
Softwarequalität

Binsenweisheiten der Softwareentwicklung

Probleme sind schwieriger als vermutet.

Entwicklungen dauern länger als erwartet.

Was schief gehen kann, geht schief.

Programme kosten mehr als erhofft.

Programme laufen nicht und keiner weiß warum.

Programme veralten, bevor sie funktionieren.

Hersteller ändern Programme schneller als Benutzer ihre Handhabung lernen.

Programme wuchern, um Speicher zu füllen und Prozessorzeit zu vergeuden.

Die Komplexität von Programmen wächst, bis keiner mehr durchblickt.

Wenn Programme nutzlos geworden sind, werden sie fürs Museum dokumentiert.

Schlagwörter

Qualitätsmerkmal, Qualitätsmodell, Qualitätsmaßnahme, Qualitätsmanagement.

Lernziele

Der Leser lernt kennen:

- Grundbegriffe der Softwarequalität,
- Softwarequalitätsmerkmale,
- Softwarequalitätsmaßnahmen.

1

Grundbegriffe der Softwarequalität

Probleme der Softwareentwicklung

- Große Programmsysteme sind schwer zu entwickeln, zu verstehen, zu warten und zu pflegen.
- Bei ihrer Entwicklung werden schwer erkennbare, aber folgenschwere Fehler gemacht.
- Geplante Termine und Kosten werden überschritten, geforderte Funktionalität und Qualität nicht erreicht.

Mängel von Softwareprodukten

Ein mangelhaftes Programm

- ⊗ ist aufwändig zu installieren,
- ⊗ funktioniert nicht korrekt, läuft unzuverlässig, unsicher,
- ⊗ verhält sich benutzerunfreundlich, wenig gebrauchstauglich,
- ⊗ reift durch „Service Packs“ und „Updates“ beim Anwender statt beim Hersteller,
- ⊗ ist schlecht dokumentiert und schwer zu durchschauen,
- ⊗ lässt sich weder leicht verbessern noch flexibel an neue Anforderungen anpassen,
- ⊗ ist unwirtschaftlich im Betrieb, kostet Geld, Zeit und Nerven.

Fehlerhaftigkeit von Software

Jedes nichttriviale Programm enthält Fehler. Bekannt werden Softwarefehler, wenn sie Pannen oder Katastrophen bewirken; dazu einige Beispiele:

- Airbus A320
 - Der Airbus A320 Kulmbach kollidiert nach der Landung auf dem Warschauer Flughafen mit einem Erdwall, weil die Bremssystemsoftware wegen eines Spezifikationsfehlers versagt. Zwei Menschen werden getötet, 56 verletzt [MCA94], [Lad98] The A320 Accident in Warsaw 14 September 1993.
- Ariane 5
 - Die ESA-Rakete Ariane 5 explodiert beim Erstflug am 4. Juni 1996. Zur Ursachenkette gehören eine fehlende Ausnahmebehandlung, ein Überlauf einer Ganzzahl, Spezifikationslücken in der von der Ariane 4 unverändert übernommenen Lenksystemsoftware, Fehler in der Systemspezifikation und im Risikomanagement. Der Materialschaden beträgt 300 Millionen Euro. Siehe Untersuchungsbericht der ESA [Lio96], Diskussion und Literaturhinweise in [JeMaR], [Lad98] The Ariane 5 Failure 4 June 1996, [Rob10], [WikAr].
- Mars Climate Orbiter
 - Die NASA-Sonde Mars Climate Orbiter verglüht im September 1999 in der Marsatmosphäre. Ursache ist ein Implementationsfehler, eine fehlende Umrechnung von Daten einer amerikanischen Einheit in die internationale SI-Einheit: „The output from the SM_FORCES application code as required by a MSOP Project Software Interface Specification (SIS) was to be in metric units of Newtonseconds (N-s). Instead, the data was reported in English units of pound-seconds (lbf-s)“ [Ste99] S. 16. Damit verglühen 160 Millionen US-Dollar Entwicklungskosten [LeiMa].

„Computer räumt Bankkonten leer“ ist inzwischen eine fast alltägliche Schlagzeile. Viren, Würmer, trojanische Pferde, Cracker und Phishing-Betrüger verderben Internetnutzern den Spaß. Andererseits verlangen Softwarehersteller von ihren Kunden, Garantieverzichtserklärungen für ihre Produkte zu akzeptieren. Dazu ein Beispiel:

Garantieverzicht

„Microsoft Windows XP Professional
ENDBENUTZER-LIZENZVERTRAG

11. AUSSCHLUSS DER GEWÄHRLEISTUNG. Die unten genannte Beschränkte Garantie ist die einzige ausdrückliche Garantie, die Ihnen gegeben wird. Sie ersetzt alle anderen ausdrücklichen Garantien (sofern vorhanden), die sich aus der Dokumentation, der Verpackung oder anderen Mitteilungen ergeben. Mit Ausnahme der Beschränkten Garantie und im größtmöglichen, durch das anwendbare Recht gestatteten Umfang stellen Microsoft und deren Lieferanten das Produkt und gegebenenfalls Supportleistungen wie besehen und ohne Garantie auf Fehlerfreiheit zur Verfügung. Sie schließen hiermit alle anderen Gewährleistungen und Garantien, seien sie ausdrücklich, konkludent oder gesetzlich, einschließlich, aber nicht beschränkt auf (sofern vorhanden) jede konkludente Gewährleistung, Garantie oder Pflicht der Handelsüblichkeit, Eignung für einen bestimmten Zweck, Zuverlässigkeit oder Verfügbarkeit, Genauigkeit oder Vollständigkeit von Antworten, Ergebnisse, fachmännische Bemühungen, Virenfreiheit und Sorgfalt – alles in Bezug auf das Produkt sowie in Verbindung mit dem Produkt zur Verfügung gestellte oder nicht erbrachte Support- oder andere Leistungen, Informationen, Software und dazugehörigen Inhalt, oder die anderweitig aus der Verwendung des Produkts entstehen – aus. Es wird auch jede Gewährleistung oder Garantie für Eigentum, ungestörte Nutzung, ungestörten Besitz, Übereinstimmung mit der Beschreibung und Nichtverletzung von Rechten Dritter in Bezug auf das Produkt ausgeschlossen.

Wenn Sie das Produkt in DEUTSCHLAND erworben haben, findet der vorstehende Ausschluss der Gewährleistung auf Sie keine Anwendung. Hinsichtlich etwaiger gesetzlicher Gewährleistungsansprüche wenden Sie sich bitte an Ihren Händler.

12. AUSSCHLUSS VON FOLGE-, ZUFÄLLIGEN UND BESTIMMTEN ANDEREN SCHÄDEN. Im größtmöglichen, durch das anwendbare Recht gestatteten Umfang sind Microsoft oder deren Lieferanten in keinem Fall haftbar für irgendwelche speziellen, zufälligen, indirekten oder Folgeschäden welcher Art auch immer oder für Strafschadensersatz (einschließlich, aber nicht beschränkt auf Schäden aus entgangenem Gewinn, Verlust von ver-

traulichen oder anderen Informationen, Geschäftsunterbrechung, Personenschäden, Verlust von Privatsphäre, Pflichtverletzung (einschließlich Pflichten nach Treu und Glauben oder Sorgfaltspflichten), Fahrlässigkeit sowie andere Vermögens- oder sonstige Schäden), die aus der Verwendung des Produkts oder der Tatsache, dass es nicht verwendet werden kann, oder aus in Verbindung mit dem Produkt bereitgestellten oder nicht erbrachten Support- oder anderen Leistungen, Informationen, Software und dazugehörigem Inhalt entstehen, oder die anderweitig aus der Verwendung des Produkts oder aus oder in Verbindung mit einer Bestimmung dieses EULAs resultieren oder in irgendeinem Zusammenhang damit stehen, selbst im Falle von Verschulden, unerlaubten Handlungen (einschließlich Fahrlässigkeit), verschuldensunabhängiger Haftung, Vertragsverletzung oder im Falle einer Verletzung der Garantie von/durch Microsoft oder deren Lieferanten, und selbst wenn Microsoft oder deren Lieferant auf die Möglichkeit solcher Schäden hingewiesen worden sind.

