Software-Messung

- Motivation
- Software-Qualitätsexperimente
- Einsatz und Anwendung von Maßen
- Maßtypen
- Software-Qualitätsmessung
- Forderungen an Maße
- Bewertung und Kalibrierung von Maßen
- Maßskalen
- Empirische Daten

Motivation
Messen

- *„When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind.“*
  (Lord Kelvin, Popular Lectures and Addresses, 1889)

- *„Was man messen kann, das existiert auch!“*
  (Max Planck, 1858 - 1947)

Motivation
Messung in der Softwareentwicklung

- Ersetzung qualitativer – oft intuitiver – Aussagen über Software durch quantitative, reproduzierbare Aussagen
- Beispiel:
  - Qualitativ, intuitiv:
    - Der Entwickler sagt: „Ich habe mein Softwaremodul ausgetestet“
  - Quantitativ, reproduzierbar:
    - „Mein Testwerkzeug zeigt zur Zeit eine Zweigüberdeckungsrate von **57% (70 von 123 Zweigen)** an. In unserer Firma gelten Module als ausreichend getestet, wenn die Zweigüberdeckungsrate mindestens **95%** beträgt. Ich muss daher noch mindestens **47** Zweige testen und erwarte aufgrund der Erfahrung mit vergleichbaren Modulen, dass ich dafür etwa **1,5 Mitarbeitertage** zusätzlichen Aufwand benötige.“

Motivation
Qualitätsmessung in der Softwareentwicklung

- Software wird heute vielfach in Anwendungsbereichen eingesetzt in denen quantitative Aussagen üblich oder notwendig sind:
  - Vertragsgestaltung: „Wir vereinbaren eine Mindestverfügbarkeit des Systems von 99,8%!“
  - Sicherheitsnachweis eines Bahnsystems beim Eisenbahnbundesamt: „Wie hoch ist das Restrisiko durch Softwarefehler?“
  - Ist die zu erwartende Anzahl an Restfehlern hinreichend klein für die Freigabe?
  - Ist die Wahrscheinlichkeit hinreichend gering, dass Softwarefehler in Steuergeräten einen Ausfall unserer Oberklassenlimousine verursachen?
  - Wir benötigen eine ausfallfreie Missionszeit von 4 Wochen. Wird das gelingen?

Motivation
Qualitätsmessung in der Softwareentwicklung: Probleme

- Die meisten Qualitätseigenschaften sind nicht direkt messbar!
  - Fehlergehalt
  - Verfügbarkeit
  - Zuverlässigkeit
  - Sicherheit
  - ...
- Man versucht die Qualitätseigenschaften ...
  - ... experimentell zu ermitteln (z.B. Zuverlässigkeit) oder
  - ... aus direkt messbaren Eigenschaften zu schließen (z.B. Fehlergehalt).

Software-Qualitätsexperimente
Stochastische Analyse von Software-Zuverlässigkeit: Situation

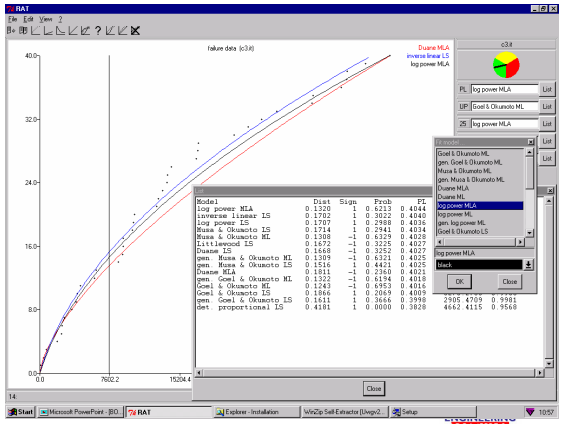
- Eigenständige Forschungsrichtung seit ca. 30 Jahren
- Wenig Einfluß auf die Softwareentwicklung in der Praxis
- Mathematische Basis zum Teil kompliziert
- Sehr viele unterschiedliche stochastische Zuverlässigkeitsmodelle
- A priori Auswahl eines Modells nicht möglich
- Bestimmung von Modellparametern notwendig
- Anwendung der Theorie in der Praxis erfordert leistungsfähige Werkzeugunterstützung

