Universität Zürich[UZH]

Institut für Informatik

Martin Glinz Thomas Fritz

Software Engineering

Kapitel 10

Messen von Software

10.1 Grundlagen

10.2 Messtheorie

10.3 Qualität von Maßen

10.4 Produktmaße

10.5 Prozessmaße

10.6 Messen mit Qualitätsmodellen

10.7 Zielorientiertes Messen

Messung

Messen (to measure) – Ein interessierendes Merkmal eines Gegenstands (oder einer Menge von Gegenständen) quantitativ erfassen

Warum messen?

- Merkmalswerte
  - vergleichen
  - bewerten
  - statistisch auswerten
- mit der Zielsetzung
  - Qualität von Produkten bzw. Prozessen lenken
  - Erfahrungen quantifizieren
  - Entscheidungsgrundlagen gewinnen
  - Prognosen stellen

Was messen?

Gegeben: eine Menge von Gegenständen

- Festlegen von zu messenden Merkmalen der Gegenstandsmenge
- Bestimmung von Merkmalseigenschaften (Beziehungen, Operationen), welche das Maß berücksichtigen muss
- Produktmaße: Messung von Software-Qualitätsmerkmalen, z.B.
  - Komplexität
  - Zuverlässigkeit
  - Effizienz
- Prozessmaße: Messung von Prozess-Qualitäten, z.B.
  - Aufwand
  - Dauer
  - Fehlerkosten

Beispiel

- Gegenstandsmenge: Programme
- Mögliche interessierende Merkmale:
  Größe, Komplexität, Effizienz, ...
- Auswahl eines zu messenden Merkmals:
  zum Beispiel Größe
- Eigenschaften des Merkmals „Größe eines Programms“:
  Die Merkmalswerte sind
  - geordnet (Beziehung: „Programm x ist größer als Programm y“)
  - additiv (Operation: „Programm x und Programm y sind zusammen gleich groß wie Programm z“)
  - vervielfachbar (Operation: „Programm x ist 2 1/2 mal größer als Programm y“)

10.1 Grundlagen

10.2 Messtheorie

10.3 Qualität von Maßen

10.4 Produktmaße

10.5 Prozessmaße

10.6 Messen mit Qualitätsmodellen

10.7 Zielorientiertes Messen

Maße als quantitative Modelle

Ein Merkmal messen – Ein quantitatives Modell des zu messenden Merkmals bilden

Das Modell

- muss die Eigenschaften des gemessenen Merkmals adäquat wiedergeben

insbesondere

- Beziehungen zwischen Merkmalsausprägungen
- Operationen auf der Menge der Merkmalsausprägungen

Beispiel

Werden die Messwerte des Merkmals Größe von Programmen als reelle Zahlen modelliert, so

- müssen die Eigenschaften des Merkmals modelliert sein:
  - vergleichbar: $x \leq y$ ✔
  - additiv: $s = q + r$ ✔
  - vervielfachbar: $y = a\,x$, (a konstant) ✔
- dürfen Eigenschaften der reellen Zahlen, die keine Entsprechung im Original haben, nicht verwendet werden, zum Beispiel
  - keine negativen Zahlen
  - keine Multiplikation

Maß und Repräsentation

Ein Maß (measure) für das Merkmal M einer Gegenstandsmenge D ist eine Abbildung μ: D $\rightarrow$ S, welche jedem d $\in$ D einen Messwert μ(d) auf einer Skala (scale) S so zuordnet, dass M und S strukturähnlich oder homomorph sind.

Strukturähnlich heißt: Zu jeder Relation R auf M gibt es eine Relation R* auf S mit

(1) $R(M(d_1), M(d_2)) \Rightarrow R^*(\mu(d_1), \mu(d_2))$

(2) $R^*(\mu(d_1), \mu(d_2)) \Rightarrow$ $R(M(d_1), M(d_2))$ und die Aussage $R(M(d_1), M(d_2))$ ist sinnvoll interpretierbar.

