Software-Qualitätsmodelle

Udo Kelter

05.05.2007

Zusammenfassung dieses Lehrmoduls

Software soll generell eine hohe Qualität haben. Diese allgemeine Anforderung kann in konkreten Einzelfällen eine ganz verschiedene Bedeutungen haben. Im Rahmen der Analyse der Anforderungen an ein System müssen daher die allgemeinen Qualitätsanforderungen heruntergebrochen werden auf möglichst objektiv prüfbare Merkmale. Zur Formulierung der Qualitätsziele benutzt man Qualitätsmodelle. Dieses Lehrmodul stellt u.a. das ISO-Qualitätsmodell vor und diskutiert, wann und wie Qualitätsziele im Entwicklungsprozeß überwacht werden.

Vorausgesetzte Lehrmodule: - keine -

Stoffumfang in Vorlesungsdoppelstunden: 1.0

Inhaltsverzeichnis

1 Motivation und Einordnung

Einen Mangel an Qualität hat vermutlich jeder, der Software benutzt, schon einmal sehr deutlich empfunden. Software-Fehler und andere Qualitätsmängel verursachen jährlich materielle Schäden in Milliardenhöhe und kosten schlimmstenfalls sogar Menschenleben. Daher ist es eine Binsenweisheit bzw. Standardanforderung, daß Software qualitativ "gut" oder "hochwertig" sein sollte.

Was diese pauschale Anforderung genau bedeutet, ist aber keineswegs klar und ohne weiteres operativ umsetzbar. Wenn in einem Entwicklungsvertrag zwischen einem Softwarehaus und einem Kunden vereinbart wäre, daß eine "qualitativ hochwertige" Software geliefert wird, wie könnte dies bei der Abnahme geprüft werden und ggf. vor Gericht bewiesen werden, daß die Software nicht hochwertig ist? Benötigt werden daher möglichst objektiv prüfbare Kriterien, die in ihrer Summe die Qualität eines Softwaresystems belegen.

Die Definitionsprobleme beginnen bereits beim Begriff Qualität, der eine sehr große thematische Spannbreite hat. Der einschlägige DIN-Standard definiert **Softwarequalität** als *die Gesamtheit der Merkmale und Merkmalswerte eines Softwareprodukts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen.* Relevant für die Qualität sind sowohl funktionale wie nichtfunktionale Eigenschaften eines Softwaresystems. Da funktionale Eigenschaften i.d.R. durch Daten- oder Funktionsmodelle (z.B. in Form von UML-Modellen) spezifiziert werden, versteht man unter Softwarequalität überwiegend nichtfunktionale Eigenschaften.

Qualität kostet Geld – der Aufwand für die erzielte Qualität muß angemessen sein. Eine pauschale Forderung nach beliebig hoher Qualität ist daher nicht sinnvoll und praktisch nicht erfüllbar. Im Rahmen des Requirements Engineering muß daher ausgehandelt werden, wie wichtig einzelne Qualitätsaspekte sind und wieviel Aufwand dort investiert werden soll.

1.1 Qualitätsziele

Die vorstehenden Betrachtungen führen aus mehreren Perspektiven heraus zur Erkenntnis, daß die gewünschten Qualitätseigenschaften explizit formuliert und schriftlich notiert werden müssen.

Unter **Qualitätszielen** (oder **Qualitätsanforderungen**) verstehen wir Spezifikationen von Mindestwerten von Qualitätsmerkmalen; Beispiele:

- Das System braucht maximal 100 MB Plattenplatz.
- Das System hat höchstens 2 Std. Ausfallzeit pro Jahr.
- Die Bedienungsanleitung ist "leicht verständlich".
- Die Dialoge des Systems bieten immer eine "klare" Orientierung an.

Die vorstehenden Beispiele illustrieren die typischen Probleme bei der Formulierung von Qualitätszielen:

- Die zugrundeliegenden Qualitätsmerkmale bzw. darin benutzte Begriffe sind nicht immer klar. Abhilfe schafft hier ein Begriffskatalog, auch als **Qualitätsmodell** bezeichnet, der entweder einem Standardkatalog entspricht oder der für ein Produkt bzw. Projekt individuell entwickelt wird.
- Die Wertebereiche (Skalen) der Qualitätsmerkmale und die Meßbarkeit differieren erheblich: im einfachsten Fall hat man numerische Werte, die mit geringem Aufwand präzise durch verfügbare Werkzeuge bestimmt werden können. Das andere Extrem sind subjektive Wertungen auf unscharf definierten Skalen.
- Die einzelnen Qualitätsmerkmale sind nicht disjunkt. Abstraktere Qualitätsmerkmale (System ist "leicht bedienbar") stellen den kondensierten Gesamteindruck dar, der aufgrund vieler Detailmerkmale entsteht. Bei den abstrakten Qualitätsmerkmalen ist das Aggregieren der Einzelwertungen nicht trivial, und die Skala für das Gesamtergebnis ist i.d.R. unscharf.

