Messen und Schätzen

Hartmut Krasemann, T-Systems

- Motivation “Was messen, wo schätzen im Projekt”
- Messungen: Aufwand, Produktgröße (LOC, FP, WP), Produktivität, Monitore
- Schätzungen: Produkt, Aufwand, SW-Gleichung, Makroschätzung
- Fehler: Mengen, Testaufwand, Behebungsaufwand, Erfahrungen
- Empfehlungen: Ein Messprogramm, Messen in agilen Projekten
- Literaturhinweise: Wo weiterlesen

Motivation:Inhalt

- Mikro- und Makroschätzung
- Wann Messen und Schätzen?
- Prozesse: Planung und Rationalisierung
- Woher Input nehmen? (globale Zahlen, Faustformeln)
- Planung: Aufwand (Termin, Kopfzahl)
- Time to Market
  Mess- und Schätzkreislauf in agilen Projekten
- Rationalisierung: Messen
- Produktivität: Programmiersprachen
- Software-Maße: LOC, FP und WP

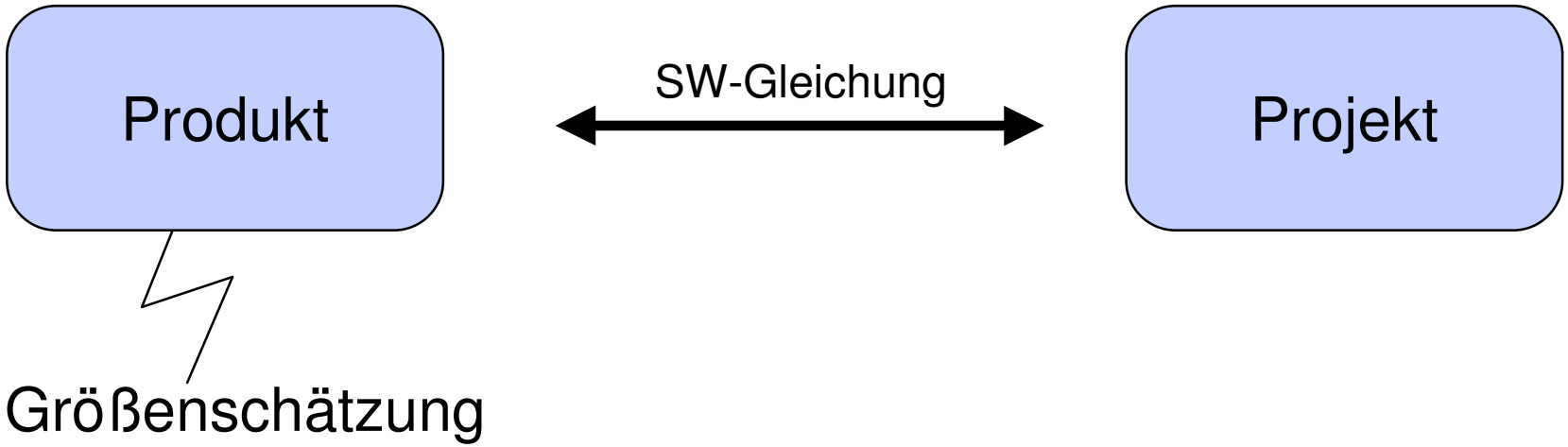
Mikroschätzung

Projektschätzung = Mikroschätzung

----------	**66**
– ----------	12
– ----------	23
• ----------	14
• ----------	9
– ----------	31
• ---------	15
• ----------	16
----------	**185**
– ----------	43
– ----------	54
• ----------	24
• ----------	13
• ----------	17
– ----------	75
• ---------	18
• ----------	17
– ...	...

