Willkommen zur Vorlesung
Softwarekonstruktion
im Wintersemester 2012 / 2013

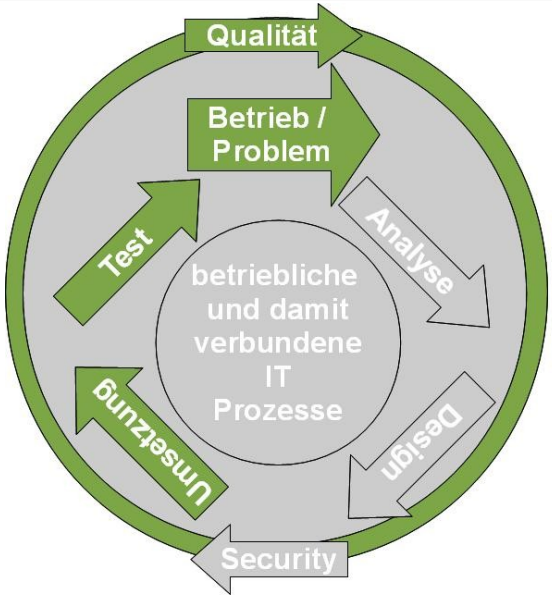
Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

1.2 Softwaremetriken

[inkl. Beiträge von Prof. Martin Glinz, Universität Zürich und Prof. Ian Sommerville, Univ. St. Andrews]

- Qualitätsmanagement
  - Grundlagen
  - Prozessqualität
  - Softwaremetriken
- Testen
- Modellgetriebene SW-Entwicklung



1.2 Software-metriken

- Einführung
- Grundlagen
- Komplexitätsmetriken
- Weitere Metriken und deren Bewertung

Software-Metriken

Software-Metriken sind ein Ansatz, mit dem Software-Prozess- und Produkt-Eigenschaften sowie ihre Beziehungen zueinander quantifiziert und gemessen werden.

Grundlegende Fragen, die wir in diesem Abschnitt betrachten werden, sind:

- Wie werden messbare Software-Merkmale ausgewählt ?
- Wie werden Software-Merkmale konsistent und objektiv gemessen ?
- Wie werden die Beziehungen zwischen verschiedenen Metriken hergestellt ?
- Wie werden Software-Metriken zur Produktivitäts- und Qualitätsverbesserung genutzt ?

Messung

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

Das heißt:

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

Warum Messen ?

Messungen helfen bei drei aufeinander aufbauenden Aufgabengebieten:

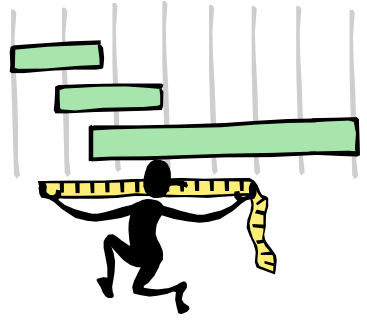
- Verstehen
  - Messung hilft zu verstehen, was während der Entwicklung eines Produkts passiert, indem sie Merkmale explizit herausstellt und quantifiziert.
- Steuern
  - Mit Hilfe von Messungen lassen sich Projekte steuern. Ausgehend von früheren Projekten lässt sich ein Erfahrungsschatz aufbauen, der die Vorgehensweise (Methoden, Techniken, ...) mit den erreichten Ergebnissen in Beziehung setzt.
  - Dieses Wissen lässt sich nutzen, um in zukünftigen Projekten Anpassungen vorzunehmen, damit die gewünschten Ziele erreicht werden.
- Verbessern
  - Die (aufgrund von Erfahrungen extrahierten) Zusammenhänge zwischen Vorgehensweisen und erreichten Zielen verfestigen sich zu allgemein akzeptierten Prinzipen, je öfter sie durch erfolgreiche Projekte bestätigt werden.
  - Bei nachfolgenden Projekten wird die Anwendung dieser Prinzipien von vorn herein geplant, Zeit und Ressourcen werden eher bereitgestellt.

Softwaremessung

- Softwaremessung bedeutet, dass einem Attribut des Softwareprodukte oder -prozesses ein numerischer Wert zugewiesen wird.
- Das erlaubt das objektive Vergleichen zwischen Techniken und Prozessen.
- Zur Softwaremessung gehört jede Art von Messung, welche ein Softwaresystem, -prozess oder zugehörige Dokumentation betrifft.
  - Z.B. die Anzahl der Codezeilen; der Fog-Index (vgl. später); die Anzahl der Personen-Tage, die für die Entwicklung einer Komponente benötigt wird.
- Das erlaubt, die Software und ihren Prozess zu quantifizieren.
- Kann verwendet werden, um Produktattribute vorherzusagen und den Softwareprozess zu steuern.

Software-Metriken: Definition

- Software-Metriken sind Messungen bestimmter Merkmale von
  - Software-Produkten
  - Software-Projekten und
  - Software-Prozessen
- zu deren Bewertung, Planung und Überwachung.



Maßtypen

- Produktmaße: Messung von Software-Merkmalen, z.B.
  - ?
- Projekt- bzw. Ressourcenmaße für Personen, Hardware, Software:
  - ?
- Prozessmaße für Software-Entwicklung und -Wartung: Messung von Prozess-Qualitäten, z.B.
  - ?

Maßtypen

- Produktmaße: Messung von Software-Merkmalen, z.B.
  - Größe, Funktionalität, Komplexität, Bedienbarkeit, Zuverlässigkeit, Sicherheit, Wartbarkeit, Übertragbarkeit, Integrierbarkeit, Effizienz
- Projekt- bzw. Ressourcenmaße für Personen, Hardware, Software:
  - ?
- Prozessmaße für Software-Entwicklung und -Wartung: Messung von Prozess-Qualitäten, z.B.
  - ?

Maßtypen

- Produktmaße: Messung von Software-Merkmalen, z.B.
  - Größe, Funktionalität, Komplexität, Bedienbarkeit, Zuverlässigkeit, Sicherheit, Wartbarkeit, Übertragbarkeit, Integrierbarkeit, Effizienz
- Projekt- bzw. Ressourcenmaße für Personen, Hardware, Software:
  - Anzahl Entwickler, % Overhead, Preis, Leistungsrate, Speicherkapazität, Geschwindigkeit, Genauigkeit, Nutzen
- Prozessmaße für Software-Entwicklung und -Wartung: Messung von Prozess-Qualitäten, z.B.
  - ?

Maßtypen

- Produktmaße: Messung von Software-Merkmalen, z.B.
  - Größe, Funktionalität, Komplexität, Bedienbarkeit, Zuverlässigkeit, Sicherheit, Wartbarkeit, Übertragbarkeit, Integrierbarkeit, Effizienz
- Projekt- bzw. Ressourcenmaße für Personen, Hardware, Software:
  - Anzahl Entwickler, % Overhead, Preis, Leistungsrate, Speicherkapazität, Geschwindigkeit, Genauigkeit, Nutzen
- Prozessmaße für Software-Entwicklung und -Wartung: Messung von Prozess-Qualitäten, z.B.
  - Aufwand, Dauer, Kosten, Fehlermeldungen, Änderungsanträge, Anforderungen, Anzahl Releases, Abstand zwischen Releases

[vgl. IEEE Standard „Dictionary of Measures to Produce Reliable Software“ (IEEE 928.1 1988)]

Verwendung von Messungen in der Qualitätssicherung

Ziel: Quantitative Aussagen bezüglich des von Natur aus abstrakten Produktes Software in Hinblick auf Qualitätsmerkmale. Gemessene Werte dahingehend beurteilen, ob sie den spezifizierten Anforderungen genügen.