1.2 Qualitätssicherung

Qualitätsziele werden leicht mit Qualitätssicherung verwechselt.

Unter **Qualitätssicherung** verstehen wir alle Maßnahmen, die dazu dienen, die angestrebten Qualitätsziele zu erreichen. Die Maßnahmen können präventiv oder reaktiv sein, an ein einzelnes Projekt gebunden sein oder zeitlich und organisatorisch darüber hinausreichen.

Qualitätsziele werden vor allem in der Analysephase projektbezogen formuliert und zwischen Auftraggeber und Auftragnehmer ausgehandelt, beide Rollen sind intensiv involviert. Im Gegensatz dazu betrifft die Qualitätssicherung den Entwicklungsvorgang[1]. Bei der vereinfachenden Annahme, daß dem Auftraggeber der Entwicklungsvorgang gleichgültig ist und nur das Endresultat zählt, ist die Qualitätssicherung nur für den Auftraggeber relevant.

Im Rahmen der Qualitätssicherung muß nicht nur die Qualität des Endprodukts, sondern auch die Qualität von relevanten **Zwischenprodukten** kontrolliert werden. Dies ist vor allem dann zu beachten, wenn die Zwischenprodukte nicht direkt oder indirekt im Endprodukt enthalten sind (z.B. als mitgelieferte Entwicklungsdokumentation) und deshalb ohnehin der Qualitätssicherung unterliegen würden.

1.3 Qualität des Produkts, Projekts bzw. Vorgehensmodells

Die Qualität eines Softwaresystems wird eigentlich nur durch das System selbst bestimmt bzw. allgemeiner gesagt durch das **Produkt**, worunter wir neben der laufenden Software auch Benutzer- und System-Dokumentation auf Papier oder anderen Distributionsmedien und beliebige weitere Produktanteile verstehen[2].

Wie das Produkt entstanden ist, spielt theoretisch keine Rolle. Die Erfahrung lehrt andererseits, daß hohe Qualität nur durch entspre-

[1] Diese Definition gilt nur im Kontext der Softwareentwicklung. In der produzierenden Industrie versteht man unter Qualitätssicherung vor allem Maßnahmen, die Fehler in der (Serien-) Produktion von Gütern entdecken sollen bzw. zu vermeiden helfen.

[2] Noch weiter betrachtet müßte auch die Unterstützung durch den Hersteller, Vorhandensein von Benutzerforen u.ä. herangezogen werden, worauf wir hier nicht eingehen.

chende qualitätsorientierte Entwicklungsmethoden erzielt wird (nicht nur bei Softwareprodukten). Hinzu kommen die Probleme, die Qualitätsziele genau zu formulieren und später tatsächlich zu messen. Ein naheliegender Ansatz besteht deshalb darin, die Produktqualität *indirekt* dadurch zu spezifizieren, daß das Produkt auf eine bestimmte Art und Weise entwickelt werden soll, also über Eigenschaften des Vorgehensmodells bzw. des Softwareentwicklungsprozesses. Dieser Grundgedanke liegt auch allen (Firmen-) Zertifizierungen oder Reifegrad-Modellen zugrunde.

Die Qualität eines Produkts ist dann indirekt als das definiert, was bei fachgerechter Anwendung des Vorgehensmodells typischerweise zu erwarten ist. Diese Art der Spezifikation von Qualitätszielen ist wegen ihrer Indirektheit zwar ungenau und nicht reproduzierbar, ist aber auch sonst durchaus üblich (Bauherr zum Architekten: "Planen Sie das Haus so, daß wir mit dem vorhandenen Budget und mit den heute verfügbaren Techniken ein möglichst hochwertiges Haus bekommen.").

Ein zusätzlicher Unsicherheitsfaktor bei der Spezifikation von Qualitätszielen über Vorgehensmodelle ist der tatsächliche Verlauf des Projekts. Die "Tagesform" der Beteiligten, die Auswahl der Entwickler und deren konkrete Qualifikation und viele weitere Details, die ein Vorgehensmodell nicht spezifizieren kann oder die abweichend vom offiziellen Vorgehensmodell praktiziert werden, können Auswirkungen auf die Qualität des Produkts haben. Für die Qualität des Produkts ist der tatsächliche Verlauf des Projekts entscheidend, nicht das theoretisch einzuhaltende Vorgehensmodell.

1.4 Reifegradmodelle

Die Erkenntnis, daß Vorgehensmodelle oft nur auf dem Papier bzw. ungelesen im Regal stehen ("Schrankware"), hat zur Anforderung geführt, ein Unternehmen möge interne Kontrollmechanismen installieren, die sicherstellen, daß die Vorgehensmodelle nur eingehalten werden. Noch besser wäre es, den Erfolg des praktizierten Vorgehensmodells auch regelmäßig zu überprüfen und ggf. Ursachen für Mißerfolge zu beseitigen, indem das Vorgehensmodell angepaßt wird, also selbst

Gegenstand eines Entwicklungsprozesses wird. Diese Aktivitäten liegen offenbar auf verschiedenen semantischen Ebenen:

1. Die unterste Ebene ist der konkrete Ablauf, den ein Projekt genommen hat oder gerade nimmt. Analog zu einem laufenden Programm in einem Betriebssystem bezeichnen wir dies als einen (laufenden oder beendeten Entwicklungs-) **Prozeß**. Der Prozeß besteht letztlich aus einer Folge konkret durchgeführter Entwicklungsschritte.