Software-Qualitätsexperimente
Stochastische Analyse von Software-Zuverlässigkeit: Theorie

$$m(t) = E(N(t))$$

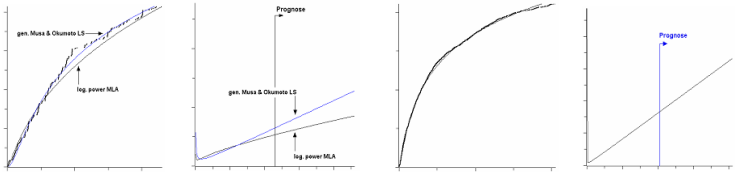
Musa bzw. Goel-Okumoto model	$m(t) = a(1-e^{-bt})$
Generalized Goel-Okumoto model	$m(t) = a(1-e^{-bt^c})$
Musa-Okumoto model	$m(t) = a\ln(bt+1)$
Generalized Musa-Okumoto model	$m(t) = a\ln(bt^c+1)$
Duane bzw. Crow model	$m(t) = at^b$
Log model	$m(t) = a\ln(bt)$
Log power model	$m(t) = a\ln^b(t+1)$
Generalized log power model	$m(t) = a\ln^b(ct+1)$
Yamada S-shape model	$m(t) = a(1-(1+bt)e^{-bt})$
Generalized Yamada S-shape model	$m(t) = a(1-(1+ct)e^{-bt})$
Geometric Moranda bzw. deterministic proportional model	$m(t) = a+b\ln(t+1)$
Littlewood model	$m(t) = c(t+a)^{-b}$
Inverse linear model	$m(t) = a(\sqrt{bt+1}-1)$

Software-Qualitätsexperimente
Stochastische Analyse von Software-Zuverlässigkeit: Praktischer Lösungsansatz

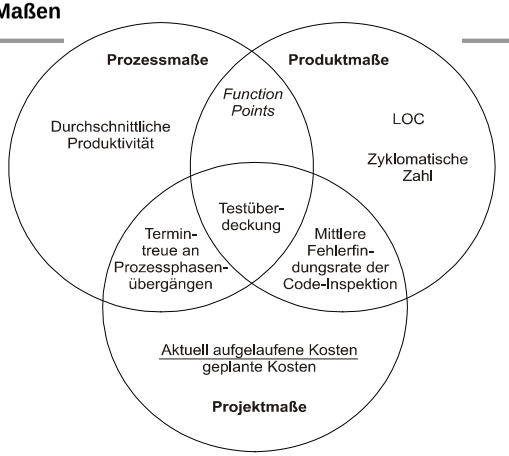


Software-Qualitätsexperimente
Stochastische Analyse von Software-Zuverlässigkeit: Praktischer Lösungsansatz

Zahlreiche Anwendungen (Verkehrstechnik, Medizintechnik, Telekommunikation, ...)



Software-Maße
Anwendung von Maßen

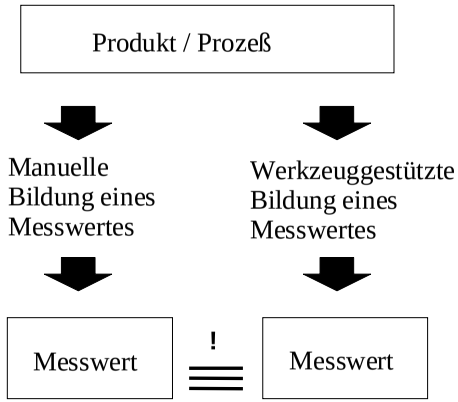


Software-Maße
Forderungen an Maße

- Einfachheit
  - Ist das Ergebnis so einfach, dass es leicht interpretiert werden kann?
- Eignung
  - Erfasst das Maß die gewünschte Eigenschaft?
- Robustheit
  - Ist der Wert des Maßes stabil gegenüber Manipulationen von unter-geordneter Bedeutung?
- Rechtzeitigkeit
  - Kann das Maß bereits zu einem Zeitpunkt gebildet werden, der noch ein rechtzeitiges Einwirken auf den Prozess gestattet?
- Analysierbarkeit
  - Ist das Maß statistisch analysierbar (z. B. numerischer Wertebereich)?
    (Für diese Eigenschaft ist maßgeblich der Skalentyp des Maßes entscheidend.)