Mögliche Ziele:

- Systemqualitätsattributen einen Wert zuweisen.
  - Durch die Messung der Eigenschaften einer System-komponente, (wie ihre zyklomatische Komplexität, vgl. später) wird versucht, Systemqualitätsattribute (wie Wartbarkeit) zu beurteilen.
- Systemkomponenten zu identifizieren, deren Qualität unter dem Standard liegen.
  - Messungen können individuelle Komponenten, deren Charakteristiken von der Norm abweichen, identifizieren. Zum Beispiel kann man die Komponenten messen, um die mit der höchsten Komplexität zu finden. Diese beinhalten am ehesten Fehler, da sie am schwersten zu verstehen sind.

Messungsannahmen

Die Möglichkeit und Sinnhaftigkeit von Softwaremessung beruht auf den folgenden Annahmen:

- Eine Softwareeigenschaft kann gemessen werden.
- Es existiert eine Verbindung zwischen den Dingen, die man messen kann, und den Dingen, die man wissen will. Man kann nur die internen Attribute messen, ist aber öfter an den externen Softwareattributen interessiert.
- Diese Verbindung wurde formalisiert und validiert.

Begrenzungen und Schwierigkeiten

- Es kann schwierig sein, herauszufinden, was gemessen werden muss, um wünschenswerte externe Qualitätsattribute zu bekommen.
- Metriken liefern immer nur punktuelle Aussagen bezüglich des untersuchten Aspekts:
  - Ermittelte Maßzahlen sind erst im Vergleich zu den Zahlen aus anderen untersuchten Programm(teil)en aussagekräftig.
  - Interpretation der Maßzahlen nur in relativ stabilen, wiederholbaren Prozessen sinnvoll.

1.2 Software-metriken

- Einführung
- **Grundlagen**
- Komplexitätsmetriken
- Weitere Metriken und deren Bewertung

Metriken als quantitative Modelle

Gegeben: eine Menge von Gegenständen

- Festlegung von zu messenden Merkmalen der Gegenstandsmenge.
- Bestimmung von Merkmalseigenschaften (Beziehungen, Operationen), welche die Metrik berücksichtigen muss.

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 (z.B. Größe: kleiner / größer)
- Operationen auf der Menge der Merkmalsausprägungen (z.B. Addition der Größe einzelner Softwarekomponenten)

Beispiel: Programmgröße (1)

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:

- ? (Beziehung: ?)
- ? (Operation: ?)

- ? (Operation: ?)

Beispiel: Programmgröße (1)

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:

- vergleichbar (Beziehung: „Programm x ist größer als Programm y“)
- additiv (Operation: „Programm x und Programm y sind zusammen gleich groß Programm y“)
- skalierbar (Operation: „Programm x ist 3 mal so groß wie Programm y“)

Beispiel: Programmgröße (2)

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

- müssen die Eigenschaften des Merkmals modelliert sein:
  - vergleichbar: ?
  - additiv: ?
  - skalierbar: ?
- dürfen Eigenschaften der natürlichen Zahlen, die keine Entsprechung im Original haben, nicht verwendet werden, zum Beispiel
  - keine Multiplikation zweier Programmgrößen.

Beispiel: Programmgröße (2)

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

- müssen die Eigenschaften des Merkmals modelliert sein:
  - vergleichbar: z.B. Größe(x) ≤ Größe(y) ok !
  - additiv: z.B. Größe(x)+Größe(y) = Größe(z) ok !
  - skalierbar: z.B. Größe(x) = 3*Größe(y) ok !
- dürfen Eigenschaften der natürlichen Zahlen, die keine Entsprechung im Original haben, nicht verwendet werden, zum Beispiel
  - keine Multiplikation zweier Programmgrößen.

Definition: Maß (informell)

- Informell ausgedrückt, besteht ein Maß 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.

[NB: Die Bezeichnungen „Maß“ und „Metrik“ haben in der Mathematik eine Bedeutung, die nicht unsere Verwendung der Begriffe hier trifft. Am ähnlichsten wäre der Begriff einer „Norm“, der allerdings auf Vektorräumen definiert ist.]

Definition: Maß (formal)

Sei M eine Menge zusammen mit Relationen und Operationen auf M.

Ein Maß ist eine Abbildung $\mu: M \rightarrow S$, welche jedem $d \in M$ einen Messwert $\mu(d)$ und jeder Relation (bzw. Operation) auf M eine Relation (bzw. Operation) auf S zuordnet, sodass die Struktur auf M bewahrt wird, das heißt:

1) Für jede Relation R auf M und Elemente $d_1, d_2 \in M$ gilt:
   $R(d_1,d_2) \Rightarrow \mu(R)\,(\mu(d_1),\mu(d_2))$
2) Für jede n-stellige Operation h auf M und Elemente $d_1,\ldots,d_n \in M$ gilt:
   $\mu(h(d_1,\ldots,d_n)) = \mu(h)\,(\mu(d_1),\ldots,\mu(d_2))$

Beispiel: Programmgröße (3)

Sei μ 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 natürlichen Zahlen
- Strukturbewahrend: Seien P1, P2 Programme und n eine natürliche Zahl. Ist μ strukturbewahrend ?

Beispiel: Programmgröße (3)

Sei μ 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 natürlichen Zahlen
- Strukturbewahrend: Seien P1, P2 Programme und n eine natürliche Zahl. Ist μ strukturbewahrend ? Ja: Die Regeln

  $P1 \leq P2 \Rightarrow \mu(P1) \leq \mu(P2)$ Vergleichbarkeit

  $\mu(P1::P2) = \mu(P1) + \mu(P2)$ Additivität

  (wobei $P1 \leq P2$ die Größen-Ordnung auf Programmen und P1::P2 die Konkatenation von Programmen) sind mit der gegebenen Abbildungsvorschrift für μ plausibel.

Skalentypen

Abhängig davon, welche Relationen bzw. Operationen auf dem Definitionsbereich des Maßes vorliegen, gibt es fünf verschiedene Skalentypen:

Typ	Relationen/ Operationen	Mögliche Analysen
Nominalskala		Reine *Kategorisierung* von Werten
Ordinalskala	< >	Skalenwerte *geordnet* und vergleichbar, Medianwert
Intervallskala	< >, Distanz	Werte geordnet, *Distanzen* bestimmbar, Mittelwert, Standardabweichung
Verhältnisskala (auch Rationalskala genannt)	< >, Distanz, (+, –) Vielfaches, %	Werte geordnet und in der Regel *additiv**, Skala hat *absoluten Nullpunkt. (* meistens, aber nicht zwingend)*
Absolutskala	< >, Distanz, (+, –) Vielfaches, %	Skalenwerte sind *absolute Größen (d.h. Nicht skalierbar)*, sonst wie Verhältnisskala.

Skalentypen: Beispiele 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: Relative Größe von Software-Komponenten in % des Gesamtsystems
- Absolutskala: Zählskala für die Anzahl der Einzelanforderungen in einer Anforderungsspezifikation

[NB: Die Beispiele sind modellierungsabhängig, d.h. bei anderer Modellierung könnten auch andere Arten von Skalen verwendet werden.]

Diskussion

Wie kann man zwei unterschiedliche Maße vereinheitlichen ? Das heißt, was kann man tun, um zwei Mengen von gleichartigen Objekten, die aber mit unterschiedlichen Maßen bewertet wurden, vergleichbar zu machen ? Wo liegen dabei Probleme ?

Diskussion

Wie kann man zwei unterschiedliche Maße vereinheitlichen ? Das heißt, was kann man tun, um zwei Mengen von gleichartigen Objekten, die aber mit unterschiedlichen Maßen bewertet wurden, vergleichbar zu machen ? Wo liegen dabei Probleme ?

