TECHNISCHE SCHULDEN IN DER SOFTWAREENTWICKLUNG

Eine Frage der Führung

Technische Schulden haben in Entstehung und Abbau vor allem mit einem Mangel an Führung zu tun. Womit noch nicht gesagt ist, wer verantwortlich ist und handeln sollte.

Technische Schulden sind allgegenwärtig. Das liegt auch daran, dass der Begriff mit mehreren Bedeutungen überladen ist. Dazu möchte ich hier nicht in die Tiefe gehen, weil das schon eine Reihe anderer dotnetpro-Artikel in den vergangenen Jahren erledigt hat. Doch für den weiteren Verlauf ist eine kleine Klassifikation hilfreich.

Der Begriff „technische Schulden" geht auf Ward Cunningham zurück, der ihn Anfang der 1990er-Jahre geprägt hat. Bemerkenswert daran ist, dass Cunningham damals an einer Finanzsoftware arbeitete. Das Wort „Schulden" wählte er, weil er der Meinung war, dass sein Auftraggeber damit etwas anfangen könne. Hätte er eine Software für den Gesundheitsbereich geschrieben, würden wir heute vielleicht über „technische Mangelernährung" nachdenken oder, wäre sein damaliger Auftraggeber ein Autokonzern gewesen, vielleicht über „technischen Rost". Doch es kam anders. Heute sprechen wir fortwährend von Schulden, obwohl die meisten von uns nicht mit einem FinTech-Unternehmen arbeiten und der Begriff oft gar keine gute Metapher darstellt.

Nehmen wir einmal an, dass wir agil an einem neuen Produkt arbeiten. Es soll ein einfach zu transportierendes Surfbrett sein. Mit unserem Team entwickeln wir einen besonderen Schaumstoff. Er ist stabil genug, um darin ein Scharnier verankern zu können, sodass wir unser späteres Surfbrett zusammenklappen können. Der Schaumstoff ist brillant. Noch nie in der Geschichte der Menschheit hat jemand einen derart brillanten Schaumstoff entwickelt. Und umweltfreundlich ist er obendrein, denn er ist kompostierbar und wird aus Atommüll produziert; und die radioaktive Strahlung geht bei der Herstellung auf magische Art und Weise verloren. Besser geht es nicht.

Im ersten Review stellen wir dann fest, dass die ursprüngliche Anforderung „Es soll platzsparend sein" ungenau war: Wir haben ein Inkrement entwickelt, das durch den Klappmechanismus mit einer geringen Grundfläche auskommt; die Stakeholder wünschen sich aber ein geringes Volumen. Also müssen wir den Schaumstoff aufblasbar machen. Das ist eine große Herausforderung, weil wir ihn dafür nicht entworfen haben. Dieses Delta meint Cunningham, wenn er von „technischen Schulden" spricht. Er meint ausdrücklich nicht, dass schlechter Code geschrieben wird.

Im obigen Beispiel: Der Schaumstoff ist brillant geworden. Makellos. Sollten wir in dieser Situation wirklich von technischen Schulden im heutigen Sinn sprechen? Ich finde, dass der Begriff hier nicht passt. Mir gefällt in diesem Fall der Ausdruck „agile Schulden" etwas besser. Aber am liebsten bezeichne ich alle Probleme, über die wir hier sprechen, als „Kursabweichung".

Das Team aus meinem Beispiel hat einen Kurs auf ein herausragendes Produkt eingeschlagen. Es hat handwerklich alles richtig gemacht. Doch während der Entwicklung zeigt sich, dass man durch eine ungenaue Anforderung oder eine fehlerhafte Kommunikation vom Kurs abgekommen ist, und zwar ohne es zu merken. Jetzt ist eine Kurskorrektur notwendig. Das klingt für mich viel netter als Schulden, die man zurückzahlen muss.

Der einzige Fall, in dem das Wort „Schulden" aus meiner Sicht angemessen ist: Man möchte sich Zeit leihen und nimmt dafür übergangsweise eine geringere Qualität in Kauf. Will man die geringe Qualität wieder loswerden, muss man die geliehene Zeit zurückzahlen. Mit Zinsen, versteht sich.