Software-Maße
Forderungen an Maße
Reproduzierbarkeit

- Ist ein Maß präzise definiert, so ist es in der Regel auch reproduzierbar (unabhängig von der Art der Bildung):



Software-Maße
Forderungen an Maße
Reproduzierbarkeit

- Beispiele:
  - McCabe's zyklomatische Zahl: e-n+2

    e = Anzahl der Kanten eines KFG; n = Anzahl der Knoten eines KFG; KFG = Kontrollflussgraph
    - Vollständig reproduzierbar
  - Lines of Code (LOC)

    Leerzeilen mitzählen? Kommentarzeilen mitzählen?
    - Bei entsprechender Festlegung vollständig reproduzierbar
  - Function Points: Manuelle Bewertung von Komplexitäten erforderlich
    - Prinzipiell nicht vollständig reproduzierbar
  - Verständlichkeit
    - Schlecht reproduzierbar

Software-Maße
Bewertung von Maßen

- Eine Empfehlung von unteren bzw. oberen Grenzen von Maßen ist schwierig.
- Welche Werte als „normal“ anzusehen sind, muss durch Erfahrungswissen ermittelt werden.
- Eine Abweichung von üblichen Werten kann ein Hinweis auf ein Problem sein; muss aber nicht.

Software-Maße
Kalibrierung von Maßen und Modellen

- Die Zuordnung zwischen Maßen und den relevanten Eigenschaften erfordert eine Kalibrierung, die ggf. veränderten Situationen angepasst werden muss.
- Es können empirische und theoretische Modelle unterschieden werden.
- Beispiel:
  - Theoretisches Modell für Aufwand (vgl. Halstead-Maße):
    $E = ... Umfang^2 ...$

    Der quadratische Zusammenhang zwischen Aufwand und Umfang ist aufgrund theoretischer Überlegungen identifiziert worden.
  - Empirisches Modell für Aufwand: E = ... Umfang1,347 ...

    Der Exponent 1,347 ist aufgrund der statistischen Auswertung von Daten ermittelt worden.

Software-Maße
Maßskalen

- Wenn man abstrakte Eigenschaften als Zahlenwert ausdrückt, so muss man sich überlegen, welche Operationen mit diesem Zahlenwert bei seiner Weiterverarbeitung sinnvoll sind.
- Beispiel:
  - Messung der Länge:
    - Brett a ist einen Meter lang. Brett b ist zwei Meter lang. Darum ist Brett b doppelt so lang wie Brett a.
    - Diese Behauptung ist sinnvoll.
  - Messung der Temperatur:
    - Heute ist es 20°C warm. Gestern war es 10°C warm. Also ist es heute doppelt so warm wie gestern.
    - Das ist falsch, die richtige Antwort ist: Heute ist es rund 3,5 % wärmer als gestern.
  - Offensichtlich gibt es einen Unterschied zwischen der Skala der Temperatur in °C und der Länge in Metern, der dazu führt, dass bestimmte Operationen auf die Temperaturskala nicht anwendbar sind.

Software-Maße
Maßskalen

- Nominalskala
  - Freie Bezeichnung bestimmter Eigenschaften mit Markierungen
  - Inventarnummern von Büchern einer Bibliothek (DV 302, PH 002, CH 056, ...)
  - Namen unterschiedlicher Requirements Engineering Methoden (SA; SADT, OOA; IM, ...)
- Ordinalskala
  - Abbildung eines geordneter Aspekts einer Eigenschaft auf eine geordnete Menge von Messwerten und zwar so, dass die Ordnung erhalten bleibt
  - Abbildung des Eintreffens von Patienten auf die Behandlungsnummern in einer Arztpraxis
- Intervallskala
  - Eine Skala, die auch dann noch gültig ist, falls Transformationen g(x) = ax + b, mit a > 0 auf sie angewendet werden.
  - Temperaturskalen in Grad Celsius oder Fahrenheit. Falls F eine Temperatur in der Fahrenheit-Skala ist, so kann die Temperatur C in der Celsius-Skala folgendermaßen errechnet werden: $C = {}^{5}/_{9}$ (F - 32). Die Relationen zwischen Temperaturen bleiben erhalten.