Antwort: Folgende alternative Vorgehensweisen sind möglich:

1) Eines der Maße als Grundlage nehmen und die Menge an Objekten, die damit nicht bewertet wurden, erneut nach diesem Maß bewerten. Kann je nach Maß sehr aufwendig (oder unmöglich) sein.
2) Abbildung mittels Abbildungsfunktion von einer Skala auf die andere. Nur möglich wenn Skalen sehr ähnlich. Sehr leicht durchführbar, aber Gefahr von Informationsverlust bzw. dem Hinzufügen von Information, die nicht in der Ursprungs-Skala vorhanden war.
3) Bildung einer neuen Skala und zweier Abbildungsfunktionen, die die gegebenen Skalen umwandeln. Problem liegt in der Bildung der neuen Skala, sodass keine Informationen verloren gehen oder hinzugefügt werden. Es besteht die Gefahr, dass lediglich eine Ordnung zwischen den Elementen der zwei gegebenen Skalen definiert wird, ohne dass sie wirklich vergleichbar wären.

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 Merkmale korreliert sein.
- Indikatormaß 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

Diskussion: Was könnten indirekte Indikatoren für den Portierungsaufwand sein ? (D.h., welche technischen Artefakte, die sich leicht zählen lassen, beeinflussen den Portierungsaufwand ?)

Beispiel: Messung von Portabilität

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

Indikator	Skala	Mess-verfahren	Plan-wert	Schwell-wert
Anzahl Betriebssystem-Aufrufe / Anzahl Prozeduraufrufe	0-100%	Zählen im Code	5%	10%
Anteil Hardware- oder Betriebs-system-abhängiger Module	0-100%	Zählen im Code	10%	15%
Anteil Nichtstandard-Codezeilen	0-100%	Zählen, vgl. mit ISO-Standard	2%	5%

Wünschenswerte Eigenschaften einer Metrik

- Validität: Misst die Metrik tatsächlich das zu messende Merkmal ?
- Aussagekraft: Sind die Messwerte sinnvoll interpretierbar ?
- Schärfe: Werden wahrnehmbar verschiedene Merkmale auf verschiedene Messwerte abgebildet ?
- 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 ? Wieviel kostet die Messung ?
- Stabilität / Reproduzierbarkeit: Wie empfindlich reagiert die Metrik auf Störungen ? Liefern mehrfache Messungen des gleichen Merkmals (durch verschiedene Leute / in verschiedenen Umgebungen) die gleichen Messwerte ?

Dynamische und statische Metriken

Dynamische Metriken...

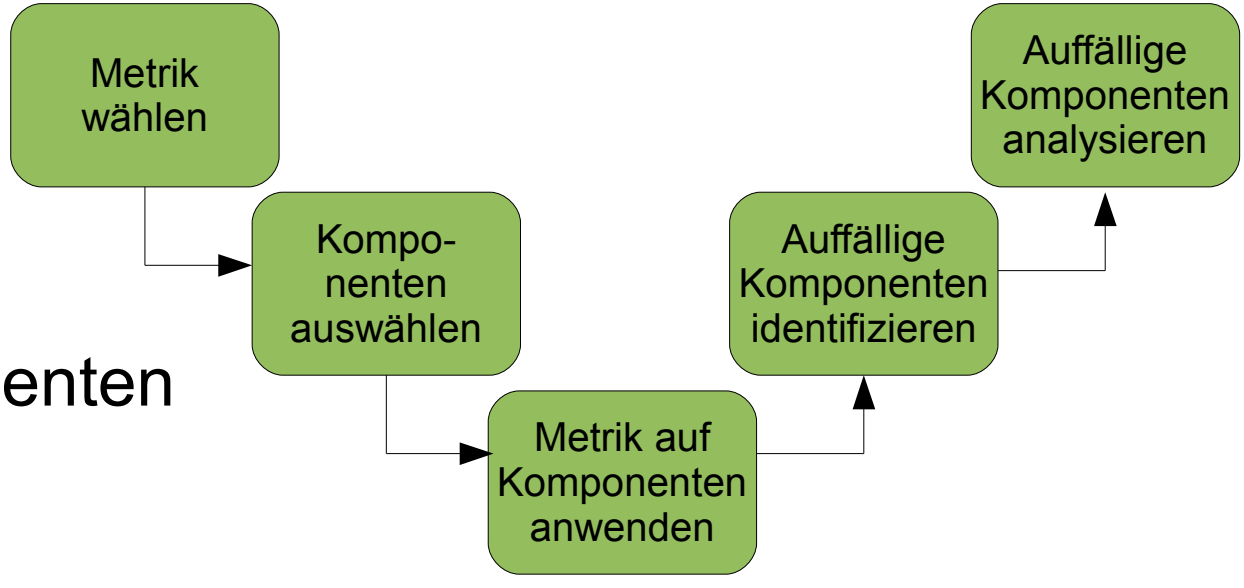
- … werden durch Messungen des Programms während der Laufzeit gesammelt
- … sind eng mit Software-Qualitätsattributen wie Effizienz und Zuverlässigkeit verbunden (es ist relativ einfach, die Antwortzeit (Performance) oder die Anzahl der Fehler (Funktionsfähigkeit) eines Systems zu messen).

Statische Metriken...

- … werden durch Messung einer Darstellung des Programms gesammelt.
- … haben eine indirekte Verbindung zu den Qualitätsattributen wie Komplexität, Verständlichkeit und Wartbarkeit (muss versuchen, eine Beziehung zwischen diesen Metriken und den Eigenschaften, wie Komplexität, Verständlichkeit und Wartbarkeit, herzustellen).

Analyse von Softwarekomponenten

- Softwarekomponenten können separat mit Hilfe einer Reihe von Metriken analysiert werden.
- Die Werte dieser Metriken können dann für verschiedene Komponenten und mit historisch gesammelten Daten, die von früheren Objekten stammen, verglichen werden.
- Anomale Messungen, die erheblich von der Norm abweichen, lassen vermuten, dass es Probleme mit der Qualität dieser Komponenten gibt.



1.2 Software-metriken

- Einführung
- Grundlagen
- **Komplexitätsmetriken**
- Weitere Metriken und deren Bewertung

Größenmaße

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

NCSS (Non-commented source statements)

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

Komplexitäts-Metriken

Komplexität – Wie komplex ist die Struktur eines Stücks Software ?

- Soll ein Indikator für Fehleranfälligkeit und Pflegbarkeit sein (umstritten).
- Skala: Metrik 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.

Es existieren viele verschiedene Komplexitäts-Metriken.
Die bekannteste ist: Zyklomatische Komplexität (McCabe).

Zyklomatische Zahl eines zyklischen Graphen

Die zyklomatische Komplexität eines Programmes wird anhand Kontrollfluss-graphen definiert. Daher zunächst einige Definitionen zu Graphen:

- Ein zyklischer Graph lässt sich als Vektorraum (VR) auffassen.
  [Grundmenge: Menge seiner Eulerschen Teilgraphen. Eulerscher Teilgraph = ein Zyklus, der alle Kanten eines Graphen genau einmal enthält.]
- Somit gibt es für den gegebenen Graphen eine Menge linear unabhängiger Zyklen (Basis des VR, erzeugt in Linearkombination alle Eulerschen Teilgraphen).

Sei G ein (zyklischer) Graph.

- Die zyklomatische Zahl (cyclomatic number) cn(G) wird definiert als Anzahl der linear unabhängigen Pfade von G (= Dimension des o.g. VR).

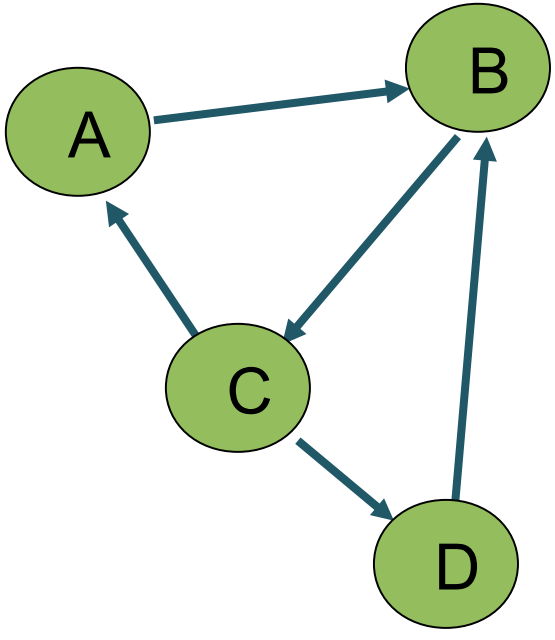
Man kann zeigen, dass cn(G) = e - v + 1 gilt, wobei:

- e = Anzahl der Kanten (edges)
- v = Anzahl der Knoten (vertices).

