstellen einer Software treten Probleme immer wieder da auf, wo das Projektteam eigentlich glaubte, dass alles richtig implementiert sei.
- Das Projektteam wirkt zunehmend ängstlicher, wenn es darum geht, das System zu verändern, und argumentiert mit: „Never change a running system".
- Das Projektteam spricht davon, dass die Software historisch gewachsen ist.
- Der Auftraggeber erhält als Antwort für die Realisierung seiner neuen Anforderung, dass dieser Sachverhalt technisch nicht umsetzbar sei.

Diese Aspekte sind einzelne Beispiele und können technische Schulden als Ursache haben. Um die Schulden zu identifizieren, sollte das Projektteam wissen, wie die technischen Schulden in einer Software in der Realität aussehen können.

## Was sind technische Schulden?

Technische Schulden in einer Software manifestieren sich in schlechter Qualität des Quellcodes. Martin Fowler nennt diese Aspekte *Code Smells* (vgl. [Fowl99]). Der Programmcode ist in diesem Fall schwer verständlich. Da die Implementierung länger dauert, verursachen Erweiterungen oder Anpassungen höhere Kosten und können zu Fehlern führen: Es treten technische Schulden auf.

Es gibt viele unterschiedliche Arten von *Code Smells*, beispielsweise *Duplicated Code*, d. h. im Quelltext eines Programms werden Duplikate verwendet. Die Programmlogiken wiederholen sich im Quellcode, obwohl die Programmstellen immer den gleichen Sachverhalt abbilden. Ein Projektteam muss die identischen Stellen bei einer Anpassung des Quellcodes doppelt pflegen. Dieser zusätzliche Aufwand erhöht die Realisierungszeit und erzeugt Fehler, weil oft nicht nur zwei, sondern viele Stellen den gleichen Sachverhalt abbilden, zum Beispiel den Aufbau einer Struktur. Bei Änderungen werden dann nicht alle Stellen konsistent geändert.

Im Quelltext einer Software kann sich eine Vielzahl weiterer Arten von *Code Smells* befinden. Je mehr Hinweise auf schlechte Qualität sich in einem System bemerkbar machen, desto höher lassen sich die angehäuften Schulden bemessen. Doch warum nimmt ein Projektteam überhaupt technische Schulden auf?

## Gründe für die Aufnahme von technischen Schulden

Martin Fowler teilt die Gründe für technische Schulden in vier Quadranten ein (vgl. [Fow09-b]). Die Gründe besitzen die Merkmale „rücksichtslos" oder „klug" in der Kombination mit „bewusst" oder „versehentlich" (**siehe Abbildung 1**).

Im Eingangsbeispiel hat das Team technische Schulden aufgenommen, damit ein Release-Termin eingehalten werden kann. Dies ist „klug" geschehen, da das Projektteam den richtigen Designweg kennt. Trotzdem wählt das Team „bewusst" eine provisorische Lösung, um den Termin noch einzuhalten (**Abbildung 1**, Quadrant Q2).

Doch es kommt in Projekten immer vor, dass das Team erst nach dem Projektabschluss oder nach einer gewissen Zeit weiß, wie die Wahl des richtigen Designs aussehen soll. Zwar handelt das Team in diesem Fall „klug", da ihm nun das richtige Design bekannt ist, „versehentlich" hat es aufgrund fehlender Erfahrung jedoch den falschen Weg gewählt (**Abbildung 1**, Quadrant Q4).

Wenn dem Team dagegen „bewusst" ist, dass für das jeweilige Problem beispielsweise Entwurfsmuster existieren, diese jedoch völlig ignoriert, ist der Grund für die Schuldenaufnahme dem Quadranten Q1 zuzuordnen. Das Projektteam kann dann als „rücksichtslos" kategorisiert werden. Es wird das richtige Design nicht akzeptieren und dieses „bewusst" nicht einsetzen. Bei der letzten möglichen Kombination hat das Team beispielsweise gar keine Vorstellungen