2. Sofern der Entwicklungsprozeß nicht völlig chaotisch abläuft, wird er durch Regeln bzw. Vorschriften gesteuert; dies Regeln bezeichnen wir als **Vorgehensmodell** oder **Prozeßmodell**. Im Gegensatz zu einem Algorithmus bzw. einem deterministischen Programm, das von einen Prozeß in einem Betriebssystem ausgeführt wird, legen Vorgehensmodelle nicht alle Schritte eines Prozesses determinstisch fest, sondern lassen viele Freiräume, u.a. weil manche Entwicklungsschritte kreative Leistungen beinhalten. Diejenigen Entwicklungsschritte, die mit Qualitätskontrollen zu tun haben, wird man in einem Vorgehensmodell aber als obligatorisch festlegen, d.h. wenn ein Prozeß konform zum Vorgehensmodell durchgeführt wurde, wurde keine vorgeschriebene Prüfung ausgelassen.

   Offengelassen haben wir bis hier bewußt die Frage, ob das Vorgehensmodell überhaupt schriftlich fixiert ist und ob irgendwie maschinell überprüft werden kann, daß der Prozeß konform zum Vorgehensmodell abgelaufen ist bzw. abläuft; letzteres ist wünschenswert und machbar, aber für unsere Betrachtungen unwesentlich.

3. Wie gesagt ist es denkbar, daß ein Vorgehensmodell unzureichend ist, weil z.B. die vorgeschriebenen QS-Entwicklungsschritte nicht ausreichen oder nicht richtig durchgeführt werden, und daß somit Qualitätsziele verfehlt werden. Um solche Defizite zu entdecken und Gegenmaßnahmen ergreifen zu können, muß parallel zu den eigentlichen Projekten ein weiterer Prozeß ablaufen, der die praktizierten Vorgehensmodelle evaluiert und weiterentwickelt; man hier von einem **Meta-Prozeß** reden, also einem Entwicklungsprozeß, der Entwicklungsprozesse beobachtet und verbessert.

Dieser Meta-Prozeß sollte idealerweise wiederum durch Regeln gesteuert werden (z.B. die Grundregel, daß jedes Entwicklungsprojekt beobachtet wird und an dessen Abschluß alle relevanten Daten archiviert werden); diese Regeln können wir als **Meta-Vorgehensmodell** bzw. **Meta-Prozeßmodell** bezeichnen.

Ein besonders hoher Reifegrad ist in einer Entwicklergruppe vorhanden, wenn ein Meta-Vorgehensmodell definiert ist und wenn laufend ein damit konformer Meta-Prozeß durchgeführt wird, der die Entwicklung projektspezifischer Vorgehensmodelle steuert, die Projekte evaluiert und ggf. die projektspezifischen Vorgehensmodelle verbessert.

Im Sinne der Qualitätsdefinition von Produkten sind Meta-Prozesse noch eine Stufe indirekter als Vorgehensmodelle. Auch im Sinne der Qualitätssicherung sind sie nur eine indirekte Maßnahme.

Formalisiert wurde das Konzept der Reifegrade durch die Reifegradmodelle, die vom Software Engineering Institute (SEI) an der Carnegie Mellon University/Pittsburgh entwickelt wurden und die unter dem Warenzeichen Capability Maturity Model (CMM) bzw. neuerdings Capability Maturity Model Integration (CMMI) u.a. Reifegradstufen für einzelne Arbeitsbereiche definieren.

2 Qualitätsmodelle

2.1 Grundbegriffe

Ein **Qualitätsmodell** ist ein Begriffsrahmen, in dem der abstrakte Begriff Qualität stufenweise in Einzelaspekte aufgelöst und so konkretisiert und präzisiert wird[3]. Im Detail bestehen es aus:

[3]Dieser Modellbegriff ist *nicht* mit dem Modellbegriff konsistent, der in der UML und anderen Modellierungssprachen benutzt wird. Dort ist ein Modell eine vereinfachte Darstellung eines realen oder geplanten Systems, das Modell ermöglicht Aussagen über das modellierte System. Aussagen über ein zu bewertendes System machen nur die Qualitätsziele; das Qualitätsmodell stellt nur den Begriffsrahmen bereit.

1. **Qualitätsmerkmalen** (engl. *quality factor*) und -Untermerkmalen (bzw. Unterbegriffen; engl. *criteria*). Diese bilden eine Begriffshierarchie, die Untermerkmale beschreiben einzelne Aspekte des übergeordneten Merkmals.