Alternative Definition zur Veranschaulichung

Alternative (äquivalente) Definition:
cn(G) = Anzahl maximal zu entfernender Kanten eines stark zusammenhängenden Graphen, um einen Spannbaum* zu erhalten.

Beispiel:



-
-

=> *cn*(G) =

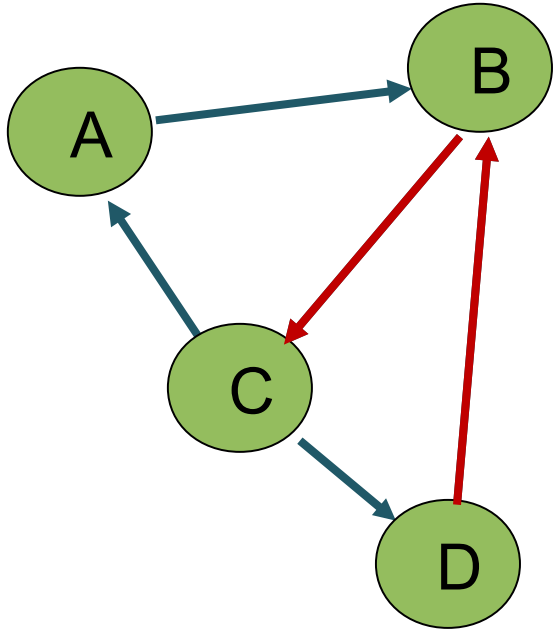
* Spannbaum = Teilgraph, der ein Baum ist und alle Knoten dieses Graphen enthält.
(Baum: Graph, in dem je zwei Knoten durch genau einen einfachen Pfad verbunden sind; einfacher Pfad enthält keinen Knoten mehrfach.)

Alternative Definition zur Veranschaulichung

Alternative (äquivalente) Definition:
cn(G) = Anzahl maximal zu entfernender Kanten eines stark zusammenhängenden Graphen, um einen Spannbaum* zu erhalten.

Beispiel:

- Entfernung der beiden Kanten B-C und D-B ergibt einen Spannbaum des Graphen.
- Mehr als zwei Kanten können nicht entfernt werden, da zwei Kanten keinen Spannbaum für vier Knoten ergeben können.

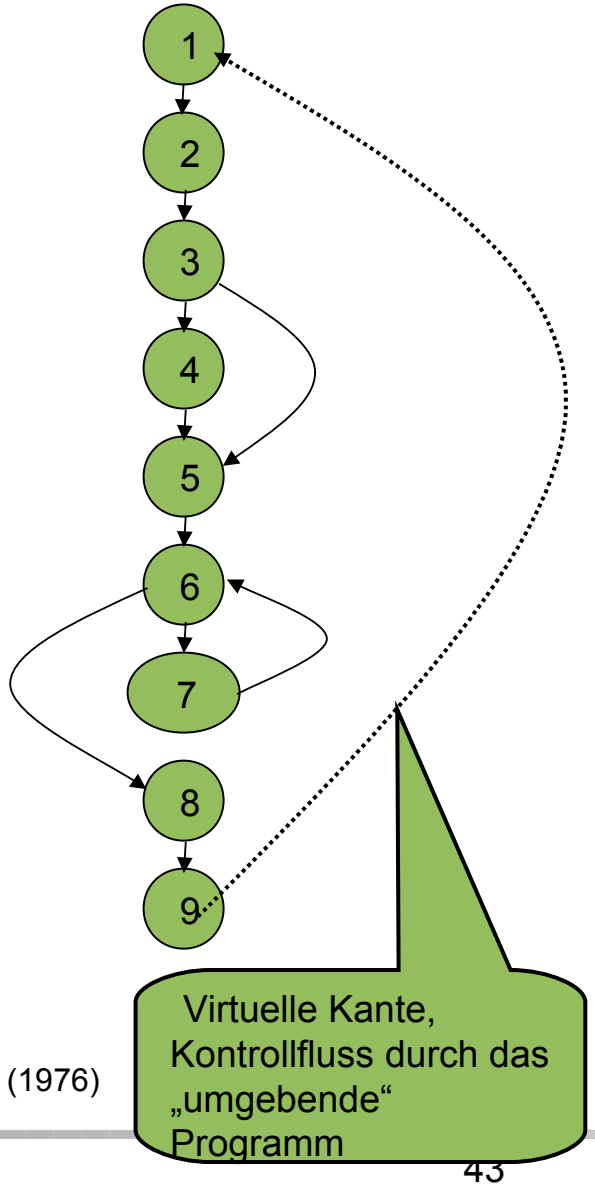


=> *cn*(G) = 2 (= 5 - 4 + 1)

* Spannbaum = Teilgraph, der ein Baum ist und alle Knoten dieses Graphen enthält.
(Baum: Graph, in dem je zwei Knoten durch genau einen einfachen Pfad verbunden sind; einfacher Pfad enthält keinen Knoten mehrfach.)

Zyklomatische Komplexität eines Programmes

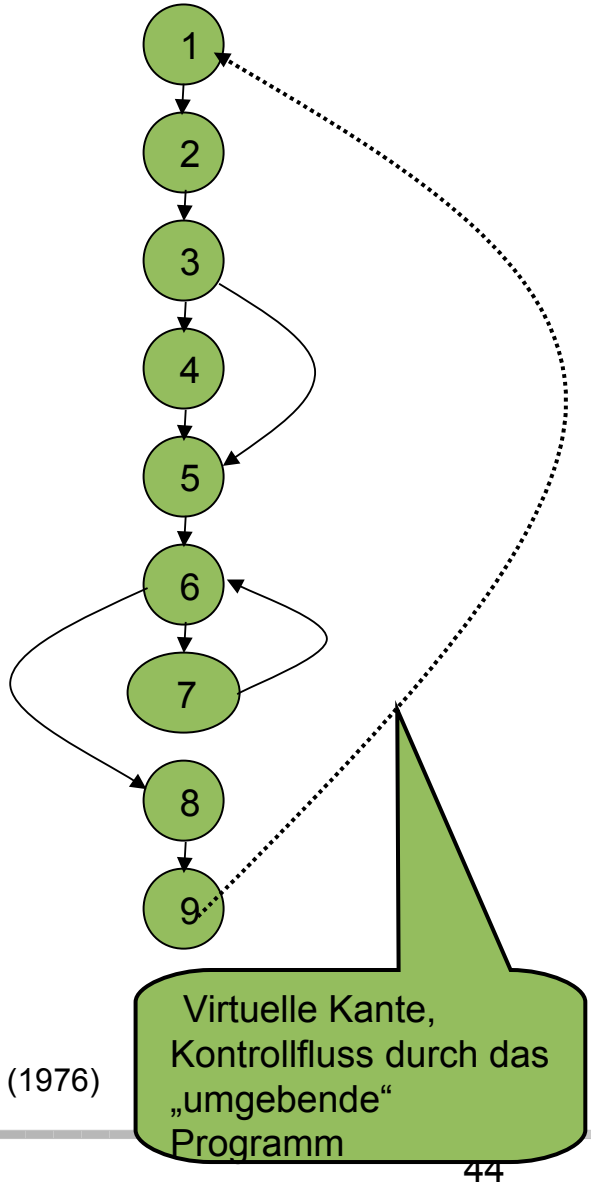
- Um diese Zahl als Metrik für Programme definieren zu können, müssen wir dem Kontrollflussgraphen eine „virtuelle“ Kante vom Endknoten zurück zum Anfangsknoten hinzufügen, um zyklischen Graphen zu erhalten.
  (Annahme: Das Programm hat nur einen Endpunkt.)
- $cc(G) = e - v + 2$ ist somit die zyklomatische Komplexität („McCabe-Metrik“) eines Kontrollflussgraphen G.
- Beispiel: $cc(G) = ?$