13. BESCHRÄNKUNG DER HAFTUNG UND VON ANSPRÜCHEN. Ungeachtet aller Schäden, die Sie aus welchen Gründen auch immer erleiden mögen (einschließlich, aber nicht beschränkt auf alle oben angesprochenen Schäden sowie in der Beschränkten Garantie alle direkten oder allgemeinen Schäden), ist die gesamte Haftung von Microsoft und deren Lieferanten unter allen Bestimmungen dieses EULAs und Ihr ausschließlicher Anspruch für alles Vorstehende (außer für Ansprüche bei Nachbesserung oder Nachlieferung, die von Microsoft bei einer Verletzung der Beschränkten Garantie gewählt wird) beschränkt auf den tatsächlich von Ihnen für das Produkt gezahlten Betrag oder US-\$ 5,00, je nachdem, welcher Betrag höher ist. Die vorstehenden Beschränkungen und Ausschlüsse (einschließlich der Abschnitte 11, 12 und wie in der Beschränkten Garantie angegeben) gelten im größtmöglichen durch das anwendbare Recht gestatteten Umfang, auch wenn ein Anspruch seinen wesentlichen Zweck verfehlt.

Wenn Sie das Produkt in DEUTSCHLAND erworben haben, finden die Beschränkungen in den vorstehenden Absätzen 12 und 13 auf Sie keine Anwendung. Stattdessen gilt Folgendes: Microsoft haftet in Fällen von Vorsatz, grober Fahrlässigkeit, bei Verletzung von Leib oder Leben sowie bei Ansprüchen nach dem Produkthaftungsgesetz nach den gesetzlichen Vorschriften. Eine weiter gehende Haftung von Microsoft besteht nicht. Dies gilt für alle Haftungstatbestände einschließlich unerlaubter Handlung. [...]

Beschränkte Garantie für Produkte, die in DEUTSCHLAND erworben wurden: [...] Die folgende Beschränkte Garantie hat Vorrang vor jeglicher Beschränkter Garantie, die in den gedruckten Materialien des Produkts oder in der Software enthalten ist.

HERSTELLERGARANTIE. Das Produkt wird für allgemeine Zwecke entwickelt und angeboten und nicht für besondere Zwecke eines Nutzers. Microsoft garantiert, dass das Produkt für einen Zeitraum von 90 Tagen ab Empfangsdatum im Wesentlichen gemäß den beiliegenden gedruckten Materialien arbeitet.

ANSPRÜCHE DES KUNDEN. Falls das Produkt dieser Garantie nicht entspricht, wird Microsoft nach ihrer Wahl entweder (a) das Produkt reparieren oder ersetzen oder (b) den von Ihnen gezahlten Preis rückerstatten. Diese Garantie gilt nicht, wenn der Fehler des Produkts auf einen Unfall, Missbrauch oder fehlerhafte Anwendung zurückzuführen ist. Für jedes Ersatz-Produkt übernimmt Microsoft eine Garantie nur für den Rest der ursprünglichen Garantiefrist oder für 30 Tage, je nachdem, welcher Zeitraum länger ist. Microsoft ist nicht haftbar für jeglichen Verlust oder Schaden, den Sie vernünftigerweise hätten vermeiden können, z. B. durch das regelmäßige Erstellen einer Sicherungskopie Ihrer Software und Dateien. Sie müssen einen Kaufnachweis und eine Kopie Ihrer datierten Rechnung vorlegen, falls dies von Microsoft gewünscht wird. Microsoft erbringt die Leistungen unter dieser Garantie vergütungsfrei, mit der Ausnahme, dass Sie für jegliche Ausgaben, die Ihnen entstehen (z. B. die Kosten der Versendung des Produkts an Microsoft), verantwortlich sind. [...]

HAFTUNGSBESCHRÄNKUNG. Microsoft haftet in Fällen von Vorsatz, grober Fahrlässigkeit, bei Verletzung von Leib oder Leben sowie bei Ansprüchen nach dem Produkthaftungsgesetz nach den gesetzlichen Vorschriften. Eine weiter gehende Haftung von Microsoft besteht nicht. Dies gilt für alle Haftungstatbestände einschließlich unerlaubter Handlung.“

Gründe für hohe Qualitätsanforderungen

Warum gewinnt das Entwickeln von Software mit hoher Qualität an Bedeutung?

- Neue, kritische Einsatzbereiche für Software
- Langfristige Kosten durch mangelnde Qualität
- Druck von Kunden, Druck durch Konkurrenz, drohender Imageverlust
- Sicherheitsrisiken für Güter, Umwelt und Leben
 - Finanziell: Haftung im Schadensfall
 - Ethisch: soziale Verantwortung, Umweltschutz
- Zufriedenheit der Entwickler mit ihrer Arbeit

Standard-Definitionen des Qualitätsbegriffs

Was ist Qualität? Standards geben allgemeine Definitionen.

- *DIN EN ISO 9000:2005:*

„3.1.1 Qualität

Grad, in dem ein Satz inhärenter Merkmale Anforderungen erfüllt.

ANMERKUNG 1 Die Benennung 'Qualität' kann zusammen mit Adjektiven wie schlecht, gut oder ausgezeichnet verwendet werden.

ANMERKUNG 2 'Inhärent' bedeutet im Gegensatz zu 'zugeordnet' 'einer Einheit innewohnend', insbesondere als ständiges Merkmal“ [ISO9000].

- *ISO/IEC 25000:2005:*

„**software quality** capability of software product to satisfy stated and implied needs when used under specified conditions“ [ISO25000].

- *IEEE Standard Glossary of Software Engineering Terminology:*

„**quality**. (1) The degree to which a system, component, or process meets specified requirements.

(2) The degree to which a system, component, or process meets customer or user needs or expectations“ [IEEE90] S. 60.

Fünf Qualitätsansätze nach David A. Garvin

Garvins Ansätze zur Qualität eines Produkts entsprechen den Sichten verschiedener Interessengruppen, vgl. [Bal98] S. 256–257.

- (1) Im **transzendenten Ansatz** ist Qualität die nur subjektiv erfahrbare Güte des Produkts. Sie lässt sich nicht konkret definieren und messen. In der Praxis führt dieser Ansatz nicht weit.
- (2) Der **kundenbezogene Ansatz** passt zur Sicht des *Marketings und Vertriebs*: Qualität ist, „wenn der Kunde zurückkommt und nicht das Produkt“. So entspricht Qualität dem Grad, zu dem das Produkt bewusste Anforderungen und unbewusste Wünsche des virtuellen Benutzers befriedigt. Da sich Menschen in Bedürfnissen unterscheiden, gibt es keine absolut qualitätvollen Produkte. Qualität bezieht sich auf eine Kundengruppe mit ähnlichen Bedürfnissen. Für die Kundengruppe nutzlose Produktmerkmale steigern die Qualität nicht. Der Ansatz spiegelt sich in den Definitionen von [ISO25000], [IEEE90] (2).
- (3) Im **produktbezogenen Ansatz** ergibt sich Qualität aus Merkmalen des Produkts, die genau definiert, aus allgemeinen Anforderungen abgeleitet, und intersubjektiv feststellbar sind. Da kein Merkmal ein Produkt vollständig beschreibt, gibt es keine eindeutige Qualität, sondern viele Qualitätsmerkmale zu verschiedenen Produktaspekten. Produkte lassen sich durch ihre Merkmale vergleichen. Dieser Ansatz passt zur Sicht der *Produktkonstrukteure*, der *Entwickler*. Er spiegelt sich in den Definitionen von [ISO9000], [IEEE90] (1).