(1) und (2) heißen auch Repräsentations- oder Homomorphiebedingung

Bedeutung der Strukturähnlichkeit (Homomorphie)

- Für Beziehungen bedeutet Strukturähnlichkeit:

  Zu jeder Beziehung b zwischen Merkmalsausprägungen von Gegenständen aus D gibt es eine Beziehung b' zwischen Skalenwerten aus S, so dass für alle $d_1, d_2 \in D$ die Regel

  $M(d_1)\ b\ M(d_2) \Leftrightarrow \mu(d_1)\ b'\ \mu(d_2)$

  plausibel (d.h. sinnvoll interpretierbar) ist

- Für Operationen bedeutet Strukturähnlichkeit:

  Zu jeder Operation $\otimes$ zwischen Merkmalsausprägungen von D gibt es eine Operation $\oplus$ zwischen Skalenwerten aus S, so dass für alle $d_1, d_2, d_3 \in D$ die Regel

  $M(d_1) \otimes M(d_2) = M(d_3) \Leftrightarrow \mu(d_1) \oplus \mu(d_2) = \mu(d_3)$

  plausibel (d.h. sinnvoll interpretierbar) ist

Bedeutung der Strukturähnlichkeit (Homomorphie) – 2

- Relationen auf der Skala S, zu denen es keine korrespondierende Relation auf den Merkmalsausprägungen gibt, dürfen nicht zur Bearbeitung von Messwerten verwendet werden.
- Formal ausgedrückt:

  Sei Q_s eine Relation auf der Skala S. Q_s darf genau dann zur Bearbeitung von Messwerten verwendet werden, wenn es eine auf M definierte Relation Q_o gibt und für alle $(s_1, s_2) \in Q_s$ gilt:

  Es existieren $d_1, d_2 \in D$ mit $\mu(d_1) = s_1$ und $\mu(d_2) = s_2$

  so, dass $Q_o\ (M(d_1), M(d_2)) \Leftrightarrow Q_s\ (s_1, s_2)$

Bemerkungen

- Etwas salopp könnte man sagen, ein Maß besteht aus
  - einer Menge von Gegenständen mit einem zu messenden Merkmal
  - einer Skala
  - einem Messverfahren (Abbildung der Gegenstände auf die Skala)
  - Strukturähnlichkeit zwischen Merkmalsmenge und Skala
- Häufig werden Maße definitorisch eingesetzt, d.h. sie definieren ein intuitives Merkmal einer Menge von Gegenständen.
- Ein Maß für Software wird in der Literatur oft auch als Metrik (metric) bezeichnet.

  Dieser Name ist jedoch schlecht gewählt, weil „Metrik“ in der Mathematik eine feste Bedeutung im Sinn eines verallgemeinerten Distanzbegriffs hat.

Beispiel

Sei γ ein Maß für das Merkmal Größe von Programmen

- Messverfahren: Zählen der Programmzeilen. Jede Zeile, die nicht leer ist oder ausschließlich Kommentar enthält, zählt als Programmzeile
- Skala: Teilmenge der reellen Zahlen
- Strukturähnlichkeit: Seien P_1, P_2, P_3 Programme mit den Größen $G(P_1)$, $G(P_2)$ und $G(P_3)$ und n eine nicht negative reelle Zahl.

  Die Regeln

  $G(P_1) \leq G(P_2) \Leftrightarrow \gamma(P_1) \leq \gamma(P_2)$ Vergleichbarkeit

  $G(P_1) + G(P_2) = G(P_3) \Leftrightarrow \gamma(P_1) + \gamma(P_2) = \gamma(P_3)$ Additivität

  $G(P_1) = n\, G(P_2) \Leftrightarrow \gamma(P_1) = n\, \gamma(P_2)$ Vervielfachung

  sind mit der gegebenen Abbildungsvorschrift für γ plausibel

Skalentypen

Abhängig davon, welche Relationen auf der Skala eines Maßes eine plausible Entsprechung im Definitionsbereich des Maßes haben, gibt es fünf verschiedene Skalentypen:

- Nominalskala (nominal scale)
- Ordinalskala (ordinal scale)
- Intervallskala (interval scale)
- Verhältnisskala (ratio scale); aufgrund einer falschen Übersetzung im Deutschen auch als Rationalskala bezeichnet
- Absolutskala (absolute scale)

Eigenschaften der Skalentypen

Typ	erlaubt	Eigenschaften
Nominalskala	= ≠	Reine *Kategorisierung* von Werten Nur nicht-parametrische Statistik
Ordinalskala	= ≠ < >	Skalenwerte *geordnet* und vergleichbar Medianwert, sonst nur nicht-parametrische Statistik
Intervallskala	= ≠ < > Distanz	Werte geordnet, *Distanzen* bestimmbar Mittelwert, Standardabweichung
Verhältnisskala (auch Rational-skala genannt)	= ≠ < > Distanz, (+, –) Vielfaches, %	Werte geordnet und in der Regel *additiv** Skala hat *absoluten Nullpunkt* Übliche parametrische Statistik
Absolutskala	= ≠ < > Distanz, (+, –) Vielfaches, %	Skalenwerte sind *absolute Größen* Sonst wie Verhältnisskala

