Softwarequalität zum Anfassen: Gibt es so etwas?

Softwarequalität ist ein sehr schwer greifbarer, abstrakter Begriff. Das ist so, trotz (oder gerade wegen?) der Vielzahl an Qualitätsmodellen in der IT. Um Softwarequalität fassbar zu machen, bräuchte man zum einen ein ganzheitliches, intuitiv verständliches Qualitätsmodell und zum anderen einen Qualitätssicherungsprozess, der die Ideen aus dem Qualitätsmodell im Projektalltag und im Softwareprodukt spürbar werden lässt. Aber was ist Softwarequalität eigentlich? Wer bestimmt, was Qualität ist? Wie kann sie gesteuert werden? Gibt es ein intuitiv verständliches Qualitätsmodell? Diesen und weiteren Fragen geht der Artikel nach.

Softwarequalität wird häufig an einer Liste von sogenannten nicht-funktionalen Eigenschaften festgemacht, die gute Software haben sollte. Diese Eigenschaften, deren Namen meistens auf „...keit“ oder „...heit“ enden, sind in der Regel eingeteilt in Haupteigenschaften und Eigenschaften, die zu einer bestimmten Haupteigenschaft gehören. Diese Einteilung kann man dann Modell oder Norm nennen. Die Zahl dieser Modelle und Normen scheint ähnlich groß zu sein wie die Zahl der darin enthaltenen „…keiten” und „…heiten“. Auf der Suche nach Softwarequalität verliert man sich also nicht nur leicht in der Vielzahl von Softwareeigenschaften, sondern auch in der Vielzahl der Modelle und Normen (**siehe Abbildung 1**).

Was ist Softwarequalität?

Definition 1 in **Kasten 1** macht deutlich, dass Softwarequalität von expliziten (bewussten) und impliziten (unbewussten) Bedürfnissen abhängt. Neben dem Anwender und dem Kunden der Software gibt es noch zahlreiche andere Akteure, die im Lebenszyklus der Software einen Stakeholder darstellen, und alle haben ihre jeweils eigenen Bedürfnisse und Erwartungen (**siehe Abbildung 2**). Jeder dieser Standpunkte hat seine Berechtigung und kann seinen Beitrag zur Softwarequalität leisten. Bei neutraler, transzendenter Betrachtung lässt sich feststellen, dass trotz aller Maßnahmen für Softwarequalität letztlich immer die subjektive Zufriedenheit der Akteure (vor allem die des Geldgebers bzw. Kunden) ausschlaggebend ist. Es gibt Verfahren und Maßnahmen zum Erreichen von Softwarequalität, die allerdings viel Erfahrung benötigen und gegebenenfalls einen Kompromiss zwischen sich widersprechenden Bedürfnissen und Erwartungen der Akteure erforderlich machen.

In Definition 2 in **Kasten 1** kommt zum Ausdruck, dass Software zwei verschiedene Aspekte beinhaltet:

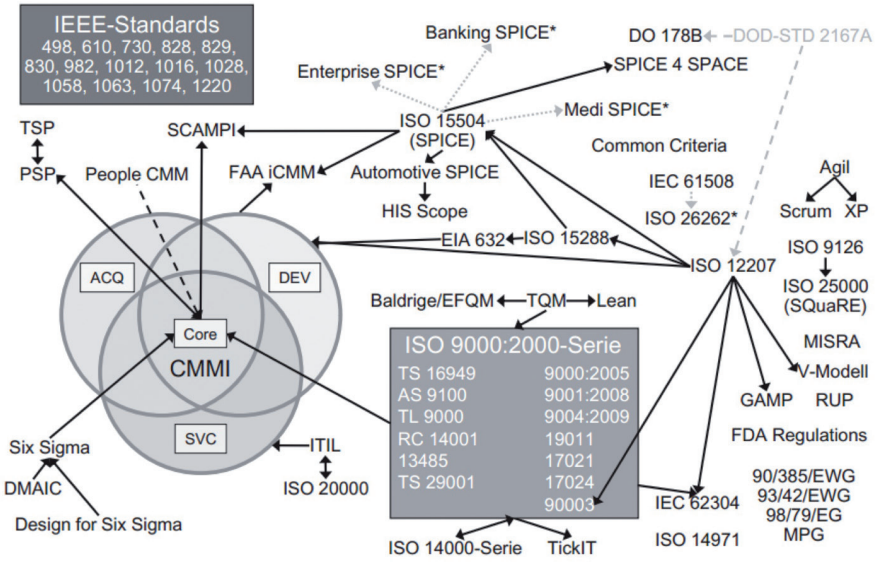

Abb. 1: Eine Übersicht existierender Modelle und Normen in der IT (aus [Wal11], S. 30).

- das Softwareprodukt
- den Softwareentwicklungsprozess

Definition 1
Softwarequalität ist die Gesamtheit von Funktionen und Merkmalen eines Softwareprodukts, das die Fähigkeit besitzt, angegebene oder implizite Bedürfnisse zu befriedigen (vgl. [Wik-a]).

Definition 2
Softwarequalität ist die Summe aller relevanten Eigenschaften eines Softwareprodukts, mit denen seine Kunden zufriedengestellt werden, und die Summe der dazu notwendigen Eigenschaften von Softwareprozessen, die zur Erstellung, zum Betrieb und zur Pflege gefordert werden (vgl. [Wal11]).

Definition 3
Softwarequalität ist eine Auszeichnung von Software für folgende drei maßgebliche Aspekte:

- *Korrektheit:* Das Produkt wird seinen funktionalen und nicht-funktionalen Anforderungen gerecht (vgl. [Wes12]).
- *Produktionseffizienz:* Die Herstellungskosten der Software entsprechen den gegebenen Rahmenbedingungen (vgl. [Wes13-b]).
- *Evolvierbarkeit:* Der Entwicklungsprozess stellt ein notwendiges Maß an Wartbarkeit der Software sicher (vgl. [Wes13-a]).

Kasten 1: Definitionen von Softwarequalität.

