www.jusletter-it.eu

Nils Fuchs

Management technischer Verschuldung

Vertragliche Instrumente zum Umgang mit technischer Verschuldung in IT-Projekten

Technische Verschuldung ist ein wesentlicher Kostenfaktor bei der Nutzung komplexer und auf längere Zeiträume ausgelegter Software bzw. IT-Systeme. Bisher gibt es dazu jedoch, zumindest in der Schweiz, nur wenig juristische Literatur. Was technische Verschuldung ist, was sie antreibt und wir man ihr entgegenwirken kann, wird in diesem Beitrag anhand eines Austausches mit IT-Experten genauer beleuchtet. Der Autor befasst sich mit den Erscheinungsformen, Treiben und der rechtlichen Einordnung technischer Verschuldung und schlägt Lösungsmöglichkeiten für ein einvernehmliches Management zwischen den Parteien vor.

Beitragsart: Wissenschaftliche Beiträge
Rechtsgebiete: IT-Recht, Vertragsrecht

Zitiervorschlag: Nils Fuchs, Management technischer Verschuldung, in: Jusletter IT 24. April 2024

Inhaltsübersicht

1. Einleitung

[1] Das Konzept der «technischen Schuld» ist nicht neu. Bereits 1992 sprach der Softwareentwickler Ward Cunningham darüber, wie mit der Implementierung von «unreifem Code» Schulden aufgenommen würden. Gewisse Schulden würden zwar die Entwicklung beschleunigen, müssten aber umgehend mit einer Code-Überarbeitung «zurückgezahlt» werden.[1] Im Kern handelt es sich bei technischer Verschuldung um eine Abweichung von der optimalen Entwicklung bzw. Programmierung eines Projektes, welche zu erhöhten Wartungs- und Weiterentwicklungskosten führt. Es wird demnach umso zeitaufwändiger und teurer, je länger man auf technischen Schulden sitzen bleibt. Wie beim Herauszögern der Rückzahlung von Bankkrediten hat es schwerwiegende und kostspielige Folgen, wenn technische Schulden nicht umgehend beglichen werden.

[2] Ein prominentes Beispiel für die Folgen technischer Verschuldung ist der sog. «Y2K Bug». Aus Platzgründen wurde bei Computersystemen bis in die 1990er Jahre die Jahreszahl mit nur zwei Ziffern erfasst (TT/MM/JJ), was beim Übergang vom Jahr 1999 zum Jahr 2000 (durch Anzeige einer Doppelnull) zu Problemen führte. Im Zuge grosser Unsicherheit über die Folgen dieser Umstellung wurden weltweit schätzungsweise 300 Milliarden Dollar für Upgrades von PC-Systemen ausgegeben.[2]

[3] Technische Schulden belasten nicht nur die Gesamtkosten eines IT-Projektes mit Erstellung und Wartung des bestehenden Funktionsumfanges, sie können auch die Einführung neuer Funktionalitäten unverhältnismässig teuer machen. Steigt die technische Verschuldung eines Projektes unkontrolliert, erhöhen sich laufend die Kosten für eigentlich banale Anpassungen.

[4] Aus juristischer Sicht ist die Regelung technischer Verschuldung auf verschiedenen Ebenen interessant. IT-Projekte setzen sich oft aus verschiedenen Dienstleistungen zusammen, rechtlich handelt es sich dabei um gemischte Verträge. Für die Beratung und Konzeption kommt in der Regel Auftragsrecht zur Anwendung, für die Erstellung eines bestimmten Programmes sind Bestimmungen des Werkvertragsrechtes anwendbar und beim Verkauf von Standardsoftware gilt in der Regel Kaufrecht.[3] Der simple «Kauf» eines Webshop-Systems kann also alle Ebenen, namentlich Beratung, Kauf von Standardsoftware, Entwicklung von Individualsoftware und Wartung bzw. Hosting umfassen. Um grösstmögliche (Rechts-)Sicherheit über den Fortbestand und das reibungslose Betreiben eines IT-Projektes zu haben, lohnt es sich also, möglichst früh und klar Instrumente zu vereinbaren, mit welchen Zusatzwünsche, Support, Wartung, Updates und nicht zuletzt die Vermeidung technischer Verschuldung geregelt werden.

[5] Ziel dieses Beitrages ist es, basierend auf einem Austausch mit Experten auf diesem Gebiet, technische Verschuldung zu definieren, ihre Treiber zu bestimmen, Möglichkeiten zu ihrer Minimierung aufzuzeigen und vertragliche Instrumente und Regelungskonzepte vorzuschlagen.

---

1 Ward Cunningham, The WyCash portfolio management system, in: OOPSLA '92: Addendum to the proceedings on Object-oriented programming systems, languages, and applications (Addendum), 1992, DOI 10.1145/157709.157715, S. 29 f.

2 Encyclopedia Britannica, Y2K bug, https://www.britannica.com/technology/Y2K-bug, Stand 4. September 2023.

3 Wolfgang Straub, Informatikrecht, 1. Aufl., Bern, 2004, S. 49–176.

2. Technische Verschuldung

2.1. Grundlagen

2.1.1. Lebenszyklus eines Programmes

[6] Um das Konzept der technischen Verschuldung zu verstehen, ist ein gewisses Grundverständnis von IT-Projektentwicklung und dem Lebenszyklus eines Programmes hilfreich. Bereits 1980 erkannte Lehman die Wichtigkeit, effektive, kosteneffiziente und zeitgerechte Software zu erstellen und zu unterhalten. Im Jahr 1977 wurden etwa 70% der Gesamtausgaben für IT-Projekte in den USA für die Programmpflege und -wartung und nur etwa 30% für die Programmentwicklung aufgewendet. Es ist indes klarzustellen, dass Programmpflege und -wartung alle Änderungen an einem Computerprogramm erfassen. Anders als physikalische Maschinen leiden Programme nicht unter Abnutzung, Verschleiss, Korrosion oder Verschmutzung. Sie ändern sich nicht, es sei denn, Menschen nehmen Anpassungen vor. Dies geschieht immer dann, wenn das aktuelle Verhalten eines Programms als falsch, unpassend oder zu eingeschränkt wahrgenommen wird. «Reparaturen» können also eine Veränderung weg vom Ursprungszugszustand bedeuten.[4] Unter die Softwarepflege und -wartung fällt somit auch deren Weiterentwicklung und Anpassung an veränderte Bedürfnisse, was die eingangs erwähnte Kostenaufteilung relativiert.

[7] Der Lebenszyklus eines Programmes (auch SDLC, Software Development Life Cycle genannt) durchläuft gemäss Lehman im Grundsatz drei Phasen: Definition, Implementierung und Wartung.[5]

[8] Ob starre, aufeinander aufbauende Phasen auf die heutigen komplexen IT-Projekte noch ohne weiteres anwendbar sind, erscheint fraglich, zumal eine agile Entwicklung zunehmend als Standard angesehen wird. Ein fliessender Übergang und ein gewisses Vor und Zurück bzw. mehrmaliges Durchlaufen dieser Lebensphasen erscheint zeitgemässer.

[9] Basierend auf der beschriebenen Grundstruktur haben sich verschiedene Methoden entwickelt, welche je nach den Anforderungen, Mitteln und der gewünschten Mitwirkung des Bestellers unterschiedliche Vorgehensweisen implizieren. Die wohl bekannteste und traditionellste Methode ist die Wasserfall-Methode, in welcher zu Beginn der Entwicklung alle Anforderungen festgelegt werden und die Entwicklung in vorgegebenen, sequenziellen Schritten abläuft.[6] Dem gegenüber steht die heute zunehmend wichtige Agile-Methode, in welcher Anforderungen laufend evaluiert werden und die Entwicklung in inkrementellen Schritten erfolgt.[7] Die Methoden unterscheiden sich also insbesondere darin, in welchem Zeitpunkt Anforderungen festgelegt werden und wie ausgeprägt die Mitwirkung des Bestellers im Entwicklungsprozess ist.

---

4 Meir M. Lehman, Programs, life cycles, and laws of software evolution, in: Proceedings of the IEEE, 1980, DOI 10.1109/PROC.1980.11805, S. 1060 f.

5 Zum Ganzen Lehman, S. 1066 (Fn. 4), gewisse Autoren sprechen sich für eine feinere Aufteilung in Bedürfnisanalyse, Design, Implementierung, Testen und Wartung aus (vgl. T. Saravanan/Sumit Jha/Gautam Sabharwal/Shubham Narayan, Comparative Analysis of Software Life Cycle Models, 2020, DOI 10.1109/ICACCCN51052.2020.9362931, S. 906), wobei es sich hierbei nach Meinung des Autors um einen etwas umfangreicheren Ausdruck derselben Überlegung handelt.

6 T. Saravanan/Sumit Jha/Gautam Sabharwal/Shubham Narayan, Comparative Analysis of Software Life Cycle Models, 2020, DOI 10.1109/ICACCCN51052.2020.9362931, S. 907 f.; vgl. auch Winston W. Royce, Managing the Development of Large Software Systems, 1970.

7 Saravanan/Jha/Sabharwal/Narayan, S. 907 f., (Fn. 6).

[10] Aus alledem folgt, dass Evolution eine intrinsische, durch Feedback angetriebene Eigenschaft von Software ist.[8] Auf Basis dieser Annahme formulierte Lehman seine «Gesetze der Software Evolution». Das erste und zweite Gesetz sind vorliegend von Bedeutung:[9]

1. *Kontinuierliche Veränderung*
   Ein Programm muss laufend angepasst werden, sonst verliert es zunehmend an Nützlichkeit.
2. *Zunehmende Komplexität*
   Mit der Entwicklung eines Systems nimmt seine Komplexität zu, wenn nicht daran gearbeitet wird, sie auf dem bisherigen Niveau zu behalten oder zu reduzieren.

[11] Zwischen diesen beiden Gesetzen besteht ein gewisses Spannungsverhältnis: Auf der einen Seite muss ein Programm laufend angepasst und weiterentwickelt werden; auf der anderen Seite führt dies zu einer höheren Komplexität, welche es wiederum schwieriger und teurer macht, auf Veränderungen zu reagieren. Wird diese Komplexität nicht aktiv beseitigt, resultiert eine technische Verschuldung.

[12] Dass insbesondere das erste Gesetz noch heute Bedeutung hat, bestätigt Vladimir Riecicky[10] mit Hinweis auf das Prinzip der Kontinuität: Software müsse die Realität widerspiegeln. Die Realität ändere sich laufend, also habe sich auch Software laufend anzupassen. Auch dem zweiten Gesetz kommt eine nicht zu unterschätzende Bedeutung zu: Zunehmende Komplexität ist ein Symptom von zu wenig Zeit für die Entwicklung und folglich zunehmender technischer Verschuldung, welche aktiv beseitigt werden muss.

2.1.2. Erscheinungsformen und Treiber

[13] Technische Verschuldung tritt in unterschiedlichen Dimensionen in Erscheinung. Gemeinsam ist ihnen oft, dass den Entwicklern nicht dasjenige Umfeld gegeben wird, in welchem die Aufarbeitung und Beseitigung technischer Verschuldung im Rahmen des gewöhnlichen Betriebes möglich ist. Es muss die Grundüberzeugung gelten, dass Entwickler einen möglichst optimalen Code schaffen wollen und dies auch (entsprechend ihrer Fähigkeiten) können, wenn ihnen das richtige Umfeld gegeben wird. Die folgend dargelegten Erscheinungsformen von technischer Verschuldung ergeben sich aus Expertengesprächen mit Stephan Sutter[11] und Vladimir Riecicky. Sie sind nicht trennscharf, gehen fliessend ineinander über und hängen in gewissem Masse auch voneinander ab.

1. *Kurzfristige «quick and dirty» Lösungen (Spaghetti-Code)*:
   Um engen Zeitplänen gerecht zu werden, sehen sich Entwickler gezwungen, rasche und unausgereifte Lösungen zu implementieren, die nicht den optimalen technischen Standards entsprechen. Dies kann zu erhöhtem Wartungsaufwand, hoher Codekomplexität, mangeln-

8 Lehman, S. 1067 (Fn. 4).

9 Lehman, S. 1068 (Fn. 4, Zusammenfassung und Übersetzung des Autors).

10 Vladimir Riecicky ist einer der Interviewpartner des Autors und Experte auf dem Gebiet der IT-Projektentwicklung.

11 Stephan Sutter ist einer der Interviewpartner des Autors und ebenfalls Experte auf dem Gebiet der IT-Projektentwicklung.

der Skalierbarkeit und einem erhöhten Risiko von Fehlfunktionen führen. Man handelt sich eine Schuld gegenüber der zukünftigen Entwicklung ein: Wird diese nicht bald «zurückbezahlt», schlägt sie sich in allen weiteren Anpassungen des Codes in Form gesteigerter Kosten nieder. Die Qualität des Codes, in welcher sich technische Verschuldung am ehesten zeigt, leidet.