T.J. McCabe: A Complexity Measure, IEEE Transactions on Software Engineering Vol. 2, No. 4, p. 308 (1976)

Zyklomatische Komplexität eines Programmes

- Um diese Zahl als Metrik für Programme definieren zu können, müssen wir dem Kontrollflussgraphen eine „virtuelle“ Kante vom Endknoten zurück zum Anfangsknoten hinzufügen, um zyklischen Graphen zu erhalten.
  (Annahme: Das Programm hat nur einen Endpunkt.)
- $cc(G) = e - v + 2$ ist somit die zyklomatische Komplexität („McCabe-Metrik“) eines Kontrollflussgraphen G.
- Beispiel: $cc(G) = 10 - 9 + 2 = 3$



T.J. McCabe: A Complexity Measure, IEEE Transactions on Software Engineering Vol. 2, No. 4, p. 308 (1976)

Geschlossene Ablaufkonstrukte bzw. mehrere Endpunkte

Bemerkung: In Programmiersprachen mit **geschlossenen Ablaufkonstrukten** berechnet sich *cc*(G) wie folgt:

- Zähle alle Verzweigungen und Schleifen (**if**, **while**, **for**, etc.).
- Addiere für jede Auswahl-Anweisung (**switch**, **CASE**) die Zahl der Fälle - 1.
- Addiere 1.

Bemerkung: Bei einem Programm mit mehreren Endpunkten muss für jeden Endpunkt eine „virtuelle“ Kante zum Startpunkt eingefügt werden, um einen zyklischen Graph (laut Definition zyklische Zahl) zu erhalten.

Zyklomatische Komplexität des Kontrollflussgraphen G eines Programms **mit mehreren Endpunkten** (und einem Startpunkt) ist daher: $cc(G) = e - v + p + 1$ (p: Zahl der Endpunkte des Programms)

Bewertung der zyklomatischen Komplexität

- Zyklomatische Komplexität höher als 10 ist nach McCabe nicht tolerabel => Überarbeitung des Programmteils !
- Problem der Validität: Ein Spaghetti-Programm und ein wohlstrukturiertes Programm des gleichen Problems können die gleiche zyklomatische Komplexität haben.
- Eignet sich die Metrik als Indikator für Fehleranfälligkeit ?
  → Nicht besser als Non-Commented Source Statements (NCSS) (vgl. Kafura und Canning, 1985)
- Verwendete Skala: Absolutskala, aber nicht additiv:
  Die Aneinanderreihung zweier Programmstücke mit den Komplexitäten c1 und c2 hat die Komplexität c1+c2-1:
  eine kontraintuitive Eigenschaft.

Zyklomatische Komplexität vs. Testaufwand

Zyklomatische Komplexität gibt Auskunft über den Testaufwand:

- Zyklomatische Komplexität = Anzahl der unabhängigen Pfade im Programmstück (siehe oben)
- Zyklomatische Komplexität – 1 = Anzahl der Entscheidungen im Kontrollflussgraph eines strukturierten Programms
  (d.h. Programmiersprache mit **geschlossenen Ablaufkonstrukten**, s.o.)
- Oft wird die 100%-ige Ausführung aller Anweisungen und Verzweigungsmöglichkeiten eines Programms verlangt – dafür könnten alle unabhängigen Pfade durch den Kontrollflussgraphen einmal »durchlaufen« werden.
- Die zyklomatische Komplexität gibt somit eine obere Grenze für die Anzahl der benötigten Testfälle zur Erreichung dieses Kriteriums an (vgl. Kapitel 2: Testen).

„Schlechte“ Maße

Typischer Vertreter eines konstruierten Maßes: Halstead-Maß (Halstead 1975).

Halstead postuliert die Bestimmbarkeit charakteristischer Kenngrößen für Programme aus vier Basisgrößen:

- N1 Zahl der Operatoren im Programm (+, -, >, <, ...)
- η1 Zahl der voneinander verschiedenen Operatoren
- N2 Zahl der Operanden im Programm (Variablen, Literale)
- η2 Zahl der voneinander verschiedenen Operanden
- N = N1 + N2 Implementierungslänge

Aus den Basisgrößen leitet Halstead verschiedene Kenngrößen ab, beispielsweise Halsteadprogrammlänge: $L = \eta1*\log_2(\eta1) + \eta2 * \log_2(\eta2)$

Die Validität der Halstead-Maße ist nie überzeugend nachgewiesen worden. Sie sind veraltet und sollten nicht mehr verwendet werden.

Weitere Komplexitätsmaße

Softwaremetrik	Beschreibung
Fan-in / Fan-out	**Fan-in** einer Funktion oder Methode X ist die Anzahl der Funktionen oder Methoden, die X aufrufen. **Fan-out** ist die Anzahl der Funktionen oder Methoden, die von X aufgerufen werden. **Hoher Fan-in:** X eng mit dem Rest des System verbunden. Änderungen an X können weitreichende Folgewirkungen haben. **Hoher Fan-out:** gesamte Komplexität von X ist hoch, da Steuerungs-logik von X die aufgerufenen Komponenten koordinieren muss.
Länge der Bezeichner	Maß der durchschnittlichen Länge von Bezeichnern (Namen von Variablen, Klassen, Methoden, etc.) in einem Programm. Je länger sie sind, desto aussagekräftiger sind sie wahrscheinlich (und damit das Programm verständlicher).
Tiefe der Verschachte-lung	Maß der Tiefe der Verschachtelung von if-Bedingungen in einem Programm. Tief verschachtelte if-Bedingungen sind schwer zu verstehen und potentiell fehleranfälliger.
Fog-Index	Maß der durchschnittlichen Länge von Wörtern und Sätzen in einem Dokument. Je höher der Fog-Index eines Dokuments ist, desto schwerer ist es zu verstehen.

Die objektorientierte Metriken-Suite „CK“

Softwaremetrik	Beschreibung
Gewichtete Methoden pro Klasse (Weighted Method Complexity, WMC)	Anzahl der Methoden in jeder Klasse, gewichtet durch die Komplexität: WMC = $\Sigma C(i)$ mit C(i) = Komplexität von Methode i Einfache Methode kann Komplexität 1 haben; größere und komplexere sehr viel höheren Werte. Komplexe Objekte sind meistens schwerer zu verstehen.
Tiefe des Vererbungsbaums (Depth of Inheritance Tree, DIT)	Maximale Tiefe der Generalisierungshierarchie. Repräsentiert die Anzahl der Ebenen in einem Vererbungsbaum, in dem Unterklassen Attribute und Operationen (Methoden) von einer Superklasse erben. Je tiefer der Baum, desto komplexer ist das Design: Man muss viele Klassen verstehen, um das unterste Blatt des Vererbungsbaums nachvollziehen zu können.
Zahl der Kinder (Number of Children, NOC)	Anzahl der direkten Unterklassen. Maß für die unmittelbaren Unterklassen einer Klasse. Misst die Breite einer Klassenhierarchie, während DIT die Tiefe repräsentiert. Ein hoher NOC-Wert kann eine hohe Wiederverwendung anzeigen. Es sollte mehr Anstrengung in die Validierung der Basisklassen fließen, da die Zahl der Unterklassen, die von ihnen abhängen, groß ist.