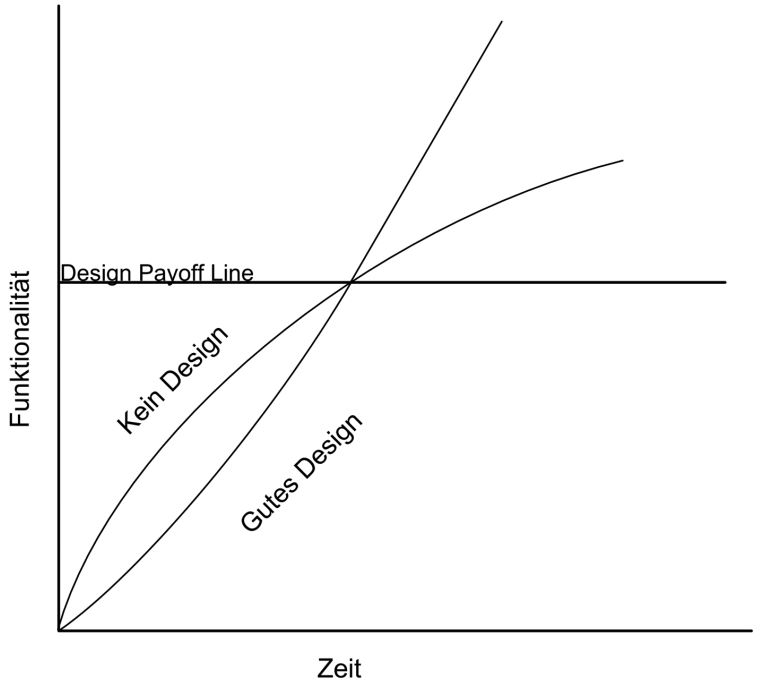


*Abb. 2: Gegenüberstellung der Kein-Design-Kurve und der Gutes-Design-Kurve.*

von Entwurfsmuster. Sie gehen „rücksichtslos" technische Schulden ein. Doch das geschieht nicht „bewusst", sondern aus Unwissenheit, daher „versehentlich". Das Team erstellt seine Anwendung „irgendwie" (**Abbildung 1**, Quadrant Q3).

Technische Schulden sollten niemals „rücksichtslos" eingegangen werden. Die Schulden können nicht kontrolliert und gegebenenfalls behoben werden. Wenn ein Projektteam technische Schulden aufnimmt, sollte es eigentlich das Ziel haben, die Schulden nur „klug" und „bewusst" (Quadrant Q2) einzugehen. Doch erfahrungsgemäß lässt es sich nicht immer vermeiden, dass technische Schulden „versehentlich" und „klug" aufgenommen werden (Quadrant Q4). Das Projektteam lernt immer im Laufe der Zeit dazu. Die Aufnahme von technischen Schulden sollte nur durch Gründe erfolgen, die den Quadranten Q2 und Q4 zugeordnet werden können.

## Ein Balanceakt

Martin Fowler stellt die folgende Hypothese auf (vgl. [Fow07]):

Es werden zwei imaginäre, stereotypische Projekte betrachtet (**siehe Abbildung 2**). Das Projektteam des ersten Projektes verwendet kein bewusstes Design und nimmt technische Schulden auf (*Kein-Design-Kurve*). Das Team des zweiten Projektes arbeitet generell mit gutem Design und vermeidet technische Schulden (*Gutes-Design-Kurve*).

Zu Beginn der Projektlaufzeit kann das Projektteam, das kein Design verwendet, mehr Funktionalität je Zeiteinheit entwickeln als das Projektteam mit gutem Design. Das ist möglich, weil letzteres Projekt zu Beginn in das Design investieren muss, während das Projekt mit keinem Design sofort mit der Realisierung beginnt. Im weiteren zeitlichen Verlauf steigt die Produktivität des Projekts mit gutem Design langsam, um sich schließlich bei weitestgehend konstanter Produktivität zu stabilisieren. Das Projekt ohne Design wird im Laufe der Zeit immer langsamer. Die Produktivität lässt aufgrund des schlechten Designs stetig nach. Ab einem bestimmten Zeitpunkt zahlt sich die Investition in gutes Design aus. Das Projekt mit gutem Design kann je Zeiteinheit mehr Funktionalität entwickeln als das Projekt, das auf ein Design verzichtet hat (*Design Payoff Line*).

