Zeitschrift: Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association suisse des électriciens, de l'Association des entreprises électriques suisses

Herausgeber: Schweizerischer Elektrotechnischer Verein ; Verband Schweizerischer Elektrizitätsunternehmen

Band: 79 (1988)

Heft: 17

Artikel: Software-Qualitätssicherung : eine Einführung

Autor: Rudin, H.

DOI: https://doi.org/10.5169/seals-904075

Nutzungsbedingungen

Die ETH-Bibliothek ist die Anbieterin der digitalisierten Zeitschriften auf E-Periodica. Sie besitzt keine Urheberrechte an den Zeitschriften und ist nicht verantwortlich für deren Inhalte. Die Rechte liegen in der Regel bei den Herausgebern beziehungsweise den externen Rechteinhabern. Das Veröffentlichen von Bildern in Print- und Online-Publikationen sowie auf Social Media-Kanälen oder Webseiten ist nur mit vorheriger Genehmigung der Rechteinhaber erlaubt. Mehr erfahren

Conditions d'utilisation

L'ETH Library est le fournisseur des revues numérisées. Elle ne détient aucun droit d'auteur sur les revues et n'est pas responsable de leur contenu. En règle générale, les droits sont détenus par les éditeurs ou les détenteurs de droits externes. La reproduction d'images dans des publications imprimées ou en ligne ainsi que sur des canaux de médias sociaux ou des sites web n'est autorisée qu'avec l'accord préalable des détenteurs des droits. En savoir plus

Terms of use

The ETH Library is the provider of the digitised journals. It does not own any copyrights to the journals and is not responsible for their content. The rights usually lie with the publishers or the external rights holders. Publishing images in print and online publications, as well as on social media channels or websites, is only permitted with the prior consent of the rights holders. Find out more

Download PDF: 16.07.2025

ETH-Bibliothek Zürich, E-Periodica, https://www.e-periodica.ch

Software-Qualitätssicherung – eine Einführung

H. Rudin

Bereits in den siebziger Jahren wurden unter dem Sammelbegriff Software-Engineering Konzepte für eine ingenieurmässige Softwareentwicklung erarbeitet. In der Praxis lässt die Anwendung dieser Konzepte jedoch noch einiges zu wünschen übrig. Hier setzen die modernen Software-Qualitätssicherungssysteme an, die derzeit in vielen Firmen in Aufbau sind. Diese können allerdings – wie erste Erfahrungen zeigen – nur zu Erfolg führen, wenn sie sorgfältig eingeführt werden und die vorhandene Firmenkultur angemessen berücksichtigen.

C'est déjà dans les années 70 qu'ont été élaborés, sous le générique ingénierie des logiciels, les concepts pour une ingénierie du développement de logiciels. Dans la pratique, l'utilisation de ce concept laisse cependant encore à désirer. C'est là qu'interviennent les systèmes d'assurance de la qualité modernes pour logiciels, actuellement en voie de mise sur pied dans de nombreuses firmes. Comme le démontrent les premières expériences, ces systèmes ne peuvent réussir que s'ils sont introduits soigneusement et tiennent convenablement compte de la culture d'entreprise en place.

Adresse des Autors
Hans Rudin, Dipl. El.-Ing. ETH,
Standard Telephon & Radio AG,
Friesenbergstrasse 75, 8055 Zürich.

Der Anteil am Bruttosozialprodukt der USA für alle Bereiche der Datenverarbeitung betrug bereits 1980 5% und dürfte bis 1990 auf mehr als 12% steigen. Software ist zu einem wichtigen Faktor unserer Industriegesellschaft geworden. So hängen auch sicherheitsrelevante Systeme, z.B. der Luftverkehr, zunehmend von Software ab. Zwischen Bedeutung und Reife der Softwaretechnologie besteht jedoch ein Ungleichgewicht. Hinter dem Schlagwort der Softwarekrise, das Ende der sechziger Jahre geprägt wurde, stehen Softwareprodukte, deren mangelnde Qualität zu teuren Ausfällen mit enormen Folgekosten führte sowie Softwareprojekte, die wegen massiver Kosten- und Terminüberschreitungen zum Teil nie beendet wurden. Schätzungen beziffern den Anteil der abgebrochenen Softwareprojekte auf 20%.