Die existierenden Erwartungen und Bedürfnisse der Akteure richten sich an beide Aspekte. Der Aspekt „Softwareprodukt“

Abb. 2: Es gibt sehr unterschiedliche Sichten auf die Software(-qualität), die sich alle aus den Rollen der jeweiligen Akteure ergeben.

stellt eher die Kundenbedürfnisse dar, der Aspekt „Softwareentwicklungsprozess“ eher die Bedürfnisse der Entwickler.

Definition 3 in **Kasten 1** erweitert Definition 2 um den Aspekt der Entwicklungskosten (b). Produktqualität (a) und Prozessqualität (c) sind gleich gewichtet. Diese drei Aspekte machen zusammen das aus, was wir unter Softwarequalität verstehen (vgl. auch [CCC-a]). Wartbarkeit ist ein Aspekt, der selten explizit, aber meistens implizit gefordert wird. Im Kano-Modell (vgl. [CCC-i]) stellt die Wartbarkeit einen Basisfaktor dar, der – wenn sie nicht ausreichend ist – als großes Qualitätsdefizit wahrgenommen wird.

Alle drei Definitionen haben das gleiche Problem: Sie beschreiben nur sehr abstrakt, was Softwarequalität sein soll. Konkret wird aber nicht deutlich, was das für die Software, d.h. das Projekt und das Produkt, bedeutet und wie man sie erreicht. Abhilfe schafft hier ein Qualitätsmodell.

Ein (be-)greifbares Qualitätsmodell

Aus Sicht des Nutzers eines Softwarequalitätsmodells ist der praktische Einsatz im Entwickleralltag von großer Bedeutung. Zu diesem Zweck wäre eine ganzheitliche Anwendbarkeit wichtig, d.h. alle wesentlichen Aspekte von Softwarequalität müssten darin enthalten sein. Außerdem müssten die „…keiten“ und „…heiten“ so zu strukturieren sein, dass eine intuitiv verständliche Ordnung entsteht. Das würde es dem Nutzer des Qualitätsmodells erlauben, die Vielzahl an Qualitätsmerkmalen leichter zu überschauen, eigene Prioritäten/Schwerpunkte zu setzen und eine persönliche oder projektspezifische Checkliste zu formulieren, deren Abarbeitung verhindert, dass wichtige Qualitätsziele übersehen werden. Eine solche intuitiv verständliche Ordnung müsste sich aus der Natur, aus dem Wesen von Software, ableiten lassen.

Definition 2 in **Kasten 1** unterscheidet zwischen Produktqualität und Prozessqualität. Diese Unterscheidung ist tatsächlich von grundlegender Bedeutung (vgl. auch [CCC-b]).

Betrachtet man das Produkt, lassen sich die beiden Teilaspekte „Anwendung“ und „Betrieb“ unterscheiden. Betrachtet man den Prozess, lassen sich die drei Teilaspekte „Auslieferung“, „Entwicklung“ und „Anforderung“ unterscheiden. Diese Einteilung stellt die Kernaussage des in **Abbildung 3** gezeigten Qualitätsmodells dar. Nach diesem Modell gibt es sechs Teilaspekte von Software, die in **Tabelle 1** erläutert werden. Diese Teilaspekte lassen sich sehr schön auf das Eisberg-Modell projizieren. Im Eisberg-Modell werden die Bedeutung und der fließende Übergang von innerer (versteckter) nach äußerer (sichtbarer) Qualität nachvollziehbar visualisiert (vgl. [CCC-c]). Die Eisberg-Metapher macht deutlich, dass die inneren Qualitätsmerkmale die äußeren

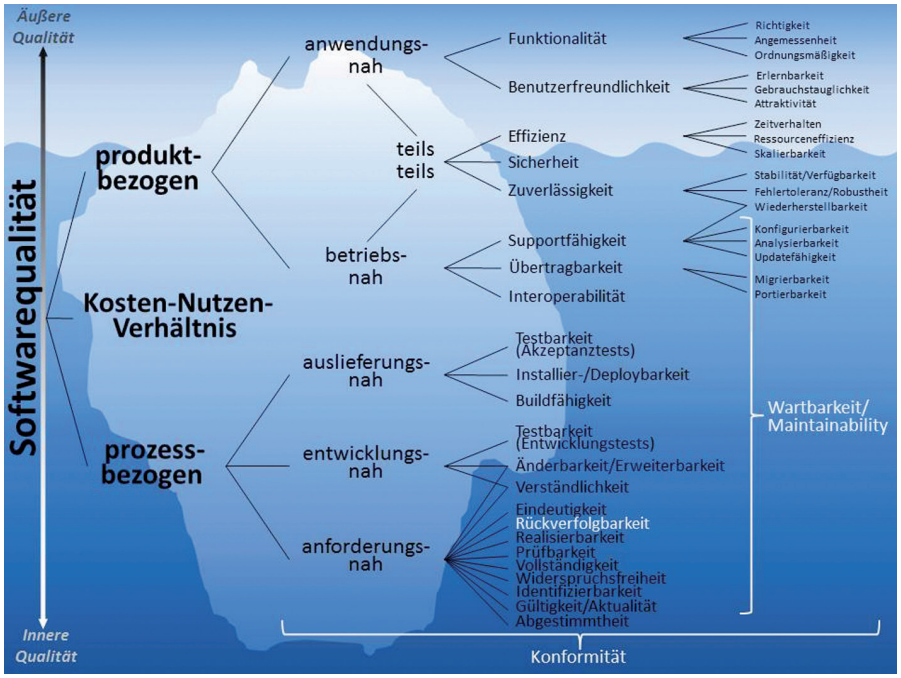

Abb. 3: Ein ganzheitliches Modell für Softwarequalität mit einer Hierarchie von Merkmalen und Merkmalsgruppen, das sich aus den unterschiedlichen Aspekten der Software ableiten lässt (siehe Defintion 3 in Kasten 1).

tragen müssen, ansonsten kann die aus dem Wasser ragende Eisbergspitze (das Softwareprodukt) nicht die erforderliche Höhe und Form erreichen.