Nach dieser Hypothese sollte die Empfehlung für ein Projektteam lauten, entweder kein Design oder gutes Design zu verwenden. Doch in einem realen Projekt ist das nicht möglich. Das liegt zum einen daran, dass ein Projektteam niemals kein Design verwenden kann. Wenn überhaupt, wird ein Team zu wenige Erfahrungen haben, wie es gutes Design gestaltet. Zum anderen kann das Projektteam niemals ein gutes Design verwenden, das technische Schulden grundsätzlich vermeidet. Das Team wird immer wieder technische Schulden aufnehmen, deren Gründe im Quadrant Q4 der **Abbildung 1** liegen.

Technische Schulden sollte man im Projekt auch nicht grundsätzlich unterlassen. Einerseits kann das strikte Vermeiden zu einem Mehraufwand führen, der gar nicht nötig ist (Stichwort: *Over-Engineering*). Andererseits können technische Schulden in verschiedenen Fällen sogar sinnvoll sein:

- Durch die frühere Auslieferung der Software kann das Projektteam wertvolle Erkenntnisse über die Funktionalität gewinnen, die es rechtfertigen können, technische Schulden aufzunehmen. Die Software kann vom Auftraggeber zum Teil sogar produktiv zum Einsatz kommen und somit früh echten Geschäftswert erzielen. Dies würde die Schuldenaufnahme nachträglich ausgleichen.
- Die technischen Schulden müssen in einem Projekt (z. B. bei einem einmaligen Migrationsprojekt) nicht immer zurückgezahlt werden. Es fallen keine Zinsen an, da das endgültige Ende des Projekts vor Erreichen der *Design-Payoff-Line* liegt.

Das Projektteam muss das optimale Design, das ein gewisses Maß an technischen Schulden enthält und somit zwischen der Kein-Design-Kurve und der Gutes-Design-Kurve liegt, individuell bestimmen.

## Maßnahmen zur Kontrolle von technischen Schulden

Die Metapher, die Ursachen und die Folgen von technischen Schulden sollten allen Projektbeteiligten klar sein. Nur wenn offen mit dem Thema umgegangen wird, können die Schulden kontrolliert werden. Dazu sollten die technischen Schulden dokumentiert, identifiziert und analysiert werden.

### Quellcode-Kommentar

Um technische Schulden zu identifizieren, müssen diese dokumentiert werden. Entsprechend sollten die Schulden im Quellcode einer Software mit Kommentaren versehen werden. Hierbei hat sich – sich je nach Ursache der technischen Schulden – folgendes Vorgehen bewährt:

- Die Schuldenaufnahme, deren Ursache dem Quadranten Q2 zugeordnet wird, sollte vom Projektteam markiert werden. Das Team weiß somit, dass dieser Bereich ein suboptimales Design enthält.
- Bei der Verursachung von Schulden aus Gründen, die dem Quadranten Q4 zugeteilt werden, ist bei der Implementierung noch nicht klar, dass Schulden aufgenommen wurden. Deswegen sollte der bereits erstellte Quellcode einer Software hinterfragt werden. Allein ein Unsicherheitsgefühl – auch schon während der Realisierung – stellt ein Zeichen dafür dar, dass diese Stelle markiert werden muss.

Wie die Dokumentation durchgeführt werden soll, muss vorher im Team einheitlich definiert werden, damit die Stellen wiederauffindbar sind. Das kann eine Markierung mit einem *To-Do*, der zusätzlichen Angabe der Ursache von technischen Schulden und einer Beschreibung sein, wie beispielsweise *<TODO> <Technische Schuldenursache> <Beschreibung>*.

**Code-Review**

Auch regelmäßige Code-Reviews können dazu beitragen, dass technische Schulden identifiziert werden. Eine Person analysiert alleine oder gemeinsam mit einem anderen Entwickler den geschriebenen Quellcode. Diese Person kann entweder aus dem Projektteam stammen oder auch projektfremd sein. Code-Reviews sind hilfreich, um eine zweite Meinung zu erhalten und damit beispielsweise ein eingefahrenes Vorgehen zu hinterfragen.