Es gibt noch ein drittes Phänomen, das gerne unter technischen Schulden subsumiert wird: miserabel geschriebener Code. Doch auch für diesen gibt es gute Gründe. Zum Beispiel programmieren Anfänger in der Regel schlechter als erfahrene Entwickler, die das schon seit Jahrzehnten machen. Dieser „schlechte" Code kann aber auch durchaus etwas Gutes haben: Er konnte mit dem verfügbaren Personal geschrieben werden. Wäre das nichts Gutes, warum sollte man es dann geschehen lassen?

Ein Aspekt dieser Angelegenheit wird gerne übersehen: Die Frage, was „schlechter Code" ist, ist nicht pauschal zu beantworten. Denn irgendjemand hat ihn so geschrieben, wie er nun mal ist. Wenn der Urheber weiß, dass er nicht gut programmiert hat, wird er hoffentlich um Hilfe bitten; andernfalls hat das Team ein ganz anderes Problem.

Also kann man davon ausgehen, dass der Urheber mit seinem Programm einigermaßen zufrieden ist. Und das hat wichtige Vorteile: Der Urheber versteht das Programm und kann es weiterentwickeln. Das ist keine Selbstverständlichkeit. Ich habe bereits Teams erlebt, in denen ein großes Know-how-Gefälle festzustellen war.

Ein Team hatte einen externen „Experten" ins Team geholt. Hätte man andere Experten gefragt, hätten sie zweifellos gesagt, dass Quellcode und Architektur, die der Externe geliefert hatte, nach „objektiven" Maßstäben außerordentlich gut waren. Das Problem war nur: Niemand aus dem Team sah sich in der Lage, an dem Programmcode Änderungen vorzunehmen. In diesem Sinne war der Code schlecht wartbar. Die objektiven Maßstäbe gibt es schlicht nicht. Quellcode ist Kommunikation. Wenn ich ihn nicht verstehe, kann ich damit nichts anfangen.

Wer ist schuld an Schulden …

Wenden wir uns der Frage zu, wer für technische Schulden verantwortlich ist. Genauer: Wer ist verantwortlich, dass sie überhaupt entstehen?

Den Anfang machen die „agilen Schulden", also jene Aufwände, die dadurch notwendig werden, dass sich das Verständnis des Entwicklungsziels im Lauf der Zeit verändert (Bild 1). Auslöser sind die Wünsche, Ideen, Anforderungen und User Stories, die das Entwicklerteam bekommt. Sie kommen vom Produktmanagement, Product Owner oder irgendeiner anderen, ranghöheren Person.

Ganz offensichtlich sind die Softwareentwickler nicht dafür verantwortlich, dass diese Schulden angehäuft werden.