In den siebziger Jahren suchte man unter dem Begriff Software-Engineering die Erstellung komplexer Software auf eine ingenieurmässige Basis zu stellen. Software-Engineering beinhaltet Prinzipien, Methoden und Sprachen zur systematischen Entwicklung sowie Wartung von Software. Heute steht uns ein ganzer Baukasten von Methoden und Sprachen zur Verfügung, um Software nach diesen Prinzipien zu entwickeln. Langsam erscheinen auch brauchbare Werkzeuge auf dem Markt, welche diese Methoden wirksam unterstützen. In der Praxis aber lässt die Anwendung des Software-Engineering zu wünschen übrig. Zwar herrscht Einigkeit, dass man nach den Methoden des Software-Engineering vorgehen müsste, aber im Projektalltag unter Termindruck, sich ändernden Anforderungen und fehlender Unterstützung durch Werkzeuge werden diese guten Vorsätze nur allzuoft fallen gelassen. Genau hier setzt die Qualitätssicherung an. In Figur 1 ist die Entwicklung dieser Disziplin in vier Stufen nach Godfrey [1] dargestellt.

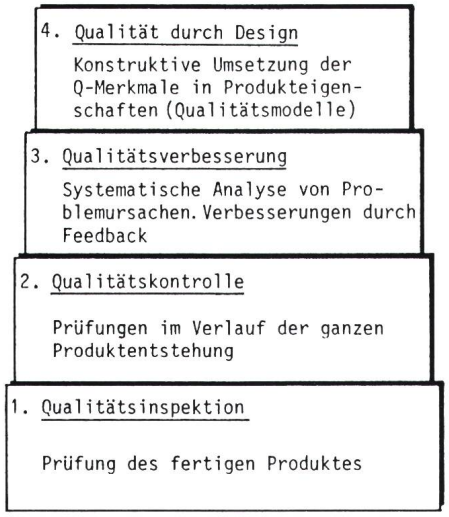
Qualität und Software

Nach DIN 55 350 ist *Qualität* die *Gesamtheit von Eigenschaften und Merkmalen eines Produktes oder einer Tätigkeit, die sich auf die Eignung zur Erfüllung gegebener Erfordernisse beziehen.*

Diese Definition lässt sich auch auf Software anwenden, und zwar sowohl auf die Tätigkeit der Softwareentwicklung als auch auf deren Resultat, das Softwareprodukt.

Die Erfordernisse, welche ein Softwareprojekt zu erfüllen hat, sind neben der Einhaltung von Terminen und Kosten das Erreichen der Sachziele, d.h. die Erfüllung der Produktanforderungen. Ein Mass dafür ist die Produktqualität. Dieser Zusammenhang zwischen Projekt- und Produktqualität ist in Figur 2 veranschaulicht.

Unter *Software* verstehen wir nach [4] die *Gesamtheit der notwendigen Programme, Daten, Abläufe, Regeln und jeglicher dazugehöriger Dokumen-*



Figur 1 Die Entwicklungsstufen der Qualitätssicherung nach Godfrey [1]

tation für die Nutzung eines Rechnersystems.

Die Qualität eines Softwareproduktes wird neben der Erfüllung von funktionalen Anforderungen, Mengen- und Leistungsanforderungen durch Merkmale bestimmt wie:

- Korrektheit
- Zuverlässigkeit
- Sicherheit
- Benutzerfreundlichkeit
- Wartbarkeit
- Wiederverwendbarkeit
- Flexibilität
- Portabilität

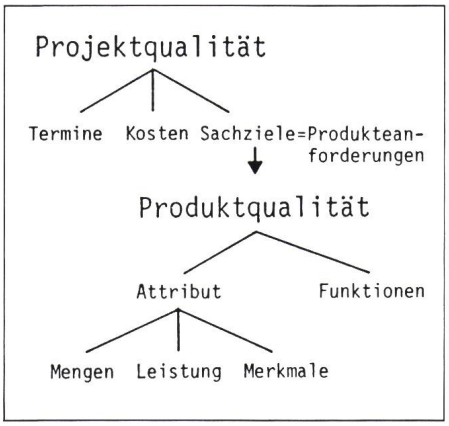
Will man eine objektive Aussage über die Qualität eines Softwareproduktes machen, so müssen diese Merkmale in den Produktanforderungen spezifiziert werden, und zwar in Form von messbaren Kenngrössen (vergleiche Abschnitt über Messbare Kenngrössen für Software-Qualitätsmerkmale.

Software ist immateriell!