Die objektorientierte Metriken-Suite „CK“

Softwaremetrik	Beschreibung
Kopplung von Klassen (CBO)	Klassen C und D sind gekoppelt, wenn Methoden von C Methoden oder Variablen von D benutzen. CBO ist ein Maß dafür, wie viele Kopplungen existieren. Ein hoher CBO-Wert zeigt, dass die Klassen in einem hohen Maße voneinander abhängig sind. Damit ist es wahrscheinlicher, dass die Änderung einer Klasse viele Klassen im Programm betrifft.
Reaktion einer Klasse (RFC)	Maß für die Anzahl der Methoden, die potentiell als Antwort auf eine Nachricht an die umgebene Klasse ausgeführt werden können. RFC ist mit der Komplexität verbunden. Je höher der RFC-Wert, desto komplexer und wahrscheinlich fehleranfälliger ist eine Klasse.
Mangel an Zusammenhalt in Methoden (Lack of Cohesion of Methods, LCOM)	Anzahl der durch die Methoden einer Klasse gemeinsam benutzten Instanzvariablen. Differenz der Anzahl von Methodenpaaren mit geteilten Attributen und der Anzahl ohne diese. Diese Metrik existiert in verschiedenen Variationen. Es ist nicht klar, ob dieses Maß irgendeine zusätzliche Information zu den anderen Metriken liefert.

Weitere Metriken für objektorientierte Programme

Kürzel	Bezeichnung	Erläuterung
NOV	Number of Variables	Anzahl der Instanzvariablen (member variables) einer Klasse
NOM	Number of Methods	Anzahl der Methoden (Operationen) einer Klasse
NORM	Number of Redefined Methods	Anzahl der in einer Klasse redefinierten Methoden

Diskussion

Komplexitätsmaße sollen unter anderem ein Indikator für Fehleranfälligkeit und Pflegbarkeit sein. Welches Problem ergibt sich, wenn ein Programmierer an einem Komplexitätsmaß gemessen wird, um die beiden Eigenschaften zu steuern ?

Diskussion

Komplexitätsmaße sollen unter anderem ein Indikator für Fehleranfälligkeit und Pflegbarkeit sein. Welches Problem ergibt sich, wenn ein Programmierer an einem Komplexitätsmaß gemessen wird, um die beiden Eigenschaften zu steuern ?

Antwort:

1) Die Komplexität von Code ergibt sich oft bereits aus der Komplexität des zu lösenden Problems und ist somit nur teilweise durch den Programmierer steuerbar. Nur wenn man mehrere Lösungen für das gleiche (oder ähnliche) Problem durch Metriken vergleicht, kann man die Qualität der Programmierung vergleichen.

2) Wenn die Metrik bekannt ist, kann der Programmierer den Code daraufhin optimieren, ohne notwendigerweise die Qualität zu verbessern.

1.2 Software-metriken

- Einführung
- Grundlagen
- Komplexitätsmetriken
- **Weitere Metriken und deren Bewertung**

Zuverlässigkeitsmaße (1)

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 to Produce Reliable Software“ enthält genaue Definitionen verschiedener Zuverlässigkeitsmaße (IEEE 928.1:1988).

Beziehungen zwischen externen und internen Attributen

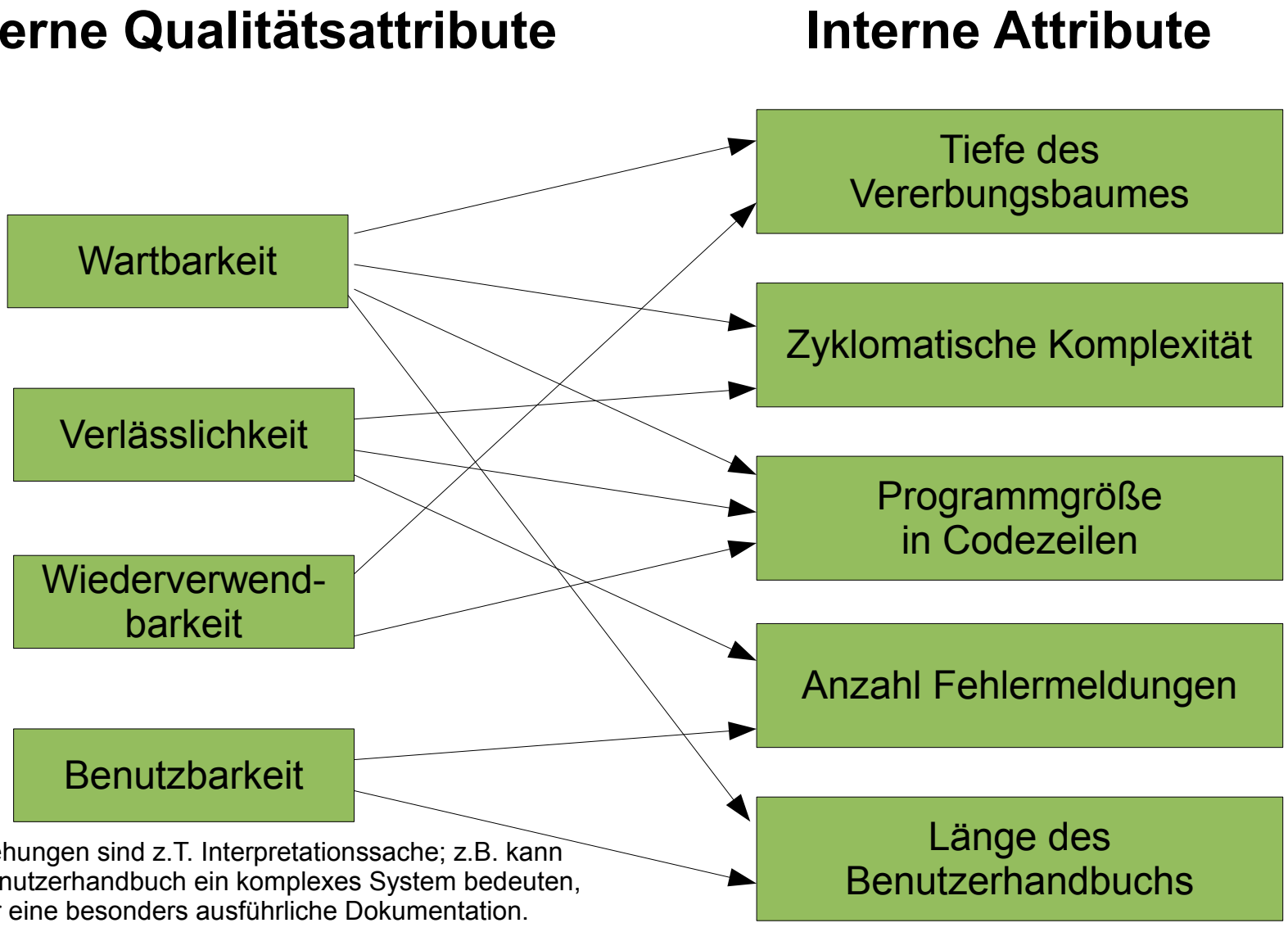
Externe Qualitätsattribute

- Wartbarkeit
- Verlässlichkeit
- Wiederverwend-barkeit
- Benutzbarkeit

Interne Attribute

- Tiefe des Vererbungsbaumes
- Zyklomatische Komplexität
- Programmgröße in Codezeilen
- Anzahl Fehlermeldungen
- Länge des Benutzerhandbuchs

Beziehungen zwischen externen und internen Attributen



NB: Die Beziehungen sind z.T. Interpretationssache; z.B. kann ein langes Benutzerhandbuch ein komplexes System bedeuten, oder auch nur eine besonders ausführliche Dokumentation.