Software-Maße
Maßskalen

- Rationalskala
  - Eine Skala, in der Messwerte zueinander in Relation gesetzt werden können (Prozentuale Aussagen über Messwerte sind sinnvoll.).
  - Länge in Metern (Es ist doppelt so weit von a nach b wie von c nach d)
  - Temperatur in Kelvin
- Absolutskala
  - Eine Skala, die die einzige Möglichkeit zur Messung des Sachverhaltes darstellt.
  - Zählen

Software-Maße
Die zyklomatische Komplexität

- Verbreitet als Maß für Komplexität.
- Oft mit der Aura einer „wichtigen“ Schlüsselgröße umgeben.
- Stammt aus der Graphentheorie (Stark zusammenhängende Graphen) und kann demnach auf Kontrollflussgraphen und damit auf Programme bezogen werden.
- Berechnungsvorschrift: e – n + 2
  (e = Anzahl der Kanten, n = Anzahl der Knoten)
- Sehr einfach zu bilden, da für Programme stark von der Anzahl der Entscheidungen abhängig.
- Als Komplexitätsmaß geeignet, falls die Anzahl der Entscheidungen viel über die Komplexität des Programms aussagt.
- Wahrscheinlich das am weitesten verbreitete Maß in Analyse- und Testwerkzeugen.

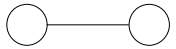
Software-Maße
Die zyklomatische Komplexität

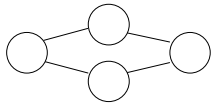
- Die zyklomatische Zahl ist eine Größe zur Messung der strukturellen Komplexität von Programmen.
- Die Basis für die Berechnung der zyklomatischen Zahl bildet der Kontrollflussgraph.
- Die zyklomatische Zahl v(G) eines Graphen G ist: v(G) = e - n + 2

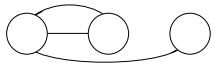
  (e - Anzahl der Kanten, n - Anzahl der Knoten)

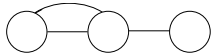
Software-Maße
Die zyklomatische Komplexität

- Zyklomatische Komplexität von Graphen

Sequenz  v(G) = 1-2+2 = 1

Auswahl  v(G) = 4-4+2 = 2

Abweisende Schleife  v(G) = 3-3+2 = 2

Nicht abweisende Schleife  v(G) = 3-3+2 = 2

Software-Maße
Aktuelle Bedeutung der Software-Messtechnik

- Eine funktionierende Software-Messtechnik ist wichtig für folgende Bereiche:
  - Flachere Management-Strukturen
  - Standards im Bereich der Software-Entwicklung
  - Die Erreichung eines hohen Capability Maturity Level (Assessments).

Software-Maße
Aktuelle Bedeutung der Software-Messtechnik: Software-Messtechnik und flache Management-Strukturen

- Der Trend im Bereich des Software-Managements geht zu flacheren Strukturen:
  - Ein Manager betreut erheblich mehr Entwickler als bisher.
  - Die Bereitstellung und Verdichtung von Informationen geschieht nicht mehr durch das mittlere Management, sondern über automatisierte Messsysteme.
  - Eingriffe des Managements sind nur dann erforderlich, wenn Messwerte auf Problemsituationen hinweisen.

=> Ein funktionierendes Messsystem ist eine wichtige Voraussetzung.