Das Qualitätsmodell in **Abbildung 3** kann als ganzheitlich betrachtet werden, da es alle Anforderungen aus dem Applikations-Lebenszyklus an die Software umfasst – von der Anforderungsanalyse (Requirements-Engineering) über Entwicklung und Betrieb bis hin zur Endanwendung (vgl. [CCC-d]). Die Benennung und Einteilung der Qualitätsmerkmale in den unteren Ebenen (rechts im liegenden Baum) ist nicht immer eindeutig (z. B. die Wiederherstellbarkeit) und teilweise subjektiv. Entscheidend sind hier aber die sechs Teilaspekte von Software, die in **Tabelle 1** näher erläutert werden. Sie zeigen die große Bedeutungsbreite von Softwarequalität. Die drei in **Abbildung 3** weiß geschriebenen Merkmale werden im Folgenden genauer betrachtet.

Aspekt	Teilaspekt	Beschreibung
produkt-bezogen	anwendungsnah	Qualitätsmerkmale dieser Kategorie beziehen sich auf die Anwendung des Softwareproduktes. Für Endanwender der Software sind diese Merkmale wichtig.
	teils anwendungs-, teils betriebsnah	Qualitätsmerkmale dieser Kategorie sind Eigenschaften, mit denen sowohl Endanwender als auch Administratoren nicht direkt und nur bedingt zu tun haben. Sind sie nicht zufriedenstellend erfüllt, bedeutet das für Endanwender eine höchstens eingeschränkte Nutzung der Software und für Administratoren zusätzliche Systemeingriffe.
	betriebsnah	Qualitätsmerkmale dieser Kategorie beziehen sich auf die Einrichtung und Pflege des Softwareprodukts in einer Test- oder Produktionsumgebung. Für Administratoren sind diese Merkmale bei der Inbetriebnahme und Wartung wichtig.
prozess-bezogen	auslieferungsnah	Qualitätsmerkmale dieser Kategorie beziehen sich auf den Bau und Test des Release-Kandidaten sowie auf die Kommunikation zwischen Betrieb und Entwicklung.
	entwicklungsnah	Qualitätsmerkmale dieser Kategorie beziehen sich auf die Umsetzung von Anforderungen in Architektur, Design und Implementierung, d. h. die Realisierung neuer Features, Fehlerbehebungen und Refactorings. Für Entwickler sind diese Merkmale wichtig.
	anforderungsnah	Qualitätsmerkmale dieser Kategorie beziehen sich auf die Ermittlung, Erstellung und Verwaltung der Anforderungen und die Kommunikation zwischen Anforderungsmanagement und Entwicklung. Für Anforderungsanalysten/Requirements-Engineers sind diese Merkmale wichtig.

Tabelle 1: Beschreibung der sechs Teilaspekte von Software aus Abbildung 3.

Wartbarkeit, Konformität und Rückverfolgbarkeit

Für die Softwarequalität haben die Qualitätsmerkmale Wartbarkeit, Konformität und Rückverfolgbarkeit (**siehe Abbildung 3**) eine herausragende Bedeutung.

Konformität ist der Grad, in dem ein Produkt oder ein Prozess vereinbarte Normen oder Vereinbarungen erfüllt. Die Konformität kann als eine Art Querschnittsmerkmal betrachtet werden. Konformität in allen Merkmalen steigert die Effizienz durch Wiedererkennung und verringerter Fehlerwahrscheinlichkeit. Das gilt für jeden Akteur in seiner jeweiligen Rolle, sei es

- der User bei der Anwendung der Software (z. B. GUI-Komponenten zeigen einheitliches Verhalten),
- der Administrator im Betrieb (z. B. Installationsskripte haben immer den gleichen Aufbau und verwenden dieselbe Technologie),
- der Entwickler bei der Implementierung (z. B. Standards im Design der Software oder einheitliche Verwendung von Tools) oder
- der Fachexperte bzw. der Anforderungsmanager (gleicher Aufbau von Use-Cases).

Der Nutzen, den die Konformität bietet, entfaltet sich allerdings nur, wenn sich die betroffenen Akteure auf eine einheitliche Vorgehensweise – einen Standard – einigen und diesen bewusst und konsequent an-

Teilaspekt	Qualitätsmerkmal	Typische Maßnahmen zur Analyse, Umsetzung oder Überprüfung
anwendungsnah	Funktionalität	Hohe Code-Coverage in Unit-Tests, gute Mischung aus Unit-, Integrations- und Systemtests (vgl. [CCC-e]).
	Benutzerfreundlichkeit (Usability)	Frühzeitige Usability-Tests, Beratung durch Spezialisten.
teils anwendungs-, teils betriebsnah	Zeitverhalten (Performance)	Zeitmessungen.
	Ressourceneffizienz	Ressourcenüberwachung.
	Skalierbarkeit	Lasttests.
	Sicherheit	Autorisierung, Authenzifizierung, Verschlüsselung.
	Stabilität/Resilience	Chaos Monkeys (vgl. [Git]).
	Fehlertoleranz/Robustheit	Manuelle Tests sowohl mit erfahrenen Power-Usern als auch mit naiven, unerfahrenen Usern.
	Wiederherstellbarkeit/Reliability	Chaos Monkeys (vgl. [Git]).
betriebsnah	Konfigurierbarkeit	Gute Architektur.
	Analysierbarkeit	Logging, Protokollierung, Monitoring.
	Update-Fähigkeit	Automatische Tests.
	Migrierbarkeit	Gute Architektur, Automatische Tests.
	Portierbarkeit/Portability	Tests auf unterschiedlichen Plattformen.
auslieferungsnah	Testbarkeit (Akzeptanztests)	Automatisierung, Oberflächentests, gute Kommunikation zwischen Fachseite und Entwicklung.
	Installier-/Deploybarkeit	Automatisierung, DevOps, Continuous Delivery.
	Buildfähigkeit	Automatisierung, Continuous Integration.
entwicklungsnah	Testbarkeit (Akzeptanztests)	Test Driven Development, Behavior Driven Development (vgl. [Wik-d]), Continuous Integration.
	Änderbarkeit/Modifiability	Separation of Concerns, Modularisierung, Kapselung.
	Verständlichkeit/Understandibility	Clean Coding (vgl. [CCC-f]), Clean Architecting (vgl. [Obe14]).
anforderungsnah	<alle>	Beobachtungs- und Befragungstechniken, Glossar, Satzschablonen, Dokumentationswerkzeug, UML, Validierungs- und Review-Techniken.