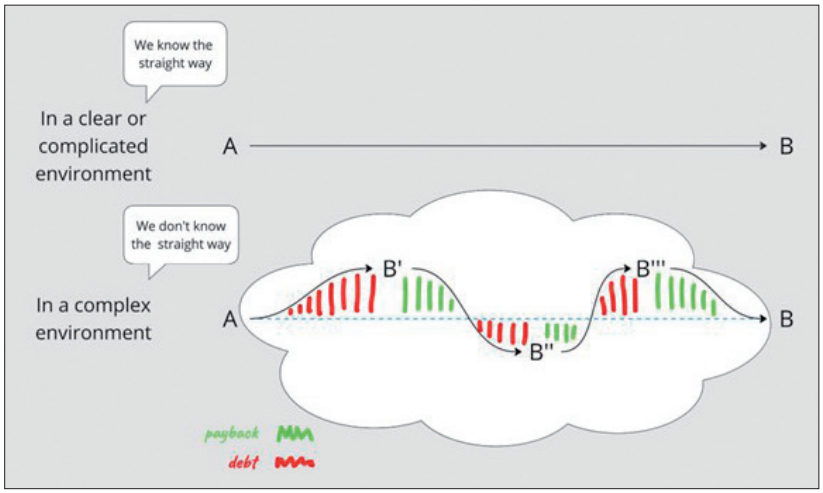


Bild: Max Diecks, kutura.digital

Agiles Arbeiten kommt in einer komplexen Umgebung zum Einsatz, wenn wir den geradlinigen Weg vom Start A zum Ziel B nicht kennen. Ein erstes Zwischenziel B' ist dann mehr oder weniger weit von der Ideallinie entfernt. Diese Abweichung brachte Ward Cunningham dazu, den Begriff „technische Schulden" zu prägen. Zahlt man sie nicht sofort zurück, entfernt man sich mit dem nächsten Ziel B'' noch weiter von der Ideallinie, was auf lange Sicht zum Bankrott führt. Nur wenn der Code „clean" ist, kann man die Schulden zügig begleichen. Zahlt man sie zurück, können die nächsten Ziele B'', B''' und so weiter näher an der Ideallinie liegen (Bild 1)

Sie entwickeln, was bestellt wird. Und das in so hoher Qualität, wie es das eigene Können und die von außen auferlegten Vorgaben zulassen. Wenn man es so betrachtet, liegt die Verantwortung außerhalb des Dev-Teams.

So ähnlich ist es auch bei den technischen Schulden, die man freiwillig und bewusst eingeht, um zum Beispiel einen vom Management vorgegebenen Termin einzuhalten. Die Argumentation ist hier die gleiche, auch wenn der Quellcode oder die Architektur dabei vielleicht Qualitätseinbußen erleiden. Wer schnell etwas liefern soll, führt womöglich das sinnvolle Refactoring einer Klasse nicht durch und ergänzt einfach ein paar Methoden, die die aktuelle Aufgabe irgendwie erledigen.

Als Letztes bleiben noch Code-Mängel, die zum Beispiel entstehen, weil es sich um ein unerfahrenes Team mit Junior-Entwicklern handelt. Jeder gibt sein Bestes, und das ist eben noch lange nicht „perfekt". Auch hier kann man die Verantwortung bei den Vorgesetzten suchen. Sie haben das Team zusammengestellt und verwalten die Fortbildungsbudgets. Was soll das Team oder gar ein einzelner Entwickler da machen?

… und wer begleicht sie?

In einer späteren Phase stellt sich die Frage, wer für den Abbau der entstandenen Schulden und Mängel verantwortlich ist. Zu diesem Zeitpunkt kann man in analoger Weise die Verantwortung ganz klar bei den ranghöheren Personen ansiedeln. Der PO priorisiert im Backlog die Feature-Tickets höher als die Schulden-Tickets? Der Vorgesetzte gibt neue Termine vor, die höchste Priorität haben? Und so weiter, und so weiter.

Kurz gesagt: Es liegt an der Führung. Nichts Neues eigentlich. Das steht ja auch schon so im Titel des Artikels. Und nun? Wie geht es weiter? Soll das heißen, Augen zu und mit Höchstgeschwindigkeit in den technischen Bankrott? Hoffentlich nicht. Irgendetwas muss sich ändern. Und das bedeutet immer: Irgendjemand muss seine Handlungsweise verändern. Da stellt sich die Frage, wer das sein kann. Wer kann das Team aus dem Schlamassel führen? Die naive Antwort lautet: die schon genannte ranghöhere Person. Und genau dort befindet sich der Knackpunkt. Das Wort „ranghöher" ist das Problem.

Vielleicht ist Ihnen aufgefallen, dass ich „Führung" und „Rang" gleichbedeutend verwendet habe. Das entspricht dem weitverbreiteten Bild, dass Vorgesetzte auch führen (sollten). Viel weiter komme ich jedoch, wenn ich Führung so fasse, wie es Svenja Hofert und Claudia Thonet formulieren [1]:

„Führung ist völlig unabhängig von einer Funktion und Position. Führung ist das Bestimmen der Richtung von Bewegung und erfolgreiches Intervenieren in kritischen Situationen."

Nach meinem Geschmack sollte der Rang, also die Position im Organigramm, beim Abbau von technischen Schulden keine Rolle spielen. Ohne in die Tiefe zu gehen, kann man hier einfach mal von flachen Hierarchien oder lateraler Führung sprechen. Letzteres ist interessant, weil das Wort „Führung" darin vorkommt. Ein Wort, das wir in keinem (deutschen) Positionstitel finden. Wir haben Abteilungsleiter, Teamleiter, Bereichsleiter, aber keine Abteilungsanführer, Teamanführer, Bereichsanführer. Die Silbe „an" habe ich eingebaut, um Konflikte mit der deutschen Geschichte zu vermeiden, denn darum geht es hier ganz sicher nicht.