Software-Maße
Aktuelle Bedeutung der Software-Messtechnik: Software-Messtechnik und Softwareentwicklungs-Standards

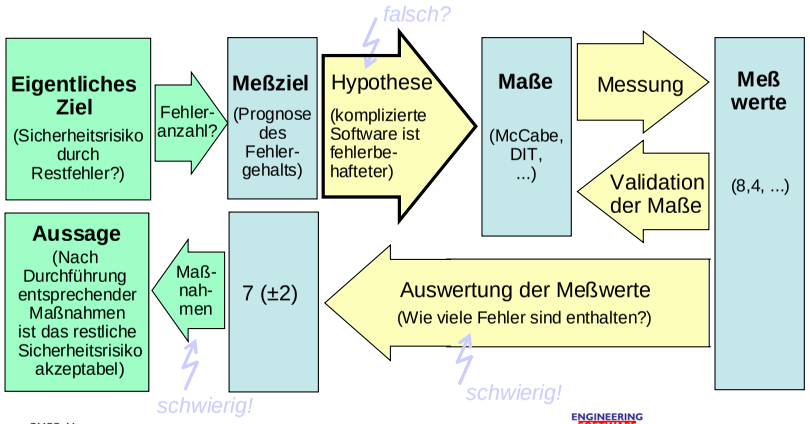
- Standards erlangen im Bereich der Software-Entwicklung zunehmend Bedeutung (z. B. ISO 9001):
  - Qualifikationsnachweis gegenüber potentiellen Kunden.
  - Marketingargument; Abgrenzung gegenüber nicht zertifizierten Mitbewerbern.
  - Wichtig im Rahmen von Produkthaftung.
  - In einigen Bereichen Voraussetzung zur Auftragserteilung.
  - Alle Standards legen besonderen Wert auf systematische Vorgehensweise, Transparenz und Kontrolle des Entwicklungsprozesses.

=> Dies kann mit Hilfe entsprechender Maße nachgewiesen werden.

Software-Maße
Aktuelle Bedeutung der Software-Messtechnik: Software-Messtechnik und das Capability Maturity Model

- Das Capability Maturity Model ordnet den Reifegrad des Software-Entwicklungsprozesses in eine von fünf Stufen ein. Die möglichen Reifegrade sind 1-initial, 2-repeatable, 3-defined, 4-managed, 5-optimized.
- Die Erreichung der Reifegrade 4 und 5 ist nur bei Existenz und Nutzung eines Messsystems möglich, das folgende Tätigkeiten ermöglicht:
  - Messung von Produktivität und Qualität.
  - Bewertung von Projekten auf der Basis dieser Messungen.
  - Erkennung von Abweichungen.
  - Treffen von Korrekturmaßnahmen im Falle von Abweichungen.
  - Identifikation und Beherrschung von Projektrisiken.
  - Prognose von Projektverlauf und Produktivität.

Software-Qualitätsmessung
Schlussfolgerungskette



Software-Qualitätsmessung
Beliebte Hypothesen in Theorie und Praxis

	/Fenton, Ohlsson 00/	/Basili, et al. 96/	/Cartwright, Shepperd 00/	/Basili, Perricone 84/	/Abreu, Melo 96/
Wenige Module enthalten die Mehrzahl der Fehler.	++	++	(+)	++	/
Wenige Module erzeugen die meisten Ausfälle.	++	/	/	/	/
Viele Fehler im Modultest bedeuten viele Fehler im Systemtest.	+	/	/	/	/
Viele Fehler im Test bedeuten viele Ausfälle im Feld.	--	/	/	/	/
Fehlerdichten korrespondierender Phasen sind über Releases hinweg konstant.	+	/	/	/	/
Umfangsmaße sind geeignet zur Fehlerprognose.	+	/	+	-	/

++: starke Bestätigung; +: schwache Bestätigung; 0: keine Aussage;
-: schwache Ablehnung; --: starke Ablehnung; /: nicht evaluiert; ?: unklar

Software-Qualitätsmessung
Beliebte Hypothesen in Theorie und Praxis: Erkenntnisse I

- Fehler sind über die Module einer Software nicht gleichmäßig verteilt, sondern konzentrieren sich in einigen wenigen Modulen
- Diese Module erzeugen den größten Teil der Probleme
- Großer Modulumfang bedeutet nicht zugleich hoher Fehlergehalt
- Viele erkannte Probleme im Test bedeuten nicht, dass eine Software in der Praxis Qualitätsmängel zeigt
- Es scheint Regeln zu geben, die dafür sorgen, dass aufeinander folgende Entwicklungen ähnliche Ergebnisse liefern