2. *Unzureichende Dokumentation*:
   Zu enge Zeitpläne können zu einer unzureichenden Dokumentation des Codes oder des Gesamtsystems und diese wiederum zu Problemen bei künftiger Wartung oder Erweiterung führen. Entwickler, die später mit dem Code arbeiten müssen, können Schwierigkeiten haben, ihn zu verstehen und Änderungen vorzunehmen. Es kommt zu teuren Zeitverlusten. Auch führt dies zur Problematik des sog. Key-Personals, wenn das System nur noch von einigen wenigen, hochqualifizierten Entwicklern verstanden wird, deren Abgang ein Abrutschen in die digitale Sklaverei bedeutet.[12] Eine unzureichende Dokumentation ist ein klares Symptom technischer Verschuldung.
3. *Mangelhafte Automatisierung von Tests*:
   Wenn unzureichende, zu wenig oder gar keine automatisierten Tests implementiert werden, um die Funktionalität und Qualität der Software zu überprüfen, kann dies zu unentdeckten Fehlern führen. Besonders bei kurzfristigen Lösungen für ein spezifisches Problem kann der Mangel an automatisierten Tests dazu führen, dass Fehler (als «Spätfolge» der kurzfristigen Lösung) erst spät im Entwicklungsprozess erkannt werden, was erhebliche Kosten und Verzögerungen verursacht. Auch erschwert ein Mangel an Tests die Übersichtlichkeit über den Code und damit auch die allenfalls nötigen Arbeiten wie Refactorings.
4. *Verwendung veralteter Technologien und Durchbrechung der Kontinuität*:
   Der Einsatz veralteter Technologien kann angehäufte technische Verschuldung bedeuten. Veraltete Technologien können Sicherheitslücken aufweisen, die vom Anbieter nicht mehr behoben werden, und es kann schwierig sein, qualifizierte Entwickler zu finden, die diese warten oder erweitern können. Dies kann so weit gehen, dass mit der verwendeten Technologie die Realität nicht mehr abgebildet werden kann, also das Prinzip der Kontinuität durchbrochen wird. In diesem Fall muss eine Ablösung der Software in Betracht gezogen werden.
5. *Teamfluktuation und Team-Mix*:
   Durch zu kurze Fristen und einen zu hohen Business-Delivery-Druck steigt die technische Schuld, die Komplexität des Codes nimmt zu und die Dokumentation nimmt ab, beides entgegen den Vorstellungen des Entwicklerteams. Wird ihnen wiederholt verunmöglicht, ihre Arbeit ordnungsgemäss und nach ihren Qualitätsansprüchen durchzuführen, resignieren sie, wodurch es zu Teamfluktuationen und einem Wissensverlust durch Abnahme von Senior-Entwicklern bzw. Zunahme von Junior-Entwicklern kommt. Verändert sich der Team-Mix eines Entwicklerteams hin zu mehr Junior-Entwicklern und weniger Senior-Entwicklern, kann dies ein Ausdruck technischer Verschuldung sein.[13]

12 Die digitale Sklaverei ist nach Vladimir Riecicky der Zustand, in dem nicht mehr die Entwickler, sondern der Code bzw. das IT-Projekt den Entwicklungsprozess steuert, da die technische Schuld derart hoch ist, vgl. die Abbildung unter https://www.k-at-r.com/transformation#h.h4iltci46zpd. Gleichzeitig kann eine Abhängigkeit von Key-Entwicklern gewollt sein, um sich «unersetzlich» zu machen, eine differenzierte Personalpolitik ist gefragt.

13 Vgl. Fn. 12.

[14] Anhand der genannten Erscheinungsformen lassen sich verschiedene Treiber technischer Schuld eruieren. In erster Linie treibt ein zu *hoher Zeitdruck* bzw. «Business-Delivery-Druck»[14], verstärkt durch ein *mangelndes Bewusstsein* (aller Parteien) für die Problematik, die Zunahme von «Spaghetti-Code» an und schafft damit technische Verschuldung. Werden zu schnell zu viele Features erwartet, ohne genügend Zeit für die Optimierung des bestehenden Codes zu gewähren, leiden die Code-Qualität und Dokumentation. Dazu kommt ein Wissensverlust aufgrund von *Teamfluktuationen*, wenn erfahrene Entwickler das Team verlassen. Weiter kann ein *Mangel automatisierter Tests* bzw. grundsätzlich *qualitätssichernder Prozesse* zu unerkannten Problemen führen, die sich erst zu spät bemerkbar machen. Nicht zuletzt treibt auch ein basierend auf einer *schlechten Architektur* entwickeltes Projekt die technische Verschuldung an und kann die Einhaltung der Kontinuität erschweren.

2.1.3. Definition

[15] Nach dem Gesagten lässt sich technische Verschuldung wie folgt definieren:

> *Technische Verschuldung ist das Ausmass an ausgelassener Arbeit, gemessen an der optimalen Entwicklung, und damit das Ergebnis (bewusster) Abkürzungen in der Entwicklung einer Software. Je länger diese Schuld nicht beseitigt wird, desto grösser sind die Auswirkungen auf Betrieb, Wartung und Weiterentwicklung.*

[16] Eine niedrige technische Verschuldung bedeutet, dass das Systemdesign sauber und leicht zu verstehen ist und die Dokumentation dazu umfassend und aktuell ist. Bei einem hohen technischen Schuldenstand ist es hingegen, ohne persönliche Anleitung durch die involvierten Systementwickler, sehr schwierig zu verstehen, wie das System funktioniert.[15]

2.2. Folgen und Messbarkeit

[17] Die Folgen technischer Verschuldung sind so vielfältig wie ihre Erscheinungsformen und hängen oft direkt mit selbigen zusammen. In erster Linie sind s*teigende Kosten* eine Hauptfolge, indem Betriebs- und Wartungskosten steigen. Auch wird die *Weiterentwicklung* durch die Notwendigkeit, technische Schulden abzuarbeiten, erschwert bzw. unverhältnismässig hohe Weiterentwicklungskosten verursachen. Weiter besteht das Risiko von *Entwicklerfrustration und dem Verlust von Fachwissen*, wenn erfahrene Entwickler resignieren und das Entwicklerteam verlassen. Nicht zuletzt leidet die *Kundenzufriedenheit* aufgrund von Zeitverzögerungen und der mit dem Zuwachs technischer Verschuldung einhergehenden Risikoerhöhung für den Fortbestand des Projektes.

[18] Um Regelungen zur Minimierung oder Beseitigung von technischer Verschuldung entwickeln zu können, muss diese in ihren verschiedenen Erscheinungsformen messbar gemacht werden. Dazu gibt es verschiedene Möglichkeiten, die die unterschiedlichen Facetten technischer Verschuldung abdecken können:

---

14 Vgl. Fn. 12.

15 Vladimir Riecicky, Knowledge Risk Exposure, 2016, Beitrag auf LinkedIn, https://www.linkedin.com/pulse/knowledge-risk-exposure-vladimir-riecicky, Stand 9. Oktober 2023.

1. *Codequalität und Architektur*: Zur Messung technischer Verschuldung im Code bestehen bereits verschiedene Softwaretools, die unterschiedliche Metriken zur Berechnung einer technischen Verschuldung verwenden. Beurteilt werden unter anderem die Architektur, das Design und der eigentliche Code, jeweils in verschiedenen Kombinationen und auf verschiedene Art und Weise.[16] Auf technischer Ebene lässt sich mittels dieser Tools technische Verschuldung klar definieren und berechnen. Das bekannteste Programm ist wohl SonarQube[17], ein «*selbstverwaltetes statisches Analysewerkzeug für die kontinuierliche Überprüfung der Codebase*»[18]. Die Messung der technischen Schuld im Code kann durch den Anbieter selbst oder auch Dritte (bspw. im Rahmen eines Gutachtens) erfolgen.
2. *Marginal Cost of Ownership*: Die sog. Marginal Cost of Ownership ist eine von Vladimir Riecicky vorgeschlagene Masseinheit zur Messung technischer Verschuldung.[19] Es handelt sich hierbei um eine Art der Grenzkostenbetrachtung, bei der die fachliche Komplexität einer Story mit den Kosten ihrer Umsetzung durch die Zeit verglichen wird. Die Grundüberlegung ist, dass die Implementierung einer ähnlich komplexen Funktion im Laufe des Projektes gleich viel kosten müsste. Ist dies nicht der Fall, kostet die Implementierung dieser ähnlich komplexen Funktion also mehr als zuvor, steigen die Grenzkosten, was ein Zeichen für eine potenzielle technische Verschuldung ist (mehr technische Verschuldung = mehr (Code-)Komplexität = mehr Zeit und Geld für Änderungen am Code). Gemessen an dieser Entwicklung lässt sich also eine Aussage über die technische Verschuldung eines IT-Projektes machen.
3. *Team-Mix*: Mitverantwortlich für den Erfolg eines IT-Projektes ist die Fähigkeitsverteilung im Team. Werden (Senior-) Entwickler nicht genügend gefördert oder zu lange einem ungünstigen Arbeitsumfeld ausgesetzt, verlassen sie das Team, woraufhin sich die verbleibenden (Junior-) Entwickler im Blindflug oder schlimmstenfalls bereits in der digitalen Sklaverei befinden. Im Blindflug schaffen die verbliebenen Entwickler unbewusst weitere technische Verschuldung.[20] Folglich lässt sich aus der Fähigkeitsverteilung und allenfalls Teamfluktuation ein Mass für mögliche technische Verschuldung ableiten.

2.3. Lösungsansätze

2.3.1. Auf Ebene des Codes

[19] Die wohl naheliegendste Lösung zur Minimierung der technischen Verschuldung ist, sie bereits auf der Ebene des Codes zu beseitigen. Dazu gibt es verschiedene Aspekte, die helfen, einen möglichst schuldarmen Code sicherzustellen.

16 Vgl. Paris C. Avgeriou/Davide Taibi/Apostolos Ampatzoglou/Francesca Arcelli Fontana/Terese Besker/Alexander Chatzigeorgiou/Valentina Lenarduzzi/Antonio Martini/Athanasia Moschou/Ilaria Pigazzini/Nyyti Saarimaki/Darius Daniel Sas/Saulo de Soares Toledo/Angeliki Agathi Tsintzira, An Overview and Comparison of Technical Debt Measurement Tools, in: IEEE Software, 2021, DOI 10.1109/MS.2020.3024958, S. 61–68 für eine grosse Übersicht über die bestehenden Programme und deren Möglichkeiten.

17 Avgeriou, S. 65 f. (Fn. 16).

18 https://www.sonarsource.com/products/sonarqube/, Übersetzung des Autors.

19 Vladimir Riecicky, Technical debt eats business case for breakfast, Beitrag auf LinkedIn, https://www.linkedin.com/pulse/technical-debt-eats-business-case-breakfast-vladimir-riecicky, Stand 09.10.2023.

20 Fn. 12.

2.3.1.1. Agile Entwicklung

[20] Agile Entwicklung ist eine Entwicklungsmethode basierend auf dem sog. Agile Manifesto.[21] Im Kern bedeutet Agilität die Fähigkeit, schnell und flexibel Veränderungen im geschäftlichen und technischen Bereich wahrzunehmen und darauf zu reagieren. Im Gegensatz dazu steht die eine lange Zeit verbreitete «Wasserfall» Entwicklungsmethode, nach welcher bereits zu Beginn der Entwicklungsarbeiten klargestellt wird, in welcher Reihenfolge welche Schritte unternommen werden müssen, um zu dem gewünschten Ergebnis zu gelangen. Vor dem Hintergrund reglementarischer Entwicklungen bspw. zur «Compliance by Design» erscheint die agile Entwicklung heute als das Mass der Dinge.[22]

[21] Eine Entwicklung anhand der im agilen Manifest darlegten Grundsätze schafft einen Mehrwert, indem in regelmässigen, kurzen Abständen (Sprints) funktionierende Softwareteile (Inkremente) erstellt werden. Dies erlaubt es, geänderten Anforderungen in jeder Phase des Entwicklungsprozesses auch kurzfristig Rechnung zu tragen und die zukünftige Arbeit (kommende Sprints) bei Bedarf anpassen zu können. Darüber hinaus wird der Besteller aktiv in den Entwicklungsprozess einbezogen, wodurch Feedback und Reflexion erleichtert werden. Die Grundsätze stellen keine formale Definition von Agilität dar, sondern sind vielmehr Leitlinien für die Bereitstellung hochwertiger Software auf agile Weise.[23]

[22] Für das Management technischer Verschuldung ausschlaggebend sind die Möglichkeit, noch während des Entwicklungsprozesses schnell auf geänderte Anforderungen zu reagieren, sowie der regelmässige Austausch zwischen Besteller und Anbieter. Auch besteht die Möglichkeit, eine Entwicklungseinheit, also einen Sprint, zugunsten der Reduktion technischer Verschuldung, anstatt zur Entwicklung einer neuen Funktion, zu verwenden. Bei der optimalen Entwicklung von Software verbindet die Produktvision den Anbieter und den Besteller. Erst im Zusammenspiel der jeweiligen Fähigkeiten und Möglichkeiten kann eine Produktvision geschaffen, verfeinert und realisiert werden. Eine agile Entwicklung stärkt diesen Gedanken und schafft eine gute Grundlage, um technischer Verschuldung bereits im Entwicklungsstadium zu begegnen.

2.3.1.2. Exkurs: DevOps[24]

[23] Ein bekannter, agiler Entwicklungsansatz stellt die DevOps-Methode dar. Diese setzt sich aus den Wörtern «Development» für Entwicklung sowie «Operations» für den Betrieb ab.[25] Die normalerweise getrennt agierenden IT-Bereiche Entwicklung und Betrieb werden durch die DevOps-Methode zusammengeführt mit dem Ziel, effizientere Softwareprodukte zu realisieren.[26] DevOps verfolgt bei der Softwareentwicklung und -bereitstellung einen iterativen Ansatz. Bei der

21 https://agilemanifesto.org/, Stand 10. Oktober 2023.