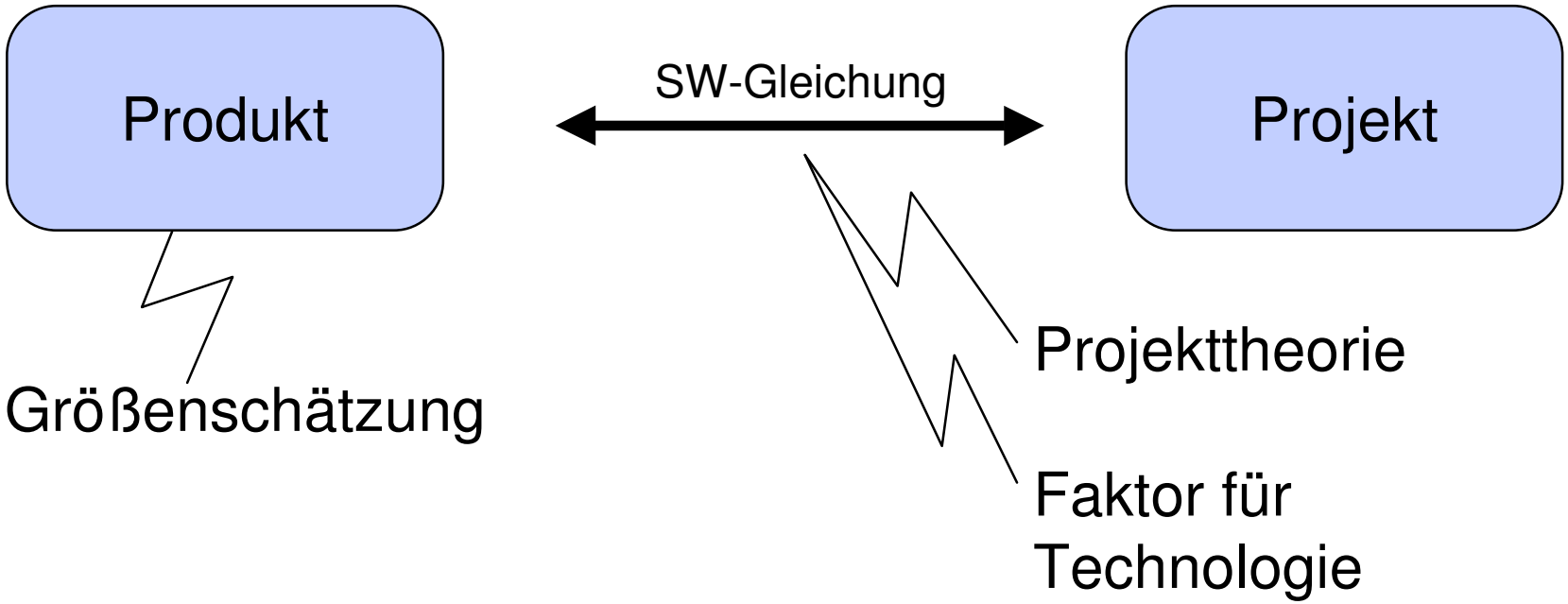
Makroschätzung

Makroschätzung: vom Produkt zum Projekt



Makroschätzung

Makroschätzung: vom Produkt zum Projekt



Messen und Schätzen
Prozesse: Planung und Rationalisierung

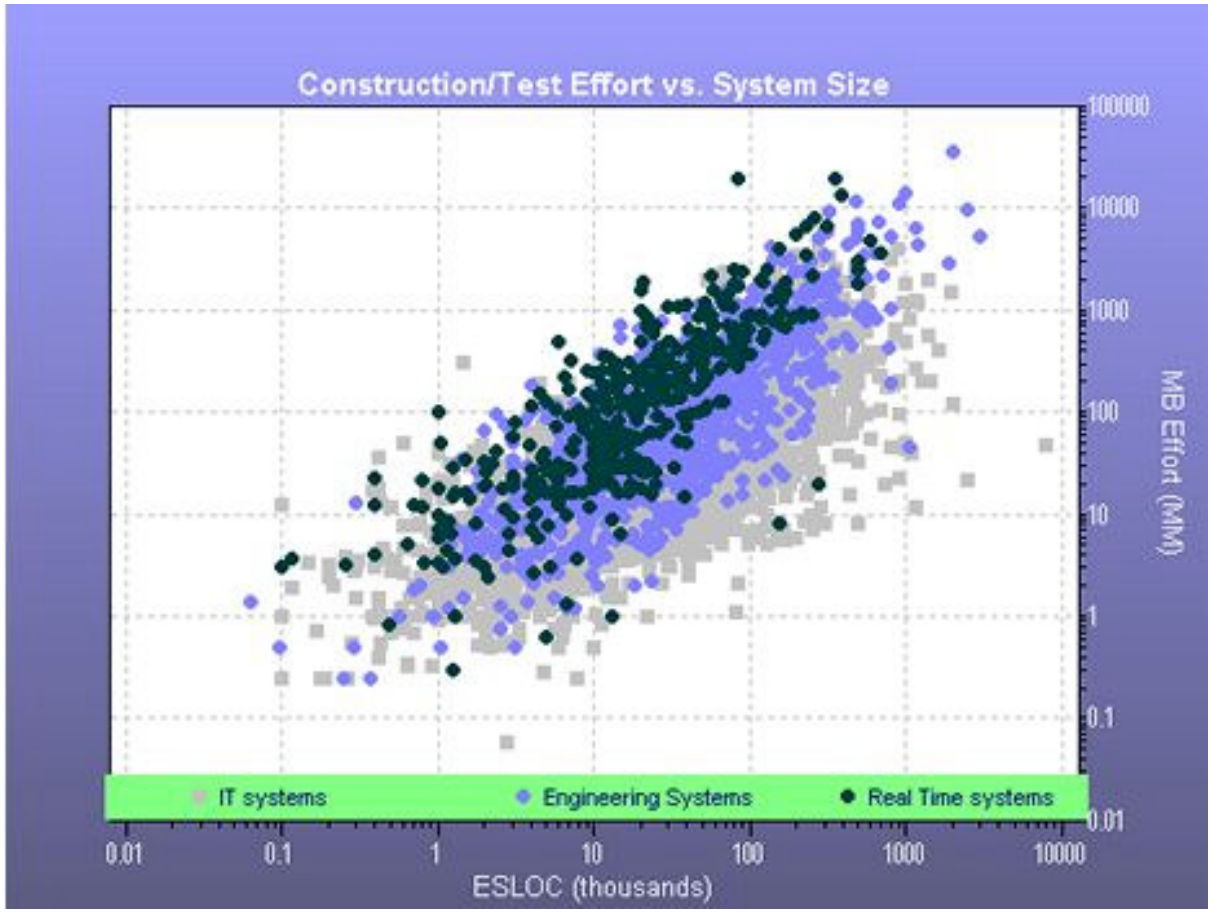
- Angebotskalkulation, Planung (Schätzen)
  - Aufwand, Zeit, Geld (frühere Messungen, “Erfahrung”)
- Projektcontrolling (Messen und Schätzen)
  - Messen: Aufwand, Qualität (Änderungshäufigkeit)
  - Schätzen: Restaufwand, Restfehler, Zeit, Geld
- Produktcontrolling (Messen)
  - Produktgröße, Fehlermenge, andere Qualitätsparameter
- Projektnachlese (Messen)
  - Produktgröße, Aufwand, Laufzeit, Qualität (Fehlermengen)
  - Produktivität berechnen, Messergebnisse aufheben

Rationalisierung

Messen und Schätzen
Woher Input nehmen?

- Globale Durchschnittswerte
  - Produktivität
  - Fehlermengen
- Faustformeln
  - Softwaregleichung von Putnam
  - empirische Annahme über Problemlösungskapazität
    => Faustformeln nach C. Jones
- persönliche Projekterfahrung
- unternehmensweite Datensammlung (Nachbereitungen)