Tabelle 2: Typische Maßnahmen, um gewünschte Qualitätsmerkmale zu realisieren.

Cost to fix a defect		Time detected				
		Requirements	Architecture	Construction	System test	Post-release
Time introduced	Requirements	1×	3×	5–10×	10×	10–100×
	Architecture	–	1×	10×	15×	25–100×
	Construction	–	–	1×	10×	10–25×

Tabelle 3: Die Beseitigung von Fehlern, die im Requirements-Engineering gemacht, aber erst nach dem Release der Software bemerkt werden, kann um den Faktor 100 teurer werden (Quelle [Wik-c]).

wenden. Das ist nur bei guter Kommunikation zwischen den Akteuren möglich, die Wildwuchs verhindert.

Die *Wartbarkeit* ist ein Art Super-Merkmal, das mehrere Teilaspekte von Software umfasst. Sie wird typischerweise auf den Quellcode bezogen, fängt aber schon viel früher im Prozess an und reicht im Prozess noch bis in den Betrieb hinein. Wartbarkeit beginnt mit der Erhebung der Anforderungen bei den Fachexperten, geht weiter mit der Qualität des Anforderungsmanagements bis hin zur Entwicklung, beinhaltet aber auch die Administration im Betrieb und reicht schließlich bis zur Produktionsreife der Software. Ist ein Glied in dieser Kette zu schwach, kann die fehlende Wartbarkeit zum Problem werden. Um Wartbarkeit zu gewährleisten, muss also jeder Akteur zum einen für sich alleine seine Kernaufgabe effizient ausführen können, zum anderen muss aber auch die Kommunikation zu den jeweiligen Kommunikationspartnern reibungsarm funktionieren.

Die *Rückverfolgbarkeit (Traceability)* wird typischerweise auf Anforderungen und auf ihre Dokumentation bezogen. Um Anforderungen effizient zu verwalten, ist es wichtig, die chronologische Änderung einer Anforderung (vertikale Traceability) und den Zusammenhang zu anderen Anforderungen im Kontext (horizontale Traceability) transparent zu haben. Diese Qualitätseigenschaft lässt sich auf den gesamten Applikations-Lebenszyklus ausdehnen, d. h. welche Anforderung steht mit welchen Codeabschnitten, mit welchen Tests, mit welcher Komponente, mit welcher Auslieferung, mit welchem Release-Kandidaten und mit welcher Funktionalität in der Anwendung in Verbindung. Fehleranalysen und -behebungen werden durch diese breite und tiefe Form der Rückverfolgbarkeit deutlich einfacher.

Bedeutung der Kommunikation

Wartbarkeit, Konformität und Rückverfolgbarkeit haben eines gemeinsam: Sie machen deutlich, dass die Kommunikation eine fundamentale Rolle im Entwicklungsprozess spielt. Gute Kommunikation ist ein entscheidender Schlüssel zur Softwarequalität. Anwender, Fachexperten, Anforderungsmanager, Entwickler, Architekten und Administratoren gehören zu einer Kommunikationskette.

Typische Schwachstellen in dieser Kette sind die Schnittstellen zwischen Anforderungsmanagement und Entwicklung sowie zwischen Entwicklung und Betrieb. Erstere liegt begründet in einer Kommunikationsbarriere zwischen den fachlichen und den technischen Gedankenwelten der Akteure (vgl. [CCC-g]). Letztere ist häufig ein organisatorisches Problem durch die Trennung von Entwicklung und Betrieb in verschiedene Abteilungen. Dieses Problems hat sich die DevOps-Bewegung angenommen (vgl. [Hüt12]), die sich für cross-funktionale Teams einsetzt, d. h. *Mixed Teams*, die nicht nur aus Entwicklern bestehen, sondern auch aus Administratoren aus dem Betrieb (Operators) und Anforderungsspezialisten aus dem Requirements-Engineering.

Innerhalb eines Teams ist eine gute Zusammenarbeit nicht automatisch gegeben. Teambildung ist ein komplexer Prozess. Um diesen bewusst und zielgerichtet zu vollziehen, bietet das *Team Clean Coding* eine Reihe von Best Practices an (vgl. [CCC-h]).

Anforderungsnahe Softwarequalitätsmerkmale

Die Qualität von Prozessen, mit denen Anforderungen ermittelt werden, und die Qualität von Dokumenten, welche die Anforderungen beschreiben, ist ein großes Thema (vgl. [Wik-b]). In Qualitätsmodellen der Softwareentwickler findet man diesen Aspekt von Software üblicherweise nicht, weil solche Modelle eher das Softwareprodukt fokussieren und weniger den Entwicklungsprozess in seiner ganzen Breite. Allerdings wirkt sich mangelnde Qualität im Requirements-Engineering massiv auf das Produkt und dessen Qualität aus. Das wird deutlich, wenn man die Behebungskosten für Fehler vergleicht, die in verschiedenen Phasen der Entwicklung gemacht und früher oder später entdeckt wurden (**siehe Tabelle 3**).

Beseitigungskosten von Fehlern im Requirements-Engineering können leicht so hoch werden, dass man den Fehler gar nicht mehr beseitigt und eine technische Schuld und ihre Folgekosten in Kauf nimmt, die aber dann während einer langen Lebensdauer der Software wiederum höher werden können als die Rückzahlung der Schuld. Soft-