22 Hauke Precht/David Saive, Compliant Programming – Juristen in der agilen Software-Entwicklung, in: InTeR, 2020, S. 13 f.

23 Torgeir Dingsøyr/Sridhar Nerur/VenuGopal Balijepally/Nils Brede Moe, A decade of agile methodologies: Towards explaining agile software development, in: Journal of Systems and Software, 2012, DOI 10.1016/j.jss.2012.02.033, S. 1214.

24 Verfasst von MLaw Vaishnavan Subramaniam, der sich in seiner Masterarbeit an der Universität Bern mit DevOps-Verträgen befasste.

25 Björn Kasteleiner/Alexander Schwartz, Informatik Spektrum 42, München 2019, S. 212.

26 Inge Hanschke: Agile in der Unternehmenspraxis: Fallstricke erkennen und vermeiden, Potenziale heben, Wiesbaden 2017, S. 49.

DevOps-Methode werden Prozesse in regelmässigen Abständen wiederholt, um neue Anforderungen, Funktionen und Verhaltensweisen, welche sich aus dem Betrieb ergeben, aufzufangen. Diese werden als DevOps-Prozesse bezeichnet.[27] Grundsätzlich umfasst der DevOps Prozess folgende Phasen: Planung, Entwicklung, Test, Bereitstellung, Inbetriebnahme, fortlaufende Überwachung, Feedback-Einholung und damit die Rückkehr zur Planung. Diese Prozesse sind jedoch nicht in Stein gemeisselt. Bei Bedarf können Sie angepasst werden, denn das Ziel von DevOps ist die ständige Optimierung und Verbesserung von Prozessen, damit die Softwareentwicklung und -bereitstellung noch effizienter erfolgt.[28] Der Leistungsumfang eines Produkts (Scope) wird, wie bei allen agilen Entwicklungsmethoden, nicht bereits von Beginn weg festgelegt, sondern es entstehen im Rahmen mehrerer, kurzer Entwicklungsphasen (Sprints) neue Prototypen. Dadurch erhält der Entwickler die Möglichkeit, rasch auf sich ändernde Erfordernisse eines Bestellers oder des Markts zu reagieren, oder sich an neue Technologien sowie Tools anzupassen.[29]

[24] Die DevOps-Methode bietet somit die Möglichkeit, technischer Verschuldung bereits während einzelner Sprints zu begegnen. Ein Vorteil von DevOps im Vergleich zu anderen agilen Entwicklungsmethoden besteht darin, dass DevOps auch die Betriebsphase miteinschliesst. Dadurch erhält der Besteller bereits im Anfangsstudium ein Feedback darüber, wie die einzelnen Softwareteile im operativen Alltag entsprechend ihres Nutzens abschneiden und welche Problemfelder noch bestehen. Der Besteller kann dadurch das Risiko einer technischen Verschuldung bereits im Vorfeld minimieren.

[25] In der Praxis besteht bei DevOps jedoch die Problematik, dass Entwicklungskosten (CapEx) und Betriebskosten (OpEx) separat betrachtet werden. CapEx bezeichnet Investitionen, die einmalig anfallen, während OpEx die wiederkehrenden, laufenden Betriebskosten umfasst. Wenn ein Teil des Softwaresystems gewartet wird, werden diese Kosten den Betriebskosten zugeordnet. Werden Softwareteile entwickelt, gehören die entstehenden Kosten zu den Entwicklungskosten. Das bedeutet im Umkehrschluss, dass sich Unternehmen entweder für CapEx oder OpEx entscheiden müssen. Dadurch werden die tatsächlichen Kosten verschleiert, da sich die Kosten eines Softwaresystems während seiner Lebensdauer zwischen diesen beiden Bereichen hin und her bewegen.[30] Für das Management technischer Verschuldung ist es daher unumgänglich, sich diesem Problem anzunehmen, bevor DevOps als Lösungsansatz in Betracht gezogen wird.

2.3.1.3. Code-Audit

[26] Ein Audit ist «eine unabhängige Prüfung eines Softwareprodukts, eines Softwareprozesses oder einer Reihe von Software-Prozessen durch einen Dritten, um die Einhaltung von Spezifikationen, Standards, vertraglichen Vereinbarungen oder anderen Kriterien zu beurteilen»[31]. Codequalität und sogar technische Verschuldung selbst lassen sich wie beschrieben mittels Analyse-

27 Hanschke S. 51 (Fn. 26).

28 Rúben Dos Santos Barros, DevOps Technologies for Tomorrow, Diss. Porto, 2016, S. 13; Antonio Capizzi/Salvatore Distefano/Manuel Mazzara, From DevOps to DevDataOps: Data management in DevOps processes, Ithaca, 2019, S. 3; Georgia König/René Kugel, in: Hans-Peter Fröschle/Ralf Oesterreich/Nikolaus Schmidt (Hrsg.): IT-Operations in der Transformation, Zukunftsweisende IT-Betriebsmodelle zwischen «Hey Joe» und «NoOps», Wiesbaden, 2022, S. 174.

29 Emily Freeman, DevOps für Dummies, Weinheim, 2020, S. 30.

30 Ron Vincent, DevOps and Finance, https://www.linkedin.com/pulse/devops-finance-ron-vincent.

31 IEEE Standard for Software Reviews and Audits, 2008, S. 5, Ziff. 3.2, DOI: 10.1109/IEEESTD.2008.4601584. Übersetzung des Autors.

software beurteilen. Ein vollständiger Code-Audit kann zudem massgeschneidert erfolgen und über die reine Verwendung eines Analysetools hinaus Änderungsempfehlungen umfassen. Ein solcher Code-Audit kann in erster Linie durch einen unabhängigen Dritten, aber auch durch den Anbieter selbst im Sinne einer Selbstkontrolle durchgeführt werden.

[27] Durch Festlegung eines Standards sowie bestimmter Kennzahlen und Messmethoden, bspw. durch Bestimmung eines reputablen Analysetools, kann eine Code-Qualität (und eine Qualität der Dokumentation) vereinbart werden, die sowohl der Anbieter als auch der Besteller nachvollziehen und messen kann.[32] Über eine allfällige technische Verschuldung im Code kann der Anbieter selbst, unter Bezugnahme auf die vereinbarten Kennzahlen, eine Aussage machen. Es können ihm auch entsprechende Sorgfaltspflichten überbunden werden, die vereinbarten Kennzahlen einhalten zu müssen. Im Zweifelsfall können die Kennzahlen durch einen unabhängigen Dritten überprüft werden.

2.3.1.4. Automatisierte Tests und Refactoring

[28] Damit der Code bereits von Beginn weg möglichst unverschuldet ist, bietet es sich an, automatisierte Testverfahren anzuwenden. Dieses Vorgehen hängt auch mit dem oben beschriebenen Code-Audit zusammen: Ziel ist es, eine hohe Code-Qualität zu gewährleisten und zu verhindern, dass Fehler und folglich technische Verschuldung erst bei einem externen Audit sichtbar werden. Es bedarf deshalb einer Übersicht durch den Softwareentwickler selbst und automatisierte Testverfahren sollten von Beginn weg vorgesehen sein.

[29] Sich selbst testender Code erlaubt es zudem, Bugs bereits bei ihrer Entstehung zu erkennen. Damit erhöht sich die Fehlerlosigkeit des Codes sowie auch die Produktivität der Entwickler, da weniger Zeit mit Debugging verbraucht werden muss. Bugs lassen sich somit von selbst identifizieren und effizient beseitigen.[33] Des Weiteren reduziert sich die Anfälligkeit für technische Verschuldung in Form von kurzfristig funktionierendem, aber langfristig problematischem Code, da die Tests fast alle potenziell betroffenen Bereiche abdecken können. Ein möglicher Blindpunkt besteht allerdingt im Auffinden von Architektur- und Designfehlern.

[30] Um den Code aufzuräumen und allenfalls nötige Anpassungen vorzunehmen (bspw. aufgrund geänderter Anforderungen, damit das Prinzip der Kontinuität nicht missachtet wird), bietet sich ein sog. Refactoring an. Refactoring ist ein Prozess, bei dem ein Softwaresystem so verändert wird, dass sich das äussere Verhalten des Codes nicht verändert, sich aber seine interne Struktur verbessert.[34] Voraussetzung sind unter anderem solide (Selbst-)Tests im Code, was die Wichtigkeit soeben erwähnter automatisierter Tests noch erhöht.[35] Ein Refactoring ist indes nicht ganz ungefährlich: Es erfordert Änderungen an funktionierendem Code, welche subtile Bugs einführen können.[36] Es gibt eine grosse Zahl verschiedener Refactoring-Szenarien, welche unterschiedliche und teils widersprüchliche Einflüsse auf die Codequalität haben können. Mit der

32 Um potenzielle tote Winkel oder Eigenheiten bestimmter Analysetools zu vermeiden, ist es empfehlenswert, mehrere Tools parallel anzuwenden.

33 Martin Fowler, Refactoring, Improving the design of existing code, Reading, Mass, 1999, S. 73 f.

34 Fowler S. 9 (Fn. 33).

35 Fowler S. 73 (Fn. 33).

36 Fowler S. 6 (Fn. 33).

richtigen Wahl lässt sich jedoch eine klare Verbesserung der Codequalität erzielen.[37] Ein Refactoring lohnt sich also nach grösseren Änderungen, sollte aber nicht ohne Not erzwungen werden.

2.3.2. Auf Ebene der Entwicklungspartner

[31] Neben der Code-Ebene kann technische Verschuldung auch auf Ebene der Entwicklungspartner angegangen und reduziert werden. Ein sowohl von Sutter als auch Riecicky hervorgehobener Aspekt ist das grundsätzliche Bewusstsein aller beteiligten Parteien über die Problematik der technischen Verschuldung und den Lebenszyklus des in Frage stehenden Produktes. Wie oben dargelegt ist ein zu hoher «Business-Delivery-Druck» einer der Haupttreiber technischer Verschuldung. Wird in zu kurzer Zeit zu viel von den Entwicklern verlangt, sowohl seitens des Managements des Anbieters als auch seitens des Kunden, müssen die Entwickler Abkürzungen nehmen, woraus suboptimaler Code und eine mangelhafte Dokumentation resultieren. Ein diesbezügliches gegenseitiges Bewusstsein kann einen Haupttreiber also dämpfen.

2.3.2.1. Team-Mix

[32] Wie in Kapitel 2.2 anhand des von Riecicky entwickelten K@R Navigators[38] dargelegt, besteht eine direkte Verbindung zwischen technischer Verschuldung und dem Team-Mix in einem Entwicklungsteam. Wird dem Team ein suboptimales Umfeld gegeben, verlassen in letzter Konsequenz essenzielle Entwickler das Team und hinterlassen ein Projekt mit hoher technischer Verschuldung und ein Team mit geringer Berufserfahrung, welche wiederum unbewusst technische Verschuldung schaffen. Es bietet sich folglich an, ein Auge auf die Zusammensetzung des Entwicklerteams zu haben und allenfalls mittels Vereinbarung sicherzustellen, dass immer eine genügende Anzahl erfahrener Entwickler involviert ist. Auch ist darauf zu achten, dass Junior-Entwickler im Team «nachgezogen» und weitergebildet werden und mit der Applikation vertraut gemacht werden.

2.3.2.2. Marginal Cost of Ownership

[33] Anknüpfend an die oben erklärte Masseinheit der Marginal Cost of Ownership lässt sich mit ihr eine weitere Metrik für die technische Verschuldung eines Projektes festlegen, auf welche sich die Parteien stützen können. So kann vereinbart werden, dass der Anbieter neben der Kostenschätzung eine Beurteilung der Komplexität der in Frage stehenden Änderung oder Weiterentwicklung vornimmt, und diese kategorisiert – bspw. mit Buchstaben. Zusätzlich können als weitere Grundlage bereits im Voraus die Bedingungen bestimmter Standardänderungen festgelegt werden, konkret innert welcher Zeit (und mit welchem Aufwand) eine Standardänderung realisiert werden muss.

[34] Es kann vereinbart werden, welches Verhältnis zwischen Komplexität und Kosten eingehalten werden soll, über welchen Zeitraum die Beurteilung erfolgt und wie bei unterschiedlichen

---

37 Jehad Al Dallal/Anas Abdin, Empirical Evaluation of the Impact of Object-Oriented Code Refactoring on Quality Attributes: A Systematic Literature Review, in: IEEE Transactions on Software Engineering, 2018, DOI 10.1109/TSE.2017.2658573, S. 66.

38 https://www.k-at-r.com/transformation#h.h4iltci46zpd, Stand 26. Februar 2024.

Verhältnissen reagiert wird (bspw. in Form von Eskalationsstufen beginnend mit einer Informationspflicht über kostenfreie Wartungssprints bis hin zu einer Vertragsauflösung). Dies schafft einerseits Raum für eine gewisse Flexibilität: Es bleibt also möglich, dass Änderungen etwas mehr oder weniger Kosten verursachen als «sie sollten». Andererseits wird eine Kennzahl geschaffen, aufgrund welcher Massnahmen ergriffen werden können, sollte sie zu lange einen bestimmten Wert unter- oder überschreiten.