   Die Merkmale sind meist noch vage und nicht direkt prüfbar oder meßbar. Beispiel: "System ist leicht erlernbar".

2. **Qualitätsindikatoren** (engl. *metrics*), die automatisiert oder manuell prüfbar oder meßbar sind.

   Beispiel: Für 73 % der Formulare des Systems liegt ein (nichtleerer) Hilfetext vor.

   Manuell zu prüfende Indikatoren haben Nachteile: der Aufwand für die Prüfung ist hoch, das Resultat ist ggf. ein "Gutachten", das seinerseits interpretiert werden muß, und die Ergebnisse sind nur bedingt reproduzibel. Maschinell zu prüfende Indikatoren sind billiger zu berechnen, schneller verfügbar und reproduzibel. Sie liefern meist numerische Werte; diese können trotz ihrer scheinbaren Objektivität völliger Unsinn sein und müssen ebenfalls sinnvoll interpretiert werden.

Für ein einzelnes Qualitäts(unter)merkmal sind i.d.R. *mehrere Qualitätsindikatoren* relevant.

Ein einzelner Qualitätsindikator ist i.d.R. für *mehrere Qualitätsmerkmale* relevant. Dies deutet bereits an, daß die Qualitätsmerkmale und -Untermerkmale meist nicht völlig überschneidungsfrei sind. Somit ist die gesamte begriffliche Verfeinerungshierarchie kein Baum, sondern nur eine Halbordnung, in der die Indikatoren die Blätter bilden.

Der Hauptunterschied zwischen Indikatoren und (Unter-) Merkmalen besteht darin, daß für die Indikatoren unabhängig voneinander konkrete Meßwerte bestimmt werden können. Diese elementaren Bewertungen müssen dann zu den höheren Ebenen der Verfeinerungshierarchie hin aggregiert (und dabei ggf. noch interpretiert) werden.

Bei der Qualitätsbewertung von Projekten oder Vorgehensmodellen kann man analog wie bei Produkten eine geeignete Konkretisierungshierarchie bilden, muß aber natürlich völlig andere Merkmale

und Indikatoren benutzen. Ferner ist die "Datenbasis" für eine konkrete Beurteilung eine völlig andere: bei Vorgehensmodellen die Dokumentation des Vorgehensmodells ("Methodenhandbuch und sonstige Dienstvorschriften"), bei Projekten die hoffentlich vorhandene Aufzeichnung aller relevanten Details des Projektverlaufs.

2.2 Qualitätsmerkmale nach ISO 9126

Die ISO-Norm 9126 (bzw. die entsprechende deutsche Norm DIN 66272) definiert einen Katalog von produktbezogenen Qualitätsmerkmalen (Projekt- oder Prozeßmerkmale werden nicht definiert). Hauptmerkmale sind:

- Funktionalität
- Zuverlässigkeit
- Benutzbarkeit
- Effizienz
- Änderbarkeit
- Übertragbarkeit

Die zugehörigen Teilmerkmale werden im Anhang der Norm nur als Vorschläge aufgeführt, müssen also fallspezifisch konkretisiert werden.

2.2.1 Funktionalität

Vorgeschlagene Teilmerkmale sind:

- **Vollständigkeit:** Alle spezifizierten Funktionen sind vorhanden.
- **Angemessenheit:** die spezifizierten bzw. realisierten Funktionen lösen die anstehenden Aufgaben.
- **Richtigkeit:** Die realisierten Funktionen liefern richtige bzw. spezifikationsgemäße Ergebnisse.

  Die Forderung nach funktionaler Korrektheit der realisierten Funktionen ist eine sehr beliebte Standardanforderung, sie macht aber erhebliche Probleme: Eine Implementierung kann immer nur gegenüber einer Spezifikation bzw. einem ähnlichen Vorprodukt korrekt sein. Diese Vorgaben sind meist lückenhaft, unpräzise oder

inkonsistent. Selbst da, wo eine vollständige Spezifikation vorliegt, scheitert der praktische Nachweis der Korrektheit am zu hohen Aufwand. Testen ist generell ungeeignet, durch Tests kann man nur die Anwesenheit von Fehlern zeigen, nicht deren Abwesenheit.
- **Interoperabilität:** Das System arbeitet mit anderen vorhandenen Systemen erfolgreich zusammen, z.B. erlaubt es den Austausch von Daten oder den gegenseitigen Aufruf.
- **Sicherheit:** Das System verhindert unberechtigte Zugriffe, die die Integrität der Daten beschädigen oder geheime Daten offenbaren oder die Verfügbarkeit des Systems beeinträchtigen.
- **(Normen-) Konformität:** Das System hält allgemeine oder spezielle Normen ein.