Die Qualität von Software wird vor allem durch ihre *immaterielle Natur* beeinflusst. Software gehorcht keinerlei physikalischen Gesetzen (wie z.B. der Schwerkraft). Daraus folgt eine Reihe von Eigenheiten:

1. Software kann wie alle immateriellen Güter *Nutzen nur durch Kommunikation* entfalten. Dabei muss Software sowohl mit dem Menschen (Entwicklungsingenieur, Benutzer) als auch mit der Maschine (Computer) kommunizieren. Die Wahl geeigneter Sprachen zur Spezifikation, Beschreibung, Codierung und zur Gestaltung der Benutzeroberfläche ist ein wichtiger Einflussfaktor der Softwarequalität.

2. Die *Funktion von Software ist nicht stetig*. Eine Brücke, die das Gewicht von zehn Lokomotiven trägt, wird unter der Last von drei Lokomotiven nicht zusammenbrechen. Erzeugt ein Programm bei zwei Eingabewerten korrekte Ausgaben, so ist nicht bewiesen, dass ein dazwischenliegender Wert auch ein korrektes Resultat erzeugt. Interpolation und Extrapolation sind bei Software nicht ohne weiteres möglich. Daraus folgt: *Software ist - ausser in trivialen Fällen - nicht vollständig prüfbar*. Wohl muss man Software testen, um Fehler zu finden und Aussagen über die Qualität zu machen, aber einen Beweis für die Korrektheit kann man damit nicht antreten, denn testen lässt sich immer nur ein geringer Bruchteil aller möglichen Fälle. Dieser Sachverhalt hat zur Erkenntnis geführt, dass man Qualität nicht in ein Softwareprodukt «hineintesten» kann, sondern dass sie hinein entwickelt werden muss. Die Qualitätssicherung von Software hat also bei der Entwicklung anzusetzen.



Figur 2 Zusammenhang zwischen Projekt- und Produktqualität nach Frühauf [2]

3. *Die Replikation von Software ist trivial*. Ein einfacher Vorgang erzeugt eine Kopie, die vom Original nicht zu unterscheiden ist. Das führt zu Konsistenzproblemen. Konfigurationsmanagement ist daher ein wichtiger Bestandteil der Software-Qualitätssicherung, aber das traditionelle Gebiet der Qualitätssicherung, die Produktion, hat bei Software keine entsprechende Bedeutung. Software ist ein Entwicklungsprodukt.

4. *Software unterliegt keiner Abnutzung*. Ausfälle sind immer die Folge von Fehlern. Unglücklicherweise hat sich der Begriff *Wartung* auch für Software eingebürgert. Wartung bei Software ist aber nie ein Wiederherstellen des ursprünglichen Zustandes, sondern die Korrektur von Fehlern, die schon immer im Produkt vorhanden waren. Üblicherweise werden unter den Begriff Wartung auch Verbesserungen und Anpassungen des Programmes subsummiert. Als Faustregel gilt, dass für alle Belange der Wartung eines Softwareproduktes ein doppelt so hoher Aufwand zu leisten ist, wie für dessen Entwicklung.

Auch der Begriff der *Zuverlässigkeit* macht bei Software Schwierigkeiten. Software verhält sich deterministisch. Erst die Benutzung zusammen mit statistisch verteilten Eingabewerten erlaubt Aussagen über die Zuverlässigkeit einer Softwareanwendung, nicht aber der Software an und für sich.

Software-Qualitätssicherung ist kein Sonderfall!

Trotz dieser Eigenheiten weisen Softwareprodukte mehr Gemeinsamkeiten als Unterschiede mit anderen Produkten menschlicher Tätigkeiten auf. Es gibt daher keinen Grund, in der Qualitätssicherung von Software die Grundsätze der traditionellen Qualitätssicherung ausser acht zu lassen und dabei die diskutierten Charakteristika als Entschuldigung heranzuziehen.

Messbare Kenngrössen für Software-Qualitätsmerkmale

Da geeignete Kenngrössen (Metriken) fehlen, ist die Messung von Qualitätsmerkmalen ein Problem. Objektive Aussagen über Softwarequalität sind aber nur durch Messungen zu erhalten.