Produktivität
globale Durchschnitte



Aufwand / LOC

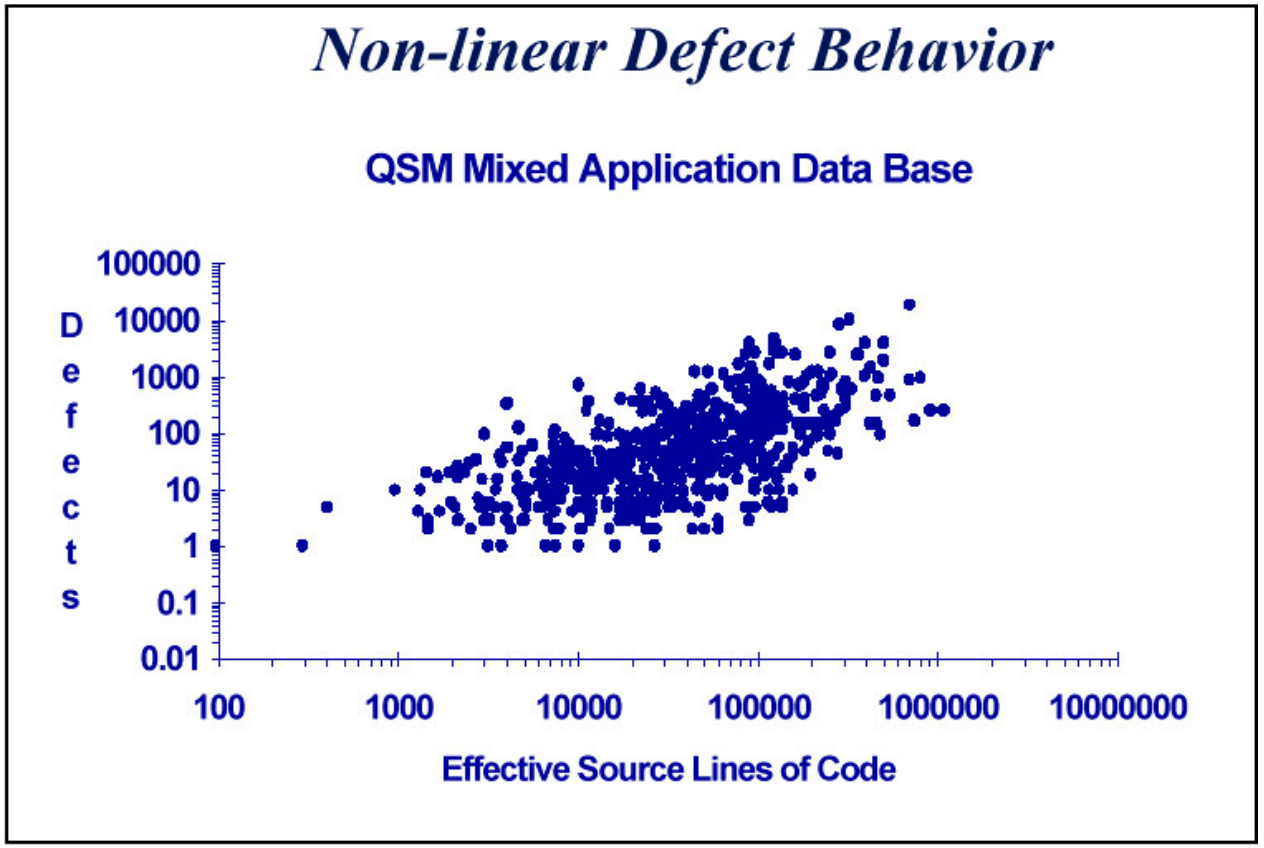
Von
http://www.qsm.com

Bei 100 kLOC:
(1 – 20.000 MM)
60 – 600 MM

---

1 bis 10 LOC / h

Fehlermengen
globale Durchschnitte



Fehlermenge / LOC

Von
http://www.qsm.com

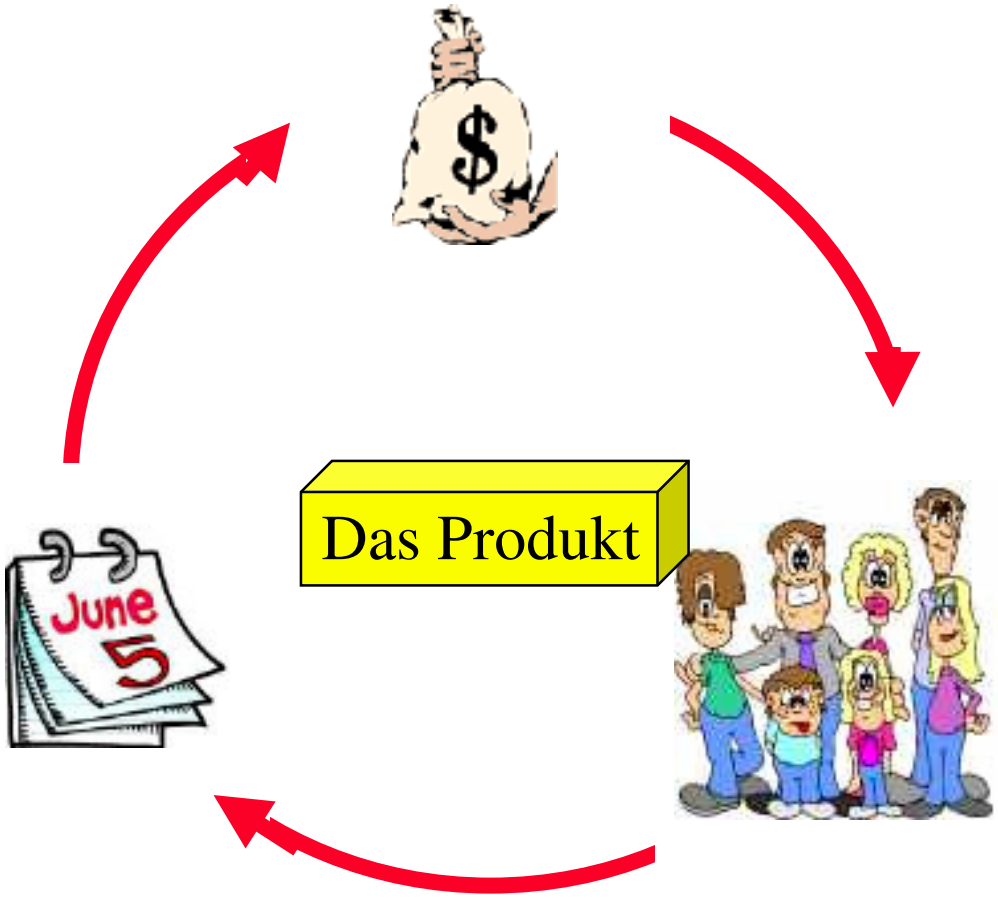
500 kLOC:
6 Fehler / kLOC

10 kLOC:
3 Fehler / kLOC

C. Jones für Java:
12,5 Fehler/kLOC

Unsicher !!!

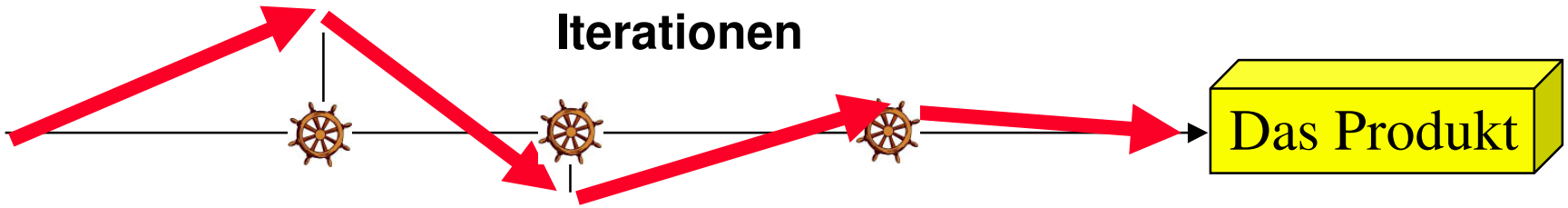
Planung:
Aufwand (Termin, Kopfzahl)

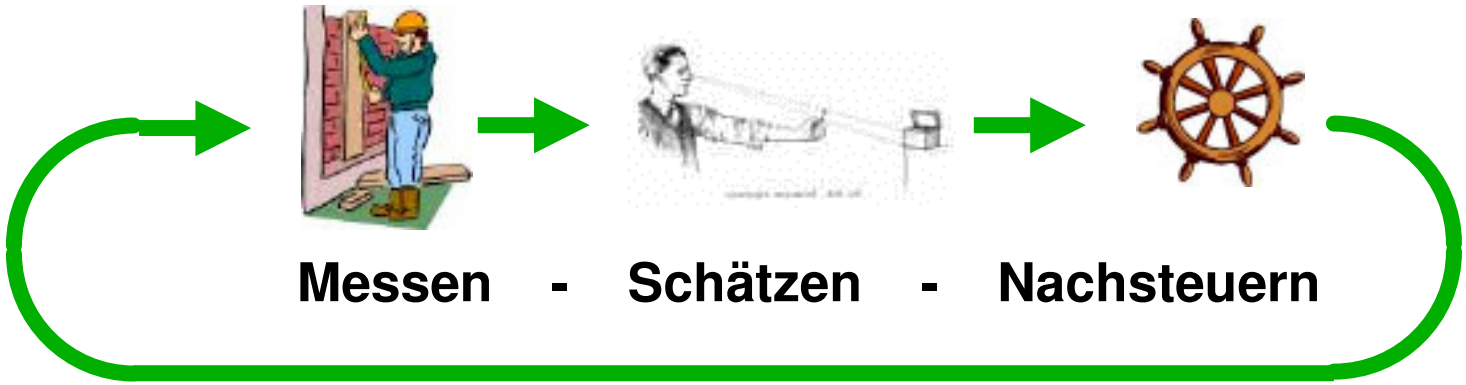


8 / 4 ≠ 2

2 * 3 ≠ 6

Time to Market
Mess- und Schätzkreislauf in agilen Projekten





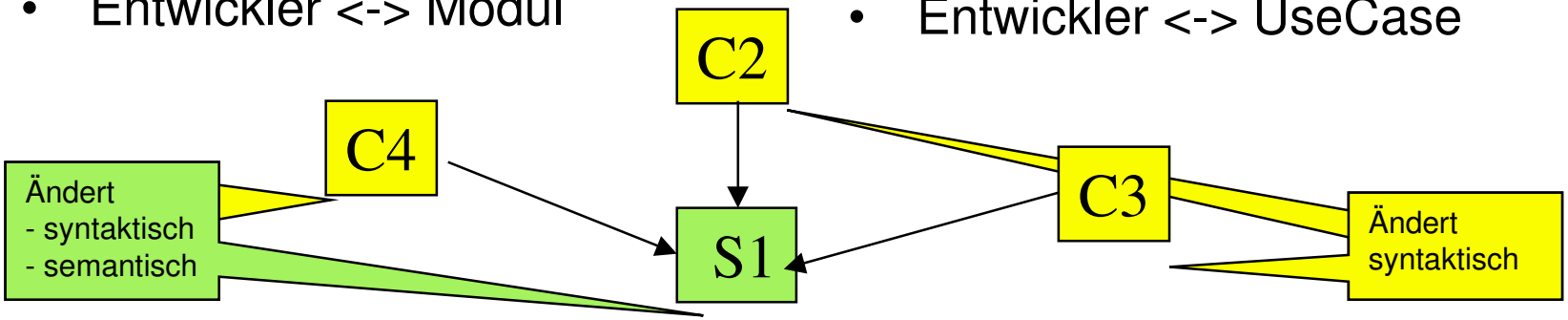
Rationalisierung Messen

- Programmiersprachen und -umgebungen
  - Verbesserungen der Hilfsmittel
    Navigieren, Generieren, Prozeßverbesserungen
  - Mächtigkeit der Umgebung (von 5 auf 10 LOC/h seit 1980 ?)
  - Mächtigkeit der Sprache (21 statt 53 LOC ?)
- Auswahl der “besten” Sprache
  - kann ein Projekt billiger machen
  - wird stark unterbewertet = selten gemacht
- Verbesserungen in einem Projekt
  - bei Projektdauer > 2 Jahre