- (4) Im **prozessbezogenen Ansatz** entsteht die Qualität des Produkts durch richtiges Anwenden von Herstellungsverfahren und Einhalten von Vorschriften. Dem guten Prozess entspringt das gute Produkt. Dieser Ansatz, der sich in der industriellen Fertigung materieller Produkte bewährt hat, passt zur Sicht des *Produktionsmanagements*. Aber Softwareentwicklung ist ein informationeller Konstruktionsprozess, kein materieller Produktionsprozess.
- (5) Der **wertorientierte Ansatz** passt zu den Sichten der *Finanzabteilung* und der *Produkttester*, die Kosten-/Nutzen- bzw. Preis/Leistungsverhältnisse untersuchen: Qualität ist eine Funktion von Kosten (Preis) und Nutzen (Leistung). Für den Produzenten ist die Qualität umso besser, je geringer die Kosten und je größer der Nutzen ist. Doch der Konsument schätzt ein unnützes Produkt nicht besser ein, wenn er es gratis erhält.

Aspekte von Qualität

Ein **Prozess** (*process*) ist ein Vorgang, der durch Eingaben, Teilprozesse und Ausgaben beschrieben ist. Ein **Produkt** (*product*) ist ein Ergebnis eines Prozesses. Nach Garvins Qualitätsansätzen sind zu unterscheiden:

- **Prozessqualität** (*process quality*), Qualität des Produktionsprozesses: Grad, zu dem Entwickler in einem Vorgehensmodell definierte Prozesse einhalten und beste Praktiken der Softwaretechnik einsetzen. Ohne Vorgehensmodell keine Prozessqualität.
- **Produktqualität** (*product quality*), Qualität des Produkts: Grad, zu dem das Produkt definierte Anforderungen erfüllt. Ohne Anforderungsdefinition keine Produktqualität. Die Norm *ISO/IEC 25000* differenziert diskussionswürdig [ISO25000]:
 - **Interne Produktqualität** untersucht nur **statische Merkmale** des isolierten Produkts, d.h. Merkmale, die sich feststellen lassen, ohne das Produkt auszuführen (z.B. durch Korrektheitsbeweise, Code-Reviews).
 - **Externe Produktqualität** untersucht **dynamische Merkmale** des Produkts in einer definierten Umgebung, d.h. Merkmale, die sich beim Ablauf des Produkts feststellen lassen (z.B. durch Tests, Messungen).
- **Gebrauchsqualität** (*Benutzungsqualität, Anwendungsqualität, quality in use*), Qualität des Produktgebrauchs: Grad, zu dem sich das Produkt für einen bestimmten Zweck in einem bestimmten Umfeld eignet. Ohne sinnvolle Anwendung und erfolgreiche Benutzung keine Gebrauchsqualität.

Idealerweise folgt aus Prozessqualität Produktqualität und daraus Gebrauchsqualität, aber real nicht zwangsläufig. „Gute“ Prozesse können „schlechte“ Produkte liefern und umgekehrt. „Gute“ Produkte können für Benutzer „unbrauchbar“ sein, wenn die umgesetzten Anforderungen nicht den Zwecken der Benutzer entsprechen. „Schlechte“ Produkte können „brauchbar“ sein.

Die Softwaretechnik hat zu jedem Qualitätsaspekt vielfältige Qualitätsmodelle entwickelt. Hier folgt ein kleiner Überblick.

Organisations- und Prozessqualitätsmodelle

Sie sind oft Teil firmenspezifischer oder normierter Vorgehensmodelle.

- Agile Methoden wie *Extreme Programming (XP)* (seit 2000)
- *V-Modell XT* (Extreme Tailoring) (Boehm 1979 – Version 1.3 2009)
- *Capability Maturity Model Integration (CMMI)* (2008)
- *IT Infrastructure Library (ITIL) V3* (2007)
- *ISO 15504: Software Process Improvement and Capability Determination (SPICE)* (2006)
- *Rational Unified Process (RUP)* (Version 9 2006)

- *EN ISO 9000* (2000)
- *Bootstrap* (1993)
- *Capability Maturity Model (CMM)* (1991)

Produkt- und Gebrauchsqualitätsmodelle

Ein **Produktqualitätsmodell** (*product quality model*) besteht aus Qualitätsmerkmalen und ihren Beziehungen zur Spezifikation von Anforderungen und zur Bewertung von Produktqualität. Ein (Software-) **Qualitätsmerkmal** (*quality factor, characteristic*) ist eine Produkteigenschaft, die sich auf eine Anforderung bezieht, einen Qualitätsaspekt erfasst und sich zur Produktbewertung eignet. Qualitätsmerkmale sind nicht binär, sondern graduell: Produkte weisen mehr oder weniger davon auf. Aber Qualitätsmerkmale sind oft nicht direkt quantifizierbar. Ein (Software-) **Qualitätsindikator** (*quality metric*) ist eine quantifizierbare (beobachtbare, messbare) Größe mit Beziehungen zu Qualitätsmerkmalen. Ein (Software-) **Qualitätsmaß** (*quality measure*) beschreibt, wie der Wert eines Indikators ermittelt und auf eine quantitative Skala abgebildet wird. Beispiele für Qualitätsmodelle:

- Zwei *Produktqualitätsmodelle* der Norm *ISO/IEC 25000* (2005) definieren die Qualitätsmerkmale Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit, Übertragbarkeit und gliedern sie in 27 Teilmerkmale [ISO25000].
- Das *Gebrauchsqualitätsmodell* der Norm *ISO/IEC 25000* definiert die Qualitätsmerkmale Effektivität, Produktivität, Sicherheit (*safety*), Benutzerzufriedenheit (*satisfaction*) [ISO25000].
- Der *Goal-Question-Metric-Ansatz (GQM)* von Basili und Rombach (1987) definiert eine Vorgehensweise, die in sechs Schritten zu spezifischen Qualitätsmodellen führt, vgl. [Bal98] S. 263–270.
- Das *Functionality-Usability-Reliability-Performance-Supportability-Modell (FURPS)* von Hewlett-Packard (1985) gliedert seine fünf Hauptmerkmale in 27 Teilmerkmale und definiert Indikatoren basierend auf Fehlern und Kosten. Es ist ein spezielles *FCM-Modell*, vgl. [Bal98] S. 260–262.
- Das *Factor-Criteria-Metrics-Modell (FCM)* von McCall (1977) gliedert Qualitätsmerkmale in Teilmerkmale (*criteria*) und ordnet ihnen Indikatoren zu. Merkmale beschreiben die Benutzersicht und sind zu spezifizieren, Teilmerkmale beschreiben die Entwicklersicht und sind zu konstruieren, Indikatoren beschreiben die Qualitätsprüfersicht und sind zu messen. FCM-Modelle können auch Prozessqualität beschreiben, vgl. [Bal98] S. 258–262.