2.3.2.3. Technische Verschuldung als Funktion und Kontinuität

[35] Nicht zuletzt kann technischer Verschuldung begegnet werden, indem man sie als Funktionsbestandteil betrachtet. Als Resultat des gegenseitigen Bewusstseins der Parteien für die Problematik kann technische Verschuldung als normaler Bestandteil eines Entwicklungsprozesses betrachtet werden. Dadurch lässt sich der Behebung allfällig angehäufter technischer Verschuldung wie der Entwicklung einer neuen Funktion begegnen. Das bedeutet, dass insbesondere diese zeitlichen und finanziellen Ressourcen für die «Aufräumarbeiten» zur Verfügung gestellt werden. Die Parteien gestehen sich damit ein, dass eine gewisse technische Verschuldung in der Entwicklung «zu erwarten» ist. Die technische Verschuldung ist jedoch kaum zu jedem Zeitpunkt vollständig fassbar. Dem könnte mit periodischen Standortgesprächen begegnet werden.

[36] Konkret sollte der Anbieter einen Überblick über die angesammelte technische Verschuldung im Projekt haben, bspw. durch Messungen wie oben beschrieben. Für diese Schuld vereinbaren die Parteien eine Informationspflicht. Sobald die Schuld einen gewissen Stand, gemessen bspw. an einer Kennzahl, erreicht hat und bereinigt werden muss, wird diese Bereinigung behandelt wie jede andere Entwicklung. Das heisst, dass auch der gleiche finanzielle Aufwand wie bei der Entwicklung einer neuen Funktion in Rechnung gestellt wird. Dieses Vorgehen funktioniert indes nur in Kombination mit den anderen Regelungsmöglichkeiten zur Minimierung der technischen Verschuldung.[39] In erster Linie soll diese also minimiert werden. Für denjenigen Teil, der auch bei treuer und redlicher Entwicklungsarbeit entsteht, kommt dann die beschriebene Vorgehensweise zum Zug.

[37] Ebenfalls als wichtiger Funktionsbestandteil sollte die Sicherstellung der Kontinuität betrachtet werden. Vladimir Riecicky betont, dass eine Software immer die Realität abzubilden hat. Wird eine essenzielle Anpassung an geänderte Umstände in der Realität, bspw. eine bevorstehende reglementarische Änderung, nicht genügend im Code abgebildet, verschlimmern sich die Konsequenzen bei einem allfälligen Eintritt der entsprechenden Veränderung in der Realität. Das gilt im Übrigen auch für die mittels Software unterstützten Unternehmensprozesse. Neben dem Abarbeiten technischer Schuld sollte also die nötige Zeit aufgewendet werden, um einen Abgleich mit der Realität zu ermöglichen, damit die zukünftige Entwicklung nicht eine unverhältnismässig grosse Abweichung des verlangten Funktionsumfanges auszugleichen hat.

39 Ansonsten läuft man Gefahr, einen Kobra-Effekt zu erzeugen, also das Problem, welches man mit der Massnahme zu beheben versucht, noch verschlimmert, siehe Horst Siebert, Der Kobra-Effekt: Wie man Irrwege der Wirtschaftspolitik vermeidet, Stuttgart, München, 2002.

2.3.2.4. Konfliktlösung und Exit-Strategie

[38] Nur am Rande mit dem Management technischer Verschuldung verknüpft, aber dennoch zu beachten, ist das Vereinbaren von Konfliktlösungsmechanismen und das Bereithalten einer Exit-Strategie. Sollte es bspw. im Zusammenhang mit steigender technischer Verschuldung zu Meinungsverschiedenheiten kommen, kann mit einem vorbestimmten Eskalationsverfahren bestenfalls ein langwieriges Gerichtsverfahren vermieden werden.[40]

[39] Als möglicher erster Eskalationsschritt bietet sich die gemeinsame Lösungsfindung zwischen den Parteien, wenn nötig auf der jeweils nächst höheren Ebene in der Organisationshierarchie (bspw. Projektteam, Leitungsausschuss, Geschäftsleitungsmitglieder) an, jeweils mit verbindlichen Fristvorgaben. Schliesslich kann die Einsetzung eines Mediators und/oder die Aufnahme einer Schiedsklausel eine Beschleunigung ermöglichen.[41]

[40] Zusätzlich kann es sich anbieten, klare Prozesse zu vereinbaren, die im Falle eines vorzeitigen Ausstieges aus dem Projekt einen möglichst sauberen Übergang der Entwicklungsgrundlagen (Codebase, Dokumentation etc.) gewährleisten. Dadurch kann die Arbeit am Projekt mit einem anderen Partner möglichst bald weitergeführt werden, während allfällige Ansprüche gegen den ursprünglichen Partner geprüft und gegebenenfalls (gerichtlich) durchgesetzt werden können. Eine Möglichkeit hierzu ist das Hinzuziehen eines «Escrow-Agenten». Bei diesem hinterlegt der Softwareersteller regelmässig den aktuellen Sourcecode und weitere Entwicklungsdokumente des Projektes. Unter gewissen, zuvor bestimmten Bedingungen kann der Besteller die entsprechenden Daten von dem Escrow-Agenten beziehen und erlangt so eine gewisse Unabhängigkeit vom Softwarehersteller.[42]

[41] Als Herausgabefälle werden meist nur Insolvenz und manchmal auch rechtskräftige Urteile vereinbart. Möglich – und in diesem Zusammenhang sinnvoll – kann die Möglichkeit sein, eine Herausgabe nur gegen Leistung eines bestimmten Geldbetrages zu erlauben. Solche Beträge sind aber in der Regel hoch, um den Anbieter zur schützen.

3. Rechtliche Möglichkeiten

3.1. Regelungsmöglichkeiten

[42] Ziel dieses Beitrages ist es, vertragliche Instrumente für den Umgang mit technischer Verschuldung aufzuzeigen. Um eine Einbettung in das bestehende Regelungssystem sicherzustellen, ist es unumgänglich, zunächst eine kurze Analyse des bestehenden gesetzlichen Rahmens durchzuführen. Diese soll den Grundstein für die anschliessenden vertraglichen Lösungsansätze legen. Die Analyse beschränkt sich auf das Schweizer Recht.

---

40 Gerichtliche Auseinandersetzungen sind bezogen auf IT-Projekte grundsätzlich eher selten, wofür es gemäss Straub eine Mehrzahl von Gründen gibt, vgl. Wolfgang Straub, Verantwortung für Informationstechnologie, Zürich, 2008.

41 Vgl. Wolfgang Straub, Legal Engineering, in: Jusletter 7. Oktober 2019, 2019, Rz. 75 ff.; siehe auch Ursula Sury/André Henri Kuhn, Digital in Law, 2. Aufl., Bern, 2021, S. 149–152.

42 Sury S. 145–150 (Fn. 41).

3.1.1. Gesetzliche Regelung

3.1.1.1. Vertragsqualifikation

[43] Zur Analyse der auf ein IT-Projekt anwendbaren gesetzlichen Bestimmungen ist zunächst eine Vertragsqualifikation vorzunehmen. Aus Platzgründen wird hierbei nur der für IT-Projekte relevante Teil genannt und auf die Arbeit von Straub[43] verwiesen.

1. *Auftrag*: Im Zusammenhang mit IT-Projekten fallen insbesondere *Beratung*, *Planung und Projektmanagement* unter die Bestimmungen des Auftragsrechtes.[44] Abzugrenzen sind bspw. ausgearbeitete Konzepte als Projektgrundlage, für welche das Werkvertragsrecht anwendbar sein kann.
2. *Werkvertrag*: In den Anwendungsbereich des Werkvertragsrechts fallen im Rahmen von IT-Projekten die Erstellung *individueller Produkte und Informationssysteme*.[45]
3. *Kaufvertrag*: Im IT-Bereich kommt Kaufvertragsrecht insbesondere für den *Verkauf von Standardsoftware* zur Anwendung. Dies gilt auch für noch zu erstellende Software, wenn es sich dabei um ein Serienprodukt handelt.[46] Allenfalls ist eine Abgrenzung zum Lizenzvertrag vorzunehmen.
4. *Gemischte Verträge*: Bei IT-Projekten handelt es sich oft um eine Kombination aus verschiedenen Leistungen und damit verschiedenen Vertragsarten. In solchen Situationen spricht man von einem gemischten Vertrag.[47] Bei gemischten Verträgen werden die Leistungen verschiedener Vertragstypen kombiniert. In der Rechtsanwendung kommt es dadurch jedoch nicht zu einer Absorption in einen einzigen Vertragstypen, sondern es werden jeweils die Regeln derjenigen Vertragsart herbeigezogen, die am besten auf die in Frage stehende Leistung passt.[48] Für ein IT-Projekt bedeutet dies, dass insbesondere Fragen der Mängelgewährleistung immer unter Zuhilfenahme des auf die in Frage stehende (mangelhafte) Leistung anwendbaren Rechtes zu beantworten sind.

3.1.1.2. Erscheinungsformen der technischen Verschuldung

[44] Zum Management der technischen Verschuldung stellt sich somit die Frage, wie diese in den verschiedenen gesetzlichen Vertragsarten zutage tritt.

1. *Auftrag*: Im Rahmen eines Projektmanagements im Sinne eines Auftrages kann das Management technischer Verschuldung einen wichtigen Bestandteil darstellen. Als Ausfluss seiner Treuepflicht hat der Beauftragte auch eine Aufklärungspflicht, unter anderem über die technische Verschuldung des Projektes. Auch kann von einem treu und redlich handelnden Experten erwartet werden, die nötigen Massnahmen zur Minimierung technischer Verschuldung von sich aus zu treffen.

---

43 Straub (Informatikrecht) (Fn. 3).

44 Straub (Informatikrecht) S. 171 f. (Fn. 3).

45 Straub (Informatikrecht) S. 105 (Fn. 3).

46 Straub (Informatikrecht) S. 91 (Fn. 3).

47 Peter Higi/Anton Bühlmann, in: Zürcher Kommentar OR, 5. Aufl., Zürich, Basel, Genf, 2019, Vorbemerkungen zum 8. Titel (Art. 253–273c) N 195 ff. (zit. ZK OR-Autor).

48 ZK OR-Higi/Bühlmann (Fn. 47), Vorbemerkungen zum 8. Titel (Art. 253–273c) N 200 f.

2. *Werkvertrag*: Bei der Erstellung von Individualsoftware spielt technische Verschuldung eine sehr grosse Rolle. Der Unternehmer erstellt für den Besteller (von Grund auf) eine Software und hat damit massgeblichen und direkten Einfluss auf die Faktoren, die zu technischer Verschuldung führen. Bei der Erstellung von Individualsoftware ist aktives Management technischer Verschuldung wohl am wichtigsten, zumal hier Ereignisse, die ausserhalb des eigentlichen Codes liegen, wie Teamfluktuation und Zeitpläne des Bestellers, hauptsächlich von Relevanz sind.
3. *Kaufvertrag*: Auch in Zusammenhang mit einem Kaufvertrag über Standardsoftware kann technische Schuld auftreten. Anders als bei einem Werkvertrag treffen die Parteien erst nach Entwicklung der Software aufeinander. In der Standardsoftware bestehende und nicht beseitigte technische Schuld kann dennoch im weiteren Betrieb des Käufers ins Gewicht fallen. Da hier kein gemeinsamer Entwicklungsprozess mit der entsprechenden Möglichkeit zur Intervention besteht, wird die Beurteilung einer technischen Schuld wohl oft vor dem Hintergrund zugesicherter oder für den Gebrauch vorausgesetzter Merkmale geschehen müssen.

3.1.1.3. Haftung

[45] Ausgehend von der oben dargelegten Klassifizierung der verschiedenen vertraglichen Leistungen in die gesetzlichen Vertragskategorien lässt sich nun darlegen, in welchem Rahmen für die Akkumulation von technischer Verschuldung bereits kraft geltenden Rechts gehaftet werden könnte. Auf detaillierte Ausführungen zu gesetzlichen Haftungsnormen wird indessen bewusst verzichtet.

3.1.1.3.1. Auftrag

[46] Der Beauftragte hat seinen Auftrag sorgfältig auszuführen. Sorgfaltsmassstab ist nach Art. 398 OR die gleiche Sorgfalt, welche von einem Arbeitnehmer im Arbeitsverhältnis erwartet wird. Verlangt wird die Sorgfalt, «*welche ein gewissenhafter Beauftragter (mit der vertraglich verlangten Qualifikation) in der gleichen Lage unter Berücksichtigung des spezifischen Vertragsinhalts bei der Besorgung der ihm übertragenen Geschäfte anwendet*.»[49] Fehlendes Fachwissen oder fehlende Erfahrung (sog. Übernahmeverschulden) kann der Beauftragte nicht haftungsmildernd einwenden.[50]

[47] In Bezug auf die Haftung für technische Verschuldung in einem IT-Projekt, mit welchem ein IT-Projektmanager beauftragt wurde, ergibt sich, dass Abweichungen von den Regeln der Kunst eine Haftung zur Folge haben können. Ausgehend von obengenannter Definition technischer Verschuldung kann dafür auch eine Haftung konstruiert werden. Unterschieden werden muss hier indes zwischen bewusst, bspw. aufgrund eines engen Zeitplanes des Bestellers, aufgenommener Verschuldung und solcher, welche auf mangelnde Sorgfalt in der Projektführung zurückzuführen ist. Nur für Letztere ist eine Haftung aus unsorgfältiger Auftragsausführung denkbar, sofern die Aufklärungs- und Treuepflichten in ersterem Fall eingehalten wurden.

---

49 David Oser/Rolf H. Weber, in: Basler Kommentar Obligationenrecht I, 7. Aufl., Basel, 2020, Art. 398 N 24 (zit. BSK OR I-Autor).