Produktivität Prozeßverbesserungen

Klassisch

- Entwickler <-> Modul



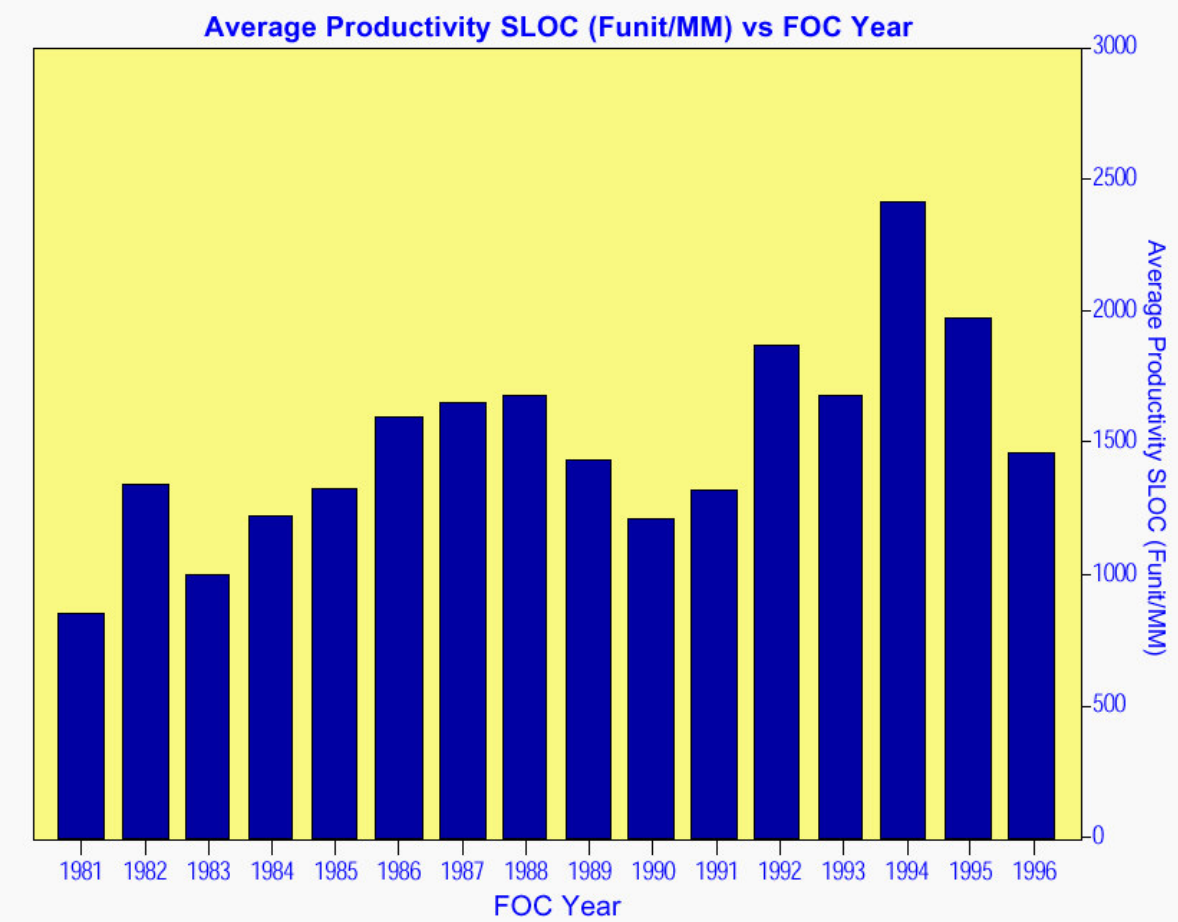
- E1 ändert,
- E1 *bittet* E2 bis E4 zu ändern
- Integrator findet Fehler:
- E1 und E4 korrigieren
- *E2 hat vergessen*
- *E3 ist in Urlaub*

Smalltalk, CVS, Eclipse

- Entwickler <-> UseCase

- E4 ändert S1, C2-C4
- E1 ändert ebenfalls
- Integrator findet Konflikte (werkzeugunterstützt) und korrigiert mit Hilfe von E1 und E4

Produktivität über alle Projekte, Programmiersprachen



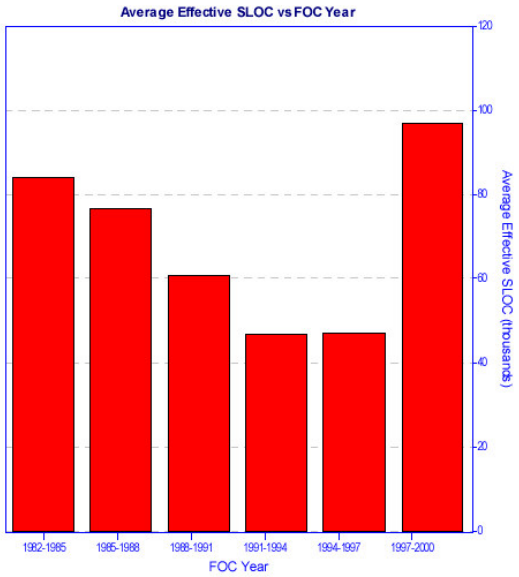
entnommen aus:
“Familiar Metric Management - QSM Database shows Drop in Productivity”
von L.H.Putnam und W. Myers
http://www.qsm.com/resources

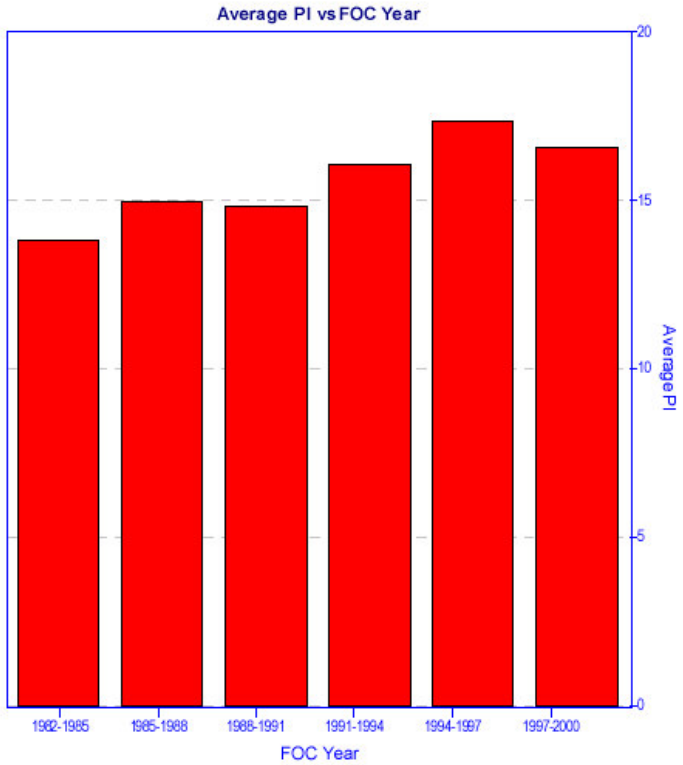
1980: 5 SLOC/h
1995: 12 SLOC/h

Effekte sind:

- Projektgröße
- Programmierumgebung
- Programmiersprache

Produktivität
Um Projektgröße korrigiert (Putnam-QSM)





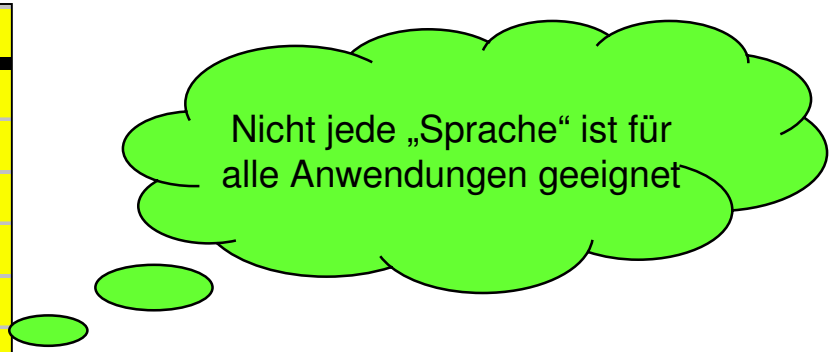
Die Projektgröße sank -
die LOC/h stiegen

From the QSM Database - Productivity Statistics Buck - 15 Year Trend, Doug Putnam

Produktivität von Programmiersprachen

- Programmiersprachen sind sehr verschieden “produktiv”
- Produktivität = Funktionsumfang pro Arbeitsstunde
- Generationen: Maschinensprache, Assembler, Hochsprachen
- Language Table (Capers Jones)

Language	Productivity Level
Assembler	1
C	2,5
Cobol, Fortran	3
C++, Java	6
M, Smalltalk	15
Excel	55



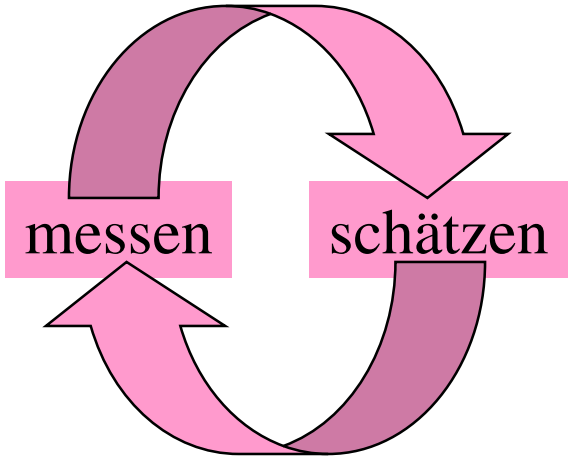
Software-Maße
LOC, FP und WP

- Source Lines Of Code (LOC, SLOC, kLOC)
  - "intrinsisch", sagt mehr über die PL als über die Funktionalität
- Funktionspunkte (FP) [IFPUG]
  - "extrinsisch", aus äußeren Eigenschaften berechnete abstrakte FP
- Widgetpunkte (neu !!!, WP)
  - "extrinsisch", zählt Widgets des GUI
- Zusammenhang: Language Table von C. Jones