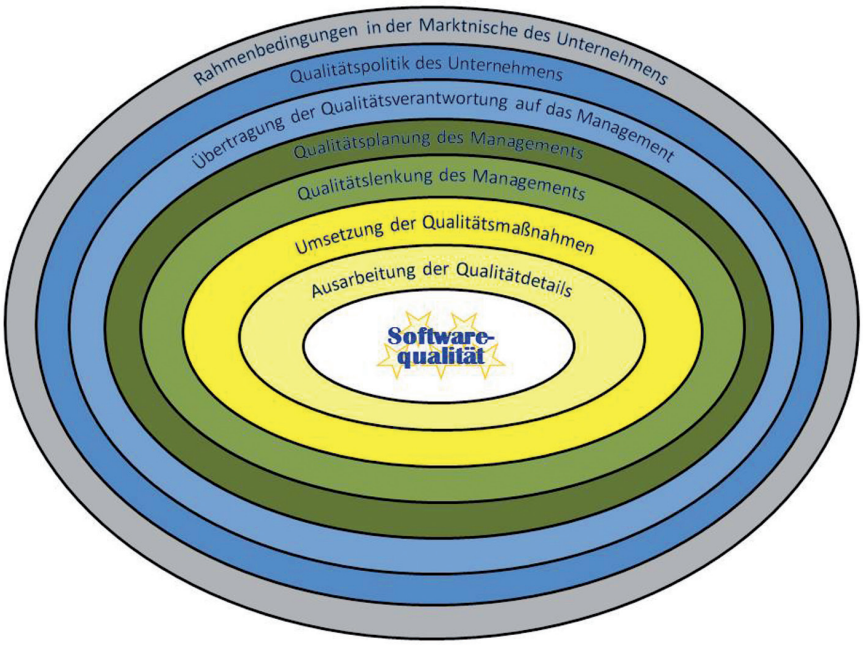

Abb. 4: Schrittweise Konkretisierung der Softwarequalität durch die Unternehmensleitung (blau), das Qualitätsmanagement (grün) und die Akteure der Umsetzung (gelb). Letztere sind typischerweise Anforderungsanalysten, Entwickler, Tester und Administratoren.

warequalität beginnt also schon mit dem Beginn des Applikations-Lebenszyklusmanagements und damit weit vor der ersten Implementierung (vgl. auch [CCC-d]). In einem ganzheitlichen Qualitätsmodell für Software, wie in **Abbildung 3** gezeigt, darf dieser Teilaspekt von Software deshalb nicht fehlen.

Qualitätssicherung und -bewusstsein

Unter *Qualitätssicherung (QS)* versteht man oftmals so etwas wie eine Testabteilung. In diesem Artikel wird darunter die Summe sämtlicher Maßnahmen verstanden, die zur Erreichung von Softwarequalität beitragen. QS betreibt jeder verantwortungsvolle Akteur in seiner jeweiligen Rolle, der ein Qualitätsbewusstsein für seine Aufgabe hat. Das Zusammenspiel aller dieser Maßnahmen ist dann effektiv, wenn das Qualitätsbewusstsein aller Akteure sich an denselben Rahmenbedingungen ausrichtet und es einen Qualitätstreiber gibt, der die QS koordiniert und Softwarequalität einfordert. Ein einheitliches und intuitiv verständliches Qualitätsmodell stellt dabei eine wesentliche Grundlage dar, in dem alle Akteure ihr Qualitätsbewusstsein abbilden können (**siehe Abbildung 2 und 3**).

Ein solches Qualitätsmodell allein – selbst wenn es allen Akteuren bewusst ist – reicht jedoch nicht aus, um Softwarequalität im Projekt und Produkt spürbar werden zu lassen. QS muss zielgerichtet, koordiniert und bewusst durchgeführt werden. Dafür sind Qualitätstreiber notwendig. Auf der Suche nach Qualitätstreibern landet man schnell beim Qualitätsmanagement. Aber was sind Qualitätsmanager? Wer beauftragt sie? Und woher wissen sie, was Qualität ist?

Priorisierung und Konkretisierung von Qualitätsaspekten

Äußere Rahmenbedingungen für die Qualitätspolitik eines Unternehmens bilden die Marktnische, in der ein Unternehmen angesiedelt ist. Durch eine sinnvolle Qualitätspolitik werden innerhalb dieses Rahmens strategische Qualitätsziele zur Leitlinie aller Softwareentwicklung im Unternehmen. Durch die Besetzung der Qualitätsmanagerrolle mit einer konkreten Person wird der Qualitätsrahmen weiter konkretisiert. Während der Planung, Lenkung und Umsetzung werden immer weitere Qualitätsentscheidungen getroffen (**siehe Abbildung 4**). Wie das Ganze konkret aussehen kann, zeigt **Tabelle 4**. Drei konkrete Unternehmen aus verschiedenen Marktnischen könnten die in der Tabelle dargestellten Schritte wie beschrieben vollzogen haben. Andere Unternehmen in diesen Branchen könnten andere Prioritäten setzen und andere Konkretisierungswege gehen, z. B. in

	Anbieter von Online-Banking-Software	Betreiber eines Social Networks	Hersteller einer Entwicklungsumgebung
Qualitätspolitik	Sicherheit über alles!	Verfügbarkeit ist oberstes Gebot!	Möglichst kurze „Time to Market"!
Übertragung der Verantwortung auf	Projektmanager: *Die wichtigsten bekannten Risiken für Webanwendungen ausschließen.*	Architekt: *Stabilität, Ausfallsicherheit und Wiederherstellbarkeit sicherstellen.*	Produktmanager: *Wartbarkeit und kurze automatisierte Auslieferungszyklen gewährleisten.*
Qualitätsplanung	Anforderungsanalysten mit der Ausarbeitung von OWASP-Angriffsszenarien beauftragen, gegebenenfalls Security-Experten ins Team holen.	Entwurf einer geeigneten Architektur.	Ermittlung geeigneter Automatisierungsmaßnahmen und Metriken zur Überwachung der Codequalität.
Qualitätslenkung	Ressourcen für eine geeignete Testumgebung beim Betrieb anfordern, Tester mit Angriffsszenarien beauftragen.	Umsetzung der entworfenen Architektur in der Entwicklung begleiten.	Ressourcen für die Einrichtung und Wartung einer Deployment-Pipeline beauftragen.
Umsetzen von Qualitätsmaßnahmen	Betrieb: Geeignete Testumgebung einrichten. Entwicklung: Tests implementieren und durchführen.	Vorgaben der Architektur bei der Implementierung bewusst berücksichtigen.	Deployment-Pipeline mit Metrikberechnung einrichten.
Ausarbeitung der Qualitätsdetails	Welche Tools werden benutzt? Gibt es ein Review-Verfahren? Wird mit TDD entwickelt?	Implementierung spezieller Tests, um Verhalten bei Ausfallfehler zu testen.	Werden Clean-Code-Developer-Regeln berücksichtigt? Wie wird die Deployment-Pipeline dokumentiert?

Tabelle 4: Beispiele, wie die schrittweise Konkretisierung von Softwarequalität praktisch aussehen kann.