50 BSK OR I-Oser/Weber (Fn.49), Art. 398 N 28.

3.1.1.3.2. Werkvertrag

[48] Damit ein Werkvertrag korrekt erfüllt ist, muss das Werk den vertraglichen Vorgaben entsprechen und die vorausgesetzten und vertraglich vereinbarten Eigenschaften aufweisen.[51] Ein Mangel am Werk ist folglich ein vertragswidriger Zustand des Werkes. Neben dem Fehlen einer hinsichtlich der Gebrauchstauglichkeit des Werkes vorausgesetzten Eigenschaft gilt auch das Fehlen einer vereinbarten Eigenschaft als Mangel.[52]

[49] Dem Besteller stehen nach Ablieferung des Werkes, abhängig von der Schwere des Mangels, gemäss Art. 368 OR vier Mängelrechte zur Verfügung: Die Wandelung, die Minderung, die Nachbesserung und ergänzend der Schadenersatz. Die Mängelrechte sind dispositiv, bei einer Freizeichnung gilt jedoch die Schranke von Art. 100 OR.[53]

[50] Betreffend die Haftung für technische Verschuldung in IT-Projekten stellt sich also wieder die Frage, in welchem Rahmen (und welcher Schwere) technische Verschuldung einen Mangel darstellt. Nicht jeder Fehler in der Informatik ist ein Mangel im Sinne des Werkvertragsrechtes[54], so ist bei komplizierten Leistungsprogrammen eine gewisse Menge an «Störungen» noch vertragsgemäss.[55] Ausschlaggebend ist, «*ob die hergestellte Software die nach dem konkreten Vertrag (z.B. in einem integrierten ‹Pflichtenheft›) vereinbarten und die ohne Vereinbarung vorausgesetzten Eigenschaften aufweist [...], oder ob sie von der vertraglichen Sollbeschaffenheit [...] abweicht*»[56].

[51] Vereinbaren die Parteien explizit die (technische) Schuldenfreiheit des Werkes bzw. die Einhaltung gewisser Qualitäts-Kennzahlen, so stellt das Vorhandensein technischer Verschuldung das Fehlen einer vereinbarten Eigenschaft und damit einen Mangel dar. Je nach vereinbartem und vorausgesetztem Gebrauch stellt eine technische Verschuldung einen schweren oder nur minder schweren Mangel dar. Zu unterscheiden ist hierbei zwischen Systemen, die als Basis für eine Weiterentwicklung dienen sollen und solchen, welche durch den Besteller nicht weiterentwickelt werden.[57] In ersterem Fall kann eine technische Verschuldung schwer ins Gewicht fallen, in letzterem Fall kann sie hingegen von geringerer Relevanz sein.[58] Abhängig davon stehen dem Besteller dann unterschiedliche Mängelrechte zu. Fälle, in denen «nur» die Beseitigung technischer Verschuldung anfällt, sofern diese noch nicht überhandgenommen hat[59], werden wohl am einfachsten mit einer unentgeltlichen Nachbesserung erledigt. Sollte der Anbieter nicht in der Lage sein, diesem Ersuchen nachzukommen, wird der Abzug am Werklohne die sinnvollste Lösung sein.

---

51 Gaudenz G. Zindel/Bertrand G. Schott, in: Basler Kommentar Obligationenrecht I, 7. Aufl., Basel, 2020, Art. 367 N 9.

52 Peter Gauch, Der Werkvertrag, 6. Aufl., Zürich, Basel, Genf, 2019, Rz. 1398; Alfred Koller, Schweizerisches Werkvertragsrecht, Zürich, St. Gallen, 2015, Rz. 514.

53 Jörg Schmid/Hubert Stöckli/Frédéric Krauskopf, OR BT, 2. Aufl., Zürich, Basel, Genf, 2016, Rz. 1729.

54 Gauch Rz. 1475 (Fn. 52).

55 Gauch Rz. 1427 (Fn. 52).

56 Gauch Rz. 1475 (Fn. 52).

57 Wobei in den wohl meisten Fällen eine gewisse Weiterentwicklung stattfinden muss, um das Produkt an neue Gegebenheiten, Softwareversionen etc. anzupassen. Massgebend ist wohl die geplante Nutzungsdauer des Werkes, bis es durch ein neues abgelöst werden soll.

58 Wobei sie aber insbesondere im Bereich steigender Wartungskosten relevant bleibt.

59 Auch hier ist die Vereinbarung von Kennzahlen zwecks Messbarkeit hilfreich.

3.1.1.3.3. Kaufvertrag

[52] Im Rahmen eines Kaufvertrages haftet der Verkäufer dem Käufer gemäss Art. 197 OR für alle zugesicherten Eigenschaften sowie dafür, dass die Sache nicht körperliche oder rechtliche Mängel hat, welche ihren Wert oder ihre Tauglichkeit zum vorausgesetzten Gebrauch aufheben oder mindern. Das Kaufvertragsrecht kommt in erster Linie bei Standardprodukten zum Einsatz. Folglich stellt sich einerseits die Frage, in welchem Rahmen technische Verschuldung als Sachmangel gilt und andererseits, ob die Schuldenfreiheit allenfalls als zugesicherte Eigenschaft gilt.

[53] Ein Sachmangel ist die «*ungünstige Abweichung der Ist-Beschaffenheit von der Soll-Beschaffenheit*»[60], welche den Wert oder die Tauglichkeit zum vorausgesetzten Gebrauch aufhebt oder erheblich mindert. Auch eine unzureichende Dokumentation kann ein Mangel sein.[61] Ausschlaggebend ist, wie sich der «vorausgesetzte Gebrauch» definiert sowie die Erheblichkeit der Minderung des Wertes oder der Tauglichkeit. Hier gilt es zu differenzieren: Handelt es sich bei der eingekauften Software um ein Standardprodukt, welches erwartungsgemäss vom Käufer nicht selbst weiterentwickelt oder angepasst werden kann, so kann eine technische Verschuldung nicht stark ins Gewicht fallen, da nicht davon ausgegangen werden muss, dass der Käufer Handlungen vornimmt, bei welchen technische Verschuldung von Relevanz ist. Hingegen bei einer Software, die als Grundlage zur Weiterentwicklung dienen soll, fällt eine bestehende technische Verschuldung im Weiterentwicklungsprozess klar ins Gewicht, weshalb der vorausgesetzte Gebrauch wohl auch eine Schuldenfreiheit oder zumindest eine tiefe technische Verschuldung umfasst. Wenn Software wie «Baumaterial» eingekauft wird, kann nach der Meinung des Autors eine (hohe) technische Verschuldung als Mangel qualifiziert werden.

[54] Wird im Rahmen der Vertragsverhandlungen zugesichert, dass die gekaufte Software frei von technischer Verschuldung ist bzw. ein bestimmter Qualitäts-Score eingehalten wurde, ist das Vorhandensein technischer Verschuldung ein Mangel im Sinne der Sachmängelgewährleistung und der Käufer kann entsprechend seine Mängelrechte geltend machen.

[55] Steht ein Mangel an der Kaufsache fest, so kann der Käufer gemäss Art. 205 OR mittels Wandelung den Kauf rückgängig machen oder durch Minderung Ersatz des Minderwerts der Sache fordern. Handelt es sich bei der Kaufsache um eine Gattungsschuld, besteht überdies die Möglichkeit einer Ersatzlieferung einer fehlerfreien Sache i.S.v. Art. 206 OR.

[56] Beim Kauf von Software handelt es sich in der Regel um ein Standard- und Serienprodukt. Damit handelt es sich grundsätzlich um eine Gattungsschuld, da es «*auf die Individualität der ausgewählten Sache nicht ankommt*»[62] und die Software ohne weitere Anpassungen verkauft wird. Weist die gekaufte Software also wider Erwarten eine (hohe) technische Verschuldung auf, besteht grundsätzlich neben Wandelung und Minderung auch die Möglichkeit einer Ersatzlieferung. Dies wird indes in den wenigsten Fällen erfolgversprechend sein, denn die Mangelhaftigkeit liegt nicht in dem spezifischen, dem Käufer gelieferten Exemplar der Software (ausser der physikalisch gelieferte Datenträger ist defekt), sondern der gesamten Software, deren Kopien verkauft werden. So wird dem Vorliegen technischer Verschuldung wohl mit einer Minderung oder in schweren Fällen einer Wandelung begegnet werden müssen.

---

60 Heinrich Honsell, in: Basler Kommentar Obligationenrecht I, 7. Aufl., Basel, 2020, (zit. BSK OR I-Autor), Art. 197 N 2.

61 Straub (Informatikrecht), S. 95 (Fn. 3).

62 BSK OR-Honsell (Fn. 60), Art. 206 N 1.

3.1.2. Vertragliche Regelung

[57] Im Schweizerischen Recht gilt der Grundsatz der Vertragsfreiheit. Die Vertragsfreiheit ist Ausfluss der Privatautonomie und umfasst unter anderem die Freiheit zu bestimmen, ob, mit wem und mit welchem Inhalt ein Vertrag geschlossen werden soll.[63] Art. 19 OR hält fest, dass der Inhalt eines Vertrages innerhalb der Schranken des Gesetzes frei festgelegt werden kann. Die oben ausgeführte Klassifikation in verschiedene Vertragstypen bleibt indes als Grundlage relevant und dient der Lückenfüllung bei offenen Fragen. Da es sich bei der Entwicklung von Individualsoftware oft um einen Werkvertrag, allenfalls mit auftragsrechtlichen Komponenten, handelt, wird sich betreffend Haftung im Folgenden daran orientiert.

[58] Ausgehend von den beschriebenen Lösungsmöglichkeiten lassen sich vertragliche Regelungen für das Management technischer Verschuldung entwickeln. Als Ausgangspunkt wird vorliegend von einer agilen Entwicklung ausgegangen, die vertraglichen Lösungen lassen sich aber, teilweise etwas angepasst, auch auf andere Entwicklungsmethoden anwenden.

3.1.2.1. Gemeinsames Bewusstsein, Definition und technische Schuld als Funktionsbestandteil

[59] Eine wichtige Grundlage für das Management technischer Verschuldung ist, wie mehrfach betont, das Bewusstsein für die Problematik. Auch für vertragliche Lösungsansätze zur Vermeidung technischer Verschuldung sind ein gegenseitiges Verständnis und eine einheitliche Definition die Grundlage. Folglich sollten sich die Parteien in einem ersten Schritt auf eine einheitliche Definition einigen. Durch die Auseinandersetzung mit den für die jeweilige Partei wichtigen Punkte bzw. Problematik kann zugleich das gegenseitige Bewusstsein gestärkt werden. In gewissem Umfang sollte der Anbieter der IT-Dienstleistung den Besteller (häufig ein IT-Laie) an die Hand nehmen und auf die Problematik der technischen Verschuldung aufmerksam machen.

[60] Wenn die Problematik der technischen Verschuldung beiden Parteien bekannt und bewusst ist, kann sie behandelt werden, als wäre sie Teil des gewöhnlichen Funktionsumfanges. Eine saubere, vollständige und schuldenfreie Entwicklung dauert unter Umständen länger und kostet mehr, als wenn Abkürzungen genommen werden (und dadurch technische Verschuldung aufgenommen wird). Genau wie es zur gewöhnlichen Entwicklung gehört, Zeit und Ressourcen für die Entwicklung einer neuen Funktion aufzuwenden, so sollte als ebenso normal die Aufarbeitung von technischer Verschuldung betrachtet werden. Die Parteien müssen sich einigen, wie viel technische Verschuldung in welchem Zeitraum als «zu erwarten» anzusehen ist. Einer darüber hinausgehenden technischen Verschuldung ist mit geeigneten Instrumenten zu begegnen und das Risiko zwischen den Parteien aufzuteilen.[64] Ziel ist es, den Entwicklern ein Umfeld zu bieten, in dem technische Verschuldung möglichst vermieden und nötigenfalls aufgearbeitet werden kann, sowie dem Besteller die Sicherheit zu geben, dass sein IT-Projekt so schuldfrei wie möglich entwickelt wird.

63 Ingeborg Schwenzer, Schweizerisches Obligationenrecht, Allgemeiner Teil, 7. Aufl., Bern, 2016, Rz. 25.01.

64 Eine in diesem Fall «übermässige» technische Verschuldung ist wohl vom Anbieter im Rahmen eines «Aufräumsprints» auf seine Kosten zu tragen.

3.1.2.2. Kennzahlen und zugesicherte Eigenschaften

[61] Damit die technische Verschuldung in einem Projekt bestimmt werden kann, muss sie messbar sein. Dazu können sich die Parteien auf Kennzahlen und Messmethoden einigen, um eine Überprüfung durch die Parteien selbst und allenfalls sachverständige Dritte zu ermöglichen. Wenngleich damit nicht alle der vielen Facetten technischer Verschuldung abgedeckt werden können, wird eine gewisse Sicherheit und nicht zuletzt ein Beweismittel in einem allfälligen Prozess geschaffen. Insbesondere die Qualität des Codes und der Dokumentation können so im Rahmen eines Audits beurteilt werden.