Language	Productivity Level	SLOC/FP	SLOC/WP
Assembler	1	320	730
C	2,5	128	290
Cobol, Fortran	3	107	245
C++, Java	6	53	120
M, Smalltalk	15	21	48

Messungen
Inhalt

- Aufwand messen
- Fehlermengen und -raten messen
- Produktgröße messen
- Produktivität messen
- Funktionspunkte
- Einwände gegen Funktionspunkte
- Widgetpunkte
- Automatisierung von Messungen
- Monitore



Aufwand messen
Stunden

- ist das einfachste: Stunden aufschreiben
- manchmal auch das einzige!
- Problem: Zielkonflikt bei der Zuordnung der Stunden:
  - optimiert für die Kunden-Rechnung (extern/intern) oder
  - optimiert für den Lernprozess des Schätzens

Fehlermengen und -raten messen,Aufwand für Test,Behebung schätzen

- Wird oft gemacht
- Erlaubt Prognosen, wenn Größe und Aufwand gemessen wird
  - eigene oder Literatur – Mittelwerte
  - Prognose Testaufwand
  - Prognose Fehlerbehebungsaufwand
- Testaufwand = A * X(pro LOC) + B * Y(pro Fehler)
- Behebungsaufwand = C * Z (pro Fehler)

Produktgröße messen

- LOC messen ist
  - sehr einfach und automatisierbar (z.B nach jedem Build, Link, ..)
- akademischer Streit um die “richtigen” LOC
  - ist irrelevant, wichtig ist Konsistenz und einfache Messbarkeit
  - z.B Brutto-LOC
- LOC ist das richtige Maß fürs Schätzen [Boehm 1981]
  - Produktivität in LOC hängt vom Jahrzehnt ab
  - Produktivität in LOC ist unabhängig von der Programmiersprache
- LOC messen nicht die Produktgröße!
  - dasselbe Programm ist in St oder M nur 1/3 so “groß” wie in Java

Produktivität messen

Eine übliche (tayloristische) Definition

Menge = Produktivität * Arbeitskapazität * Arbeitszeit

Menge = Produktivität * Aufwand

- Intrinsisch (dieselbe Sprache):
  Größe [LOC] = Produktivität$_i$ [LOC/h] * Aufwand [Mh]
- Extrinsisch (auch verschiedene Sprachen):
  Größe [WP] = Produktivität$_e$ [WP/h] * Aufwand [Mh]

Definition via SW-Gleichung (siehe http://www.qsm.com)

Produktgröße = Produktivität * Arbeitskapazität$^{1/3}$ * Zeit$^{5/3}$

Funktionspunkte

- FP ist ein “virtuelles” extrinsisches Maß [Albrecht 1979]
  - extrinsisch: völlig unabhängig von der Sprache
  - virtuell: einen FP kann man nicht identifizieren
  - (abhängig von der Architektur, deshalb leichter Wandel)
- FP ist ein Meßverfahren in 2 wesentlichen Schritten
  - ungewichtete FP aus (gewichteten) Schnittstellen-Artefakten
  - FP durch Gewichten mit Architektur-Eigenschaften
  - Anleitung durch ein genormtes Regelwerk [IFPUG94]
- FP messen die Produktgröße
  - dasselbe Programm hat in St oder M genausoviele FP wie in Java

Funktionspunkte

- Ein Formular für Funktionspunkte nach [IFPUG94]

Funktionsart	Komplexität	Gewicht	Anzahl
Eingabedaten	einfach	3	
	mittel	4	
	komplex	6	
Ausgabedaten	einfach	4	
	mittel	5	
	komplex	7	
Abfragen	einfach	3	
	mittel	4	
	komplex	6	
Anwenderdateien	einfach	7	
	mittel	10	
	komplex	15	
Referenzdateien	einfach	5	
	mittel	7	
	komplex	10	
		Summe: FPtab =	

FP = FPtab * (0.65 + 0.01 * Korrekturfaktor)
Korrekturfaktor = SUM (F_i)

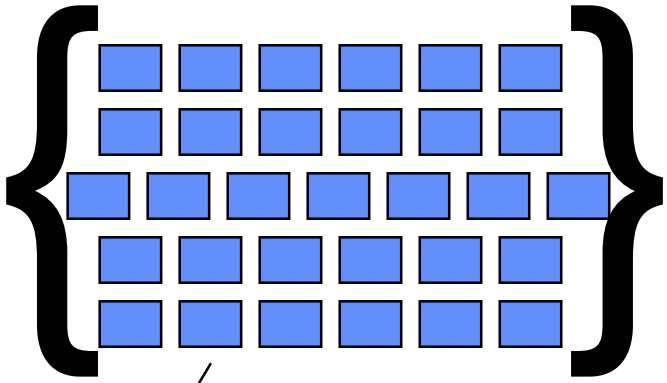
Einflußfaktoren F_i: (Polaritätsprofil: **Werte von 0=ohne Bedeutung bis 5=sehr wichtig vergeben, addieren**)

Backup/Recovery-Verfahren erforderlich?	
Datenkommunikation erforderlich?	
Verteilte Verarbeitung?	
kritische Performance?	
komplexe Nutzungsumgebung?	
Online-Dateneingabe benötigt?	
Dateneingabe über Screen-Folgen?	
Online-Update benötigt?	
komplexe Inputs,Outputs, Dateien, Abfragen?	
komplexe interne Verarbeitung?	
Wiederverwendung des Codes?	
Konvertierungen und Installation wichtig?	
mehrere Installationen, mehrere Kunden?	
besondere Benutzerfreundlichkeit?	
Summe = Korrekturfaktor =	

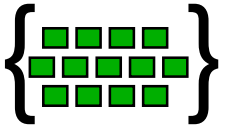
0 <= Korrekturfaktor <= 70

Funktionspunkte Einwände

Das Programm:



Maß: M: {Elemente} → N



LOC ist ein Maß !

aber: Was messen wir da?

die elementaren FP werden durch Gewichten bestimmt

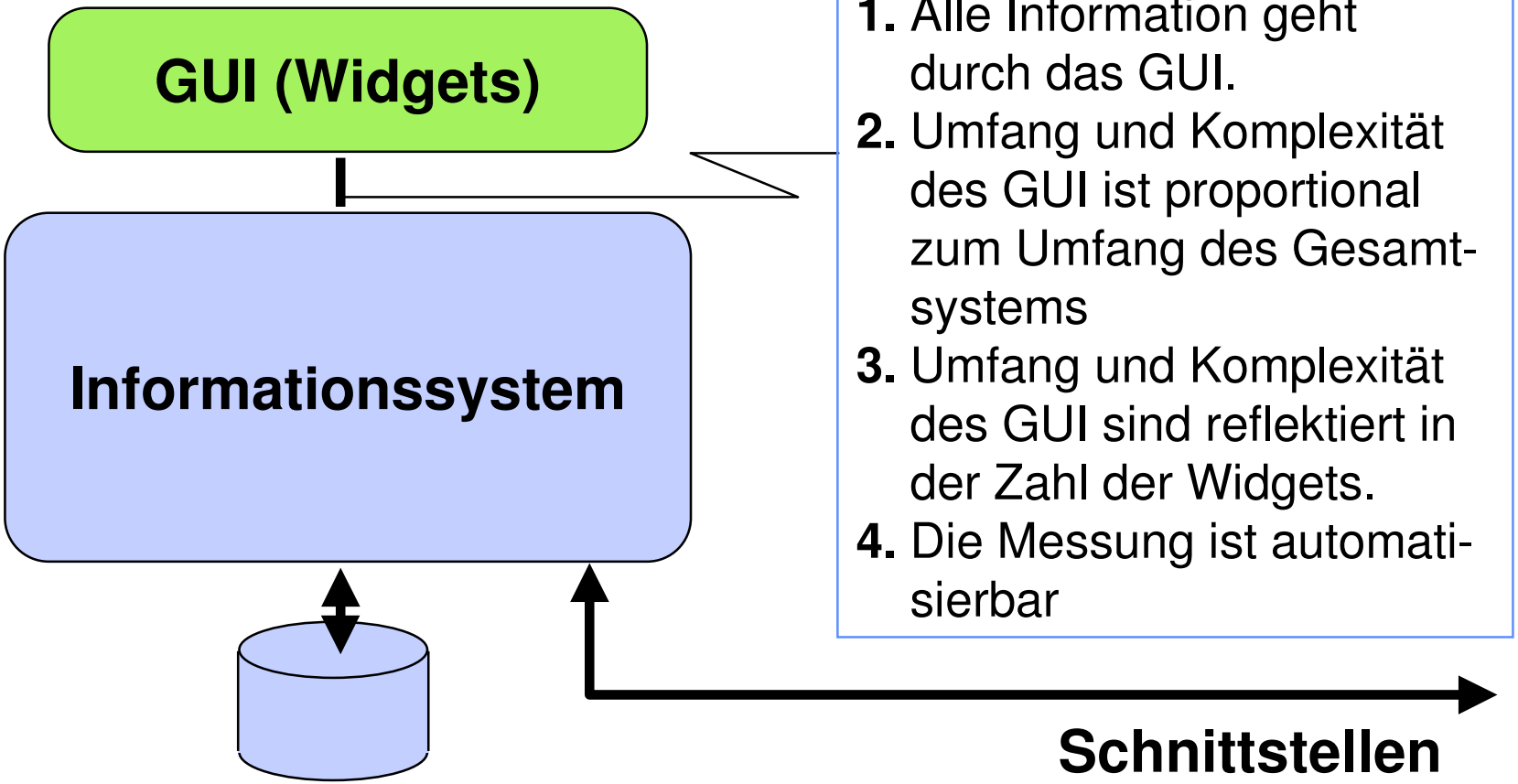
FP selbst sind messbar !