* Verhältnisskalen sind meistens additiv, müssen es aber nicht zwingend sein

Formale Definition der Skalentypen – 1

- Formal werden die Skalentypen definiert über die Art der erlaubten Skalentransformationen:

  Mit welchen Transformationen kann eine Skala in eine äquivalente andere Skala des gleichen Typs überführt werden?

- Sekundär ergeben sich daraus die bekannten Eigenschaften der verschiedenen Skalentypen (Vergleichbarkeit, Addititvität, etc.)

- Hinweise
  - Um einfacher rechnen zu können, werden bei Intervall- und Verhältnisskalen üblicherweise Teilmengen der reellen Zahlen als Wertebereich verwendet.
  - Ist eine Skala additiv, so muss es mindestens eine Verhältnisskala sein. Die Umkehrung gilt nicht immer.

Formale Definition der Skalentypen – 2

- Nominalskala: Anwendung einer beliebigen injektiven Funktion
  ($f(x_1) = f(x_2) \Leftrightarrow x_1 = x_2$) auf alle Skalenwerte ergibt wieder eine Nominalskala
- Ordinalskala: Anwendung einer streng monotonen Funktion
  ($\forall x_1, x_2: x_1 > x_2 \Rightarrow f(x_1) > f(x_2) \vee \forall x_1, x_2: x_1 > x_2 \Rightarrow f(x_1) < f(x_2)$)
  auf alle Skalenwerte ergibt wieder eine Ordinalskala
- Intervallskala: Anwendung einer Lineartransformation
  ($f(x) = ax + b$) auf alle Skalenwerte ergibt wieder eine Intervallskala
- Verhältnisskala: Multiplikation aller Skalenwerte mit konstantem Faktor
  ($f(x) = ax$) ergibt wieder eine Verhältnisskala
- Absolutskala: Keine Transformation möglich

Skalen für Maße im Software Engineering

Beispiele

- Nominalskala: Testergebnisskala mit den Werten {erfüllt, nicht erfüllt, nicht getestet}
- Ordinalskala: Eignungsskala mit den Werten { – –, –, o, +, ++ }
- Intervallskala: Datumskala für Zeit
- Verhältnisskala: Anzahl-Codezeilen-Skala für Programmgröße
- Absolutskala: Zählskala für die Anzahl der Einzelanforderungen in einer Anforderungsspezifikation

Mini-Übung 10.1

Geben Sie zu folgenden Maßen je eine möglichst starke Skalentransformation an und begründen Sie daraus, welche Skala das betreffende Maß hat.

a) Leistungsbewertung mit Punkten, 0 = schlechteste, 100 = beste Leistung

b) Leistungsbewertung mit Noten, 1 = schlechteste, 6 = beste Leistung

c) Menge der gefundenen Fehler im Systemtest

Direkte und indirekte Maße

Direkte Maße

- Interessierende Merkmale in einfacher Weise direkt messbar
- Beispiele: Kosten, Durchlaufzeit

Indirekte Maße

- kein direktes Maß vorhanden oder Messung zu teuer
- messbare Indikatoren bestimmen
- Indikatoren müssen mit dem zu messenden Merkmal korreliert sein
- Indikatormaße bilden zusammen indirektes Maß für interessierendes Merkmal
- Beispiele: Portabilität, Benutzerfreundlichkeit

Beispiel: Messung von Portabilität

- Direktes Maß
  - Verhältnis Portierungsaufwand / Neuentwicklungsaufwand
  - Messung zu teuer
- Indirektes Maß mit drei Indikatoren

Indikator	**Skala**	**Messverfahren**	**Planwert**	**Schwellwert**
Anzahl Betriebssystem-Aufrufe / Anzahl Prozeduraufrufe	0-100%	Zählen im Code	5%	10%
Anteil hardware- oder betriebssystemabhängiger Module	0-100%	Zählen im Code	10%	15%
Anteil Nichtstandard Codezeilen	0-100%	Zählen, vgl. mit ISO-Standard	2%	5%

10.1 Grundlagen

10.2 Messtheorie

10.3 Qualität von Maßen

10.4 Produktmaße

10.5 Prozessmaße

10.6 Messen mit Qualitätsmodellen

10.7 Zielorientiertes Messen

Eigenschaften guter Maße – 1

Es gibt bessere und schlechtere Maße.