**Pair Programming**

Beim *Pair Programming*, einer sehr konsequenten Form des Code-Reviews, realisieren zwei Entwickler eine Anforderung, indem sie den Quellcode gemeinsam an einem Rechner schreiben. Die eine Person entwickelt den Programmcode, während die andere über die Implementierung nachdenkt und Probleme und Anregungen unverzüglich weitergibt. Abwechselnd nehmen beide Entwickler die jeweils andere Rolle ein. Durch den gemeinsamen Erfahrungsaustausch können die Schulden identifiziert und es kann entschieden werden, ob die Aufnahme sinnvoll ist.

**Analytische Systeme**

Analytische Systeme sollten im Projekt verwendet werden, um die technischen Schulden transparent zu machen. Beispielsweise misst die Software „Sonar" (vgl. [Son]) Metriken zur Softwarequalität einer Anwendung. Anhand verschiedener Verfahren (z. B. über das Auffinden von *Duplicated Code*) werden technische Schulden gemessen. Zusätzlich wird ein Geschäftswert für das Zurückzahlen der Schulden festgelegt. Das Projektteam kann die Kosten für die Aufnahme von technischen Schulden kontrollieren.

## Tilgen von technischen Schulden

Damit die aufgenommenen Schulden nicht überhand nehmen, müssen sie getilgt werden. Das geschieht durch das Überarbeiten des Quellcodes, das so genannte *Refactoring* (vgl. [Mar08]). Bei der Refaktorisierung wird der Programmcode überarbeitet und die dokumentierten Stellen mit der schlechten Qualität werden durch ein optimales Design ersetzt. Dies kann auch durch die Analyse des Quellcodes nach *Code Smells* erfolgen. Je nach Art der identifizierten *Code Smells* sind unterschiedliche Refaktorisierungsmaßnahmen durchzuführen. Der *Duplicated Code* wird durch das Extrahieren von redundanten Programmstellen zu einer neuen Methode behoben.

Die Voraussetzung für die Refaktorisierung sollte die Testabdeckung des Quellcodes einer Anwendung sein. So ist gewährleistet, dass das System weiterhin fehlerfrei funktioniert, wenn das Projektteam eine Programmstelle überarbeitet. Ein *Continuous Integration System*, wie z. B. „Jenkins" (vgl. [Jen]), führt die definierten Tests in regelmäßigen Abständen aus und prüft die Stabilität der Anwendung.

## Fazit

Die Metapher der technischen Schulden steht in Softwareentwicklungsprojekten für die Entscheidung, auf eine optimale Lösung zu verzichten, um einen kurzfristigen Vorteil zu erreichen. Die Schulden werden vom Projektteam während der Realisierung aufgenommen und manifestieren sich in der schlechten Qualität des Quellcodes eines Systems. Ein Projektteam sollte technische Schulden nur aus Gründen aufnehmen, die dem Quadranten Q2 und Q4 in Abbildung 1 zugeordnet werden können. Das optimale Design für ein Projekt enthält technische Schulden und liegt zwischen der Kein-Design-Kurve und der Gutes-Design-Kurve. Die aufgenommenen technischen Schulden sollten dokumentiert werden, damit das Projektteam diese kontrollieren und darüber entscheiden kann, ob die Schulden behoben werden sollen. Das Tilgen der Schulden wird durch Refaktorisierungsmaßnahmen durchgeführt. ■

## Literatur & Links


**[Cun]** W. Cunningham, Debt Metaphor, siehe: http://www.youtube.com/watch?v=pqeJFYwnkjE

**[Fowl99]** M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts, Refactoring, Addison-Wesley 1999

**[Fow07]** M. Fowler, Design Stamina Hypothesis, siehe: http://martinfowler.com/bliki/DesignStaminaHypothesis.html

**[Fow09-a]** M. Fowler, TechnicalDebt, siehe: http://martinfowler.com/bliki/TechnicalDebt.html

**[Fow09-b]** M. Fowler, TechnicalDebtQuadrant, siehe: http://martinfowler.com/bliki/TechnicalDebtQuadrant.html

**[Jen]** Jenkins, siehe: http://jenkins-ci.org/

**[Mar08]** R. C. Martin, Clean Code – A Handbook of Agile Software Craftsmanship, Prentice Hall Int. 2008

**[Son]** Sonar, siehe: http://www.sonarsource.org/

**[Wik]** Wikipedia, Schulden, siehe: http://de.wikipedia.org/wiki/Schulden