[62] Eine erste Möglichkeit besteht in der Vereinbarung der Nutzung eines bestimmten Tools zur Messung technischer Verschuldung wie bspw. dem meistverbreiteten Tool SonarQube[65]. Dieses misst und beurteilt eine Vielzahl von Aspekten wie die Codekomplexität, die Zuverlässigkeit und insbesondere die Wartbarkeit[66], zu welcher die technische Verschuldung gehört.[67] SonarQube vergibt dann unter anderem für die «Technical Debt ratio» einen Score von A bis E. Die Parteien können somit vereinbaren, welche Scores für eine Abnahme des Projektes verlangt werden. So kann bei jeder Änderung des Codes eine Beurteilung der Komplexitätserhöhung erfolgen und diese, wenn nötig, aktiv behandelt werden. Zeigt die Entwicklung der Kennzahlen ein negatives Bild, kann die Ergreifung technischer Massnahmen, wie bspw. ein Refactoring, vereinbart werden.

[63] Es handelt sich bei den vereinbarten Kennzahlen um ein Qualitätsmerkmal, welches die Parteien vertraglich als vorausgesetzte Eigenschaft der geschuldeten Sache vereinbaren können. Eine Abweichung davon würde einen Mangel darstellen und dem Besteller die entsprechenden Mängelrechte zur Verfügung stehen. Um die Feststellung solcher Mängel noch vor einer allfälligen Klage zu erleichtern, können die Parteien vereinbaren, einen externen Code-Audit durchführen zu lassen. Dessen Kosten können, bei entsprechender Vereinbarung, der «unterliegenden» Partei überbunden werden. Sollte noch ein stärkeres Mittel gewünscht sein, bietet es sich an, die Beurteilung der technischen Schuld im Rahmen eines Schiedsgutachtens i.S.v. Art. 189 ZPO durchführen zu lassen, wobei den Parteien bewusst sein sollte, dass hierfür beträchtliche Kosten entstehen können.

[64] Eine weitere Möglichkeit besteht in der Vereinbarung einer Grenzkostenbetrachtung im Sinne der MCO. Die Parteien vereinbaren in diesem Fall, neben der Kostenschätzung für eine Story auch eine (technische) Komplexitätsschätzung vorzunehmen. Für diese beiden Werte wird dann ein Verhältnis vereinbart, welches in einem bestimmten Zeitraum gewahrt werden muss. Entwickelt sich das Verhältnis dahin, dass für gleich komplexe Inkremente höhere Kosten anfallen, und dies nicht innert einer festgelegten Frist behoben wird, fällt bspw. ein Wartungssprint auf Kosten des Anbieters an.

---

65 Vgl. Avgeriou (Fn. 16) für eine Übersicht.

66 Unter anderem anhand von Code smells, siehe Fowler (Fn. 33).

67 Es gibt gleich mehrere Metriken, die sich spezifisch mit der technischen Verschuldung auseinandersetzen, vgl. die SonarQube Dokumentation, https://docs.sonarsource.com/sonarqube/latest/user-guide/metric-definitions/, Stand 25. Oktober 2023.

3.1.2.3. Team-Mix

[65] Die Zusammensetzung insbesondere des Entwicklerteams kann entscheidend für den Erfolg eines IT-Projektes sein. Wie dargelegt kann sowohl die Seniorität der Entwickler sowie das Ausscheiden eines (für die Entwicklung massgebenden) Entwicklers während des laufenden Entwicklungsprozesses zu einer grossen technischen Verschuldung führen.

[66] Da sich die Geschäftsleitung des Entwicklerteams wohl kaum Vorgaben zur internen Personalpolitik vom Besteller machen lässt, können die Parteien bspw. eine gewisse Anzahl an Senior-Entwicklern[68], die an dem Projekt arbeiten, bestimmen. So können bspw. in einem Rahmenvertrag die Rollen im Projektteam definiert werden. Im Falle des Ausstieges eines Senior-Entwicklers aus dem Projekt müsste dann innert einer bestimmten Frist ein Ersatz für die Arbeit an dem Projekt bereitgestellt werden, wobei es sich anbietet, bereits auf dem Projekt arbeitende Junior-Entwickler entsprechend weiterzubilden. Gleichzeitig lässt sich in diesem Rahmen die minimale nötige und die maximal zulässige Teamgrösse festlegen.

[67] Der Rahmenvertrag eines Projektes entspricht oft einem Auftragsverhältnis, allenfalls mit Elementen anderer Vertragstypen. Der Beauftragte hat den Auftrag gemäss Art. 298 OR persönlich auszuführen, grundsätzlich jedoch mit der Möglichkeit, Hilfspersonen beizuziehen. Da der Beizug von Hilfspersonen vertraglich ausgeschlossen werden kann,[69] muss es nach der Meinung des Autors auch möglich sein, die Zulässigkeit von Hilfspersonen auf bestimmte Personen oder Rollen zu beschränken. Zudem sollte sich der Anbieter (und Beauftragte) auch verpflichten können, eine bestimmte Zusammensetzung von Hilfspersonen verschiedener Qualifikationen im Entwicklerteam sicherzustellen. Eine Abweichung einer solchen Regelung wäre eine Nicht- oder Schlechterfüllung des Vertrages, welcher über Art. 97 OR abgewickelt werden könnte.

3.1.2.4. Vergütungsmodelle und Incentivierung

[68] Für die Vergütung eines agil entwickelten Projektes gibt es verschiedene Vergütungsmodelle mit unterschiedlicher Risikoverteilung.[70] Neben einer Vergütung nach Aufwand haben sich Modelle wie der agile Festpreis, Stückpreise, Change for Free und weitere[71] entwickelt, welche der agilen Entwicklung besser gerecht werden. Den Parteien stehen dementsprechend viele Möglichkeiten offen, um eine Vergütung zu vereinbaren, die ihren Vorstellungen einer optimalen Entwicklung am besten gerecht werden. Über die Vergütung kann viel gesteuert werden, wenn bspw. erst nach mangelloser Abnahme der volle Betrag ausbezahlt wird, wodurch die Risikoverteilung zwischen den Parteien etwas ausgeglichen werden kann. Auch kommt hier zum Ausdruck, wie viel technische Verschuldung die Parteien noch als «im Rahmen der zu erwartenden Entwicklung abzuarbeiten» betrachten.

[69] Vor dem Hintergrund technischer Verschuldung sollte darauf geachtet werden, dass das Risiko für die Entstehung technischer Verschuldung fair verteilt wird. Eine reine Vergütung nach Aufwand bspw. kann zu einer Verzerrung der Projektkosten führen, da der Besteller das gesam-

68 Die genaue Definition dessen, was für das jeweilige Projekt ein «Senior»-Entwickler ist, sollte zudem vertraglich festgehalten werden.

69 Schmid/Stöckli/Krauskopf Rz. 1906 (Fn. 53).

70 Vgl. dazu unter anderem Wolfgang Straub, Verträge für agil geführte Projekte, in: Jusletter 21. Dezember 2015, Rz. 44.

71 Siehe Übersicht in Straub (Agil geführte Projekte), Rz. 44 ff. (Fn. 70).

te Risiko trägt, auch wenn ein gewisser Aufwand für die Beseitigung technischer Verschuldung anfällt, die eigentlich der Anbieter zu tragen hätte. Nach der Meinung des Autors bieten sich insbesondere der agile Festpreis[72] und/oder eine Vergütung von Stückpreisen[73] an. Letztere funktioniert gut mit der Behandlung von technischer Verschuldung als Funktionsumfang: Durch festgelegte Stückpreise pro Story kann eine Bereinigung technischer Verschuldung als gewöhnliche Story abgearbeitet und die geleistete Arbeit entsprechend in Rechnung gestellt werden.[74]

[70] Eine weitere Möglichkeit, die zuvor beschriebenen Lösungsansätze in der Praxis umzusetzen, ist eine Incentivierung, also das Schaffen von Anreizen. Wie besprochen, kann technische Verschuldung mit entsprechendem Bewusstsein und achtsamer Entwicklung tief gehalten werden. Um technische Verschuldung zu minimieren, lassen sich Anreize vereinbaren, die eine schuldenfreie Entwicklung belohnen (bspw. mit Criteria of Satifaction in der Definiton of Done).

[71] Anknüpfend an die obengenannten Kennzahlen und als Gegenstück zur Vereinbarung von Eskalationsstufen bei negativer Entwicklung kann also vereinbart werden, dass bei konstant guter Entwicklung (die entsprechenden Scores bleiben in einem bestimmten Zeitraum konstant gut) eine Belohnung fällig wird.

[72] In einem lange dauernden Entwicklungsprozess gewinnen insbesondere die Dokumentation und das Know-how des Teams an Bedeutung. Die Aufrechterhaltung dieses Know-hows kann für den Anbieter eine kostspielige Angelegenheit werden, da sie Zeit und Ressourcen bindet, ohne dass ein neues Produkt oder Feature geliefert werden kann. Um dem entgegenzuwirken, kann auch hier mit Anreizen gearbeitet werden, so könnte bspw. eine Pauschale zur Aufrechterhaltung des Know-hows vom Besteller an den Softwareentwickler entrichtet werden. So kann sich der Besteller darauf verlassen, dass das geschaffene Wissen erhalten bleibt und der Anbieter kommt nicht in Versuchung, aus Kostengründen die Wissenserhaltung zu vernachlässigen. Eine Überprüfung ist für den Besteller hingegen anspruchsvoll.

[73] Schliesslich lassen sich die oben betreffend Team-Mix gemachten Lösungsvorschläge mit Anreizen begünstigen. Besonders die laufende Weiterbildung bereits auf dem Projekt arbeitender Junior-Entwickler kann für den Besteller von Vorteil sein, da viel Know-how behalten werden kann und bei einem allfälligen Ausscheiden eines Senior-Entwicklers keine neue, externe Person in das Projekt eingearbeitet werden muss.

3.1.2.5. Konfliktlösung

3.1.2.5.1. Eskalationsstufen und ADR

[74] Generell, aber auch in Bezug auf das Management technischer Verschuldung ist es wichtig, Konfliktlösungsmechanismen zu vereinbaren. Dies nicht aus Misstrauen den Entwicklungspartnern gegenüber, sondern zur Schaffung von geregelten Vorgehensweisen und um möglicherweise auftretende Meinungsverschiedenheiten möglichst einvernehmlich zu lösen. Dies auch vor dem Hintergrund, dass Probleme in IT-Projekten durch gerichtliche Schritte nur schwer zu lö-

72 Straub (Agil geführte Projekte), Rz. 47 ff. (Fn. 70).

73 Straub (Agil geführte Projekte), Rz. 54 (Fn. 70).

74 Dies sollte allerdings an gewisse Qualitätskriterien und Zeiträume gebunden werden. Mit anderen Worten müssen sich die Parteien einigen, wie viel technische Verschuldung in welchem Zeitraum als zu erwarten anzusehen ist. Darüber hinausgehende technische Verschuldung aufgrund bspw. mangelhafter Arbeit kann nicht mit diesem Mechanismus begegnet werden.

sen sind,[75] das Projekt während der Dauer eines Prozesses «brach» liegt[76] und die «Behandlung» technischer Verschuldung mit ihrer Zunahme zunehmend teurer wird («Zinsen und Zinseszinsen»). Eine schnelle Streitbeilegung dürfte im Interesse beider Parteien liegen.

[75] In erster Linie kommt hierzu eine Parteiverhandlung in Frage, also das Führen von Verhandlungen. Vertraglich ist zu vereinbaren, wer mit wem zu welchem Zeitpunkt und wie lange zu verhandeln hat (zuerst im Projektteam, dann im Führungsausschuss etc.). Den Parteien soll damit ermöglicht werden, stets zu wissen, wer erste Ansprechperson für Konflikte ist (bspw. mit einem Single Point of Contact) und Meinungsverschiedenheiten möglichst schlank zu bereinigen.[77] Auch kann vereinbart werden, in welchem Zeitpunkt ein Audit durch einen Dritten erfolgen soll.

[76] Sollten die Parteien nicht zu einer Einigung gelangen, gibt es weitere ADR-Verfahren, die einem (langwierigen) Gerichtsverfahren vorausgehen oder dieses ersetzen können:

[77] Bei einer Mediation versuchen die Parteien unter Beizug einer unabhängigen Drittperson ohne Entscheidbefugnis, ihre Streitigkeit unter Berücksichtigung aller Interessen einvernehmlich zu lösen. Sie ist auch in Art. 213 ff. ZPO festgehalten. Ziel ist «*das Schaffen von Rahmenbedingungen, innerhalb derer die Konfliktparteien durch wechselseitiges Ausloten von Verhandlungsspielräumen eine Lösung suchen und dabei von einer Mediatorin oder einem Mediator unterstützt werden*»[78]. Da der Mediator über keine Entscheidbefugnis verfügt, liegt der Fokus nicht auf dem Analysieren oder Aufrechnen vergangener Fehler, sondern die Parteien entscheiden selbst als Experten für ihre Auseinandersetzung über eine geeignete Lösung. Unter anderem sollen die Bedürfnisse der Parteien in der Gegenwart und Zukunft berücksichtigt werden und eine beidseitige Gewinnmaximierung angestrebt werden.[79] Die Vorteile der Mediation liegen insbesondere in der hohen Kontrolle und Gestaltungsmöglichkeit der Parteien, der Schnelligkeit und der Vertraulichkeit.[80] Da die Mediation von der Privatautonomie geprägt ist, ist es wichtig, dass die Parteien die Grundzüge der Mediation zuvor festlegen. Insbesondere, ob sie zwingend vor einer Klage durchgeführt werden muss, wie sie eingeleitet und abgeschlossen wird und wann sie allenfalls als gescheitert zu betrachten ist. Ob eine Mediationsabrede gerichtlich durchsetzbar ist, ist hingegen aufgrund der Freiwilligkeit des Verfahrens in der Lehre und Rechtsprechung umstritten.[81]

[78] Steht eine tatsächliche Fragestellung zur Diskussion, können die Parteien über die streitige Tatsache ein Schiedsgutachten i.S.v. Art. 189 ZPO einholen. Der Schiedsgutachter entscheidet dann verbindlich über die streitige Tatsache. Anders als bei einem Parteigutachten ist sodann auch der staatliche Richter gemäss Art. 189 Abs. 3 ZPO an das Schiedsgutachten gebunden. Für die Parteien ist dies in erster Linie ein Mittel zur Prozessvermeidung.[82] Der Schiedsgutachter kann überdies von den Parteien selbst bestimmt werden, wobei auch juristische Personen in Fra-

75 Wolfgang Straub, Legal Engineering, in: Jusletter 7. Oktober 2019, 2019, Rz. 75.

76 Sury S. 151 (Fn. 41).

77 Vgl. Übersicht in Straub (Legal Engineering), Rz. 75 ff. (Fn. 75).

78 Jonas Fischer/Anne Mirjam Schneuwly, ADR Alternative Dispute Resolution, Zürich, St. Gallen, Baden-Baden, 2021, S. 222 f.