Eigenschaften eines guten Maßes:

- Validität

  Misst das Maß tatsächlich das zu messende Merkmal?

- Aussagekraft

  Sind die Messwerte sinnvoll interpretierbar?

- Schärfe

  Werden wahrnehmbar verschiedene Merkmale auf verschiedene Messwerte abgebildet?

Eigenschaften guter Maße – 2

- Auswertbarkeit

  Welche Auswertungen (z.B. Statistik) sind auf den Messwerten möglich?

- Verfügbarkeit
  - Kann ein Merkmal zu dem Zeitpunkt gemessen werden, wo die Messwerte benötigt werden?
  - Wie viel kostet die Messung?

- Stabilität / Reproduzierbarkeit
  - Wie empfindlich reagiert das Maß auf Störungen?
  - Liefern mehrfache Messungen des gleichen Merkmals (durch verschiedene Leute / in verschiedenen Umgebungen) die gleichen Messwerte?

Mini-Übung 10.2

Beurteilen Sie die Eigenschaften des wie folgt definierten Maßes M_{port} für Portabilität:

Sei P ein Programm, dessen Portabilität zu messen ist, und sei

- E_p der Aufwand zur Portierung von P auf eine neue Zielplattform
- E_n der Aufwand, P für die neue Zielplattform neu zu entwickeln

$M_{port} \equiv (1 - E_p / E_n)$

10.1 Grundlagen

10.2 Messtheorie

10.3 Qualität von Maßen

10.4 Produktmaße

10.5 Prozessmaße

10.6 Messen mit Qualitätsmodellen

10.7 Zielorientiertes Messen

Was messen

- Welche quantitativen Merkmale hat ein Stück Software?

- Es gibt eine Unzahl möglicher Maße.
- Das *IEEE Standard Dictionary of Measures of the Software Aspects of Dependability* definiert eine Reihe wichtiger Maße (IEEE 928.1-2005)

- Ausgewählte, wichtige Merkmale von Software:
  - Größe
  - Komplexität
  - Zuverlässigkeit
  - Funktionalität

Größenmaße

- Größe – Wie umfangreich ist die Software?
- Skala: Verhältnisskala
- Mögliche Maße: NCSS, Anzahl Zeichen

NCSS (Non-commented source statements)

- Zählung der Codezeilen ohne Kommentar- und Leerzeilen
- Genaue Zählregeln erforderlich
- Programmiersprachenabhängig
- Leicht messbar

Komplexitätsmaße

- Komplexität – Wie komplex ist die Struktur eines Stücks Software?
- Soll ein Indikator für Fehleranfälligkeit und Pflegbarkeit sein (umstritten)
- Skala: Maß wird in der Regel so konstruiert, dass mindestens eine Intervallskala resultiert
- Additivität: kontrovers
  - Problem: Die Kombination zweier Teilprogramme kann komplexer sein als die Summe der Komplexitäten der Teile → nicht additiv
  - Aus Bequemlichkeitsgründen wird in vielen Komplexitätsmaßen die Additivität angenommen
- Mögliche Maße: Eine Unmenge (Zuse 1990). Die bekanntesten sind:
  - Zyklomatische Komplexität (McCabe)
  - Software Science (Halstead)

Zyklomatische Komplexität (McCabe 1976)

- Messung des Flussgraphen G eines Programms mit der Formel

  v(G) = e - n + 2p

  - e Zahl der Kanten
  - n Zahl der Knoten
  - p Zahl der Endpunkte des Programms

- In Programmiersprachen mit geschlossenen Ablaufkonstrukten berechnet sich v(G) in einfacher Weise wie folgt:
  - Zähle alle Alternativen und Schleifen (**if**, **while**, **for**, etc.)
  - Addiere für jede Auswahl-Anweisung (**switch**, **CASE**) die Zahl der Fälle - 1
  - Addiere 1