Aufgaben von Qualitätsmanagement

- **Qualitätspolitik** (*quality policy*) beschreibt allgemeine Absichten und Ziele zur Qualität.
- **Qualitätsplanung** (*quality planning*) legt Qualitätsziele und -modelle, Qualitätsanforderungen an Prozess und Produkt, sowie Prozesse, Betriebsmittel und Personal zum Erreichen der Ziele fest.
- **Qualitätslenkung** (*quality control*) zielt darauf, Qualitätsanforderungen durch administrative und konstruktive Maßnahmen zu erfüllen.
- **Qualitätssicherung** (*quality assurance*) zielt darauf, Qualitätsanforderungen durch analytische und diagnostische Maßnahmen zu erfüllen.
- **Qualitätsprüfung** (*quality test*) überwacht die Umsetzung der konstruktiven Maßnahmen und führt die analytischen Maßnahmen durch.
- **Qualitätsverbesserung** (*quality improvement*) zielt darauf, das allgemeine Qualitätsniveau durch Mängel- und Fehleranalyse und Prozessverbesserung zu erhöhen.

2

Softwarequalitätsmerkmale

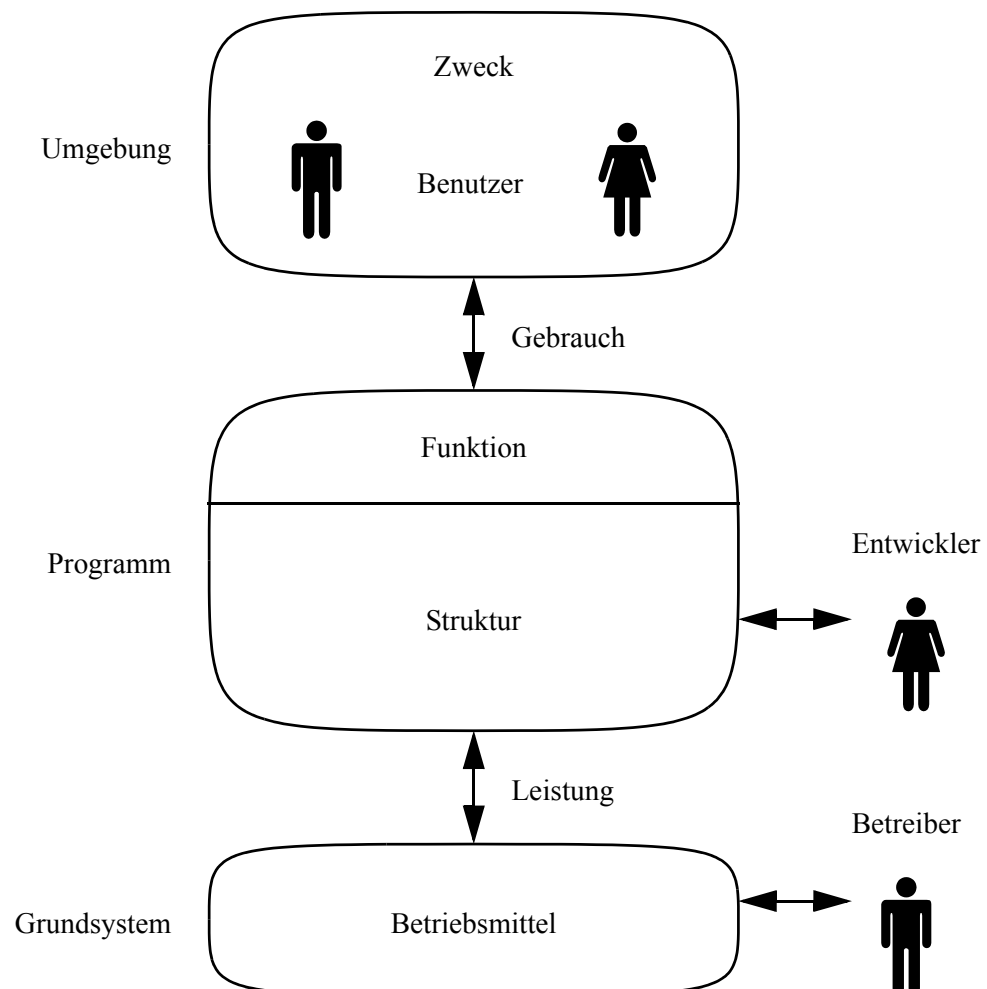
Wie bei anderen Produkten sind bei Programmen diese Aspekte zu unterscheiden, siehe Bild 1:

- **Gebrauch** (*use*): Wozu ist das Programm nütze, wie gut erfüllt es seinen Zweck, wie gut kann ich damit umgehen?
- **Funktion** (*function*): Was kann das Programm, welche Funktionen bietet es?
- **Struktur** (*structure*): Wie ist das Programm realisiert, aufgebaut?
- **Leistung** (*performance*): Welche Arbeit leistet das Programm mit welchem Aufwand?

Entsprechend teilen wir Softwarequalitätsmerkmale in diese Gruppen ein:

- **Gebrauchsqualitätsmerkmale** betreffen den zweckdienlichen Gebrauch des Programms in den sozialen, organisatorischen Umgebungen der Benutzer.
- **Funktionale Qualitätsmerkmale** betreffen die externe Funktion des Programms, das durch seine Schnittstellen festgelegte und beobachtbare Verhalten aus Benutzersicht.
- **Strukturelle Qualitätsmerkmale** betreffen die interne Struktur des Programms, die hinter seinen Schnittstellen verborgenen Implementationen aus Entwicklersicht.
- **Leistungsmerkmale** betreffen die Leistung des Programms bezogen auf benötigte Betriebsmittel und Zeit aus Betreibersicht.

Bild 1
Aspekte eines
Programms



Gruppenübergreifendes Qualitätsmerkmal: **Konformität** (*Ordnungsmäßigkeit; compliance*) ist der Grad, zu dem das Produkt Gesetze, Normen, Vorschriften u.ä. erfüllt.

Im Folgenden erscheinen Teilmerkmale in Aufzählungen unterhalb ihres Obermerkmals. Bei Merkmalen, die in der Form „Leichtigkeit, mit der das Programm...“ definiert sind, bedeutet „leicht“ soviel wie „in kurzer Zeit mit geringem Aufwand an Betriebsmitteln und Personal“.

2.1

Gebrauchsqualitätsmerkmale

Effektivität (*effectiveness*) ist der Grad, zu dem Benutzer das Programm für ihre Zwecke einsetzen können.

Zeitverhalten (*time behaviour*): Aus Benutzersicht soll das Programm Aufgaben möglichst schnell durchführen.

- **Programmproduktivität** (*program productivity*): Menge der nützlichen Arbeit, die das Programm pro Zeiteinheit erledigt; hoher **Durchsatz** (*throughput*) bei Funktionsausführung im Stapelbetrieb.
- **Reaktionsfähigkeit** (*responsiveness*): Das Programm soll so reagieren, dass sich Benutzer nicht langweilen; kurze **Antwort-** und **Verarbeitungszeiten** (*response time*) bei interaktiver Funktionsausführung.

Benutzerproduktivität (*user productivity*) ist die Leichtigkeit, mit der Benutzer ihre Ziele mit dem Programm erreichen, d.h. mit möglichst geringem Aufwand, wenigen Handgriffen und Mausklicks.

Benutzerzufriedenheit (*satisfaction*) ist das Gefühl, das der Gebrauch des Programms bei Benutzern erzeugt.

Attraktivität (*attractiveness*) ist der Reiz, den das Programm auf Benutzer ausübt.

Ästhetik (*aesthetics*) ist die Schönheit der visuellen Gestalt der Benutzungsoberfläche des Programms.