79 Fischer/Schneuwly, S. 222–224 (Fn. 78).

80 Fischer/Schneuwly, S. 247 (Fn. 78).

81 Fischer/Schneuwly, S. 358 f. (Fn. 78).

82 Luca Angstmann, Das Schiedsgutachten im schweizerischen Recht (Diss.), in: Zürcher Studien zum Privatrecht, Band 303, Rz. 19.

ge kommen, was es den Parteien erlaubt, eine Fachstelle oder einen Sachverständigen mit der nötigen fachlichen Expertise für die Beurteilung ihrer Streitigkeit zu bestimmen.[83]

[79] Schliesslich kann die Beurteilung der gesamten Streitigkeit den staatlichen Gerichten entzogen und mittels Schiedsklausel einem Schiedsgericht i.S.v. Art. 353 ff. ZPO zugewiesen werden. Die Parteien ermächtigen damit ein von ihnen bestimmtes Schiedsgericht, über ihre Streitigkeit eine verbindliche und vollstreckbare Entscheidung zu treffen.[84] Dies hat wiederum den Vorteil, dass die Parteien hohe Kontrolle über das Verfahren haben und die Schiedsrichter oder Schiedsinstitution selbst bestimmen können, und so die nötige fachliche Expertise der beurteilenden Personen sicherstellen können.[85] Eine gültig vereinbarte Schiedsklausel ist verbindlich, ein dennoch angerufenes staatliches Gericht müsste seine Zuständigkeit nach Art. 61 ZPO ablehnen.

3.1.2.5.2. Escrow

[80] Im Rahmen eines Escrow-Agreements hinterlegt der Softwareentwickler den Sourcecode und weitere Entwicklungsunterlagen bei einem Dritten «Escrow-Agenten». Rechtlich kann dies als Zwei- oder Dreiparteienverhältnis ausgestaltet werden, wobei für die vorliegende Arbeit nur das Dreiparteienverhältnis mit Softwareentwickler, Escrow-Agent und Besteller von Relevanz ist.

[81] Im Rahmen eines solchen Dreiparteienvertrages vereinbaren die Parteien die Hinterlegung der Entwicklungsunterlagen bei einer bestimmten Stelle und legen die Bedingungen fest, unter denen die Unterlagen dem Besteller herausgegeben werden sollen. Dies erlaubt dem Besteller, eigenständig an die Daten zu gelangen, wenn bestimmte Bedingungen erfüllt sind.[86] Für zusätzlichen Schutz gegen einen Konkurs des Anbieters muss der Anbieter dem Escrow-Agenten das fiduziarische Eigentum an den Entwicklungsunterlagen übertragen.[87] Als hauptsächliche Herausgabegründe nennt Fröhlich-Bleuler bspw. den Gesellschafterwechsel, den Konkurs des Anbieters oder die Unfähigkeit des Anbieters, das Projekt fertigzustellen oder Mängel zu beheben, wobei in diesen Fällen wohl ein rechtskräftiges Urteils notwendig ist.[88]

[82] Es ist zu beachten, dass auch hier Meinungsverschiedenheiten auftreten können, für welche auf die soeben beschriebenen Instrumente, insbesondere die Mediation, zurückgegriffen werden sollte, damit nicht auch hierüber Prozess geführt werden muss.[89]

[83] Der Hauptzweck der Hinterlegung des Sourcecodes besteht darin, eine gewisse Unabhängigkeit des Bestellers vom Softwareentwickler sicherzustellen und eine zügige Herausgabe zu ermöglichen. Dies auch, damit der Zugriff auf die Daten für den Besteller nicht während eines allfälligen Gerichtsprozesses blockiert ist.

---

83 Angstmann, Rz. 194 ff. (Fn. 82).

84 Fischer/Schneuwly, S. 425 f. (Fn. 78).

85 Mittlerweile bestehen auf IT Spezialisierte Schiedsinstitutionen wie die ITDR (https://www.itdr.ch/) oder die SGOA (https://sgoa.eu/).

86 Sury S. 149 (Fn. 41).

87 Gianni Fröhlich-Bleuler, Softwareverträge, 2. Aufl., Bern, 2014, Rz. 1419.

88 Fröhlich-Bleuler, Rz. 1422 ff. (Fn. 87).

89 Sury, S. 149 (Fn. 41).

3.2. Musterklauseln

[84] Den folgenden Musterklauseln liegt jeweils die Annahme zugrunde, dass die Parteien einen sonst vollständigen Vertrag mit Präambel, Zielsetzung etc. vereinbaren. Die Klauseln verstehen sich als beispielhafte Darstellung der oben beschriebenen Konzepte und müssten jeweils auf den Einzelfall angepasst werden. Platzhalter sind in kursiver, *blauer Schrift* gehalten.

3.2.1. Grundlagen

3.2.1.1. Technische Verschuldung – Definition

[85] Die Parteien sind sich der Problematik der technischen Verschuldung bewusst. Sie anerkennen, dass es sich bei technischer Verschuldung um eine Abweichung von der optimalen Codeentwicklung, entstanden durch (bewusste) Abkürzungen in der Entwicklung, handelt, und dass sie in erster Linie im Rahmen von mangelhaftem Code und unzureichender Dokumentation zutage tritt. Die Parteien bemühen sich gemeinsam, ein Umfeld zu schaffen, in welchem eine technische Verschulung minimiert wird.

[86] Die Parteien anerkennen insbesondere, dass die Haupttreiber technischer Verschuldung ein zu hoher Zeitdruck und unrealistische Terminpläne sind. Jede Partei hat die Obliegenheit, die jeweils andere Partei auf ein Verhalten hinzuweisen, das zu einer höheren technischen Verschuldung führt. Die bewusste Inkaufnahme technischer Verschuldung durch den Besteller führt nicht zu einer Haftung des Anbieters.

3.2.1.2. Technische Verschuldung – Bewusstsein und Bereinigungssprints

[87] Die Parteien anerkennen, dass es im Rahmen des Entwicklungsprozesses zu technischer Verschuldung kommen kann. Um diese möglichst tief zu halten, vereinbaren die Parteien, jeden *5. Sprint*[90] oder spätestens vor jedem Release, einen Sprint zur Beseitigung technischer Verschuldung, insbesondere zur Bereinigung des Programmcodes und der Dokumentation, aufzuwenden. Sollte ein Bereinigungssprint nach der Beurteilung der Anbieterin nicht nötig sein, da der Code keine technische Verschuldung aufweist, verpflichtet sich die Bestellerin zur Leistung einer Prämie gemäss *Klausel Y*[91]. Die Anbieterin haftet in diesem Fall für die Folgen einer wider Erwarten vorhandenen technischen Verschuldung. Der Besteller hat das Recht, auf die Durchführung eines Bereinigungssprints zu bestehen, wenn die vereinbarten Scores in *Klausel X* nicht eingehalten werden.[92]

90 Die Häufigkeit ist projektbezogen und im Einzelfall zu bestimmen.

91 Sinnvoll wäre nach der Meinung des Autors ein entsprechender Absatz in der Klausel, die die Vergütung festlegt.

92 Ziel dieser Klausel ist es, eine regelmässige Bereinigung des Codes und der Dokumentation fest in das Projekt aufzunehmen, damit eine allenfalls angehäufte technische Verschuldung regelmässig beseitigt wird. Gleichzeitig soll eine möglichst schuldenfreie Entwicklung gefördert werden, indem der Anbieterin eine Prämie ausgerichtet wird, sollte keine Bereinigung nötig sein.

3.2.1.3. Entwicklungsstandards

[88] Der Anbieter verpflichtet sich, die Entwicklung nach den *höchsten Branchenstandards*[93] durchzuführen. Insbesondere verpflichtet er sich dem bewussten und aktiven Management von technischer Verschuldung. Er sorgt insbesondere für eine genügende Testautomatisierung und ergreift alle nötigen Massnahmen zur Minimierung technischer Verschuldung. Die Beurteilung der Einhaltung geltender Branchenstandards erfolgt auf Verlangen einer Partei mittels Code-Audits durch einen Dritten gemäss *Klausel X*. Im Streitfall wird die Beurteilung durch ein Schiedsgutachten vorgenommen, wobei die das Gutachten verlangende Partei dessen Kosten zu tragen hat.[94] Die Bestimmung des Schiedsgutachters erfolgt, wenn sich die Parteien nicht innert *10 Tagen* einigen können, durch den Obergerichtspräsidenten des Kantons X.[95]

3.2.2. Vergütung

3.2.2.1. Vergütungsmodell[96]

[89] Die Parteien vereinbaren eine Vergütung *nach Story Punkten*. Der Stückpreis pro Story Punkt wird auf *CHF xx.xx* festgelegt. Für die regelmässige Bereinigung technischer Verschuldung gemäss *Klausel X* wird ein Aufwand von *1* Story Punkt(en) festgelegt.[97]

alternativ

[90] Die Parteien vereinbaren eine Vergütung *mittels agilen Festpreises*. [Ausführungen zum Ablauf].[98] Für die regelmässige Bereinigung technischer Verschuldung gemäss *Klausel X* wird ein eine Vergütung von *CHF xx.xx* vereinbart.

3.2.2.2. Prämien[99]

[91] Die Bestellerin verpflichtet sich zur Entrichtung der folgenden Prämien:

0.5 Story Punkte/CHF x.xx, wenn aufgrund fehlender technischer Verschuldung kein Bereinigungssprint gemäss *Klausel X* nötig ist.[100]

---

93 Allenfalls anwendbare bestehende Standards oder Normen, sonst auch individuell vereinbarte Werte und Pflichten.

94 Dieser letzte Satz ist optional und über dessen Verwendung im Einzelfall zu bestimmen. Die Kostentragung ist eine Anlehnung an die Michigan Mediation, welche Sanktionen vorsieht, sollte ein (schieds-)gerichtliches Urteil nicht wesentlich von der im Mediationsverfahren vorgeschlagenen Lösung abweichen. Bei einem vollständigen Obsiegen der das Gutachten verlangenden Partei sind die Kosten natürlich zur Schadensposition hinzuzuschlagen.

95 Es lohnt sich, im Voraus den Schiedsgutachter oder die den Schiedsgutachter bestimmende Person festzulegen, damit es nicht zu unnötigen Zeitverlusten kommt.

96 Diese Klausel dient der Vollständigkeit und ist rein exemplarisch und nicht vollständig. Die jeweilige Vergütung ist für den jeweiligen Einzelfall individuell zu erarbeiten und vereinbaren.

97 Ziel dieser Regelung ist es, Bereinigungssprints wie die Entwicklung neuer Funktionen zu behandeln und zu verrechnen. Gleichzeitig sollte der Bereinigungsaufwand beim Softwareentwickler durch saubere Entwicklung möglichst tief gehalten werden, weshalb für die Bereinigung eine festgelegte statt einer variablen Entschädigung vereinbart wird.

98 Vgl. Straub (Agil geführte Projekte), Rz. 47 ff. (Fn. 70) und Opelt/Gloger/Pfarl/Mittermayr, Der agile Festpreis: Leitfaden für wirklich erfolgreiche IT-Projekt-Verträge, 2. Auflage, München 2014, S. 6 ff.

99 Auch diese Klausel ist unvollständig und dient der Unterstreichung der für das Management technischer Verschuldung relevanten Regelungen.

100 Diese Prämie soll den Softwareentwickler dazu motivieren, die technische Verschuldung in der Entwicklung möglichst tief zu halten. Ein regelmässiges Aufräumen wird von den Parteien vereinbart und als «normal» betrachtet. Leistet der Softwareentwickler nun besonders gute und damit schuldenfreie Arbeit, kann der Bereinigungssprint

3.2.3. Kennzahlen

3.2.3.1. Tools

[92] Die Parteien kommen überein, dass das Softwareanalysetool *X* (bspw. SonarQube) als massgebende Metrik für die technische Verschuldung des Projektes verwendet wird. Der Anbieter orientiert den Besteller am Ende jedes Sprints schriftlich über die aktuellen Scores und welche Handlungen er, wenn nötig, zu deren Verbesserung vorsieht. Die Parteien vereinbaren die folgenden Scores in den folgenden Kategorien als Zielwerte:

Kategorie 1: A/Kategorie 2: B/Kategorie 3: B/Kategorie 4: A.[101]

[93] Eine Abweichung von diesen Scores nach unten wird im Rahmen des nächsten Bereinigungssprints korrigiert. Sollten die Scores nach der Bereinigung noch immer eine Abweichung nach unten aufweisen, so ist der Anbieter verpflichtet, auf eigene Kosten nachzubessern.