- *Qualitätsplanung* ist jener Teil des Qualitätsmanagements, der auf das Festlegen der Qualitätsziele und der notwendigen Ausführungsprozesse sowie der zugehörigen Ressourcen zum Erreichen der Qualitätsziele gerichtet ist.
- *Qualitätslenkung* ist jener Teil des Qualitätsmanagements, der auf die Erfüllung der Qualitätsanforderungen ausgerichtet ist […]. Er umfasst dabei Arbeitstechniken und Tätigkeiten sowohl zur Überwachung eines Prozesses als auch zur Beseitigung von Ursachen nicht zufriedenstellender Ergebnisse.
- *Qualitätsprüfung* ist jener Teil des Qualitätsmanagements, der eine Konformitätsbewertung durch Beobachten, Beurteilen, Messen und Vergleichen vornimmt.
- *Qualitätsdarlegung* ist jener Teil des Qualitätsmanagements, der alle geplanten und systematischen Tätigkeiten des Qualitätsmanagementsystems dokumentiert und entsprechende Nachweise über ausgeführte Tätigkeiten und erzielte Ergebnisse bereitstellt.

Kasten 2: Definitionen der Teilaspekte (Verantwortlichkeiten, Aufgaben) von Qualitätsmanagement für Dienstleistungen nach [Bru13].

Schritt 3 andere Verantwortliche beauftragen, die andere Schwerpunkte setzen, oder mehrere Qualitätsbeauftragte wie einen Projekt- und einen Produktmanager einsetzen. Verantwortung kann auch einem ganzen Team anvertraut werden. Im Umfeld von „Agile“ und „DevOps“ wird dies durch die Schlagwörter „cross-funktionales Team“, „Team-based“, „Whole-Team“ oder „One-Team-Approach“ deutlich.

Qualitätsziele und -treiber

Ein Aspekt von Qualität sind die dafür nötigen Kosten (siehe Definition 3 in **Kasten 1 und Abbildung 3**). Derjenige, der das größte Interesse an Qualität haben sollte, ist der Auftraggeber, der Geldgeber oder derjenige, der das investierte Kapital verwaltet und dafür verantwortlich ist. Alle Investitionen werden in einem bestimmten Kontext getätigt, d.h. in einer speziellen Branche und einem speziellen Marktsektor mit einer bestimmten Absicht. Dieser Investitionskontext bildet den großen Rahmen für die Softwarequalität (**siehe Abbildung 4**). In diesem Rahmen sind mit den Investitionen auch bestimmte strategische Ziele einer Organisation/eines Unternehmens verbunden, die den Qualitätsrahmen weiter einschränken.

Initial liegt die Verantwortung, für Softwarequalität zu sorgen, bei dem Kapitalverantwortlichen, d.h. bei der Geschäftsführung eines Unternehmens, das die Software herstellt oder die Herstellung beauftragt. Diese initiale Aufgabe besteht darin, sowohl allgemeine Qualitätsziele zu formulieren, die für den Erfolg des Unternehmens und seiner Investitionen als maßgeblich angesehen werden, als auch das Qualitätsmanagement zu beauftragen, das dann die Verantwortung als Qualitätstreiber übernimmt. Mit der Auswahl der Qualitätsmanager und mit ihren persönlichen Präferenzen schränkt sich der Rahmen der Softwarequalität weiter ein (**siehe Abbildung 4 und Tabelle 3**).

Qualitätsmanagement und -steuerung

Qualitätsmanagement ist ein Regelkreis, ein Zyklus von Verantwortlichkeiten, der zu einem Prozess stetiger Verbesserung führt. Im einfachsten Fall besteht der Zyklus aus den drei Verantwortlichkeiten/Phasen: Planung, Umsetzung und Kontrolle. Im Qualitätsmanagement werden diese „Qualitätsplanung“, „Qualitätslenkung“ und „Qualitätsprüfung“ genannt (**siehe Abbildung 5 und Kasten 2**).

Die Abgrenzung der Teilaspekte ist nicht immer eindeutig, vor allem zwischen Qualitätslenkung und Qualitätsprüfung. Durch wiederholte Prüfungen, überarbeitete Planungen und nachjustierte Lenkung werden der Prozess des Qualitätsmanagements selbst und das Produkt, das dabei entsteht, immer weiter optimiert: Qualität wird geschaffen. Durch die Wahl ihrer Mittel treffen Qualitätsmanager Entscheidungen, setzen Schwerpunkte und machen Vorgaben, die den Qualitätsrahmen weiter einschränken und konkretisieren (**siehe Abbildung 4**). Die Qualitätsmanager beauftragen dann die Akteure der eigentlichen Umsetzung und geben diesen den erarbeiteten Qualitätsrahmen mit.

Wenn dieser Optimierungszyklus bewusst angewandt wird, kann die Qualität gezielt gesteuert werden. Steuerung heißt dabei nicht, dass den Akteuren der Umsetzung Vorgaben bis ins kleinste Detail gemacht werden. Es geht darum, Akzente zu setzen und den Qualitätsrahmen so einzuschränken, dass bei der Umsetzung eine Richtung eingeschlagen wird, die zu den gesteckten