Global wird noch einmal gewichtet

Die Wiederholbarkeit ist Diskussionsgegenstand (±26%):

(Kemerer: “Reliability of function point measurement: a field experiment”, Communications of the ACM, 36, 1993, p.85)

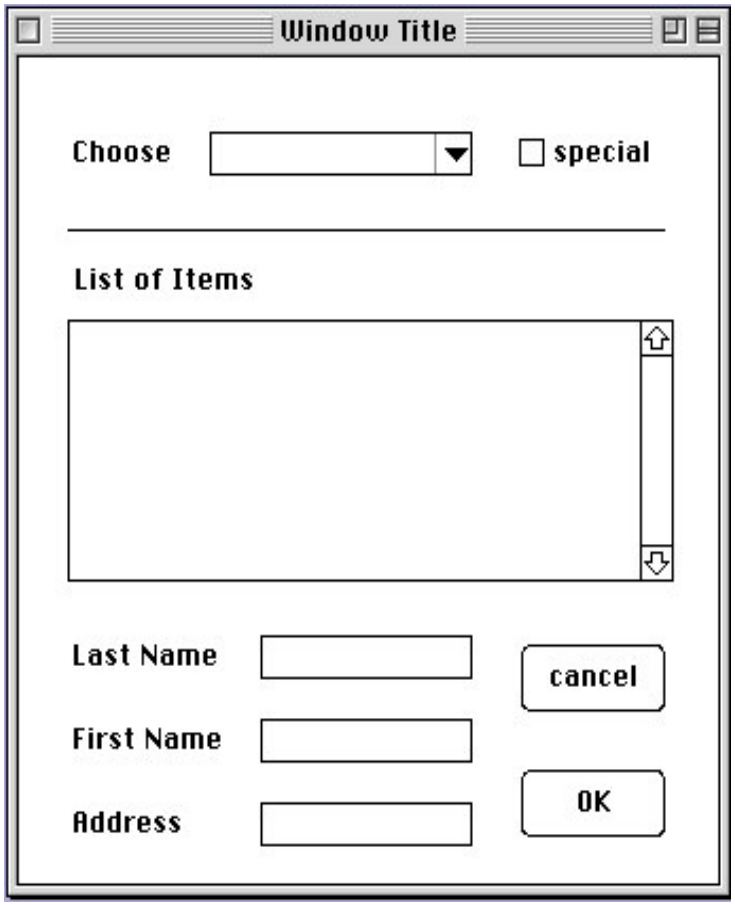
WidgetPunkte Motivation



WidgetPunkte Definition

- Die Systemgröße ist proportional zur GUI-Größe
  - bei GUI-dominierten Informationssystemen
- Die GUI-Größe (in WP) ist die Summe aller Widgets in allen Fenstern
- Das Widget ist das elementare Maß der WP
  - weil Widgets in allen Sprachen sehr ähnlich sind
  - Widgets sind sehr leicht zu zählen, besonders automatisch
  - alle Widgets haben gleiches Gewicht
  - Komplexität des Programms spiegelt sich in Komplexität des GUI
  - WP sind exakt reproduzierbar

WidgetPunkte ein Beispiel

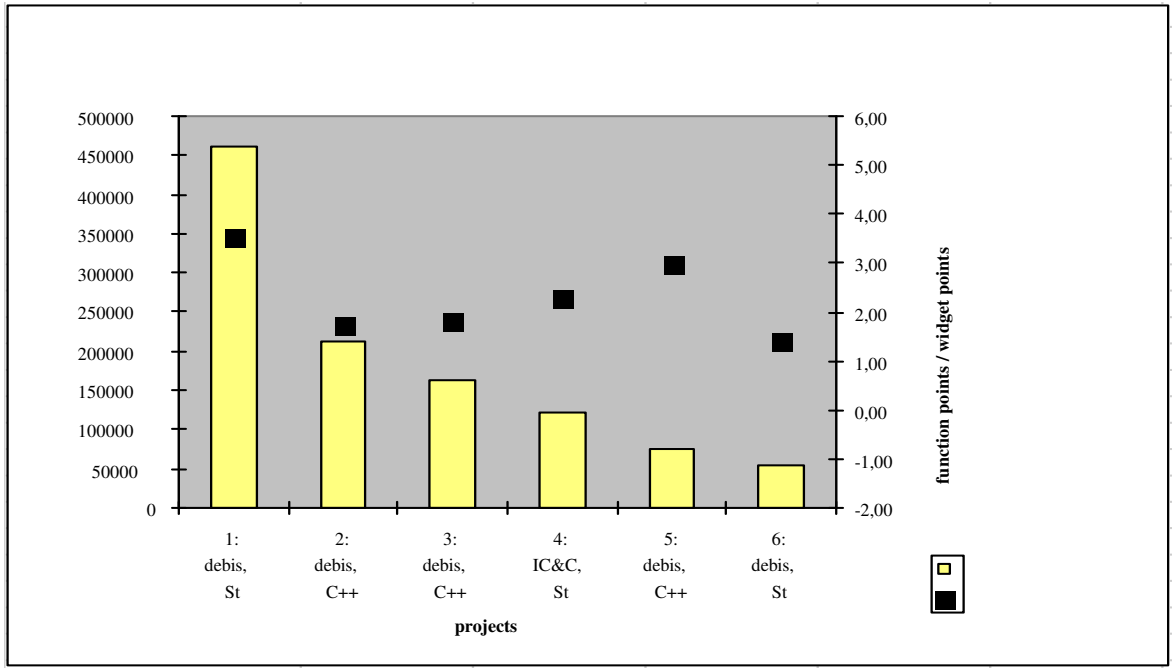


Programm / Version:	Fenster / Notizbuchblatt:	
Widget-Art		**Anzahl**
Fenster / Notizbuchblatt		.....**1**
† berschrift		.....**1**
Eingabe / Ausgabe (incl. 1 Zeile je Tabelle)		
Ein/Ausgabefelder		.....**3**
Aktionsknšpfe		.....**2**
Comboboxen		.....**1**
MenŸknšpfe		.....
Radioknšpfe		.....
CheckBoxen		.....**1**
Texteditoren		.....
Zusammengesetzt		
Listen		.....**1**
Tabellen		.....
NotizbŸcher		.....
freie Rollbalken		.....
Beschreibend		
Beschriftungen		.....**6**
Knopf-Beschriftungen		.....**2**
SpaltenŸberschriften		.....
GruppierungskŠsten		.....
Trennlinien		.....**1**
MenŸ		
MenŸs		.....**4**
MenŸeintrŠge		.....
weitere Widgets		
	Widgetpunkte (Summe):	**23**

WidgetPunkte Grenzen

- Nur GUI-Anwendungen !!! Genauigkeit steigt mit GUI-Größe
- Weitere Statistik dringend erwünscht !
- Erster Vergleich
- mit (backfired !)
- Funktionspunkten:

- 1 WidgetPunkt = 2,28 ± 0,34 FP



Automatisierung von Messungen

- Ohne Automatisierung keine Messungen!
- beim täglichen oder wöchentlichen Build
  - LOC zählen: < cat * | wc >
  - WP zählen nur wenig schwieriger: zuerst anhand des GUI-Frameworks die Zählstrategie festlegen und testen!
- Die erste OO-Metrik erhält man
  - durch Zählen der LOC pro Klasse, dann erst summieren
- Änderungsraten messen
  - zählen der neugelieferten Klassen (pro Modul...)