3.2.3.2. Grenzkostenbetrachtung MCO[102]

[94] Als weitere Metrik zu Messung technischer Verschuldung vereinbaren die Parteien die Betrachtung der Grenzkosten im Sinne der sog. Marginal Cost of Ownership. Dazu vereinbaren sie den folgenden Mechanismus:

[95] Unabhängig von der Kostenschätzung für jede Story (Anzahl an Story Punkten) führt der Anbieter eine Beurteilung der technischen Komplexität jeder Story durch. Die Beurteilung erfolgt durch eine technisch qualifizierte Fachperson des Anbieters und wird getrennt von der Kostenschätzung auf einer Skala von *A (wenig komplex) bis E (sehr komplex)* beurteilt. Diese Beurteilung wird dem Besteller schriftlich kommuniziert. Der Anbieter legt dem Besteller Rechenschaft darüber ab, wie sich die Beurteilung der technischen Komplexität im Verhältnis zum Kostenaufwand im Laufe des Projektes entwickelt.

[96] Sollten die Kosten für eine Story der gleichen Komplexitätsklasse seit Beginn des Projektes[103], um mehr als *X*%[104], (heruntergerechnet auf den Umfang der Story[105] bspw. pro Story Punkt, nach Bereinigung der Teuerung) gestiegen sein, anerkennen die Parteien dies als Symptom technischer Verschuldung und der Anbieter verpflichtet sich, auf seine Kosten die nötigen Handlungen vorzunehmen, um die technische Verschuldung zu reduzieren.

---

ausbleiben und die Zeit für die Entwicklung neuer Funktionen verwendet werden. Der Besteller spart sich damit, die vollen Kosten für einen Bereinigungssprint zu bezahlen, während der Softwareentwickler zusätzlich zu den erarbeiteten Story Punkten eine Prämie erhält.

101 Die relevanten Kriterien sind stark projekt- und toolabhängig und werden in Zusammenarbeit beider Parteien erarbeitet werden müssen.

102 Diese Regelung funktioniert besonders gut zusammen mit der Vereinbarung eines agilen Festpreises i.S.v. Straub (Agil geführte Projekte), Rz. 47 ff. (Fn. 70), da in diesem Rahmen bereits ein möglichst repräsentatives Epic ausgesucht wird, wodurch Vergleichswerte, auch betreffend die unterschiedlichen Komplexitätsgrade von Funktionen, bestehen.

103 Oder einem anderen sinnvollen Zeitpunkt, der Zeitraum ist projektabhängig und im Einzelfall zu beurteilen.

104 Der genaue Prozentsatz ist projektabhängig und im Einzelfall zu beurteilen.

105 Dieser Zusatz soll dafür sorgen, dass die Kosten verschieden umfangreicher aber gleich komplexer Storys miteinander verglichen werden kann. Eine Story der Komplexität A kann bei gleicher Komplexität umfangreicher sein und dadurch mehr kosten, weshalb die Kosten pro Storypoint (oder einer anderen Einheit) ausgewiesen werden müssen.

3.2.3.3. Code Audit

[97] Die Parteien vereinbaren, den Softwarecode regelmässig einer Überprüfung auf Qualitätsmerkmale und insbesondere technische Verschuldung zu unterziehen (sog. Audits). Die Parteien bringen der jeweils anderen Partei die Ergebnisse des von ihnen in Auftrag gegebenen Audits zur Kenntnis. Der Anbieter verpflichtet sich, auf Verlangen des Bestellers einem sachverständigen Dritten zur Durchführung eines Audits Zugang zum Sourcecode und allen weiteren notwendigen Unterlagen zu geben. Die Kosten für den Audit eines Dritten trägt diejenige Partei, die diesen verlangt. Zeigt der Audit Mängel an der Qualität oder technische Verschuldung auf, so trägt der Anbieter die Kosten für deren Beseitigung sowie den Audit, es sei denn, es handelt sich um durch den Besteller bewusst aufgenommene technische Verschuldung.[106]

3.2.4. Entwicklungsteam – Zusammensetzung und Qualifikation

3.2.4.1. Grösse und Qualifikation

[98] Die Parteien vereinbaren, die Grösse des Entwicklerteams auf *X* (bspw. 10) Personen zu beschränken. Als Ansprechperson wird dem Besteller eine Person als Single Point of Contact, sowie eine Stellvertretung, genannt.

[99] Das Entwicklerteam hat, neben dem Projektleiter des Anbieters, aus mindestens *X* (bspw. 4) Senior-Entwicklern zu bestehen. Als Senior-Entwickler gilt vorliegend ein Mitarbeiter, welcher *X Qualifikation* aufweist und mindestens *Y Jahre* Berufserfahrung aufweist. Der Anbieter legt dem Besteller regelmässig Rechenschaft über Veränderungen in der Personalzusammensetzung des eingesetzten Teams ab. Er ist verpflichtet, eine Unterschreitung der vereinbarten Anzahl von Senior-Entwicklern innert *Frist* zu berichtigen.

3.2.4.2. Weiterbildung

[100] Der Anbieter ist für die interne Weiterbildung des Entwicklerteams besorgt. Er stellt die Weiterbildung der Junior-Entwickler, die zum Erhalt der vereinbarten Teambesetzung notwendig ist, sicher.

[101] Für den Erhalt des Know-hows des Entwicklerteams und die interne Weiterbildung entrichtet der Besteller eine Pauschale von *CHF X* pro Betriebsjahr der Software, zahlbar im Voraus. Der Anbieter verpflichtet sich, eine aktuelle und vollständige Dokumentation aller entwicklungsrelevanten Dokumente zu führen.

3.2.5. Konflikteskalationsverfahren

3.2.5.1. Eskalationsstufen

[102] Die Parteien verpflichten sich, im Falle von Meinungsverschiedenheiten oder Streitigkeiten, innert *10 Tagen* Verhandlungen zur gemeinsamen Lösungsfindung in den folgenden Instanzen aufzunehmen, bevor rechtliche Schritte (insbesondere die Mediation gemäss Mediationsklausel) ergriffen werden:

[106] Eventuell kann die Wartungspflicht auf schwerwiegende Mängel begrenzt und der Auditor verpflichtet werden, sich zum Schweregrad des Mangels und zur Kostenverteilung zu äussern.

[103] Verhandlungsinstanzen:

1. Der zuständige Teamleiter des Entwicklerteams mit dem zuständigen Projektleiter des Bestellers. Bei Scheitern der Verhandlung oder fehlender Entscheidungskompetenz
2. Der Account Manager des Softwareherstellers mit dem Account Manager des Bestellers. Bei erneutem Scheitern der Verhandlungen oder fehlender Entscheidungskompetenz
3. Der CEO der Softwareherstellerin mit dem CEO des Bestellers. Die genaue Benennung der zuständigen Personen erfolgt in *Anhang 1*.

3.2.5.2. Escrow

[104] Die Parteien vereinbaren die Hinterlegung des Quellcodes und aller weiteren Herstellungsunterlagen bei *Escrow-Anbieter* mit separater Vereinbarung. Der Anbieter verpflichtet sich, dem *Escrow-Anbieter* das fiduziarische Eigentum an den hinterlegten Sachen und Daten zu übertragen. Die Hinterlegung dauert ab Beginn der Entwicklung bis zur erfolgreichen Abnahme der letzten Arbeiten.

[105] Die Herausgabe der hinterlegten Dokumente erfolgt ausschliesslich in den folgenden Fällen:[107]

1. Gesellschafterwechsel von *Person X* beim Anbieter;
2. Konkurs oder Insolvenzverfahren des Anbieters;
3. Einleitung eines Gerichts- oder Schiedsverfahrens nach gescheiterter Mediation, wobei die die Herausgabe verlangende Partei bei Unterliegen Ersatz für den durch die Herausgabe entstandenen Schaden, insbesondere den entgangenen Gewinn, zu leisten hat.[108] Der Besteller darf ausserdem die Herausgabe der Daten an einen von ihm bestimmten Dritten zwecks Durchführung eines Audits verlangen, sollt der Anbieter seiner Herausgabepflicht in *Klausel X* nicht nachkommen.

[106] Streitigkeiten betreffend die Herausgabe unterliegen den gleichen Eskalationsmechanismen wie die restlichen Streitigkeiten und Meinungsverschiedenheiten.

3.2.5.3. Mediations- und Schiedsklausel

[107] Siehe bspw. Musterklauseln der ITDR, https://www.itdr.ch/en/downloads/.

[107] Die Herausgabegründe sind projektbezogen und im Einzelfall zu erarbeiten. Die vorliegend genannten Gründe sind nicht abschliessende Beispiele. Ob dem Besteller mit Abschluss der Entwicklung neben dem Programm (Objektcode) auch der Quellcode übertragen wird (das Urheberrecht liegt beim Entwickler), hängt von der zwischen den Parteien getroffenen Vereinbarung ab. Falls es zu einer Übertragung kommen soll, wäre der Abschluss der Entwicklung ein weiterer Herausgabegrund.

[108] Zweck des 3. Herausgabegrundes ist es zu ermöglichen, dass das Projekt während eines möglicherweise langwierigen Verfahrens allenfalls mit einem Dritten weiterverfolgt werden kann und nicht blockiert wird.

4. Fazit

[108] Technische Verschuldung ist kein neues Phänomen und doch hat es nicht an Relevanz verloren. Oft durch Zeitmangel entstanden, bedarf es einer bewussten Beseitigung technischer Verschuldung, um die nachhaltige und schuldenfreie Entwicklung eines IT-Projektes sicherzustellen. Während ein gemeinsames Bewusstsein und ein entsprechendes Entwicklungsumfeld eine wichtige Grundlage bilden, darf nicht unterschätzt werden, in welcher Beziehung und Abhängigkeit die Parteien eines IT-Projektes stehen. Während grundsätzlich angenommen werden darf, dass die Parteien redlich und in Harmonie auf eine erfolgreiche Entwicklung hinarbeiten, lohnt es sich doch, Kontrollmechanismen zu vereinbaren.

[109] Abschliessend lässt sich folgendes festhalten:

[110] Die Zunahme einer gewissen technischen Verschuldung ist in einem langjährigen IT-Projekt zu erwarten. Deren Beseitigung hat genauso zum Entwicklungsprozess zu gehören wie die Entwicklung neuer Funktionen, was beiden Parteien bewusst sein muss. Steigende Kosten für die Entwicklung neuer Funktionen sind nicht nur Symptom, sondern auch Indikator technischer Verschuldung.

[111] Damit ein Überblick über die technische Verschuldung im Projekt möglich ist, muss diese laufend anhand verschiedener Metriken gemessen werden. Auf der technischen Ebene bestehen Analysetools, welche diese Beurteilung laufend vornehmen können. Aber auch den Risiken mangelhafter Projektführung sollte die nötige Aufmerksamkeit geschenkt werden.

[112] Entwickelt sich die technische Verschuldung in eine Richtung, welche die zu erwartende technische Verschuldung übersteigt, sind entsprechende Massnahmen anhand des vereinbarten Risk-Shares nötig. Oft wird der Softwareentwickler wohl auf eigene Kosten «aufräumen» müssen, wobei die Ermittlung der Quelle der technischen Verschuldung (bspw. zu enge Zeitpläne) nicht unbeachtet bleiben sollte.

[113] Die vertragliche Regelung von IT-Projekten muss den verschiedenen Bestandteilen eines IT-Projektes und den unterschiedlichen Vertragsarten Rechnung tragen. Es bietet sich an, der Entwicklung mittels Rahmenvertrag einen Mantel zu geben. Regelmässige Abnahmen durch den Besteller schaffen für beide Parteien (Rechts-)Sicherheit.

[114] Während eine Eskalation bei Meinungsverschiedenheiten möglichst vermieden werden sollte, ist eine vorgängige Regelung von (zeit-)effizienten Mechanismen von Vorteil, um den Fortbestand und die Weiterentwicklung eines IT-Projektes sicherzustellen, bspw. durch vertragliches Accountmanagement und Konflikteskalationsverfahren.

[115] Schliesslich lässt sich feststellen, dass sich technische Verschuldung vermeiden lässt. Wird sauber und qualitativ hochwertig programmiert, die Dokumentation laufend nachgeführt, das Entwicklerteam mit dem richtigen Personal besetzt und ein vernünftiger Zeitplan vorgegeben, so lässt sich die technische Verschuldung auf ein Minimum reduzieren. Softwareentwickler und -abnehmer gehen oft ein langjähriges Verhältnis ein und sind voneinander abhängig. Ein Bewusstsein für diese Situation und eine solide vertragliche Grundlage, die gute Arbeit belohnt, sind der Grundstein für eine schuldenfreie Entwicklung.

MLaw Nils Fuchs schloss mit seiner Masterarbeit zum Thema «Vertragliche Instrumente zum Management technischer Verschuldung» sein Masterstudium an der Universität Bern ab. Der vorliegende Beitrag stellt einen Zusammenzug seiner Erkenntnisse dar.

Der Autor dankt insbesondere Dr. Wolfgang Straub für die wohlwollende Betreuung seiner Masterarbeit und den zahlreichen hilfreichen Inputs, sowie MLaw Vaishnavan Subramaniam für seinen Beitrag zu DevOps-Verträgen.