Abgesehen davon hat das Wort „Anführer" einen ziemlich guten Klang. Bei einem Anführer denken wir an klassische Banküberfallfilme (Heist-Movies [2]) wie etwa „Oceans Eleven". Von der moralischen und juristischen Komponente der illegalen Handlung einmal abgesehen haben sie einen Riesenvorteil: den charismatischen Anführer. Ihm folgt nicht nur der Rest der Bande, sondern auch der Zuschauer bereitwillig. ▶

Erst der Akt des Folgens macht den Anführer zu einem solchen. Wenn keiner mitmacht, ist er nur irgendein Kerl. Die Vorgesetzten, die mir beruflich begegnen, sind mehrheitlich nicht solche Typen. Sie haben nicht die Ausstrahlung von George Clooney. Man folgt ihnen nicht, sondern tut, was sie sagen, weil sie den Paycheck unterschreiben. Das zeigt sich in Studien und Umfragen immer wieder. Wer an dieser Stelle sagt: „Mein Chef ist anders, ich würde für ihn durchs Feuer gehen", kann sich glücklich schätzen. Die Regel ist es nicht.

Die meisten Entwickler, mit denen wir arbeiten, glauben, dass Personen außerhalb des Dev-Teams eine sehr schlechte Vorstellung von technischen Schulden in der eigenen Software haben (Bild 2)

Nur am Rande sei festgestellt, dass sich an unserer emotionalen Bindung rein gar nichts ändert, wenn wir aus dem „Teamleiter" einen „Team Lead" machen. Da steckt dann zwar das richtige Wort drin, aber die meisten Team Leads haben mit einem Anführer so viel gemeinsam wie ein Lichtjahr mit einer Zeiteinheit.

So weit, so gut. Nun habe ich den Mangel an Führung in dieser Angelegenheit beklagt, doch sachliche Gründe dafür, dass die Vorgesetzten besser nicht für technische Schulden verantwortlich gemacht werden sollten, habe ich noch nicht angeführt. Dabei ist die Sache ganz einfach: Sie wissen schlicht und ergreifend nichts von den technischen Schulden. Oder sie verstehen es nicht. Oder beides.

Über viele Jahre habe ich Entwicklern, mit denen ich gearbeitet habe oder die an einem unserer Workshops zu technischen Schulden teilgenommen haben, immer wieder die gleichen Fragen gestellt.

Die erste lautete: „Wie gut können die Personen außerhalb des Dev-Teams den Umfang/das Ausmaß der technischen Schulden einschätzen?" Die Antworten waren stets eindeutig (Bild 2): Nach Meinung der Entwickler wissen andere Personen ziemlich wenig über die technischen Schulden in der eigenen Software.

Die Selbsteinschätzung (Bild 3) sieht ganz anders aus. Auf die Frage, wer über den Umgang mit technischen Schulden entscheide, die Entwickler oder die anderen, sind die Antworten ebenfalls deutlich auf einer Seite: die anderen (Bild 4).