[Quelle: Ian Sommerville: Software Engineering, 9th edition]

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

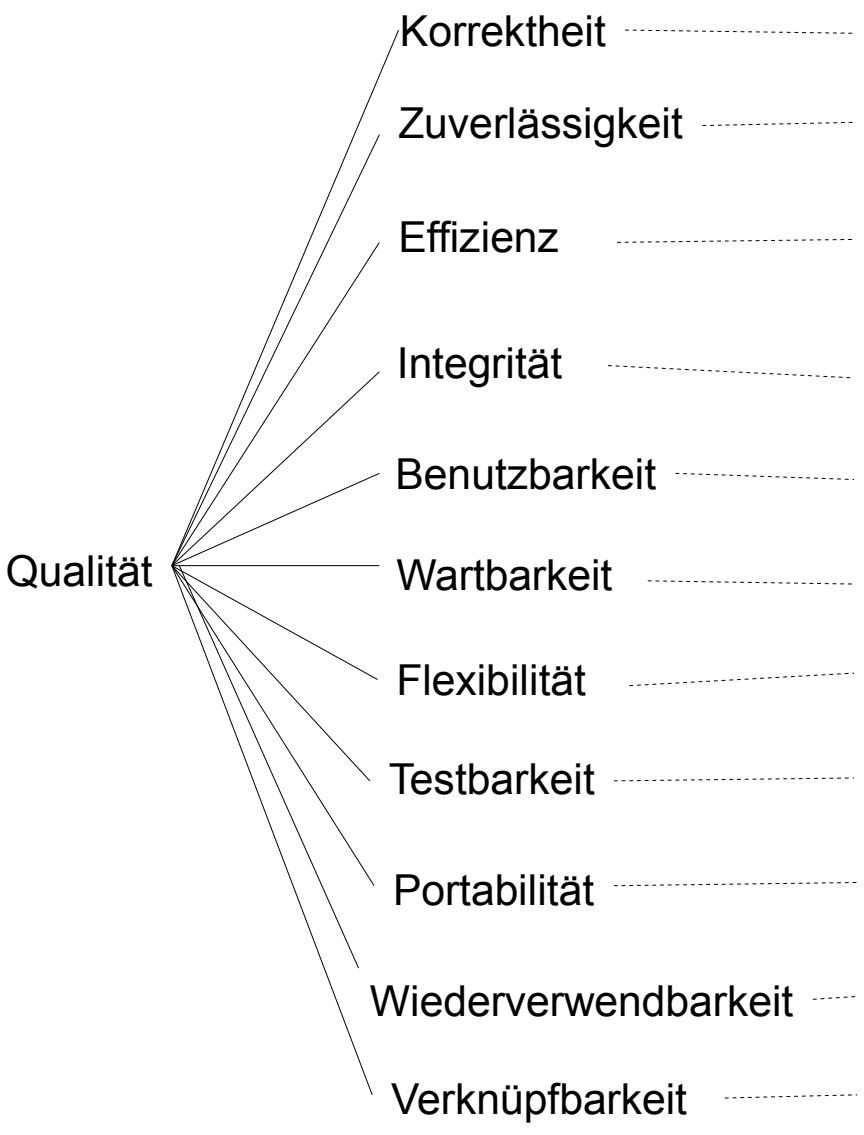
Brauchbare Messwerte entstehen nur, wenn sie geeignet erfasst werden:

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

Charakteristika von Qualitätsmodellen

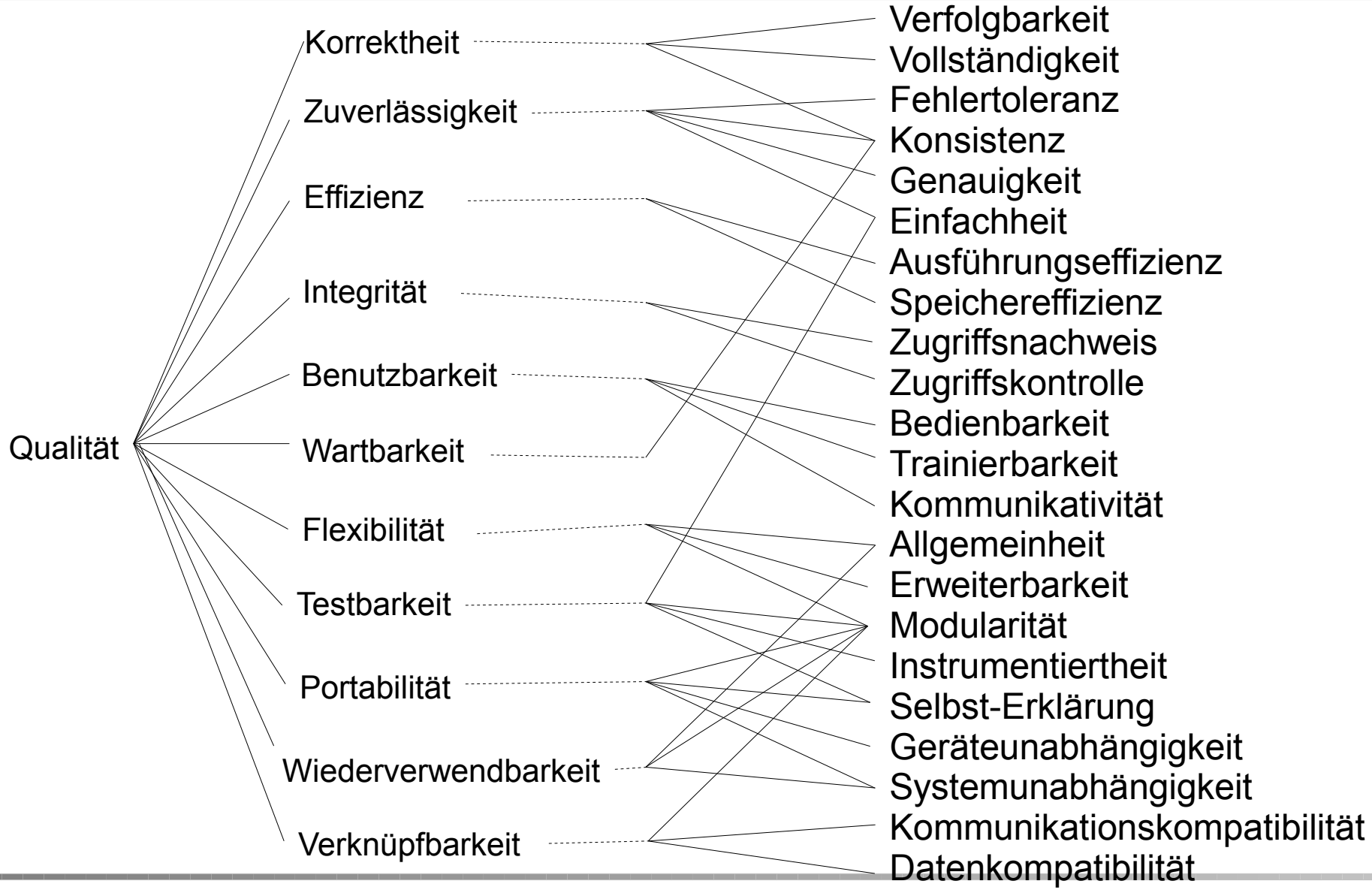
- Ein Qualitätsmodell ermöglicht die Messung von Qualität mit Hilfe eines standardisierten Modells.
- Ein Qualitätsmodell besteht typisch aus
  - einer Menge „allgemeingültiger“ Qualitätsziele (Faktoren)
  - einem Satz charakteristischer Merkmale zu jedem Faktor
  - messbaren Kenngrößen zu jedem Merkmal.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

Das Qualitätsmodell von McCall



Verfolgbarkeit
Vollständigkeit
Fehlertoleranz
Konsistenz
Genauigkeit
Einfachheit
Ausführungseffizienz
Speichereffizienz
Zugriffsnachweis
Zugriffskontrolle
Bedienbarkeit
Trainierbarkeit
Kommunikativität
Allgemeinheit
Erweiterbarkeit
Modularität
Instrumentiertheit
Selbst-Erklärung
Geräteunabhängigkeit
Systemunabhängigkeit
Kommunikationskompatibilität
Datenkompatibilität

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.