2.2.2 Zuverlässigkeit

Vorgeschlagene Teilmerkmale sind:

- **Reife:** Hier wird unterstellt, daß Systeme nie ganz fehlerfrei sind; dieses Qualitätsmerkmal besagt, daß Fehler selten sind und/oder keinen großen Schaden anrichten.
- **Fehlertoleranz:** Hier wird unterstellt, daß die "Umgebung" (Hardware, Software, andere Systeme, mit denen kommuniziert wird) ggf. nicht spezifikationsgemäß arbeitet; in dieser Lage soll das zu beurteilende Systeme trotzdem kontrolliert und soweit möglich sinnvoll weiterarbeiten, nicht abstürzen usw., obwohl es eigentlich nicht wirklich arbeiten kann[4].

[4]Hier liegt m.E. eine ziemlich unklare Abgrenzung zu Angemessenheit bzw. Korrektheit vor: daß die Umgebung nicht spezifikationsgemäß arbeitet, erweitert nur die Spezifikation des Verhaltens der Umgebung um weitere Sonderfälle; wie auf einen solchen Sonderfall zu reagieren ist, müßte eigentlich durch die Systemspezifikation definiert sein, wenn dort nichts steht, müßten sich die Entwickler eine Reaktion des Systems ausdenken, die den Benutzer hoffentlich zufriedenstellt, d.h. sie legen die Spezifikationen fest. Fehlertoleranz reduziert sich dann auf Angemessenheit der nachträglich erweiterten Spezifikation bzw. Korrektheit der Implementierung dieser erweiterten Spezifikation.

– **Robustheit:** Das System reagiert auf jede beliebige, noch so "unsinnige" Eingabe oder Benutzung in kontrollierter und sinnvoller Weise, stürzt jedenfalls nicht ab; es besteht sozusagen den "Elchtest". Bei interaktiven Systemen besteht ein beliebter Streßtest darin, mit flachen Händen auf der Tastatur extrem viele Tasten sehr schnell zu betätigen.

  "Unsinnige" Eingaben oder Benutzungen sind indessen völlig normal insofern, als die Spezifikation auch hier eine Reaktion des Systems vorgeben müßte; ist das nicht der Fall, liegt eher ein Fall unvollständiger funktionaler Spezifikationen vor.

  Ein Versagen bei einem Streßtest wird meist dadurch verursacht, daß die Verarbeitungskapazität des Prozessors oder anderer Ressourcen nicht ausreicht; diese Art der (Nicht-) Robustheit ist auf den ersten Blick eine Eigenschaft des Gesamtsystems aus Software und Prozessor bzw. weiteren Ressourcen. Wenn hingegen die Überlastung der Ressourcen durch eine geschicktere Programmierung vermieden werden kann (was nicht immer der Fall ist), dann ist der Mangel an Robustheit der Implementierung anzulasten.
– **Wiederherstellbarkeit:** Dieses Merkmal betrifft die Fähigkeit, nach einem Ausfall oder einer Störung schnell wieder die Systemleistungen zu erbringen, Daten zu restaurieren usw.

2.2.3 Benutzbarkeit

Viele Systeme werden von mehreren signifikant verschiedenen Benutzergruppen (Anfänger - Fortgeschrittene, verschiedene Rollen) benutzt, wobei einzelne Benutzergruppen ggf. nur Teile des Systems nutzen. Die Teilmerkmale des Merkmals Benutzbarkeit müssen für jede Benutzergruppe und deren Nutzungsverhalten individuell bewertet werden.

Die meisten Benutzbarkeitsmerkmale werden schon durch die Spezifikation des Systems festgelegt, nicht erst durch die Implementierung. Für viele Teilmerkmale sind ferner Produktanteile relevant, die nicht Teil des laufenden Systems sind: Benutzerdokumentation: Funktionsbeschreibungen, Tutorial, Nachschlagewerke usw.

Vorgeschlagene Teilmerkmale sind:

- **Verständlichkeit** des Verhaltens und der Konzepte des Systems
- **Erlernbarkeit** der Bedienung bzw. notwendigen Eingaben
- **Bedienbarkeit:** Aufwand (z.B. Zahl der Interaktionen) bei der Benutzung des Systems
- **Attraktivität**
- **Konformität:** Einhaltung von Normen oder Vereinbarungen zur Benutzbarkeit

Die meisten Benutzbarkeitsmerkmale sind insb. für interaktive Softwaresysteme relevant. Weiter detailliert wird das Thema Benutzbarkeit durch ein eigenes Gebiet, die **Software-Ergonomie**. Hierzu ist ein weiterer Standard vorhanden: DIN 66234, Teil 8: Grundsätze der Dialoggestaltung.

Die Benutzbarkeitsmerkmale setzen natürlich die korrekte Funktion bzw. ausreichende Performance voraus und sind insofern darauf aufbauende Kriterien.

2.2.4 Effizienz

Effizienz hängt mit Effektivität zusammen und wird leicht damit verwechselt:

- **Effektivität** ("Performance") besagt, daß eine bestimmte Leistung erzielt wird, z.B. ein bestimmter Durchsatz (Transaktionen pro Sekunde, Übertragungsraten usw.) oder eine bestimmte mittlere Antwortzeit. Erbracht wird die effektive Leistung durch ein Gesamtsystem bestehend aus Ressource (z.B. Rechner-Hardware) und Ressourcennutzer (z.B. Applikation)
- **Effizienz** bezeichnet die gute Ausnutzung der Ressourcen, also die absolute erbrachte Leistung in Relation zu den verbrauchten Ressourcen.