Was man nicht messen kann, das kennt man nicht! (Kelvin)

In den letzten Jahren wurde viel Aufwand getrieben, geeignete Kenngrössen für die Komplexität von Software zu entwickeln. Ein Beispiel dafür ist die Zyklometrische Zahl $V(G)$ nach McCabe [7]:

$$V(G) = E - n + 2p$$

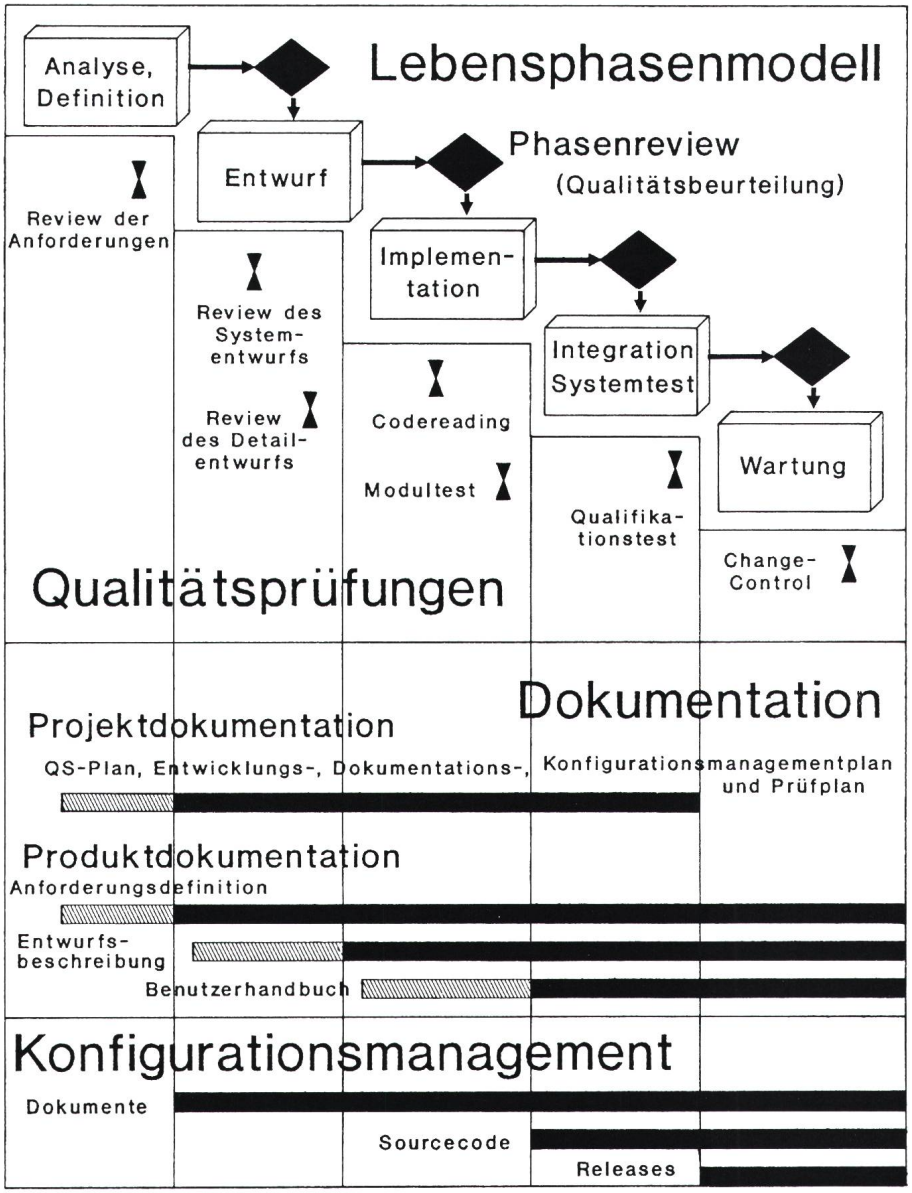
mit Anzahl Kanten E, Anzahl Knoten n und Anzahl verbundener Komponenten p. Sie repräsentiert die Anzahl linear unabhängiger Wege durch den Kontrollflussgraphen G eines Programmes.

Messungen von Kenngrössen sollte man nicht auf das Produkt beschränken; es ist auch sinnvoll, Kenngrössen des Projektes zu messen. Eine einfache Art von Messungen sind Zählungen: Eine Kenngrösse für das Qualitätsmerkmal *Portabilität* eines Softwareproduktes ist zum Beispiel der Anteil der Module, welche Betriebssystemaufrufe enthalten. Für das Projekt ist der Anteil der Dokumente, die einer Review unterzogen wurden, eine Kenngrösse für das Qualitätsmerkmal *Frühe Fehlererkennung*.