2.2

Funktionale Qualitätsmerkmale

Funktionalität (*functionality*) ist die Gesamtheit der Funktionen des Programms; somit ist es hier ein Kollektivbegriff, kein Qualitätsmerkmal.

Einsetzbarkeit ist die Leichtigkeit, mit der sich das Programm zur Benutzung bereitstellen lässt.

- **Installierbarkeit** (*installability*): Leichtigkeit, mit der sich das Programm in einer bestimmten Umgebung installieren lässt.
- **Konfigurierbarkeit**: Leichtigkeit, mit der sich das Programm für einen bestimmten Zweck konfigurieren lässt.
- **Verknüpfbarkeit** (*interoperability*): Leichtigkeit, mit der das Programm mit bestimmten anderen Programmen zusammenwirkt.
- **Austauschbarkeit** (*replaceability*): Leichtigkeit, mit der das Programm ein anderes Programm ersetzen kann.

Zuverlässigkeit (*reliability*) ist der Grad, zu dem das Programm unter bestimmten Bedingungen für eine bestimmte Zeit funktioniert. Benutzer wollen darauf vertrauen, dass das Programm stets fehlerfrei läuft und vernünftige Ergebnisse liefert.

- **Funktionsabdeckung** (*Vollständigkeit; completeness*): Grad, zu dem das Programm an es gestellte funktionale Anforderungen erfüllt.
- **Korrektheit** (*correctness*): Grad, zu dem die Implementation des Programms die Erwartungen der Benutzer, die Anforderungen und seine Spezifikation erfüllt. Nach dem produktbezogenen Qualitätsansatz ist Korrektheit der Grad, zu dem das Pro-

programm gestellte Anforderungen erfüllt. Formal ist das Programm korrekt, wenn seine Implementation seiner Spezifikation entspricht. (Daher kann man nicht über Korrektheit reden, wenn die Spezifikation fehlt.) Der Ablauf eines korrekten Programms liefert bei zulässiger Eingabe korrekte Ausgabe.

- **Genauigkeit** (*accuracy*): Nähe berechneter Werte zu erwarteten Werten (durch Güte des Algorithmus bestimmt).
- **Genauigkeit** (*precision*): Nähe zweier benachbarter Werte eines Wertebereichs (durch Anzahl der Bits des Datentyps bestimmt).
- **Sicherheit**
 - **Ausfallsicherheit** (*Reife; maturity*): Seltenheit, mit der interne Störungen oder Fehler zum Versagen, „Absturz“ oder „Zusammenbruch“ führen.
 - **Sicherheit für die Umwelt** (*safety*): Seltenheit, mit der der Programmbetrieb zu nicht verantwortbaren Risiken für Mensch und Umwelt führt.
 - **Sicherheit gegen Angriffe** von außen auf das Programm (*security*): Grad des Schutzes, der unbefugten Benutzern versehentlichen und vorsätzlichen Zugang zum Programm verwehrt und unberechtigte Zugriffe auf Daten verhindert.
- **Verfügbarkeit** (*availability*): Grad, zu dem das Programm ausführbereit ist, Ausfälle selten vorkommen und die Ausfallzeiten kurz sind.
- **Wiederherstellbarkeit** (*recoverability*): Leichtigkeit, mit der nach einem Ausfall das Programm wieder funktionstüchtig wird und verlorene oder beschädigte Daten restauriert werden.
- **Fehlertoleranz** (*fault tolerance*): Grad, zu dem interne Fehler und Störungen des Grundsystems lokal bleiben, nicht nach außen sichtbar werden und auf höheren Ebenen ablaufende Programme nicht behindern.

Benutzbarkeit (*Anwendbarkeit, Gebrauchstauglichkeit; usability, user friendliness*) ist der Grad, zu dem vorgesehene Benutzer das Programm angenehm, frustrations- und stressfrei handhaben können.

- **Verständlichkeit** (*understandability*): Leichtigkeit, mit der Benutzer die Konzepte und Funktionen des Programms verstehen können. Das erfordert einfache Konzepte und durchschaubare Benutzungsdokumente.
- **Erlernbarkeit** (*learnability*): Leichtigkeit, mit der Benutzer lernen, das Programm handzuhaben. Das erfordert klare Benutzungsoberflächen, selbsterklärende Namen für Dienste, vernünftige Systemreaktionen.
- **Angemessenheit** (*suitability*): Grad der Eignung der Funktionen des Programms für die Aufgaben des Benutzers, und Kombinierbarkeit der Funktionen zur Lösung größerer Aufgaben.
- **Durchführbarkeit** (*operability*): Leichtigkeit, mit der Benutzer ihre Aufgaben mit dem Programm lösen können.
- **Robustheit** (*robustness*): Angemessenheit, mit der das Programm auf ungewöhnliche Benutzungen reagiert. Grundsatz: Der Benutzer darf alles! Ein robustes Programm erkennt „falsche“ Eingaben von Benutzern und behandelt sie mit hilfreichen Meldungen, sodass Benutzer ihre „Fehler“ erkennen und korrigieren können. Der Ablauf eines nicht robusten Programms kann dagegen durch „falsche“ Eingabe in einen fehlerhaften Zustand geraten und „abstürzen“.

2.3

Strukturelle Qualitätsmerkmale

Ein komplexes Programm ist leider immer fehlerbehaftet, Fehlerbehebung – **Wartung** (*maintenance*) – daher eine wesentliche Daueraufgabe, die durch **Wartbarkeit** (*maintainability*) des Programms erleichtert werden soll. Dazu gehören:

- **Analysierbarkeit** (*analyzability*): Leichtigkeit, mit der sich Mängel und Fehlerursachen diagnostizieren und änderungsbedürftige Teile bestimmen lassen.
- **Instandsetzbarkeit**: Leichtigkeit, mit der sich Fehler korrigieren und Verbesserungen einbringen lassen.
- **Stabilität** (*stability*): Seltenheit des Auftretens unerwarteter Wirkungen von Verbesserungen.

Neben Wartung bedürfen Programme ständiger **Pflege** (*maintenance, support*), um sie an neue Anforderungen und Umgebungen anzupassen. Die Bezeichnung **Pflegbarkeit** (*supportability*) ist unüblich. **Änderbarkeit** (*modifiability, changeability*) ist die Leichtigkeit, mit der sich Änderungen am Programm durchführen lassen. Wir differenzieren nach dem Zweck der Änderung:

- **Portierbarkeit** (*Übertragbarkeit, Portabilität; portability*): Leichtigkeit, mit der sich das Programm auf ein anderes Grundsystem, eine andere Hardware- oder Software-Plattform übertragen lässt.
- **Anpassbarkeit** (*Adaptierbarkeit; adaptability*): Leichtigkeit, mit der sich das Programm an veränderte Aufgaben und Anforderungen der Benutzer oder veränderte Umgebungen anpassen lässt.
- **Erweiterbarkeit** (*extensibility, augmentability*): Leichtigkeit, mit der sich mit dem Programm zusätzliche Aufgaben und Anforderungen der Benutzer erfüllen lassen.
- **Prüfbarkeit** (*testability*): Leichtigkeit, mit der sich das geänderte Programm prüfen lässt.