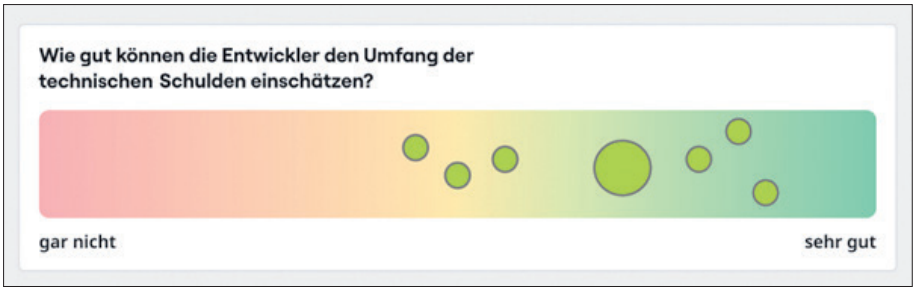
Die spannende Frage: Warum lassen die Entwickler das zu? Vielleicht ist das so eine Mentalität der Art „Der Kunde ist König" oder „Wer die Kapelle bezahlt, bestimmt, was gespielt wird". Klingt vernünftig und kulturell gelernt, ist aber mindestens in gewissen Situationen nicht zielführend. Man stelle sich den Flugpassagier vor, der dem Piloten in wilden Turbulenzen Anweisungen gibt, wie er die Maschine zu fliegen hat. Kommt natürlich nicht vor, schließlich heißt der Chef an Bord nicht umsonst Flug„kapitän". Hier kommen Kompetenz und Rang zusammen. Diesen Satz kann man nicht oft genug schreiben und lesen: *Hier kommen Kompetenz und Rang zusammen.*

Wenn es um technische Schulden in der Softwareentwicklung geht, ist das aber nicht der Fall. Entscheidet der Rang und nicht die Kompetenz über den Auf- und Abbau von technischen Schulden, setzt die Abwärtsspirale ein: *Bei technischen Schulden sind Kompetenz und Rang getrennt.*

Führungswechsel

Um das zu verhindern, gibt es einen Ausweg: Die Entwickler (= Kompetenz) müssen beim Thema technische Schulden die Führung übernehmen. Zumal dieses der Mehrzahl der Entwickler auch keineswegs egal ist (Bild 5). Die konkrete Ausgestaltung dieser Idee hängt zweifellos vom Unternehmenskontext und der vorherrschenden Kultur ab. Im weiteren Verlauf des Artikels gehe ich zuerst von einer Unternehmenskultur aus, in der man miteinander sprechen kann.

Eine grundlegende Unterscheidung besteht darin, ob die Entwickler offen oder verdeckt die Führung übernehmen. Da die verdeckte Vorgehensweise einige Risiken mit sich bringt, beginne ich mit der offenen Führung.



Bei der Selbsteinschätzung wird das Bild erwartungsgemäß positiver. Entwickler denken, dass sie technische Schulden in der eigenen Software relativ gut beurteilen können. Bemerkenswert ist, dass bei dieser Frage die Spreizung deutlich größer ist. Die Selbsteinschätzung gerät aber nicht in den roten Bereich (Bild 3)

Mit offener Führung meine ich, dass alle Beteiligten sich darüber einig sind, dass technische Schulden und weitere Code-Mängel in der Verantwortung der Entwickler liegen. Für diese Aussage bekommt man sehr leicht vom Management grünes Licht. Wer will sich schon mit Technik abgeben? Aber dazu gehört untrennbar auch, dass die Entwickler autonom entscheiden können, wann und wie sie technische Schulden abbauen. Ein wesentliches Merkmal dieser Aussage ist, dass alle Beteiligten den Entwicklern zuge-

stehen, zu neuen Feature-Wünschen oder Terminvorgaben Nein sagen zu können.

In der Praxis beobachte ich eher den Fall, dass beispielsweise Terminvorgaben des Managements als nicht verhandelbar wahrgenommen werden. In der Folge verzichten die Entwickler auf dringend notwendige Modernisierungen der Software, und wenn das über lange Zeit geschieht, rutscht man in den technischen Bankrott. Dann ist das Wehklagen groß und man macht sich auf die Suche nach den Schuldigen. Aus der Luft gegriffene Terminvorgaben werden dabei in der Regel nicht als Ursache erkannt.