Monitore
Produktivität, Qualität, Testfortschritt

- Produktivitätsmonitor
  - benutzt einfach die LOC-Zählung (und Aufwandsmessung)
  - überzeugt das Management
- Qualitätsmonitor
  - Vergleich der WP mit den LOC
    - Bei Refaktoring gehen WP rauf, LOC runter
  - LOC/Klasse-Metrik
  - Änderungsrate pro Klasse, Modul, ...
- Testfortschrittsmonitor, Qualitätsmonitor
  - messen der Fehlermenge (gegen LOC, pro Modul, ...)
  - ... und dann Konzentration der Tests

Monitore
Iterationsmonitor

- Nachsteuern in agilen Projekten
  - aus LOC, WP jeweils hochrechnen aufs Gesamtprodukt
  - aus Aufwand, LOC hochrechnen auf Termin und Kosten
  - aus OO-Qualitätsmetriken (LOC/Klasse) Qualifikationsprobleme finden
  - aus Änderungsraten Spezifikationsprobleme oder Qualifikationsprobleme finden
  - aus Fehlerraten Qualifikationsprobleme finden und / oder Testaufwand umverteilen

Schätzungen
Inhalt

- Das Prinzip Schätzung
- Projekt vs. Produkt
- Mikro und Makro-Schätzung
- Projektschätzung = Mikroschätzung
- Makroschätzung: Software-Gleichung
- Makroschätzung: Faustformeln nach C.Jones
- Beispiele
- Technologie trennen von Projekteinflüssen

Schätzung
Das Prinzip Schätzung

- Schätzen heißt Wahrsagen, das Ziel ist: Fehler minimieren
- Schätzen = Zerlegen + Vergleichen

  Entwurf Erfahrung (gemessene Projekte, Produkte)

- Fehlerquellen:
  - Unvollständigkeit der Zerlegung
  - Unzulässigkeit des Vergleichs
- Maßnahmen:
  - Fehlermargen notieren
  - mehrere Schätzer einsetzen
  - feiner Zerlegen (ist aber mehr Arbeit!)

Schätzung
Projekt vs. Produkt

- Produktschätzung ist linear
  - Zerlegung in bekannte Teile (Entwurf)
  - Übertragung der Größe bekannter Teile
  - Zusammenrechnen

- Projektschätzung ist nichtlinear
  - Zerlegung in Aufgaben
  - davon sind einige: Herstellung der Produktteile
  - Einarbeitung, Spezialisierungen, Knowhow-Übertrag
  - Abhängigkeiten der Aufgaben voneinander

Mikro und Makro-Schätzung

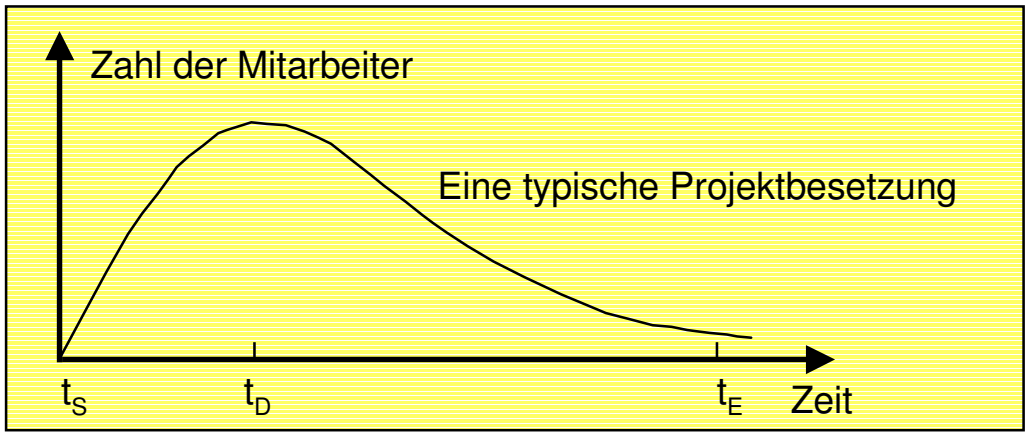
- Jede Projektschätzung beinhaltet eine Produktschätzung
- Mikroschätzung
  - Zerlegung des Produktes implizit in der Projektzerlegung
- Makroschätzung
  - erst Produktschätzung (Mikro oder Makro, in LOC, FP, WP)
  - dann übertragen auf Projekt
  - die SW-Gleichung ist nichtlinear

Mikroschätzung

Projektschätzung = Mikroschätzung

----------	**66**
– ----------	12
– ----------	23
• ----------	14
• ----------	9
– ----------	31
• ---------	15
• ----------	16
----------	**185**
– ----------	43
– ----------	54
• ----------	24
• ----------	13
• ----------	17
– ----------	75
• ---------	18
• ----------	17
– ...	...

Makroschätzung
Projektbesetzung nach Norden

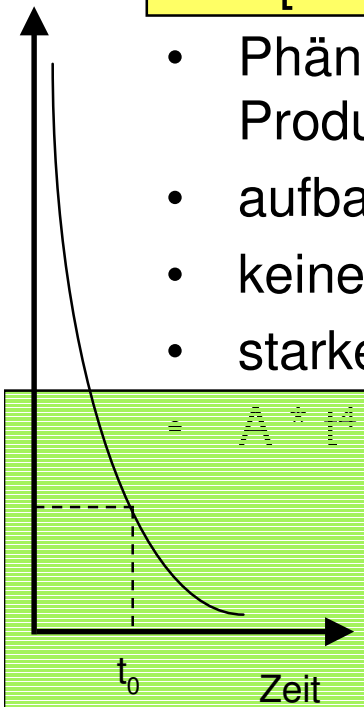


$$M = A*t/t_D^2*\exp(-t^2/2t_D^2)$$

- Projekttheorie (Norden) Rayleigh-Verteilung
  - Zerlegung in kleinstmögliche Aktivitäten,
  - Vorgänger - Nachfolger und maximale Parallelisierung

Makroschätzung
Software-Gleichung von Larry Putnam

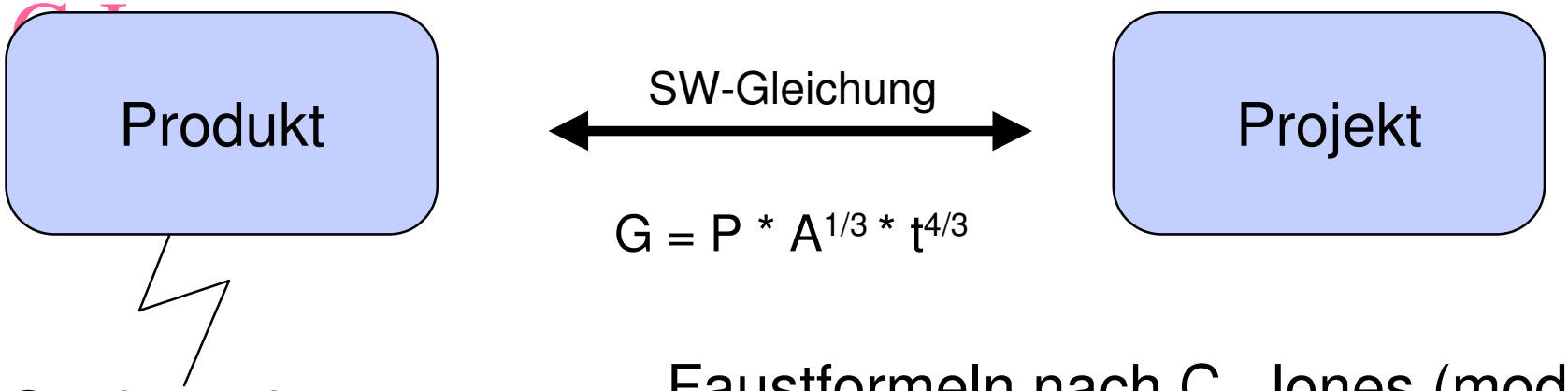
- $G[LOC] = P * A^{1/3} * t^{4/3}$
- Phänomenologische Gleichung für den Zusammenhang zwischen der Produktgröße G[LOC], dem Aufwand A[Ph] und der Projektdauer t
- aufbauend auf einer Projektbesetzung à la Norden
- keine freien Parameter außer „Technologiefaktor“ P
- starke Abhängigkeit des Aufwands von der Zeit



$dA/A = -4*dt/t$ wenn:

- keine Aufgaben mehr teilbar
- keine “Luft” auf dem kritischen Pfad

Makroschätzung
SW-Gleichung und Faustformeln nach C. J



Größenschätzung

$G^3 = P^3 * A * t^4$
$G^3 = P^3 * M * t^5$
$G^3 = P^{3*}c * G * t^5$
$G^2 = c' * t^5$

Faustformeln nach C. Jones (mod.)