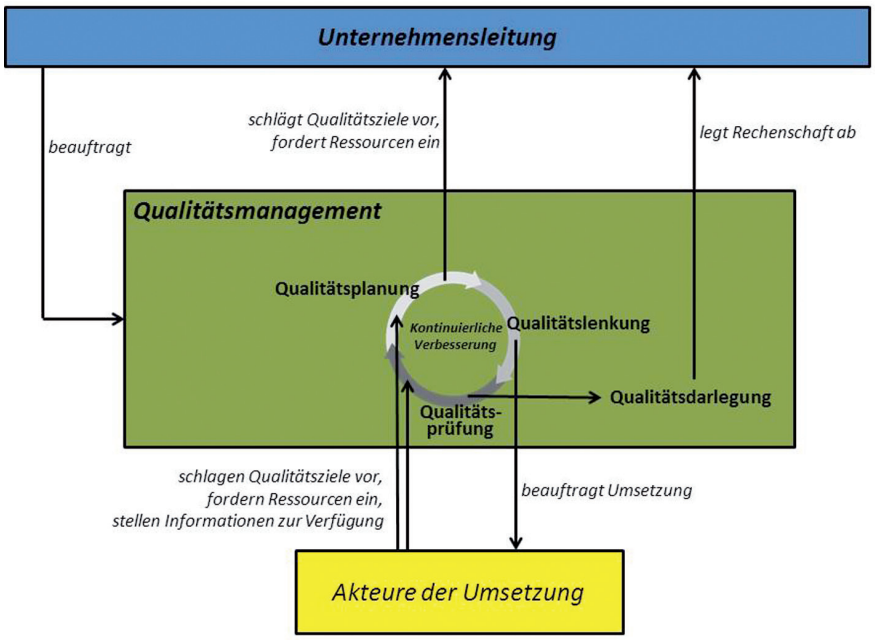

Abb. 5: Qualitätsmanagement ist ein kontinuierlicher Verbesserungszyklus, mit dem die Unternehmensleitung und die Akteure der Umsetzung regelmäßig interagieren (siehe Kasten 2).

Literatur & Links

[Bru13] M. Bruhn, Qualitätsmanagement für Dienstleistungen, Springer, 2013

[CCC-a] Clean Coding Cosmos, Was ist Softwarequalität?, siehe:
http://clean-coding-cosmos.de/der-entwicklerkosmos/softwarequalitaet-1

[CCC-b] Clean Coding Cosmos, Produktqualität vs. Prozessqualität, siehe:
http://clean-coding-cosmos.de/der-entwicklerkosmos/softwarequalitaet-4

[CCC-c] Clean Coding Cosmos, Innere und äußere Qualität, siehe:
http://clean-coding-cosmos.de/der-entwicklerkosmos/softwarequalitaet-3

[CCC-d] Clean Coding Cosmos, Das Application Life Cycle Management (ALM), siehe
http://clean-coding-cosmos.de/der-entwicklerkosmos/sep-alm-1/sep-alm-2

[CCC-e] Clean Coding Cosmos, siehe: Saubere Tests, siehe:
http://clean-coding-cosmos.de/der-entwicklerkosmos/dievierdimensionen/dimension-wissen-1/dimension-wissen-5

[CCC-f] Clean Coding Cosmos, Sauberer Code, siehe:
http://clean-coding-cosmos.de/der-entwicklerkosmos/dievierdimensionen/dimension-wissen-1/dimension-wissen-3

[CCC-g] Clean Coding Cosmos, Die Akteure im ALM, siehe: http://clean-coding-cosmos.de/der-entwicklerkosmos/sep-alm-1/sep-alm-3

[CCC-h] Clean Coding Cosmos, Team Clean Coding, siehe: http://clean-coding-cosmos.de/team-clean-coding

[CCC-i] Clean Coding Cosmos, Kundenzufriedenheit, siehe:
http://clean-coding-cosmos.de/der-entwicklerkosmos/softwarequalitaet-1/softwarequalitaet-2

[Git] GitHub, Chaos Monkey, siehe: https://github.com/Netflix/SimianArmy/wiki/Chaos-Monkey

[Hüt12] M. Hüttermann, DevOps für Developer, Apress, 2012

[Obe13] R. Oberrath, J. Vollmer, Clean Coding Cosmos: Teil 1 – Kosmologie für Softwareentwickler, in: OBJEKTspektrum 6/2013, Kasten 4

[Obe14] R. Oberrath, J. Vollmer, Clean Coding Cosmos: Teil 3 – Kosmische Effizienz durch Team Clean Coding, in: OBJEKTspektrum 2/2014, Kasten 4

[Wal11] E. Wallmüller, Software Quality Engineering, Hanser, 2011

[Wes12] R. Westphal, Clean Code Developer: Korrektheit als Wert, in: OBJEKTspektrum 5/2012

[Wes13-a] R. Westphal, Clean Code Developer: Teil 4 – Produktionseffizienz als Wert, in: OBJEKTspektrum 2/2013

[Wes13-b] R. Westphal, Clean Code Developer: Teil 4 – Produktionseffizienz als Wert, in: OBJEKTspektrum 2/2013

[Wik-a] Wikipedia, Softwarequalität, siehe: http://de.wikipedia.org/wiki/Softwarequalit%C3%A4t

[Wik-b] Wikipedia, Anforderungsanalyse (Informatik), siehe: https://de.wikipedia.org/wiki/Anforderungsanalyse_%28Informatik%29

[Wik-c] Wikipedia, Software testing, siehe: https://en.wikipedia.org/wiki/Software_testing

[Wik-d] Wikipedia, Behavior Driven Development, siehe: https://de.wikipedia.org/wiki/Behavior_Driven_Development

Zielen führt. Viele Details sind oftmals unnötig oder es ist gar nicht möglich, sie vorzugeben, weil sie zu technisch sind. Wie viele Details sinnvoll sind, müssen die Beteiligten im Einzelfall klären. Je mehr Details vom Qualitätsmanagement vorgegeben werden, desto größer ist das Risiko, dass Konflikte zwischen Qualitätsmanagement und Akteuren entstehen.

Jeder erfahrene Akteur hat in langjähriger Arbeit Überzeugungen gewonnen, die mit den Vorgaben des Qualitätsmanagements kollidieren können. Nicht erkannte oder ignorierte Konflikte stellen eine große Gefahr für die Softwarequalität dar. Effizient ist das Qualitätsmanagement dann, wenn diese Art von Konflikten klein gehalten wird – sei es durch die Vermeidung von Detailreichtum oder durch gute Kommunikation und Überzeugung.

Qualitätsumsetzung und -details

Im Qualitätsrahmen, den das Qualitätsmanagement vorgibt, realisieren die Akteure der Umsetzung (Anforderungsanalysten, Entwickler, Architekten, Administratoren, Tester) ihre Aufgabe. Sie erstellen Anforderungen, schreiben Quellcode, stellen Softwareinstallationen zur Verfügung und testen diese. Bei diesen Aufgaben müssen im Detail viele einzelne Entscheidungen getroffen werden, die alle zusammen zur Softwarequalität beitragen (**siehe Abbildung 5 und Kasten 2**). Qualitätsaspekte, die das Qualitätsmanagement nicht berücksichtigt haben, setzen engagierte Akteure im Rahmen ihres Spielraumes um. Ist der Spielraum dafür zu klein, können sie dem Qualitätsmanagement entsprechende Qualitätsziele vortragen und Ressourcen dafür anfragen.