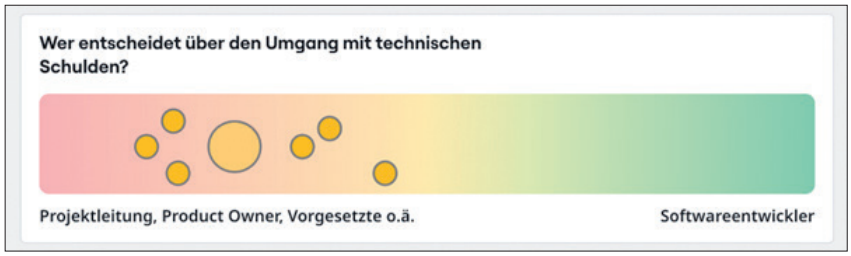
Wie kann man dazu kommen, dass die Entwickler die Führung übernehmen? Der erste und wichtigste Punkt besteht darin, dass man zuerst klärt, wie der Handlungsspielraum im Moment überhaupt aussieht. Ich habe mit einer ganzen Reihe von Teams gearbeitet, die mir mit voller Überzeugung berichteten, was sie alles nicht entscheiden dürfen. Als ich die jeweiligen Vorgesetzten oder die Geschäftsführung darauf ansprach, gab es erstaunte Gesichter: In den meisten Fällen sahen die Vorgesetzten die Entscheidungsbefugnis sehr wohl beim Team.

Kurz gesagt: Die Verantwortlichkeiten und der Handlungsspielraum, den beide Seiten sehen, sind oft unterschiedlich. Deshalb lautet mein erster Tipp: Klärt als Team mit allen relevanten Personen, wer welche Verantwortung im Bereich technischer Schulden hat. Wer diese Personen sind, hängt vom jeweiligen Kontext ab. Manchmal ist es der Product Owner, manchmal ein Vorgesetzter, und manchmal wird ein Product Owner als Vorgesetzter wahrgenommen und bekommt kein Veto.

Es empfiehlt sich auch, etwaige Terminvorgaben vorsichtig zu hinterfragen. Mancher Manager meint, man müsse stramme Termine vorgeben, damit die Mitarbeiter richtig spuren. Wenn dieser Glaubenssatz vorherrscht, die Zeitvorgaben ohne Widerspruch bleiben und das Projekt später im technischen Bankrott endet, haben alle verloren. Also sollte man zumindest eine Klärung versuchen.

Für das Gespräch kann es hilfreich sein, den Begriff der technischen Schulden zu überdenken und eventuell durch einen anderen zu ersetzen. Dafür gibt es mehrere Gründe: Erstens erfasst der Begriff nicht alle Aspekte von Problemen adäquat. Zweitens sind beide Worte irreführend, Schulden und Technik. Schulden macht man in der Regel bewusst, aber viele Code-Probleme entstehen im Lauf der Zeit „von selbst". Und die Probleme nur auf „Technik" zu reduzieren senkt das Interesse von Vorgesetzten oder Kunden rapide. Außerdem untergräbt es die Glaubwürdigkeit der Entwickler, wenn sie dauernd berichten, dass mit der „Technik" etwas nicht stimme. Es gibt durchaus Vorschläge für ein anderes Vokabular, etwa Softwareverschleiß oder Softwareerosion [3]. Ich verwende gerne das Segeln als Metapher, wie oben bereits angedeutet. In der klassischen Navigation (das heißt ohne GPS-gesteuerte Autopiloten und Ähnliches) sind Kurskorrekturen [4] etwas völlig Normales.

Wikipedia schreibt dazu: „Der direkte Weg von einem Ausgangspunkt (origin) zum Ziel (destination) führt über den Kurs. Das ist die Verbindungslinie beider Punkte [...]. Bei einer Störung, beispielsweise eine Strömung oder Querwind, wird das Ziel verfehlt. Der zurückgelegte Weg, der Weg über Grund oder Track, weicht vom Kurs ab. Der Schiffs- oder Flugzeugführer steuert gegen."

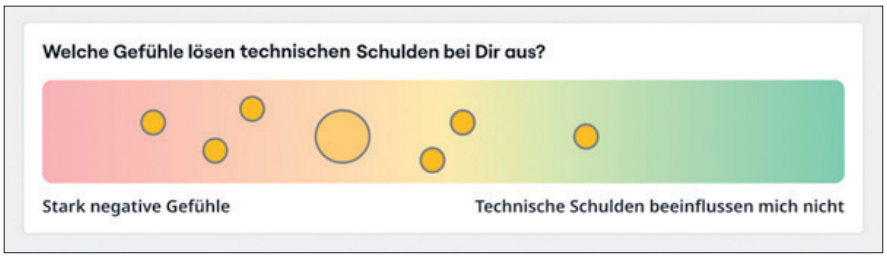


Diejenigen Personen, die die technischen Schulden am wenigsten einschätzen können, sind nach Meinung der Entwickler diejenigen, die mehrheitlich die Entscheidungen darüber treffen (Bild 4)