Zyklomatische Komplexität – 2

- Problem der Validität:
  - Ein Spaghetti-Programm und ein wohlstrukturiertes Programm des gleichen Problems können die gleiche zyklomatische Komplexität haben
  - Sind diese Programme auch intuitiv gleich komplex?
- Eignet sich das Maß als Indikator für Fehleranfälligkeit?
  → Nicht besser als NCSS (Kafura und Canning, 1985)
- Skala: Verhältnisskala, aber nicht additiv: Die Aneinanderreihung zweier Programmstücke mit den Komplexitäten v_1 und v_2 hat die Komplexität v_1+v_2-1: eine kontraintuitive Eigenschaft
- Nützlich zur Berechnung der Zahl der Programmzweige beim strukturorientierten Test (vgl. Kapitel über Testen)

Software Science (Halstead 1975)

- Typischer Vertreter eines konstruierten Maßes
- Halstead postuliert die Bestimmbarkeit charakteristischer Kenngrößen für Programme aus vier Basisgrößen:
  - N_1 Zahl der Operatoren im Programm (+, -, >, <, ...)
  - η_1 Zahl der voneinander verschiedenen Operatoren
  - N_2 Zahl der Operanden im Programm (Variablen, Literale)
  - η_2 Zahl der voneinander verschiedenen Operanden

Software Science – 2

- Aus den Basisgrößen leitet Halstead verschiedene Kenngrößen ab, beispielsweise
  - Programmlänge

    $N = \eta_1 ld\eta_1 + \eta_2 ld\eta_2$
  - Zahl der Elementarentscheidungen

    $E = N\eta_1 N_2\ ld(\eta_1 + \eta_2) / 2\eta_2$
  - Programmierdauer T

    $T = E / S$ [sec] mit $S \approx 18$
- Die Validität der Halstead-Maße ist nie überzeugend nachgewiesen worden
- Software Science ist veraltet und sollte nicht mehr verwendet werden.

Zuverlässigkeitsmaße

MTTF (Mean Time to Failure)

- Das am häufigsten verwendete Zuverlässigkeitsmaß
- Messung der Zeit, die im Mittel zwischen zwei Fehlern verstreicht
  - in Betriebsstunden
  - oder Zahl von Transaktionen
- Prognose der MTTF
  - Testreihe mit zufälligen Testdaten
  - Verteilung der Testdaten muss der erwarteten Verteilung der Daten im realen Betrieb entsprechen
  - Prognoseaussagen mit statistischen Verfahren
  - Der geforderte Konfidenzgrad der Prognose bestimmt die Zahl der zu testenden Fälle

Zuverlässigkeitsmaße – 2

Fehlerdichte

- Die Fehlerdichte (gemessen in Anzahl Fehler / 1000 NCSS) ist ein weiteres mögliches Maß für Zuverlässigkeit.

Hinweis:

Das *IEEE Standard Dictionary of Measures of the Software Aspects of Dependability* enthält genaue Definitionen verschiedener Zuverlässigkeitsmaße (IEEE 928.1-2005)

Was messen

Typische interessierende Maße sind:

- Entwicklungskosten
- Produktivität
- Termin- und Kostentreue (Vergleich SOLL - IST)
- Fehler- bzw. Defektraten
- Fehler- bzw. Defektkosten
- Qualitätskosten

- Zur Berechnung dieser Maße müssen Basisgrößen gemessen werden

Basisgrößen

- Aufwand total
- Aufwand für Qualitätsmaßnahmen (insgesamt / nur für Fehlerbehebung)
- Durchlaufzeit
- Anzahl gefundener Fehler (vor / nach Produktfreigabe)
- Produktgröße (z.B. NCSS)

Voraussetzungen

Brauchbare Messwerte entstehen nur, wenn sie geeignet erfasst werden:

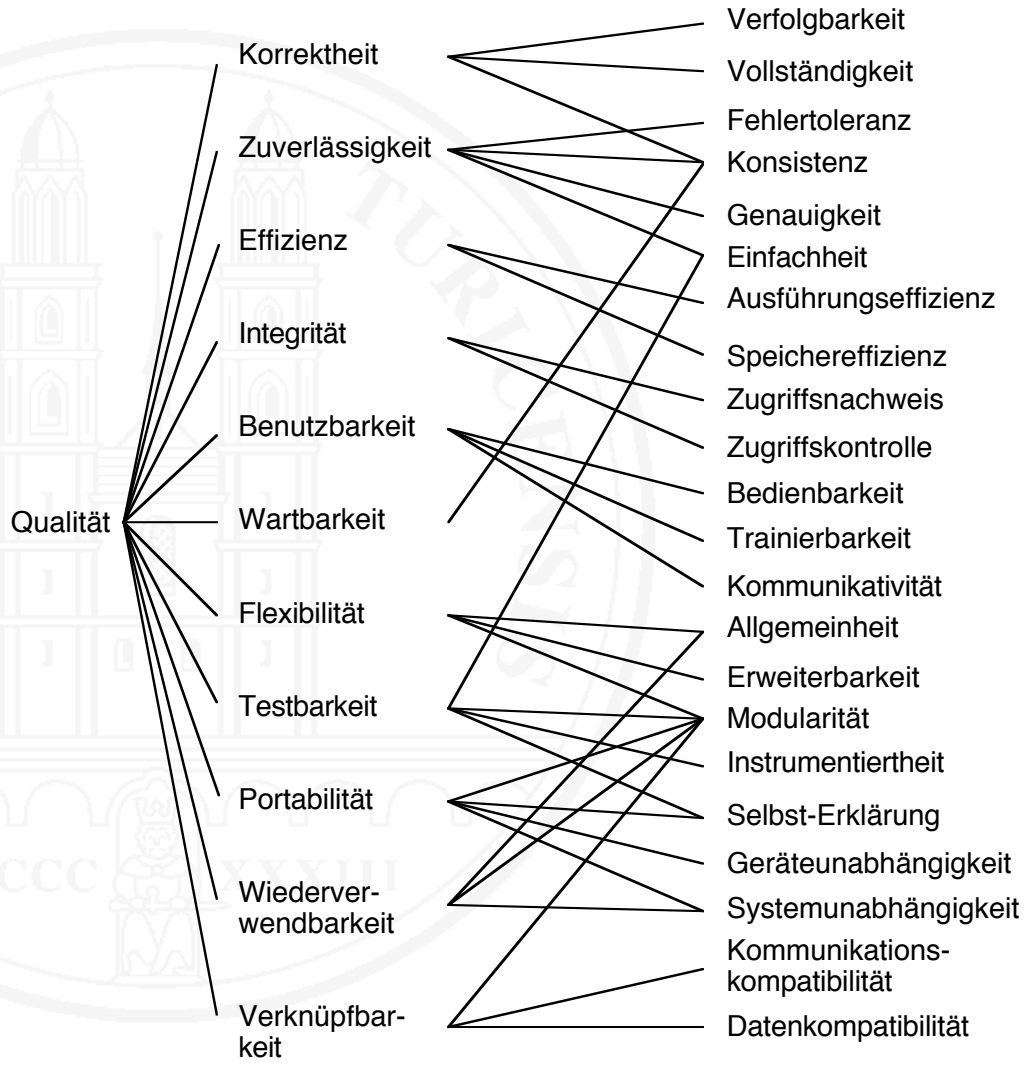
- Genaue Zählregeln
- Klar definierte Prozesse, die zum Beispiel präzise festlegen, wann ein Projekt beginnt bzw. endet
- Messwerkzeuge
- Teilautomatische Erfassung

Charakteristika von Qualitätsmodellen

- Ein Qualitätsmodell ermöglicht die Messung von Qualität mit Hilfe eines standardisierten Modells
- Ein Qualitätsmodell besteht typisch aus
  - einer Menge „allgemeingültiger“ Qualitätsziele (Faktoren)
  - einem Satz charakteristischer Merkmale zu jedem Faktor
  - messbaren Kenngrößen zu jedem Merkmal
- Typische Beispiele
  - Qualitätsmodell von McCall (1980)
  - Qualitätsmodell der ISO/IEC 9126* (vgl. Kapitel 2)

*Kürzlich ersetzt durch die neue Norm ISO/IEC 25010:2011 *Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models.* Diese Norm verwendet ein gegenüber ISO/IEC 9126 leicht verändertes Qualitätsmodell.