  Effizienz kann sich auf beliebige Leistungsarten (s. oben Effektivität) und beliebige Typen von Ressourcen beziehen: CPU-

Leistung, Plattenplatz, Hauptspeicherplatz, Plattenzugriffe, übertragene Datenmengen, usw.

Effizienz ist im Gegensatz zur Effektivität keine Eigenschaft des Gesamtsystems, sondern bezieht sich nur auf Ressourcennutzer, also i.d.R. die Software.

Effizienz wird oft als Quotient aus Leistung dividiert durch Verbrauch dargestellt; dies vermittelt aber den i.a. falschen Eindruck, man könne durch Verdopplung der Ressourcen auch die Leistung verdoppeln; die meisten Softwaresysteme verhalten sich nicht in diesem Sinne linear.

Bei Effizienzmerkmalen und damit zusammenhängenden Qualitätszielen ist oft unklar, ob ein Auftraggeber überhaupt daran diesen Eigenschaften der Software interessiert ist oder ob nicht eigentlich Effektivitätsmerkmale des Gesamtsystems aus Basissystemen und Softwareprodukt interessieren. Beispielsweise könnte sich bei der Planung eines Informationssystems ergeben, daß es pro Sekunde 10 Transaktionen verarbeiten können muß. Angenommen, die vorhandene Serverhardware schafft nur 5 Transaktionen pro Sekunde mit einer kostengünstig realisierbaren Implementierung des Systems. Die Frage stellt sich nun, ob man besser den Server aufrüstet oder besser durch eine aufwendigere Programmierung oder Vermeidung unnötiger Lasten die Effizienz der Implementierung erhöht. Die eigentlich relevante Anforderung ist hier eine absolute Leistungsanforderung an das Gesamtsystem, die Effizienzanforderung ergibt sich nur indirekt.

2.2.5 Änderbarkeit

Ein System gilt als gut änderbar, wenn der Aufwand für die Durchführung von (zu erwartenden) Änderungen (Korrekturen, Verbesserungen oder Anpassungen) gering ist. Die zeitintensiven Tätigkeiten bei Änderungen sind:

– die innere Funktionsweise des Systems verstehen, Fehlerursachen analysieren bzw. lokalisieren, von Änderung betroffene Stellen bestimmen

- System an den betroffenen Stellen ändern
- geändertes System prüfen und testen.

Hoher Aufwand, der für diese Tätigkeiten anfällt, wirkt sich unmittelbar auf den gesamten Änderungsaufwand aus. Die wichtigsten Teilmerkmale der Änderbarkeit besagen daher, daß der Aufwand für die vorstehenden Haupttätigkeiten gering ist:

- **Analysierbarkeit**
- **Modifizierbarkeit**
- **Prüfbarkeit**
- **Stabilität:** besagt, daß Änderungen nur selten unerwartete Nebenwirkungen erzeugen.

Während alle vorstehenden Merkmale **externe Qualitätsmerkmale** sind, also ohne Kenntnis der inneren Struktur des Systems beobachtbar bzw. meßbar sind, ist die Änderbarkeit ein **internes Qualitätsmerkmal**, das nur von der inneren Struktur des Systems abhängt. Es ist daher für die üblichen Benutzersichten auf ein System völlig irrelevant. Es interessiert nur denjenigen, der die spätere Weiterentwicklung des Systems durchführen bzw. finanzieren muß.

2.2.6 Übertragbarkeit

Ein System gilt als gut übertragbar, wenn der Aufwand, die Software in eine andere Umgebung (Hardware, Betriebssystem, Datenbankmanagementsystem und ähnliche Basissysteme oder Bibliotheken) zu portieren, gering ist. Vorgeschlagene Teilmerkmale sind u.a.:

- **Anpaßbarkeit**: dies betrifft die Änderungen, erforderlichen die für die Anpassung an die andere Umgebung erforderlich sind.

  Offensichtlich ist dies aber eigentlich nur eine Facette der Änderbarkeit; Anpassungen an eine sich ändernde Umgebung gelten als Standardursache für die Softwarewartung.
- **Installierbarkeit** in neuer Umgebung.

Ein Softwaresystem sollte natürlich schon in der ursprünglichen Umgebung gut installierbar sein und geeignete Werkzeuge zur Installation und Anpassung der Installation an eventuelle Modifikationen in den Basissystemen beinhalten. Letzteres ist eine Standard-Anforderung an ein System aus Sichtweise der Nutzerrolle Administrator. Die Installierbarkeit ist daher eher ein Teilmerkmal der Funktionalität.