Wart- und Änderbarkeit kann man nur indirekt in Programme hineinkonstruieren, indem man andere Merkmale beachtet. Nützlich sind z.B. **Verständlichkeit** von Entwürfen und Spezifikationen und **Lesbarkeit** (*readability*) von Programmen. **Einfachheit** (*simplicity*), **Einheitlichkeit** (*Konsistenz; consistency*), **Selbsterklärlichkeit** (*self-descriptiveness*), **Strukturiertheit** (*structuredness*), **Kompaktheit** (*conciseness*) tragen dazu bei. **Modularität** (*modularity*) fördert Wart- und Änderbarkeit durch **Lokalität** (*locality*), da Fehler leicht in Modulen zu lokalisieren sind und für viele Anpassungen nur Implementationen einzelner Module zu ändern, Schnittstellen zu erweitern oder weitere Module hinzuzufügen sind. Methoden, aber auch Mittel wie Programmiersprachen unterstützen Wart- und Änderbarkeit mehr oder weniger gut.

Wiederverwendbarkeit (*reusability*) ist der Grad, in dem sich Teile des Programms in anderen Programmen, für andere Aufgaben, in anderen Zusammenhängen wiederverwenden lassen. Wiederverwendbare Teile können Entwürfe, Spezifikationen und Implementationen sein. Einheiten der Erweiterung und der Wiederverwendung sind Module und Klassen. Das Erweitern eines Programms ist ein Spezialfall des Wiederverwendens. Module sind durch Benutzen wiederverwendbar, Klassen auch durch die objektorientierte Technik des Erweiterns.

2.4

Leistungsmerkmale

Leistungsmerkmale fassen wir unter dem Begriff **Leistungsfähigkeit** (*performance*) zusammen. Das Programm soll seine Aufgabe bzgl. bestimmter quantifizierbarer Leistungsgrößen möglichst gut erfüllen. Die Leistung eines Motors misst man in Kilowatt, wie misst man die Leistung von Software? Keine leichte Frage, doch spielen Beziehungen zwischen „produzierten Daten“, „Raum“ und „Zeit“ eine Rolle.

Effizienz (*efficiency*) ist das Verhältnis zwischen der Funktionalität des Programms und seinem Bedarf an Betriebsmitteln. Das **Verbrauchsverhalten** (*resource behaviour*) des Programms ist durch die Quantität der belegten Betriebsmittel und die Dauer ihrer Belegung beschrieben. Aus Betreibersicht soll das Programm die Betriebsmittel zur Durchführung der Aufgaben gut nutzen, d.h. sparsam damit umgehen:

- Prozessorzeit,
- Hauptspeicherbedarf,
- Hintergrundspeicherbedarf,
- Festplattenzugriffe,
- Netzlast usw.

Programme sollen möglichst wenig Speicherplatz belegen und möglichst schnell laufen, d.h. möglichst wenig Prozessorzeit beanspruchen. Zwar werden Speicher und Prozessoren durch den Preisverfall bei der Hardware immer billiger, doch wird ein Anwender unter zwei funktional und qualitativ gleichwertigen Programmen jenes vorziehen, das weniger Speicher verschwendet und schneller läuft.

Ein Ziel von Programmverbesserung ist, die Effizienz eines Programms zu erhöhen, ohne es funktional oder qualitativ zu ändern. Zwischen Speicherbedarf und Prozessorzeit gibt es oft einen Konflikt; man kann selten beides gleichzeitig verbessern. Ein schneller Algorithmus kann mehr Speicher brauchen, ein speichersparender kann langsamer laufen. Effizienz kann auch mit anderen Qualitätsmerkmalen konfliktieren. Dann ist abzuwägen, welchen Preis man für Zuverlässigkeit, Benutzbarkeit, Wartbarkeit, Änderbarkeit und Wiederverwendbarkeit zu zahlen bereit ist.

3 Softwarequalitätsmaßnahmen

Wie erzielt man Qualität? Wie erreicht man, dass Prozess und Produkt Qualitätsanforderungen erfüllen? Wodurch erhält das Programm geforderte Qualitätsmerkmale?

- **Konstruktive Qualitätsmaßnahmen** sorgen dafür, dass geforderte Qualitätsmerkmale in den Prozess oder das Produkt hineinkonstruiert werden. Ziel ist, Fehler von vornherein zu vermeiden und Qualität zu erzeugen.
- **Analytische Qualitätsmaßnahmen** beobachten, testen, messen, prüfen, analysieren, diagnostizieren den Prozess oder das Produkt, um festzustellen, ob geforderte Qualitätsmerkmale vorhanden sind. Ziel ist, Fehler im Nachhinein zu entdecken und Qualität zu bewerten.

Qualität kann *nie* in einen Prozess oder ein Produkt hineinbeobachtet, hineingetestet, hineingemessen, hineingeprüft, hineinanalysiert, hineindiagnostiziert werden! Man kann nur nachträglich ein schlechtes, fehlerhaftes Produkt entfehlern und verbessern. Deshalb rangieren konstruktive Qualitätsmaßnahmen vor analytischen. Ziel ist, durch konstruktive Maßnahmen Produkte mit geforderten Qualitätsmerkmalen herzustellen und anschließend Aufwand für analytische Maßnahmen zu sparen. Bild 2 idealisiert die Beziehungen. Beispiel: Ein *Java*-Programm ist per Konstruktion portierbar; man braucht dieses Merkmal seines Bytecodes nicht analytisch festzustellen.

Fehler

Der Begriff **Fehler** (*error*) bedeutet vielerlei: Ein **Fehlverhalten** (*anomaly, failure, malfunction*) ist eine unerwartete Situation beim Ausführen eines Programms: falsches, ungenaues Ergebnis (*incorrect, inaccurate result*), Störung (*disturbance, inconvenience*), Absturz (*crash*). Eine **Fehlerursache** (*bug, defect, fault, flaw, glitch*) ist eine statische Eigenschaft des Programms, seines Entwurfs, seiner Spezifikation oder seiner Implementation. Sie ist entstanden durch einen **Denkfehler** (*fault, mistake*) des Entwicklers. Die Norm *ISO/IEC 25000* nennt den Unterschied zwischen einer externen Qualitätsanforderung und der festgestellten externen Qualität **Defekt** (*defect*), den Unterschied zwischen einer internen Qualitätsanforderung und der festgestellten internen Qualität **Nonkonformität** (*nonconformity*) [ISO25000].