Das Qualitätsmodell von McCall



Vor- und Nachteile standardisierter Qualitätsmodelle

+ Qualitätsfaktoren sind über Merkmale und Kenngrößen nachvollziehbar und messbar definiert
+ die Vorstellungs- und Begriffswelt über Qualitäten wird vereinheitlicht

– Kausale Zusammenhänge zwischen Kenngrößen, Merkmalen und Faktoren sind hypothetisch und nicht statistisch abgesichert
– Standardisierte Qualitätsmodelle nehmen keine Rücksicht auf die individuellen Qualitätsforderungen von Projekten/Produkten

  Beispiel: im Modell nach McCall kommt Verfügbarkeit nicht vor. Dies ist jedoch für manche Systeme (z.B. Telefonvermittlung) ein extrem wichtiger Qualitätsfaktor

Ansätze für die Definition von Maßen

- Definitorischer Ansatz: Ausgehen von irgendwie definierten Maßen (z.B. Formeln von Halstead) mit der Hypothese, dass diese Maße eine gesuchte Größe messen

  ➠ Praktisch, aber Gefahr irrelevanter, nicht validierter Messungen

- Bequemlichkeitsansatz: Das messen, was einfach zu messen ist

  ➠ Einfach und billig, aber Gefahr sinnloser Messungen

- Zielorientierter Ansatz: Ausgehen von einem zu erreichenden qualitativen Ziel, Suche nach Maßen, welche dieses Ziel quantitativ charakterisieren