2.2.7 Kritik am ISO-Qualitätsmodell

Wie schon oben erwähnt ist es sehr schwierig, für beliebige Kontexte sinnvolle Hierarchien von Qualitätsmerkmalen zu definieren, ferner sind bei den gröberen Qualitätsmerkmalen Überlappungen in gewissen Ausmaß kaum vermeidbar. Dies zugestehend muß trotzdem angemerkt werden, daß das Qualitätsmodell der ISO nicht als optimal angesehen werden kann:

- Die Trennung zwischen Übertragbarkeit und Änderbarkeit (und anderen Qualitätsmerkmalen) ist sehr unsauber.
- Fehlertoleranz und Robustheit sind überwiegend Eigenschaften der Vollständigkeit und Angemessenheit der Systemspezifikation und teilweise Korrektheitseigenschaften.
- Wichtige Qualitätsmerkmale wie Wiederverwendbarkeit können keinem der Hauptmerkmale direkt zugeordnet werden oder sind nur sehr versteckt vorhanden.

Insgesamt läßt daher die Disjunktheit und Vollständigkeit der Qualitätsmerkmale durchaus zu wünschen übrig.

Ferner ist problematisch, daß der Bezugsgegenstand der Qualitätsmerkmale nicht einheitlich und oft unklar ist, damit auch die Instanz, die für die Definition der Qualitätsziele zuständig und für die Qualitätssicherung verantwortlich ist:

- Einige Qualitätsmerkmale (Beispiel: Erlernbarkeit) sind überwiegend Eigenschaften der *Spezifikation* des Softwaresystems, nicht der

Implementierung, daher nur bedingt vom Entwickler zu verantworten.

- Teilweise sind die Qualitätsmerkmale Eigenschaften der *Implementierung* (Beispiele: Effizienz, Robustheit).
- Teilweise sind die Qualitätsmerkmale abhängig von der Installation des *Gesamtsystems* bestehend aus der betrachteten Software und der unterliegenden Plattform (Hardware, Betriebssystem, sonstige Basissysteme).

Die vorstehenden Kritikpunkte treffen allerdings mehr oder minder auch auf andere vorgeschlagene Merkmalshierarchien zu.

2.3 Alternative Qualitätsmodelle

In der Literatur sind diverse weitere Merkmalshierarchien publiziert, einige sind zugeschnitten auf speziellere Klassen von Software. Stellvertretend sei hier nur die häufig zitierte Merkmalshierarchie [GrC87] genannt, die im Rahmen des Software Metric Program von Hewlett-Packard entwickelt wurde. Die Ebene dieser Merkmalshierarchie enthält folgende Einträge (die die Abkürzung FURPS ergeben):

- functionality
- usability
- reliability
- performance
- supportability

Die 2. Ebene enthält insgesamt 25 Teilmerkmale. Insgesamt sind viele Ähnlichkeiten zur ISO-Merkmalshierarchie vorhanden.

2.4 Qualitätsindikatoren

Wie schon in der Einleitung erwähnt ist ein **Qualitätsindikator** eine prüfbare oder objektiv meßbare Eigenschaft; wenn nicht anders erwähnt, beziehen wir uns immer auf Eigenschaft eines zu bewertenden Softwaresystems.

Einen Sonderfall unter den Qualitätsindikatoren stellen Qualitätsmaße dar. Ein **Qualitätsmaß** ist ein *quantitativer* Qualitätsindikator; er ist definiert durch:

1. Skala (kontinuierlich oder diskret) mit Qualitätswerten;
2. Methode, wie der Qualitätswert, den ein konkretes Produkt hat, bestimmt wird; diese Methode kann maschinell oder manuell durchführbar sein.

Es gibt eine sehr große Vielfalt an Qualitätsindikatoren; die konkrete Auswahl in einem Projekt hängt natürlich zunächst von den allgemeinen Qualitätszielen und den darin auftretenden Qualitätsmerkmalen ab. Selbst wenn diese gegeben sind, verbleibt i.d.R. weiterer Spielraum bei der Auswahl der Indikatoren.

Metriken. Vielfach werden auch sogenannte Softwaremetriken als Qualitätsmaße benutzt. An dieser Stelle muß vor einem leider vorhandenen Begriffswirrwarr gewarnt werden:

- In der Mathematik versteht man unter einem **Maß** eine Funktion, die einem mathematischen Objekt eine Größe (als Zahlenwert) zuordnet. Der oben definierte Begriff Qualitätsmaß ist hiermit konsistent.
- Unter einer **Metrik** versteht man in der Mathematik eine Funktion, die *zwei* mathematischen Objekten einen Abstand zuordnet; die (absolute) Größe der Objekte spielt dabei keine Rolle.
- Als **Softwaremetrik** werden bestimmte Funktionen bezeichnet, die eine Software-Einheit auf einen Zahlenwert abbilden. Ein simples Beispiel ist der Umfang eines Systems in "lines of code" (LOC); der Umfang ist eine reine Größenangabe und als solcher nicht gut oder schlecht. Andere Softwaremetriken liefern Werte, die direkt zur Qualitätsbeurteilung dienen können.