Tatsachen

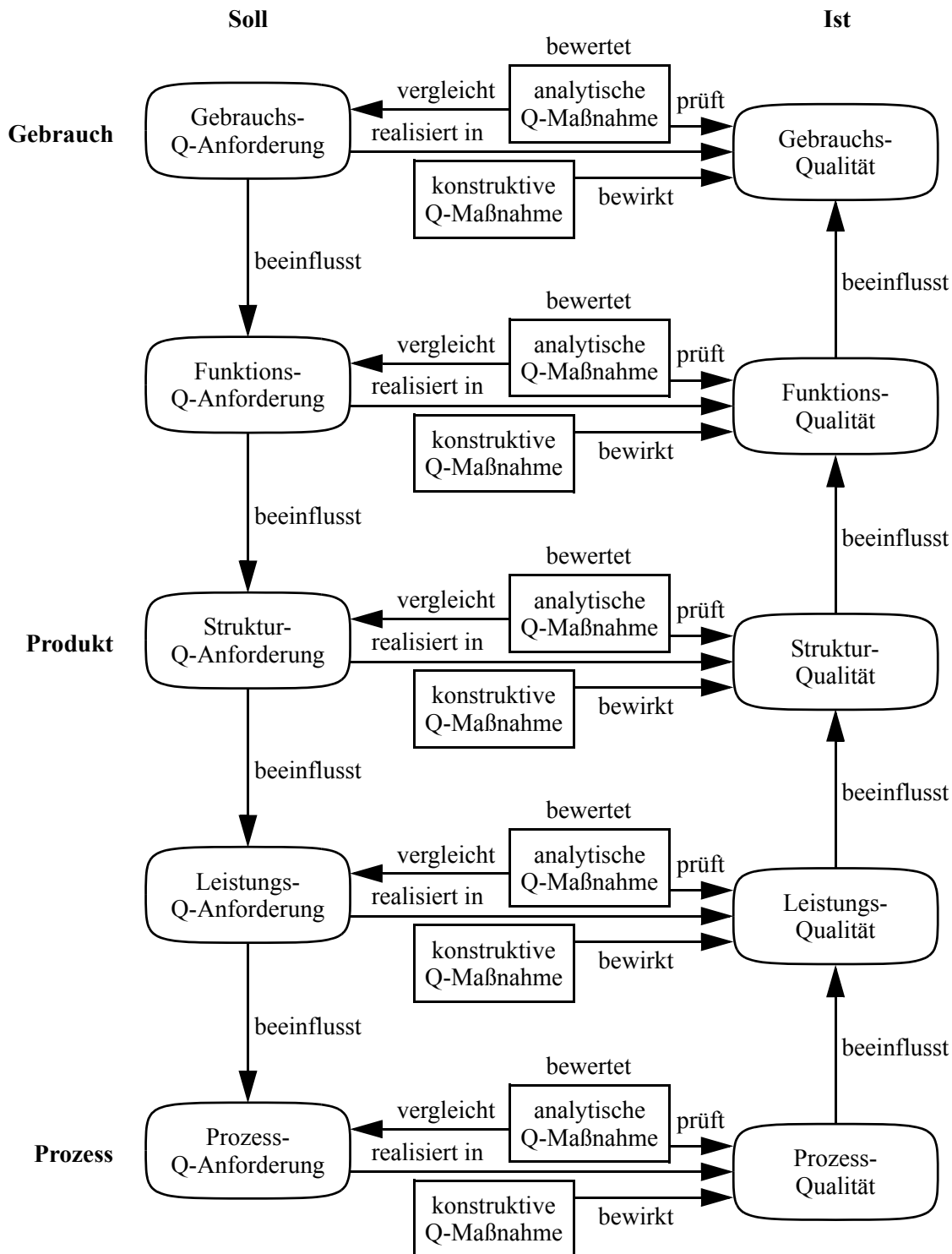
Einfache Fehlerursachen können schwere Fehlersituationen bewirken. Fehler in frühen Phasen wiegen schwerer als Fehler in späten Phasen. Je früher ein Fehler begangen und je später er erkannt und beseitigt wird, desto höher sind seine Kosten.

Leitlinie 1

Vermeide, entdecke,
korrigiere Fehler

- (1) Lieber Fehler von vornherein vermeiden als Fehler im Nachhinein beheben.
- (2) Lieber nur wenige leichte Fehler als viele schwere begehen.
- (3) Lieber Fehler spät als früh begehen.
- (4) Lieber Fehler früh als spät entdecken.
- (5) Lieber Fehler früh als spät korrigieren.

Bild 2 Qualitätsanforderungen und Qualitätsmaßnahmen



Nun ein Überblick zu Konzepten konstruktiver und analytischer Qualitätsmaßnahmen:

3.1 Konstruktive Qualitätsmaßnahmen

Zur folgenden Gliederung vgl. [Bal98] S. 280.

Organisatorische Maßnahmen

- *Checklisten*
Beispiel: Schablonen und Fragenkataloge unterstützen Einheitlichkeit von Struktur und Inhalt von Dokumenten.
- *Richtlinien*
Beispiel: Programmierrichtlinien unterstützen Lesbarkeit von Programmen.

Technische Maßnahmen

- *Methoden*
Beispiele für „von vornherein“: Spezifikation durch Vertrag (*design by contract*) unterstützt Korrektheit. Komponentenorientierung unterstützt Wiederverwendbarkeit mit Techniken zum Erstellen wiederverwendbarer Komponenten.
Beispiel für „im Nachhinein“: **Restrukturierung** (*Umgestaltung, Refaktorisierung, refactoring*) bedeutet Verbesserung der Struktur eines Programms, ohne sein beobachtbares Verhalten zu ändern, um damit Lesbarkeit, Verständlichkeit, Wartbarkeit und Erweiterbarkeit zu verbessern.
- *Sprachen*
Beispiel: Höhere Programmiersprachen unterstützen strukturelle Qualitätsmerkmale besser als Assemblersprachen.
- *Werkzeuge*
Beispiel: Ein Konfigurationsverwaltungssystem unterstützt Konfigurierbarkeit.

3.2 Analytische Qualitätsmaßnahmen

Die Begriffe Validierung und Verifikation sind in der Literatur vielfältig definiert.

Validierung

Unter einer **Validierung** (*validation*) verstehen wir

- (1) einen Prozess, in dem geprüft wird, ob ein Softwareprodukt Anforderungen genügt,
- (2) ein positives Ergebnis eines solchen Prozesses, d.h. einen Nachweis dafür, dass das Softwareprodukt die Anforderungen erfüllt.

Bei einem **validierten** Produkt ist nachgewiesen, dass es vereinbarten Anforderungen entspricht. Das Produkt muss nicht ein komplettes fertiges System sein, sondern es kann sich um jede Art von Teil- und Zwischenprodukt handeln: Pflichtenheft, Benutzerhandbuch, Entwurfsdokument, usw. Die Anforderungen liegen u.U. nicht vollständig, eindeutig und konsistent in formaler Notation schriftlich fixiert vor – möglicherweise existieren sie nur in Köpfen von Individuen wie Auftraggeber, Benutzer, Entwicklungsleiter in Form von Vorstellungen, Wünschen usw. Bei der Validierung geht es also darum, die Frage

„Haben wir das richtige Produkt entwickelt?“

positiv zu beantworten.

Die Norm *ISO/IEC 25000* nennt die Prüfung der externen Qualität gegen die externen Qualitätsanforderungen **Validierung**, die Prüfung der internen Qualität gegen die internen Qualitätsanforderungen **Verifikation** [ISO25000]. Das *V-Modell* versteht unter **Validation** die Prüfung der Eignung des Produkts für seinen Einsatzzweck, unter **Verifikation** die Prüfung der Übereinstimmung des Produkts mit seiner Spezifikation.

Verifikation

Unter einer **Verifikation** (*verification*) verstehen wir

- (1) einen Prozess, in dem geprüft wird, ob bei der Entwicklung eines bestimmten Softwareprodukts ausgehend von bestimmten Vorprodukten bestimmte Transformationsschritte korrekt durchgeführt wurden,
- (2) ein positives Ergebnis eines solchen Prozesses, d.h. einen Nachweis dafür, dass das Softwareprodukt korrekt aus den Vorprodukten entwickelt wurde.

Bei einem **verifizierten** Produkt ist nachgewiesen, dass es auf richtige Weise erstellt wurde. Bei der Verifikation geht es also darum, die Frage

„Haben wir das Produkt richtig entwickelt?“

positiv zu beantworten. Ein Beispiel dafür ist die Verifikation des Entwurfs oder der Implementation eines Programms bezüglich seiner Spezifikation, d.h. der Nachweis, dass sich das Programm so verhält, wie es seine Spezifikation vorschreibt. Die Vorprodukte stellen i.d.R. eine vollständige, eindeutige und konsistente Beschreibung der Funktion des Produkts dar, meist auf höherem Abstraktionsniveau und in formaler Notation. Bei formaler Verifikation spricht man auch von **Korrektheitsbeweisen** (*correctness proof*), die analog zu mathematischen Beweisen geführt werden. In diesem Verständnis zeigt „Verifikation [...] mit mathematischen Mitteln die Konsistenz zwischen der Spezifikation und der Implementation einer Systemkomponente“ [Bal98] S. 396, ausführlich S. 446–463.