  ➠ Fokussierte Messung, zielbezogene Interpretation der Messwerte

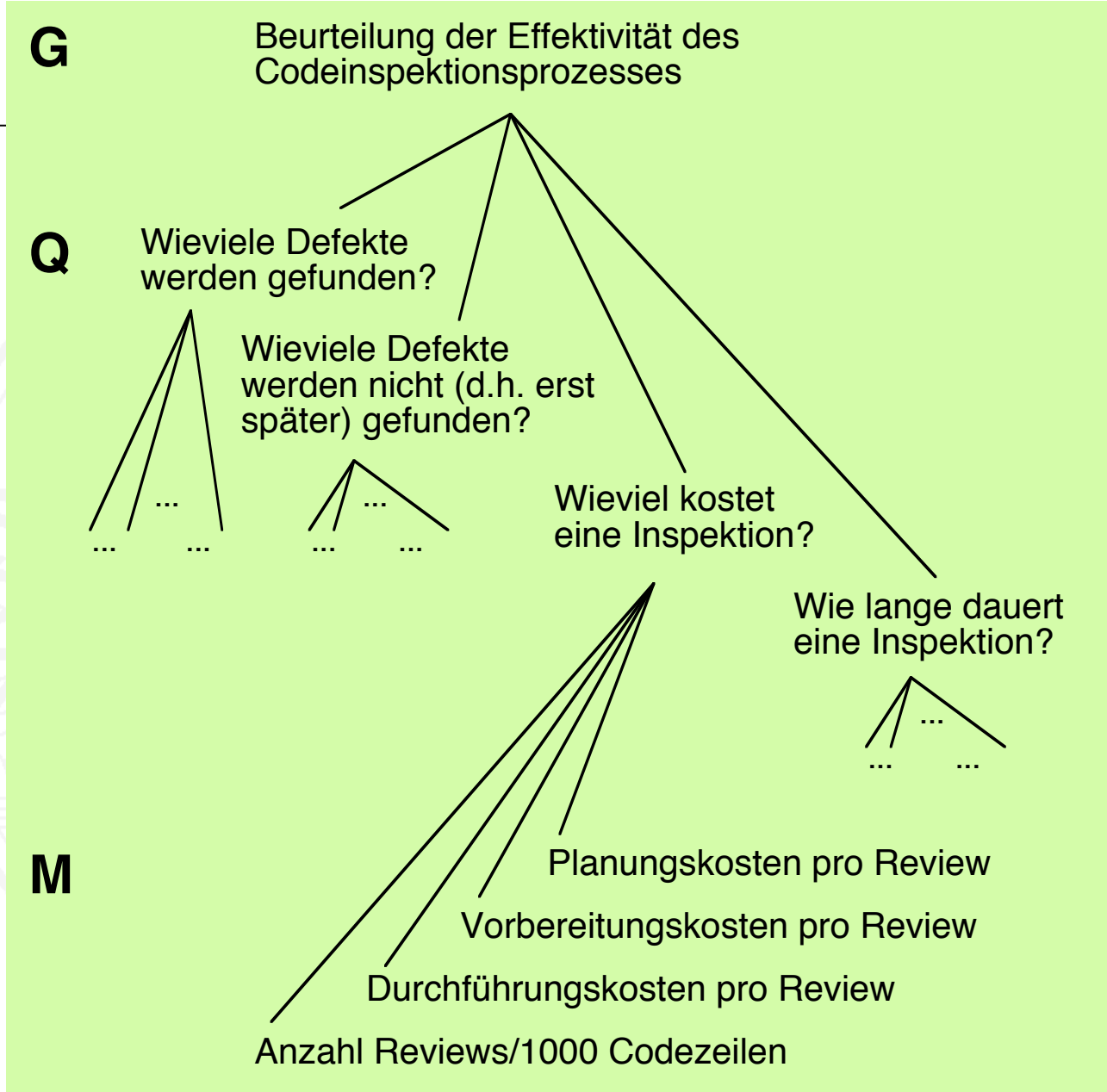
Der GQM (Goal-Question-Metric)-Ansatz

Bekanntester zielorientierter Ansatz (Basili und Rombach 1988, Basili 1992)

- Dreistufiges Vorgehen
  - **G**oal — Festlegen eines Qualitätsziels
  - **Q**uestion — Wie müssen die Fragen lauten, mit denen die Zielerreichung festgestellt wird?
  - **M**etric — Welches sind die Maße, mit denen die Fragen quantitativ beantwortet werden können?

- Vorteile
  - Es wird nur das gemessen, was dazu beiträgt, die gesetzten Ziele zu erreichen
  - Die Interpretation der ausgewählten Maße ist festgelegt

Beispiel


G
Beurteilung der Effektivität des Codeinspektionsprozesses
Q
Wieviele Defekte werden gefunden?
Wieviele Defekte werden nicht (d.h. erst später) gefunden?
Wieviel kostet eine Inspektion?
Wie lange dauert eine Inspektion?
M
Planungskosten pro Review
Vorbereitungskosten pro Review
Durchführungskosten pro Review
Anzahl Reviews/1000 Codezeilen

Literatur

A.J. Albrecht (1979). Measuring Application Development Productivity. *Proceedings Guide/Share Application Development Symposium* (Oct. 1979). 83–92

V. Basili, H.D. Rombach (1988). The TAME Project: Towards Improvement-Oriented Software Environments. *IEEE Transactions on Software Engineering* **14**(6):758–773.

V. Basili (1992): *Software Modeling and Measurement: The Goal-Question-Metric Paradigm*. Technical Report, University of Maryland, CS-TR-2956.

V. Basili, G. Caldiera, D. Rombach (1994). Goal Question Metric Paradigm. In J. J. Marciniak (ed.): *Encyclopedia of Software Engineering* **1**. New York: John Wiley&Sons. 528–532.

S.D. Conte, H.E. Dunsmore, V.Y. Shen (1986). *Software Engineering Metrics and Models*. Menlo Park, Ca.: Benjamin/Cummings.

Gilb, T. (1988). *Principles of Software Engineering Management*. Wokingham, etc.: Addison-Wesley.

IEEE (1990). *Standard Glossary of Software Engineering Terminology*. IEEE Std 610.12-1990. IEEE Computer Society Press.

R.B. Grady, D.L. Caswell (1987). *Software Metrics: Establishing a Company-Wide Program*. Englewood Cliffs, N.J.: Prentice Hall,

J.A. McCall, Matsumoto, M.T. (1980). *Software Quality Measurement Manual*, Vol. II. Rome Air Development Center, RADC-TR-80-109-Vol-2.

M.H. Halstead (1975). *Elements of Software Science*. Amsterdam: Elsevier North Holland.

Literatur – 2

IEEE 982.1 (2005). *IEEE Standard Dictionary of Measures of the Software Aspects of Dependability*. IEEE Standard 982.1-2005.

IEEE 1061 (1998). *IEEE Standard for a Software Quality Metrics Methodology*. IEEE Standard 1061-1998.

IFPUG (1994). *Function Point Counting Practices Manual, Release 4.0*. International Function Point Users Group. Westerville, Ohio (USA).

D. Kafura, J. Canning (1985). A Validation of Software Metrics Using Many Metrics and Two Resources. *Proc. 8th International Conference on Software Engineering (ICSE 1985)*. 378–385.

T.J. McCabe (1976). A Complexity Measure. *IEEE Transactions on Software Engineering* **SE-2**(4):308–320.

G.M. Weinberg (1993). *Quality Software Management. Vol 2: First Order Measurement.* New York: Dorset House.

H. Zuse (1990). *Software Complexity: Measures and Methods*. Amsterdam: De Gruyter.