Frage:

- **Wie können die wenigen besonders fehlerhaften Module erkannt werden?**

Software-Qualitätsmessung
Beliebte Hypothesen in Theorie und Praxis

	/Fenton, Ohlsson 00/	/Basili, et al. 96/	/Cartwright, Shepperd 00/	/Basili, Perricone 84/	/Abreu, Melo 96/
Code-Komplexitätsmaße sind geeignete Mittel zur Fehler-prognose.	besser als Umfangs-maße: -	WMC: +	WMC: /	besser als Umfangs-maße: -	MHF: +
		DIT: ++	DIT: ++		AHF: 0
		RFC: ++	RFC: /		MIF: +
		NOC: ?	NOC: ?		AIF: (+)
		CBO: ++	CBO: /		POF: +
		LCOM: 0	LCOM: /		COF: ++

- Objektorientierte Maße:
  - WMC (*Weighted Methods per Class*)
  - DIT (*Depth of Inheritance Tree*)
  - NOC (*Number Of Children*)
  - CBO (*Coupling Between Object-classes*)
  - RFC (*Response For a Class*)
  - LCOM (*Lack of Cohesion on Methods*)
  - MHF: *Method Hiding Factor*
  - AHF: *Attribute Hiding Factor*
  - MIF: *Method Inheritance Factor*
  - AIF: *Attribute Inheritance Factor*
  - POF: *Polymorphism Factor*
  - COF: *Coupling Factor*

Software-Qualitätsmessung
Beliebte Hypothesen in Theorie und Praxis: Erkenntnisse II

- Einzelne einfache Komplexitätsmaße (z.B. McCabe zyklomatische Zahl) funktionieren nicht besser als Umfangsmaße (z.B. LOC)
- Spezifischere Komplexitätsmaße zeigen oft eine gute Prognosequalität für den Fehlergehalt

Schlussfolgerung:

- **Eine geeignete Kombination von geeigneten Komplexitätmaßen gestattet eine gezielte Identifikation der fehlerhaften Module**

Software-Qualitätsmessung
Beliebte Hypothesen in Theorie und Praxis

	/Fenton, Ohlsson 00/	/Basili, et al. 96/	/Cartwright, Shepperd 00/	/Basili, Perricone 84/	/Abreu, Melo 96/
Modellbasierte *(Shlaer-Mellor)* Maße sind geeignet zur Fehlerprognose.	/	/	Events: ++	/	/
Modellbasierte Maße sind geeignet zur Prognose des Codeumfangs.	/	/	States: ++	/	/

Software-Qualitätsmessung
Beliebte Hypothesen in Theorie und Praxis: Erkenntnisse III

- Aus Softwareentwürfen können Messwerte gewonnen werden, die es gestatten, Codeumfänge und Fehlergehalte frühzeitig zu prognostizieren

Software-Qualitätsmessung
Schlußfolgerungen

- Statistische Verfahren zur Feststellung der Softwarezuverlässigkeit sind theoretisch fundiert und praktisch anwendbar.
- Einige plausibel erscheinende Hypothesen im Bereich der Qualitätsmessung sind empirisch falsifiziert worden, aber es existiert Evidenz, dass ...
  - ... sich Fehler in wenigen Modulen konzentrieren und ...
  - ... diese Module identifiziert werden können durch Messung ...
    - ... der Code-Komplexität und/oder
    - ... von Komplexitäten des zugrundeliegenden Entwurfsmodells.
- Die Prognose von Fehlergehalten ist auf Basis einzelner Maße (sog. univariate Analyse) nicht möglich; aber eine Nutzung einer geeigneten Kombination von Maßen (sog. multivariate Analyse) kann verlässliche Aussagen liefern.
- Es ist zu erwarten, dass die Bestimmung von Prognosemodellen anhand abgeschlossener Projekte möglich ist, da die Annahme von Ähnlichkeiten zwischen aufeinander folgenden Projekten empirisch belegt ist.

Software-Messung
Literatur

- Halstead M.H., Elements of Software Science, New York: North-Holland 1977
- Zuse H., Software Complexity - Measures and Methods, Berlin, New York: De Gruyter 1991