Nichtausführende Prüfmethoden

- Statische Analysen, Anwendungen von Maßen
- Inspektionen, Code-Reviews, Walkthroughs
- Symbolische Ausführungen, Ablaufverfolgungen, statische Tests

Simulationen

Bei einer **Simulation** wird ein bestimmter Aspekt eines Systems in einem ausführbaren Modell dargestellt; bei der Ausführung des Modells wird das Verhalten des Aspekts geprüft und daraus werden Rückschlüsse auf das Verhalten des Systems gezogen.

Beispiele für simulierte Aspekte: Leistungsverhalten; bei nebenläufigen Systemen das Synchronisations- und Kommunikationsverhalten von Prozessen.

Tests

Bei einem **Test** wird ein ausführbares Softwareprodukt – also ein Programm, eine Programmkomponente oder ein Programmsystem – aktiviert und sein Verhalten beobachtet, d.h. es wird mit Eingabedaten versorgt und die erzeugten Ausgabedaten werden untersucht. Edsger Dijkstras oft zitierte Aussage charakterisiert sehr gut die Möglichkeiten und Grenzen des Testens:

„Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence“¹ [Dij72].

Ein Test ist **erfolgreich**, wenn Fehler auftreten. Ein intensiver, erfolgloser Test lässt den Tester hoffen, dass das Programm seine Aufgabe in Anwendungen erfüllen wird. Bei einem **getesteten** Programm ist nur gezeigt, dass es *manchmal* fehlerfrei läuft. Bei Tests geht es also darum, die Frage

„Ist das entwickelte Programm fehlerhaft?“

positiv zu beantworten.

¹ „Das Testen von Programmen kann ein sehr effektiver Weg sein, um die Anwesenheit von Fehlern zu zeigen, aber es ist hoffnungslos unangemessen, um ihre Abwesenheit zu zeigen.“

Fazit

Es ist wünschenswert, Softwareprodukte möglichst weitgehend zu validieren und verifizieren. Leider reichen heutige Methoden nicht aus, um die Fehlerfreiheit von Endprodukten zu garantieren. Deshalb ist die praktische Prüfung des Ablaufverhaltens von Programmen durch Tests unumgänglich und zentral für die Qualitätssicherung.

„Debugging is at least twice as hard as writing the program in the first place. So if your code is as clever as you can possibly make it, then by definition you're not smart enough to debug it.“
 Brian Kernighan

4

Aufgaben

Aufgabe 1

Beziehungen zwischen
Qualitätsmerkmalen

Diskutieren Sie Beziehungen, Zusammenhänge, Abhängigkeiten, Konflikte zwischen den Qualitätsmerkmalen!

Aufgabe 2

Bedeutung von
Qualitätsmerkmalen

Diskutieren Sie, welche Rollen die Sichten der Beteiligten bei der Bewertung von Qualitätsmerkmalen spielen!

5

Literatur

- [Bal98] Helmut Balzert: *Lehrbuch der Software-Technik. Software-Management Software-Qualitätssicherung Unternehmensmodellierung mit CD-ROM*. Spektrum Akademischer Verlag, Heidelberg, Berlin (1998) Band 2, 769 S.
- [Dij72] Edsger W. Dijkstra: *The Humble Programmer. ACM Turing Award Lecture 1972*. Comm. ACM, Vol. 15, No. 10 (October 1972) S. 859–866, <http://portal.acm.org/citation.cfm?doid=355604.361591> (Zugriff 2010-08-14)
- [IEEE90] IEEE Standards Board: *IEEE Standard Glossary of Software Engineering Terminology*. IEEE Std 610.12-1990, The Institute of Electrical and Electronics Engineers, New York (approved September 28, 1990) 83 S., <http://www.idi.ntnu.no/grupper/su/publ/ese/ieee-se-glossary-610.12-1990.pdf> (Zugriff 2010-08-12)
- [ISO9000] Deutsches Institut für Normung e.V., International Organization for Standardization: *DIN EN ISO 9000:2005. Qualitätsmanagement, Qualitätsmanagementsysteme – Grundlagen und Begriffe (ISO 9000:2005)* (2005-12)
- [ISO25000] International Organization for Standardization, International Electrotechnical Commission: *ISO/IEC 25000:2005. Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE* (2005-08) deutsch: *Software-Engineering – Qualitätskriterien und Bewertung von Softwareprodukten (SQuaRE) – Leitfaden für SQuaRE*
- [JeMAr] Jean-Marc Jézéquel, Bertrand Meyer: *Design by Contract: The Lessons of Ariane*. Eiffel Software (1985–2010) <http://archive.eiffel.com/doc/manuals/technology/contract/ariane/> (Zugriff 2010-08-13)
- [Lad98] Peter B. Ladkin (compiled): *Abstracts of References and Incidents*. University of Bielefeld, Faculty of technology, Networks and distributed Systems (Latest ed. 5 May 1998, 1999-02-08, Last modification 1999-06-15) <http://www.rvs.uni-bielefeld.de/publications/Incidents/DOCS/FBW.html> (Zugriff 2010-08-13)
- [LeiMa] Bernd Leitenberger: *Mars Climate Orbiter*. <http://www.bernd-leitenberger.de/mco.shtml> (Zugriff 2010-08-13)

- [Lig09] Peter Liggesmeyer: *Software-Qualität. Testen, Analysieren und Verifizieren von Software*. Spektrum Akademischer Verlag, Heidelberg, Berlin (2009) 2. Aufl., 526 S.
- [Lio96] Jacques-Louis Lions u.a.: *ARIANE 5 Flight 501 Failure – Report by the Inquiry Board*. Paris (19 July 1996) Kopie des Originals mit Folien „ESA Directorate of Launchers, CNES: *Flight A501 Immediate Post-Accident Analysis*“, 60 S., <http://esamultimedia.esa.int/docs/esa-x-1819eng.pdf> (Zugriff 2010-08-13), <http://www.di.unito.it/~damiani/ariane5rep.html> (Zugriff 2010-08-13)
- [MCA94] Main Commission Aircraft Accident Investigation Warsaw: *Report on the Accident to Airbus A320-211 Aircraft in Warsaw on 14 September 1993*. (March 1994) transcribed by Peter B. Ladkin (6 March 1996, 1999-02-08 Last modification 1999-06-15) <http://www.rvs.uni-bielefeld.de/publications/Incidents/DOCS/ComAndRep/Warsaw/warsaw-report.html> (Zugriff 2010-08-13)
- [Rob10] Ken Robinson: *Ariane 5 Flight 501 Failure—A Case Study of Errors*. School of Computer Science and Engineering, University of New South Wales (16th December 1996, revised 15th April 2010) 8 S., <http://www.cse.unsw.edu.au/~se4921/PDF/ariane5-article.pdf> (Zugriff 2010-08-13)
- [Ste99] Arthur G. Stephenson u.a.: *Mars Climate Orbiter Mishap Investigation Board Phase I Report* (November 10, 1999) 44 S., ftp://ftp.hq.nasa.gov/pub/pao/reports/1999/MCO_report.pdf (Zugriff 2010-08-13)
- [WikAr] Wikipedia: *Ariane V88*. (letzte Änderung 15. Juni 2010, 09:26 Uhr) http://de.wikipedia.org/wiki/Ariane_V88 (Zugriff 2010-08-13)