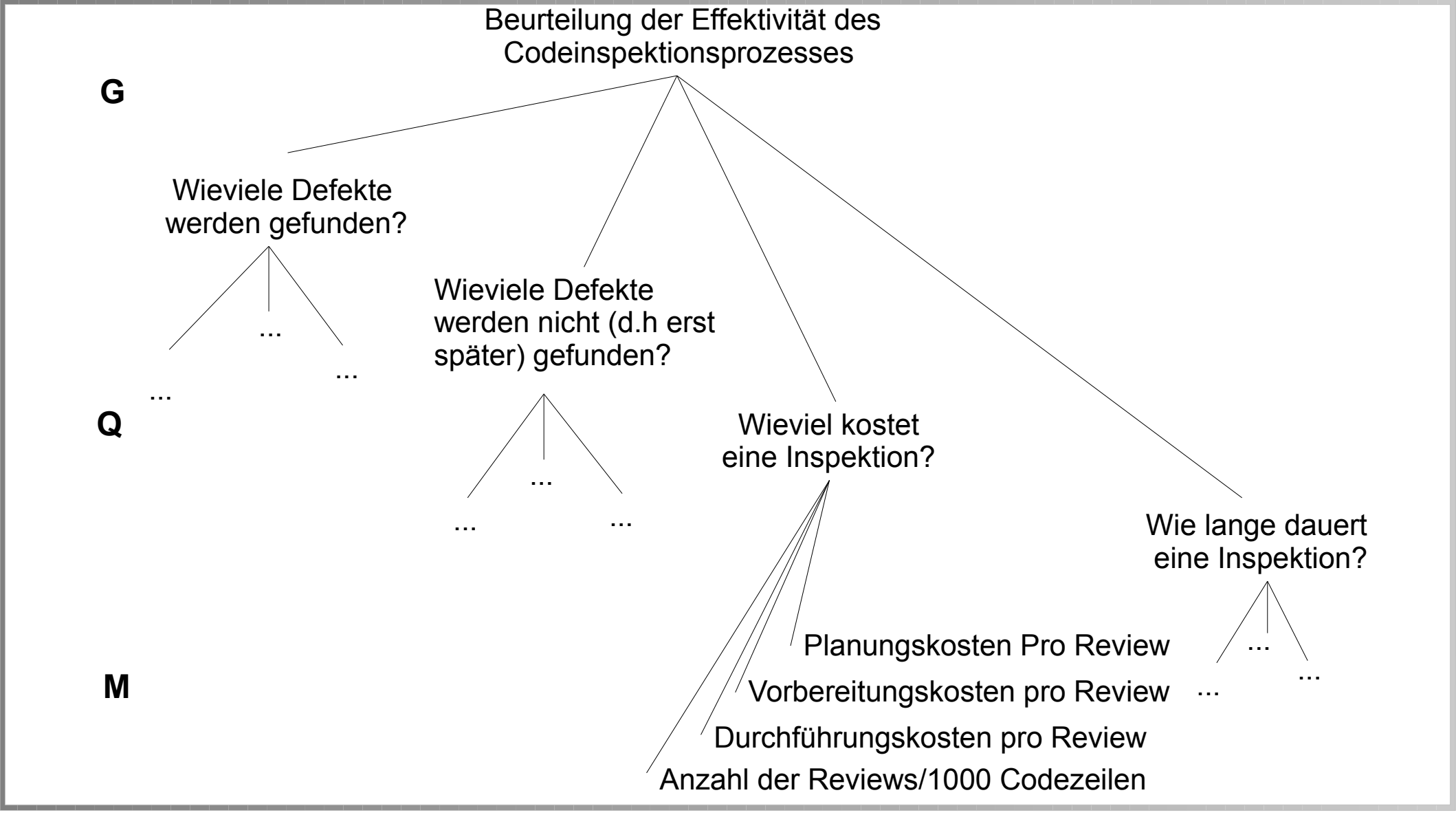
  ➠ Fokussierte Messung, zielbezogene Interpretation der Messwerte

Der GQM-Ansatz (Goal-Question-Metric)

Bekanntester zielorientierter Ansatz (Basili und Rombach 1988)

- 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: Goal-Question-Metric



Überraschungen bei Messungen

Beispiel: Die Reduzierung der Anzahl der Fehler in einem Programm führt zu einer höheren Aufrufrate der Hilfe.

- Das Programm wird nun als zuverlässiger betrachtet und hat dadurch einen breiteren, vielfältigeren Markt. Der prozentuale Anteil an Benutzern, die die Hilfe aufrufen, mag sinken, der reale Anteil aber wird steigen.
- Ein zuverlässiges System wird anders benutzt, als ein System, in dem die Benutzer um die Fehler herum arbeiten. Das führt zu einer höheren Aufrufrate der Hilfe.

Probleme mit Messung in der Industrie

- Es ist unmöglich, die Rendite zu berechnen, die eine Einführung eines organisatorischen Metrikprogramms mit sich bringt.
- Es gibt keine Standards für Softwaremetrik oder standardisierte Prozesse für Messung und Analyse.
- In viele Firmen sind die Softwareprozesse nicht standardisiert, wenig definiert und kontrolliert.
- Die meiste Forschung über Softwaremessung befasst sich mit Code-basierter Metrik und plangetriebenen Entwicklungs-prozessen. Allerdings wird mittlerweile mehr und mehr Software durch Konfiguration von ERP Systemen oder Components-off-the-Shelf (COTS) entwickelt.
- Messungen einzuführen, verlangt zusätzlichen Aufwand bei den Arbeitsprozessen.

Schwierigkeiten bei der Auswahl geeigneter Metriken

Problem: Es gibt für viele indirekt zu messende Aspekte von Software keine einzelne, optimal geeignete Metrik.

Beispiel-Resultat (Nagappan, Ball, Zeller: Mining metrics to predict component failures. ICSE 2006):

- 15 Metriken wurden auf ihre Eignung zur Vorhersage von Fehlerhäufigkeiten in 5 industriellen Software-Projekten untersucht.
- Für jedes der 5 Projekte gab es mindestens eine Metrik, die mit der Fehlerhäufigkeit korrelierte.
- Keine der untersuchten Metriken korrelierte in allen 5 Projekten mit der Fehlerhäufigkeit.

Diskussion

„Qualitätsmaße verbessern ein Produkt nicht. Das tun Qualitätssicherungsmaßnahmen. Und die tun das auch ohne gemessen zu werden.“ Wie stehen Sie zu dieser Aussage ?

Diskussion

„Qualitätsmaße verbessern ein Produkt nicht. Das tun Qualitätssicherungsmaßnahmen. Und die tun das auch ohne gemessen zu werden.“ Wie stehen Sie zu dieser Aussage ?

Antwort: Die Aussage ist zunächst korrekt. Allerdings sind in der Praxis nur begrenzte finanzielle, personelle und zeitliche Ressourcen for Qualitätssicherung vorhanden. Daher müssen Prioritäten gesetzt werden. Metriken können dabei behilflich sein, wenn man sich der diskutierten Einschränkungen bewusst ist.

Zusammenfassung Softwaremetriken

- Softwaremessungen können verwendet werden, um Daten über die Software und ihren Prozess zu sammeln.
- Es gibt verschiedene Arten von Metriken wie Produktmetriken, Projektmetriken, Prozessmetriken.
- Metriken der Produktqualität können hilfreich sein, um anomale Komponenten zu identifizieren, die womöglich Qualitätsprobleme haben.
- Metriken geben eine Möglichkeit für Vergleiche verschiedener Teile eines Softwaresystems oder verschiedener Entwicklungsprozesse.
- Metriken können Abschätzungen für Dauer, Kosten, etc. liefern.
- Metriken sind Instrumente die dem Projektmanagement Entscheidungshilfen liefern.