In der Literatur wird das Thema *Metriken* viel diskutiert; dem steht eine sehr bescheidene Anwendung in der Praxis gegenüber. Die Weiterentwicklung der Software-Qualitätssicherung hängt aber massgeblich davon ab, wieweit es gelingt, die Qualität von Softwareprodukten und -projekten zu messen.

Software-Qualitätssicherung durch planerische und analytische Massnahmen im Projekt

Solange jedes Softwareprojekt nach eigenen Regeln abgewickelt wird, ist schwierig festzustellen, was die Qualität beeinflusst und wie sie zu kontrollieren ist. Ein naheliegender Ansatz zur Software-Qualitätssicherung konzentriert sich daher auf das Projekt. Ein nach Richtlinien der Qualitätssicherung ablaufender Entwicklungsprozess soll in der Folge die Qualität des Softwareproduktes sicherstellen. In Figur 3 sind vier Grundelemente dieser projektorientierten Software-Qualitätssicherung skizziert und werden im folgenden kurz beschrieben. Für eine ausführliche Diskussion sei auf [5; 6] verwiesen.



Figur 3 Grundelemente der projektorientierten Software-Qualitätssicherung in vereinfachter Darstellung

Software-Lebensphasenmodell

Die Unterteilung eines Softwareprojektes in zeitlich aufeinanderfolgende Phasen verfolgt das Ziel, den Entwicklungsprozess überschaubar und prüfbar zu halten. Für jede Phase sind die zu erbringenden Resultate im voraus festzulegen, und am Ende ist in einer *Phasenreview* zu überprüfen, ob diese Ziele erreicht wurden. Hier ist das Management angesprochen. Seine Aufgabe ist, über die Fortsetzung des Projektes bzw. über eine Kurskorrektur zu entscheiden.

In letzter Zeit wird unter Stichworten wie *Rapid Prototyping, Successive Versions Model* [7] und *Continuous Delivery* Kritik am klasischen Phasenmodell geübt. Meiner Ansicht nach stehen die neuen Konzepte nicht wirklich im Widerspruch zum Prinzip des Phasenmodells. Der Gegensatz ist vielmehr auf dessen zu enge Interpretation zurückzuführen. In der Regel werden die Phasen nach ihren Haupttätigkeiten benannt, was aber nicht heissen soll, dass entsprechende Aktivitäten nicht auch in anderen Phasen notwendig sind. Rückgriffe auf Haupttätigkeiten früherer Phasen sind in der Praxis nicht zu vermeiden. Ziel ist aber, diese Rückgriffe möglichst auf die benachbarte Phase einzuschränken, ihre Anzahl zu begrenzen und sie in einem geordneten Verfahren abzuwickeln. Anpassungen des Phasenmodells an das Anwendungsgebiet, die eingesetzte Technologie und die Firmenkultur sind in der Praxis unumgänglich.

Qualitätsprüfungen

Hier geht es nicht nur um das Testen, d.h. die Prüfung des Codes durch seine Ausführung, sondern um die Überprüfung von Tätigkeiten und Arbeitsergebnissen während des ganzen Entwicklungsprozesses. Neben Messungen, die im Abschnitt Kenngrössen behandelt wurden, sind *technische Reviews* ein wirksames Instrument der Qualitätssicherung. Statistiken zeigen, dass sich der Aufwand für Reviews im Durchschnitt mit einer Ersparnis vom doppelten Betrage bei der Fehlerkorrektur auszahlt. Je früher ein Fehler im Lebenszyklus entdeckt wird, desto kleiner sind die Kosten, um diesen Fehler zu beheben. Reviews sind daher insbesondere in der Analyse- und Systementwurfsphase von grossem Nutzen.

Erfolgreich sind Reviews nur, wenn gewisse Spielregeln eingehalten werden: In einer gut vorbereiteten Reviewsitzung mit kompetenten Teilnehmern wird der Reviewgegenstand unter der Leitung eines Moderators Schritt für Schritt besprochen. Dabei werden alle Fehler, Probleme und Un-

klarheiten protokolliert, ohne dass Lösungsmöglichkeiten ausdiskutiert werden oder dass sich der Autor zu verteidigen hätte. Wichtig ist, dass der Reviewgegenstand und nicht der Autor kritisch geprüft wird. Die Weiterbearbeitung der gefundenen Fehler muss in der Folge geregelt sein.