Diese Störungen haben wir in der Softwareentwicklung fortwährend, zum Beispiel:

- Neue Anforderungen lassen die getroffenen Architekturentscheidungen in einem neuen Licht erscheinen.
- Plötzlich bekannt gewordene Sicherheitslücken in einer verwendeten Bibliothek verlangen nach sofortigem Eingreifen.
- Der neue Mitarbeiter kennt unsere Coding Practices noch nicht, aber genau jetzt haben wir keine Zeit, seinen (funktionierenden) Code zu überarbeiten.
- Der nächste Termin drängt und „Programm läuft" ist wichtiger als „Code ist clean".

Wichtig bei diesen Beispielen ist: Niemand handelt hier falsch. Es gibt keinen Schuldigen. Es gibt nur Ereignisse, die nach einer Kurskorrektur verlangen. Und wie beim Segeln ▶



Technische Schulden lassen die Mehrheit der Entwickler nicht kalt. Vorgesetzte sollten auch diesen Aspekt hinterfragen, um rechtzeitig reagieren zu können, bevor die Unzufriedenheit zu groß wird (Bild 5)

stellt sich die Frage, wann ich den Kurs korrigiere. Je länger ich warte, desto größer wird die Abweichung zum Ziel.

Aus meiner Sicht ist es unausweichlich, dass die Stimme der Entwicklerteams absolut gleichwertig ist, wenn es darum geht, den neuen Kurs zu wählen. Dazu gehört ein Veto-Recht, um zu große Kursabweichungen zu verhindern. Dies einzufordern ist Aufgabe der Entwickler.

Tun sie das nicht, bleibt noch die verdeckte Führung. Damit ist gemeint, dass die Teams die Aufwände für die Kurskorrektur in anderen Arbeitspaketen „verstecken". Wer mit Story Estimation arbeitet, kann zum Beispiel nach dem Schätzen den ermittelten Wert immer ein wenig erhöhen, um in dem Overhead das Refactoring oder Ähnliches unterzubringen. Alternativ kann man die Velocity senken und in der damit frei gewordenen Zeit die notwendigen Arbeiten unterbringen. Eine Alternative ist die bekannte 80:20-Regel: Vier Tage pro Woche arbeitet das Team an neuen Features, an einem Tag zahlt man Schulden zurück.

Mir gefallen diese verdeckten Ansätze nicht sehr gut, weil es an der – aus meiner Sicht notwendigen – Kommunikation aller Beteiligten fehlt. Ein Product Owner, mit dem ich gearbeitet habe, hat sich regelmäßig darüber beklagt, dass die Entwickler „den Maschinenraum polieren". Aus seiner Sicht fand zu viel Arbeit am Code statt. Die Gefahr ist nicht von der Hand zu weisen und besonders groß, wenn die Arbeiten verdeckt stattfinden.

Kommunikation

Die Ursache dieser unterschiedlichen Wahrnehmungen besteht darin, dass beide Seiten – Entwickler und Business-Leute/Stakeholder/Kunden – die jeweils andere Seite nicht gut genug verstehen. Die Entwickler beschweren sich über unrealistische Termine, und die Stakeholder beklagen die faulen Entwickler. Das eine ist so falsch wie das andere. Meistens zumindest. Der Ausweg ist leider schwierig, weil hier sehr unterschiedliche Menschen zusammenkommen.

Der Ausweg heißt: Kommunikation auf Augenhöhe. Wenn ich als Entwickler Teil des Business bin, verstehe ich, welchen Nutzen die Termine haben. Welches Ergebnis erzielt meine Firma denn, wenn wir einen Termin einhalten? Welchen Nutzen und welchen Schaden richte ich eventuell an, wenn ich den Code „perfekt" haben möchte? Solange ich als Entwickler an den Business-Entscheidungen nicht beteiligt bin, stehen alle Entscheidungen über das Aufnehmen oder die Rückzahlung von technischen Schulden auf tönernen Füßen. Eine Kurskorrektur kann ich nur seriös durchführen, wenn ich beides kenne: Position und Ziel. Wer nur eines kennt, kommt vielleicht nicht lebend an.