  Im mathematischen Sinne sind die Softwaremetriken Maße; die eigentlich bessere Bezeichnung Softwaremaß ist aber nicht üblich.
- Der Begriff **Qualitätsmetrik** wird gleichbedeutend zu Qualitätsmaß benutzt.

2.5 Entwicklung von projektspezifischen Qualitätsmodellen und Qualitätszielen

Qualitätsziele müssen im Prinzip projektspezifisch ausgehandelt und fixiert werden. Im einfachster Fall kommt man mit den Begriffen und Definitionen eines Standard-Qualitätsmodells incl. bekannter Indikatoren aus; dann braucht man nur noch Mindestwerte der Indikatoren festzulegen.

Falls man mit einem Qualitätsmodelle "von der Stange" nicht auskommt, muß man ein projektspezifisches Qualitätsmodell entwickeln, und dann auf dieser Basis Qualitätsziele formulieren. Für die Entwicklung von Qualitätsmodellen und Qualitätszielen sind diverse Vorgehensweisen vorgeschlagen worden, also Vorgehensmodelle für den Teilbereich "Definition von Qualitätszielen". Hierbei muß analysiert bzw. entschieden werden,

- welche Qualitätsziele und darin auftretende Qualitätsmerkmale im konkreten Fall überhaupt interessieren,
- in welche quantifizierbaren Untermerkmale die relevanten Qualitätsmerkmale zerlegt werden können,
- welche konkreten Maße zur Messung der Untermerkmale dienen können,
- ob und wie diese Maße praktisch berechnet werden können,
- wie die einzelnen Meßergebnisse aggregiert bzw. interpretiert werden können und
- ob die so agggregierten Bewertungen letztlich auf der Ebene der ursprünglichen Qualitätsmerkmale eine wirklich brauchbare Aussage liefern.

Das bekannteste ist der Goal Question Metric (GQM) Ansatz [BaR88], der die vorstehenden Schritte detaillierter beschreibt, der allerdings in der Praxis noch durch durch eine vorgegene Merkmalshierarchie unterstützt werden sollte.

3 Qualitätsziele und Entwicklungsstufen

Analog zu üblichen funktionalen Anforderungen an ein zu entwickelndes System stellt sich auch bei Qualitätszielen die Frage, in welcher Entwicklungsstufe sie wie detailliert ausgearbeitet werden.

Als Beispiel betrachten wir ein zu entwickelndes Informationssystem mit WWW-basierter Bedienschnittstelle, darin das Merkmal Benutzbarkeit, Untermerkmal Erlernbarkeit, und das Qualitätsziel, daß die Benutzung des Systems "leicht erlernbar" sein soll. Diese vage Qualitätsanforderung präzisieren wir durch die Anforderungen, daß (a) jederzeit über eine Hilfefunktion eine Auskunft verlangt werden kann, was im aktuellen Systemzustand als nächstes zu tun ist, und daß (b) jederzeit eine Sitemap aufgerufen werden kann.

Wichtig ist hier die Beobachtung, daß wir ein Qualitätsziel in einer sehr frühen Entwicklungsstufe (vor dem funktionalen Detailentwurf) endgültig in bestimmte funktionale Anforderungen übersetzt haben. Wie in dem Beispiel die Auskunft und die Sitemap aufgerufen werden können, bleibt noch festzulegen, ebenfalls zu beantworten ist die architektonische Frage, welches Teilsystem die Hilfetexte verwaltet und formatiert und die Sitemap generiert. Jedenfalls enthält die detaillierte funktionale Spezifikation und der Grobentwurf unseres Systems bereits die fast komplette Lösung zu der ursprünglichen Qualitätsanforderung. Der weitere Entwicklungsprozeß hat kaum noch Einfluß darauf, wie gut oder schlecht die ursprüngliche Qualitätsanforderung erfüllt wird.

Anders gesagt unterscheiden sich Qualitätsanforderungen und die üblichen funktionalen Anforderungen deutlich darin, welche Entwicklungsphasen für die Erfüllung der Anforderungen entscheidend sind und wo dementsprechend Qualitätssicherungsmaßnahmen anzusetzen sind:

– Die Erfüllung funktionaler Anforderungen ist vor allem durch Entwurfs- und Programmierfehler gefährdet. Dementsprechend sind auf diesen Entwicklungsstufen geeignete QS-Maßnahmen vorzusehen (Testen, Design Reviews usw.).

– Die Erfüllung vieler Qualitätsanforderungen ist vor allem durch Gestaltungsfehler gefährdet. Dementsprechend sind in den frühen Entwicklungsstufen QS-Maßnahmen vorzusehen (Rapid Prototyping, Usability Engineering usw.)

Literatur

[BaR88] Basili, V.R.; Rombach, H.D.: The TAME project - Towards improvement-oriented software environments; IEEE Transactions on Software Engineering 14:6, p.758-773; 1988

[GrC87] Grady, R.B.; Caswell, D.L.: Software Metrics: Establishing a Company-Wide Program. Prentice-Hall, Englewood Cliffs, NJ; 1987