Dokumentation

Die im Verlaufe der Softwareentwicklung entstehende Dokumentation lässt sich in Projekt- und Produktdokumente unterteilen. *Projektdokumente* dienen der Planung und Abwicklung des Softwareprojektes, ihre Gültigkeit ist auf die Projektdauer beschränkt. *Produktdokumente* beschreiben das entstehende Softwareprodukt. Nach unserer Definition sind sie integraler Bestandteil der Software.

Dokumentationsnormen mit Musterdokumenten für Produkt- und insbesondere Projektdokumente helfen den Software-Entwicklungsprozess zu standardisieren und übersichtlich zu halten; sie haben die Funktion von Checklisten. Auch Vorgaben, wann ein Dokument im Lebensphasenmodell zu entstehen hat, leisten einen Beitrag zur Qualitätssicherung. Erarbeitet man zum Beispiel den *Qualifikationstestplan* bereits beim Festlegen der Anforderungen, so führt dies zu messbaren Spezifikationen und hilft Lücken in der Anforderungsdefinition zu finden.

Speziell zu erwähnen ist noch der *Qualitätssicherungsplan*, der alle qualitätssichernden Massnahmen mit zugehörigen Verantwortlichkeiten in einem Projekt festlegt. Eine brauchbare Vorlage für einen solchen Plan bietet die IEEE Norm 730 [3].

Konfigurationsmanagement

Softwarepakete setzen sich aus einer Vielzahl von Elementen (Module, Dokumente) zusammen, die untereinander konsistent sein müssen. Änderungen sind zwar rasch durchgeführt, die Gefahr des Konsistenzverlustes aber ist dabei gross. Darum ist jedes Element in eine Konfigurationsverwaltung aufzunehmen. Der Einsatz von Softwarewerkzeugen ist dabei sehr zu empfehlen. Im Konfigurationsplan muss festgelegt werden, welche Elemente (Dokumente, Sourcecode) zu welchem Zeitpunkt in die Konfigurationsverwaltung aufgenommen werden und wie bei Änderungen vorzugehen ist.

Systemelement	Wichtige Anforderungen in Stichworten
Führungsaufgaben	
Organisation	QS-Organisation festgelegt. Von Produktentstehung unabhängiger Qualitätsleiter mit direktem Zugang zur Geschäftsleitung.
QS-Handbuch	QS-System nach Handbuch eingeführt und aktualisiert. Erklärung der Geschäftsleitung zur Einhaltung der Norm, QS-Organigramm, QS-Ablaufregelungen gemäss dieser Tabelle, Audit-Plan.
Software-QS-Audit	Auf der Grundlage des Auditplans, der die zu überprüfenden Funktionen und die Häufigkeit der Auditdurchführung festhält, werden in den Audits die Einhaltung und die Zuverlässigkeit des QS-Systems überprüft.
Verbesserungsmassnahmen	Systematische Analyse und Behebung von Fehlerursachen und Überprüfung der Verbesserungsmassnahmen auf ihre Wirksamkeit.
Projekt-QS-Plan	Er ist vor der Entwicklung zu erstellen. Änderungen unterliegen der Genehmigungspflicht. Er muss Entwicklungs-, Konfigurations-, Dokumentations- und Prüfplan enthalten.
QS-Ablaufregelungen	
Produktanforderungen	Vertragsprüfung unter Mitwirkung der SW-Entwicklung. Detaillierte Spezifikation der Funktionsanforderungen und Merkmale.
Entwicklung	Gemäss Entwicklungsplan. Aufteilung in kontrollierbare Arbeitspakete. Darstellung des SW-Lebenszyklus. SWQS für zugekaufte SW.
Konfigurationsverwaltung	Regelungen zur Identifikation und Verwaltung von SW-Komponenten und deren Versionen. Rückverfolgbarkeit. Änderungsverfahren und Zugriffsschutz.
SW-Dokumentation	Vollständig und aktuell gemäss Dokumentationsplan. Festlegung der Verantwortung für Erstellung, Prüfung, Freigabe und Änderung von Dokumenten.
Prüfungen	Nach Prüfplan in geeigneter, validierter Testumgebung mit Bewertung der Resultate.
Regeln, Richtlinien und Verfahren	Für Entwicklung, Dokumentation und Prüfung mit Überwachung und Anpassung an ihre Anwendung.
Beistellungen vom Auftraggeber	Geeignete QS-Massnahmen und Verantwortlichkeiten.
Zulieferungen	SW-Ersteller ist verantwortlich, dass zugelieferte SW den spezifizierten Anforderungen entspricht. Er überprüft QS-System des Unterlieferanten.
Behandlung von Fehlern	Dokumentation und Behebung von Fehlern in allen SW-Komponenten unter Konfigurationsverwaltung.
Qualitätsnachweise	Zeigen, dass spezifizierte Anforderungen erfüllt und QS-Massnahmen durchgeführt sind. Regelung der Aufbewahrung.