Wenn Entwickler über Softwarequalität sprechen, ist stets die Rede von Fehlerfreiheit, Wartbarkeit, Testabdeckung. Doch zur Qualität meines Produkts gehört auch, dass es Umsatz und Gewinn erzeugt. Das ist eine schöne, einfache Weltsicht. Wir entwickeln ein Programm, das wir verkaufen. Verdienen wir mit einem Feature mehr Geld als vorher, dann ist es ein gutes Feature.

Leider ist diese einfache Weltsicht in Konzernen aufgrund der Größe oft nicht anwendbar. Bei interner Software, die ein Rädchen im großen Getriebe ist, ist die Kopplung „Feature => Umsatz/Gewinn" nicht leicht herzustellen. Trotzdem stehe ich zu der Grundaussage. Personen auf unterschiedlichen Hierarchieebenen betrachten sehr unterschiedliche Qualitätsmerkmale. Wer wesentliche Merkmale außer Acht lässt, wird schlechtere Entscheidungen für den Kurswechsel treffen. Da ich sowohl als Manager als auch als Entwickler nur einen Ausschnitt der „gesamten" Qualität sehen kann, braucht es einen Austausch über die verschiedenen Perspektiven.

Die Organisation einer Firma in Abteilungen (lateral) und in Hierarchien (vertikal) behindert diesen Austausch. Es kommt zur Silobildung auf beiden Achsen. Um sie aufzubrechen, braucht es einen Dialog, den beide Seiten anstoßen können. Da die höheren Ränge aufgrund des fehlenden Kontakts zur Software die Probleme erst spüren, wenn es richtig weh tut, sind die Entwickler gefordert, frühzeitig in die Kommunikation zu gehen, Verantwortungsübernahme anzubieten, die nötigen Rechte (insbesondere das Veto-Recht) einzufordern und diese Rechte und Pflichten anzuwenden.

Wer sich ein wenig auf Management-Sprache einstellen will, kann sich mit dem Begriff „Andon" vertraut machen. Dieses aus dem Japanischen stammende Wort wird meist mit einer Reißleine in Verbindung gebracht, die die Mitarbeiter in den Toyota-Werken bereits vor Jahrzehnten ziehen konnten und sollten, wenn sie Qualitätsprobleme erkannten. Es „bedeutet in diesem Kontext, dass jeder Mitarbeiter befugt ist, die Fertigungslinie bei Problemen unverzüglich anzuhalten, damit Qualitätsmängel sofort behoben werden können" [5].

Denkt man an die Kosten, die ein Stillstand bedeutet, ist das erhebliche Macht in der Hand jeder einzelnen Person in der Fertigung. Denkt man an die Kosten, die Qualitätsmängel der Produkte auf der Straße bedeuten, ist es klar, weshalb die Manager diese Macht gerne abgeben. Auch hier ist die Führungsmacht vom Rang zur Kompetenz gewandert. Was hält uns davon ab, eine analoge Regelung für uns Entwickler einzufordern? ■

[1] Svenja Hofert, Claudia Thonet, Der agile Kulturwandel, www.dotnetpro.de/SL2504-05TechnischeSchulden1
[2] Heist-Movie, https://de.wikipedia.org/wiki/Heist-Movie
[3] Softwareverschleiß versus technische Schulden, www.dotnetpro.de/SL2504-05TechnischeSchulden2
[4] Kurskorrekturen auf Wikipedia, www.dotnetpro.de/SL2504-05TechnischeSchulden3
[5] Andon (Prozessoptimierung), www.dotnetpro.de/SL2504-05TechnischeSchulden4

Stefan Mintert
begleitet Organisationen in der agilen Transformation und auf dem Weg zu Agile Leadership. Seine Mission sieht er darin, Menschen dabei zu helfen, sich im Rahmen ihrer Arbeit zu entwickeln und zu wachsen.
stefan.mintert@kutura.digital

dnpCode A2504-05TechnischeSchulden