Qualitätsforderung und Transparenz

Das Qualitätsmanagement ist die „Middleware“ zwischen Unternehmensleitung und den Akteuren der Umsetzung. Letztere stellen den Qualitätsmanagern alle nötigen Informationen zur Verfügung, damit die bisher erreichte Qualität und die vorgenommenen Maßnahmen im Rahmen der Qualitätsprüfung (**siehe Abbildung 5**) beurteilt werden können. Je höher der vorgegebene Detailierungsgrad des Qualitätsrahmens ist, desto schwieriger kann die Prüfung für die Qualitätsmanager werden und desto unangenehmer für die Akteure der Umsetzung, die sich dadurch überwacht fühlen können. Die Qualitätsmanager müssen im Rahmen der Qualitätsdarlegung (**siehe Abbildung 5**) ihrerseits bei der Unternehmensleitung Rechenschaft ablegen. Darin legen sie den Erfolg ihrer Arbeit dar und benötigen dafür Informationen von den Akteuren der Umsetzung. Auf diese Weise werden die Auswirkungen aller Maßnahmen für die QS transparent.

Qualitätskosten oder -investitionen

Qualität ist nichts, was automatisch oder einfach nebenbei entsteht. Qualität kostet – und zwar in allen genannten Ebenen:

- Die *Akteure der Umsetzung* müssen die Bedeutung von technischen Details für die Softwarequalität erkennen lernen und ausreichend Zeit haben, sorgfältig zu arbeiten. Außerdem benötigen sie eine geeignete Infrastruktur für ihre Arbeit (Hardware, Software, Räumlichkeiten).
- Das *Qualitätsmanagement* muss diese Infrastruktur von der Unternehmensleitung einfordern. Außerdem stellt die Arbeit der Qualitätsmanager selbst eine Qualitätsinvestition dar, die finanziert werden muss.
- Und schließlich kostet es die *Unternehmensleitung* Zeit und Gedanken, eine erfolgreiche Unternehmensstrategie auszuarbeiten und bewusste Qualitätsziele für das Qualitätsmanagement zu formulieren.

Was ist die Alternative zu diesen Qualitätskosten? Entweder ein nicht steuerbarer Wildwuchs an QS-Maßnahmen oder aber der Versuch, Software im Blindflug zu entwickeln. Das Ergebnis ist bei beiden Alternativen gleich: Es bleibt dem Zufall überlassen, ob, welche und wie viel Qualität geschaffen wird. **Tabelle 3** zeigt außerdem, dass – wenn zu spät begonnen wird, die Qualität zu kontrollieren und zu verbessern – die Kosten unnötig und massiv nach oben getrieben werden können. Vor diesem Hintergrund erscheint es sinnvoll, frühzeitig und bewusst in Qualität zu investieren und den dafür nötigen Aufwand nicht einfach als Kosten, sondern als Investitionen in langfristige Qualität und Erfolg zu betrachten.

Schwergewichtigkeit und Skalierbarkeit

Abbildung 5 formalisiert den Prozess des Qualitätsmanagements. Dadurch kann der Eindruck entstehen, dass Qualitätsmanagement ein schwergewichtiges Bürokratiemonster sein muss, das viel Overhead erzeugt. Bei sehr großen Projekten ist es bestimmt notwendig, diesen Prozess mit einem strengen Formalismus durchzuführen. Der Grad dieses Formalismus kann sich aber an der Projektgröße orientieren. In Projekten mit nur sehr wenigen Akteuren kann ein erfahrenes Teammitglied, z. B. ein Senior Developer, Architekt oder Projektleiter die Aufgaben des Qualitätsmanagements mit gewöhnlichen Standard-Tools wahrnehmen. Diese Aufgabe kostet ihn vielleicht 25 oder 50 Prozent seiner gesamten Arbeitszeit. Bei größeren Projekten muss eine Person gefunden werden, die überwiegend als Qualitätsmanager arbeitet. In noch größeren Projekten muss entsprechend mehr Manpower in das Qualitätsmanagement investiert werden, bis sich vielleicht eine ganze Qualitätsabteilung mit spezialisierten Tools dieser Aufgabe widmet. Qualitätsmanagement kann also durchaus auch leichtgewichtig durchgeführt und skaliert werden.

Fazit

Zwei Dinge machen Softwarequalität im Projektalltag (be-)greifbar: Erstens ein intuitiv verständliches, ganzheitliches Qualitätsmodell, das alle wesentlichen Eigenschaften von Software nachvollziehbar strukturiert, sodass aus einer Flut von Qualitätsmerkmalen eine praxisnahe Übersicht und hilfreiche Checkliste entsteht. Zweitens ein gezielter, einheitlicher Qualitätssicherungsprozess, der alle Akteure miteinschließt. Dieser Prozess beginnt bei der Unternehmensleitung mit der Formulierung strategischer Qualitätsziele, führt über das Qualitätsmanagement (dem Zyklus von kontinuierlicher Verbesserung) und endet bei den Qualitätsdetails, die die ausführenden Akteure realisieren.

Bei diesem Qualitätssicherungsprozess hilft die Verwendung eines ganzeinheitlichen Qualitätsmodells. Es ermöglicht, alle Akteure in diesem Prozess anzusprechen und so zu einem einheitlichen Qualitätsverständnis zu kommen. Auf diese Weise wird Softwarequalität steuerbar und dieser abstrakte, diffuse Begriff wird zu etwas, das sich im Projekt und im Produkt anfassen und begreifen lässt. ||

Der Autor

|| Dr. Reik Oberrath
(R.Oberrath@iks-gmbh.com)
ist als IT-Berater bei der IKS Gesellschaft für Informations- und Kommunikationssysteme tätig. In der Softwareentwicklung legt er großen Wert auf Automation, Kommunikation sowie Verständlichkeit von Code und Prozessen. Er ist Mitbegründer der Knowledge Base „Clean Coding Cosmos“.