Tabelle I Zusammenstellung von QS-Anforderungselementen (Stufe A) aus SAQ 222 [4]

Software-Qualitätssicherungssysteme

Ein Qualitätssicherungssystem umfasst die Integration der ersten drei Stufen im Entwicklungsschema der Figur 1. Es hat zum Ziel, für die systematische Anwendung der beschriebenen Grundelemente in einer Firma zu sorgen. Zu einem Qualitätssicherungssystem gehören noch weitere Elemente. Insbesondere muss die Wirksamkeit des Software-Qualitätssicherungssystems in Systemaudits regelmässig überprüft werden. Diese werden, im Gegensatz zu Reviews, immer von einer unabhängigen Stelle durchgeführt.

In der Schweiz existiert eine Empfehlung der Schweizerischen Arbeitsgemeinschaft für Qualitätssicherung (SAQ), welche Anforderungen an Software-Qualitätssicherungssysteme stellt [4]. Die Tabelle I führt die Anforderungselemente in Stichworten auf. Im Trend der Internationalisierung von Normen wurde darauf ver-

zichtet, mit dieser Empfehlung eine schweizerische Norm anzustreben. In der ISO (International Standard Organisation) wird gegenwärtig eine entsprechende Norm ausgearbeitet, die inhaltlich sehr ähnliche Forderungen stellt und sich an die allgemeine Qualitätssicherungsnorm ISO 9001 anlehnt.

Software-Qualitätssicherung durch konstruktive Massnahmen am Produkt

Der projektorientierte Ansatz des Software-Qualitätssicherungssystems bildet den Rahmen, welcher durch planerische Massnahmen Qualität ermöglicht, bzw. durch analytische Massnahmen Abweichungen zu erkennen hilft. Die Mittel, um Qualität konstruktiv in ein Produkt hineinzuentwickeln, sind die Prinzipien, Methoden, Sprachen und Werkzeuge des Software-Engineering. Für eine Übersicht des umfangreichen Gebietes sei auf das Buch von *R. Fairley* [7] verwiesen. Hier sei aus der Sicht der Software-Qualitätssicherung nur eine kurze Gegenstandsbestimmung vorgenommen.

Prinzipien

Im Bereich des Software-Engineerings haben sich eine ganze Reihe von allgemeinen Handlungsgrundsätzen entwickelt, wie zum Beispiel die hierarchische Modularisierung eines Softwaresystems, deren Befolgung einen Beitrag zur Qualitätssicherung leistet. Dabei ist zu beachten, dass die Prinzipien des Software-Engineering nicht unabhängig von der sich rasch entwickelnden Technologie sind. So war zum Beispiel früher das speichereffiziente Programmieren ein vorherrschendes Prinzip, heute hat es seine Bedeutung zugunsten des übersichtlichen, strukturierten Programmierens verloren. Wie man in der Praxis sieht, macht dieses Umdenken Schwierigkeiten. Ähnliche Probleme stehen uns beim vermehrten Einsatz nichtprozeduraler Sprachen bevor.

Methoden

Methoden sind auf Prinzipien beruhende Vorgehensweisen zur Erreichung eines Zieles. Während sich frühere Software-Engineering-Methoden vor allem auf den Programmierstil konzentrierten (z.B. strukturierte Programmierung), gilt heute das Interesse der Spezifikation und dem Entwurf von Softwaresystemen. In der Praxis wird gerade in diesen frühen Phasen oft sehr wenig systematisch gearbeitet, der Grundstein für die Qualität des resultierenden Produktes wird aber hier gelegt.

Die *Qualitätsmodellierung* ist eine Software-Engineering-Methode, welche direkt beim Begriff der Qualität ansetzt. Dabei werden die Qualitätsmerkmale der Anforderungsdefinition in quantifizierbare Basiseigenschaften des Produktes aufgeschlüsselt und für jede Basiseigenschaft messbare Kenngrössen und Zielvorgaben festgelegt. Die Basiseigenschaften werden im Entwurf und in der Implementation explizit in das Produkt hineinentwickelt und die entsprechenden Kenngrössen im Verlauf der Entwicklung gemessen.

Sprachen

Programmiersprachen setzen den Systementwurf in eine für den Computer interpretierbare Form um. In der Anfangszeit waren die Sprachen maschinenorientiert, und der Mensch musste sich der Maschine anpassen. Programmieren und vor allem das Ändern der schwer durchschaubaren Programme war eine mühsame, fehlerträchtige Arbeit. Hier hat sich die Situation sehr stark gebessert. Moderne Programmiersprachen, wie zum Beispiel Modula-II und Ada, haben sich von der Maschine gelöst, ermöglichen Algorithmen aus Anwendersicht zu formulieren und erlauben, ja erzwingen sogar teilweise die Anwendung moderner Software-Engineering-Prinzipien.

Weniger entwickelt sind die sprachlichen Mittel für die früheren Phasen der Softwareentwicklung. Anforderungen an Softwaresysteme werden heute zumeist noch in natürlicher Sprache festgelegt, und auch Entwurfsbeschreibungen enthalten neben einigen informalen Graphiken in der Regel vor allem Prosa. Natürliche Sprache ist für diesen Zweck aber zu unpräzise. Eine Verbesserung der Situation ist durch den Einsatz von halbformalen Sprachen zu erwarten, deren Aufbau und Syntax formalen Regeln genügt und deren Semantik durch natürliche Sprachelemente bestimmt wird.

Werkzeuge

Werkzeuge dürfen nie die Kenntnis und das Verständnis der zugrundeliegenden Methoden ersetzen. Ist diese Voraussetzung erfüllt, leisten nach den Prinzipien der Qualitätssicherung ausgewählte Werkzeuge einen wichtigen Beitrag zur Qualität von Projekt und Produkt. Insbesondere erlauben Werkzeuge ohne wesentlichen Zusatzaufwand die Erfassung von Qualitätskennzahlen für Projekt und Produkt. In der Praxis ist die Werkzeugunterstützung in der Implementationsphase traditionell stark (Compiler, Linker, Debugger). Eine Qualitätsverbesserung ist heute vor allem durch den Werkzeugeinsatz in den frühen Phasen zu erreichen. Seit einiger Zeit sind auch Werkzeuge zur eigentlichen Qualitätsprüfung von Software verfügbar. Statische und dynamische Analysatoren erlauben die Ermittlung von Kennzahlen, wie z.B. die Testabdekkung.

Software-Qualitätssicherung in der Praxis

Die beiden Ansätze zur Software-Qualitätssicherung, der Projektansatz des Qualitätssicherungssystems und der Produktansatz des Software-Engineering, sind keine Alternativen. In der industriellen Praxis müssen beide Ansätze zur Software-Qualitätssicherung verfolgt werden. Will man die Anwendung von Software-Engineering nicht dem Zufall überlassen, so ist der Aufbau eines Software-Qualitätssicherungssystems unabdingbare Voraussetzung für Softwarequalität. Anderseits bestimmt letztlich die Anwendung des Software-Engineering die Qualität eines Softwareproduktes. Die Fähigkeit der Software-Ingenieure, Kreativität in der Anwendung des Software-Engineerings im systematischen Rahmen der Qualitätssicherung zu vollbringen, ist der Weg zu Qualitätssoftware.

Literatur

[1] *A. B. Godfrey:* The history and evolution of quality in AT&T. AT&T Techn. J. 65(1986)2, p. 9...20.

[2] *K. Frühauf:* Grundsätze zur Software-Qualitätssicherung. Techn. Rdsch.-(1988)8, S. 58...63.

[3] Standard for software quality assurance plans. ANSI/IEEE Standard 730-1984.

[4] Anforderungen an Qualitätssicherungssysteme von Software-Erstellern. SAQ-Empfehlung 222. Bern, Schweizerische Arbeitsgemeinschaft für Qualitätsförderung, 1987.

[5] Software-Qualitätssicherung - Aufgaben, Möglichkeiten, Lösungen. DGQ-NTG-Schrift Nr. 12-51. Berlin/Offenbach, VDE-Verlag, 1986

[6] *K. Frühauf, J. Ludewig* und *H. Sandmayr:* Software-Projektmanagement und -Qualitätssicherung. Zürich, Verlag der Fachvereine an der ETH, 1988.

[7] *R. Fairley:* Software engineering concepts. New York a.o., McGraw-Hill, 1985.