(Minimum Leute, konstante Problemkapazität)
$M[MA] = G[LOC] / 19200\ [^{LOC}/_{MA}]$

$t[Mte] = (G[LOC] / 128)^{2/5}$
$A = M * t$

Makroschätzung
Beispiele für Faustformeln nach C.Jones

1 PM = 2000/12 h = 166,6... h

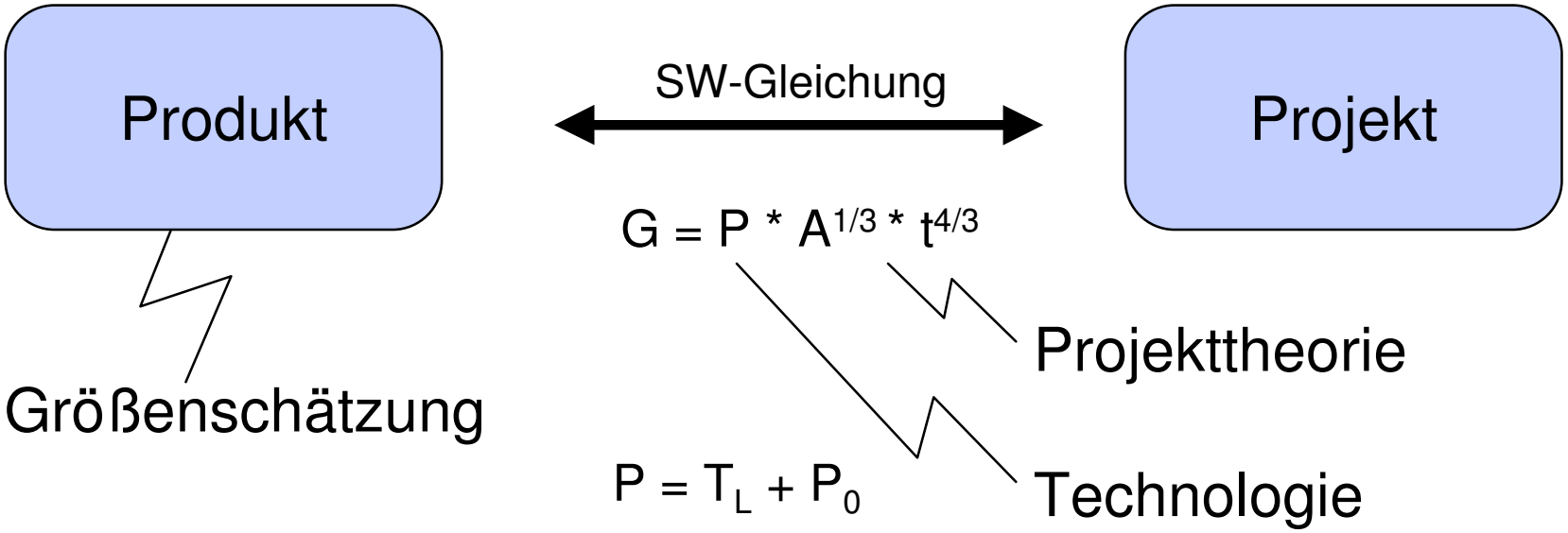
Projekt/ Firma	Technologie	LOC gemessen	Aufwand nach Faustformeln	gemessener Aufwand [h]	Abweichung [%]
1/1	Smalltalk	60.082	6.108	5.360	+ 14
2/1	Smalltalk	500.000	118.637	94.700	+ 25
3/1	Java	32.400	2.573	2.550	+ 1
4/1	C++	132.000	18.385	12.500	+ 47
5/2	Smalltalk	122.000	16.465	7.700	+ 114

nach den Faustformeln hätte dieses Projekt
6,35 MA und 15,5 Mte Laufzeit, also Aufwand = 98,8 PM = 16465 h.

Mit der SW-Gleichung umskaliert auf die tatsächliche Laufzeit von 20 Monaten ergibt sich ein Aufwand von 16465 h * (15,5/20)^4 = 16465 h * 0,36 = 5940 h.

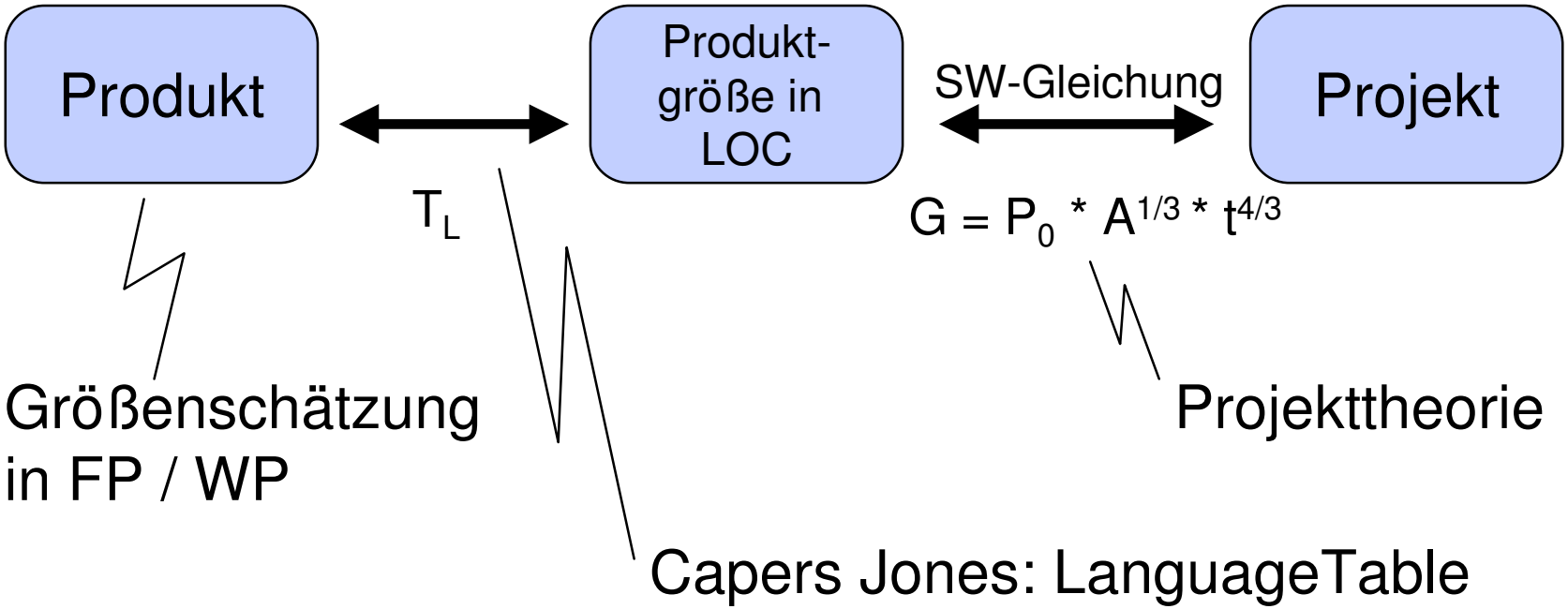
An diesem Projekt waren statt 2 allerdings 5 Personen beteiligt.

Makroschätzung
SW-Gleichung:Technologie&Projekteinflüsse



P.V.Norden: On the Anatomy of Development Projects, 1960
L.H.Putnam: A General Empirical Solution on the Macro Software Sizing and Estimating Problem, 1978

Makroschätzung
Technologie trennen von Projekteinflüssen



Makroschätzung
Beispiel: Technologie & Projektgröße

Das Produkt hat 10.000 FP

- Realisierung in C++
  - 530.000 LOC
  - 28 MA
  - 28 Mte Laufzeit
  - 784 PM

- Realisierung in St
  - 210.000 LOC
  - 11 MA
  - 20 Mte Laufzeit
  - 220 PM

auf einen Faktor 2 genau!

Vorbehaltlich der Verfügbarkeit von Smalltalkern

Vorbehaltlich der Verfügbarkeit benötigter Technologien ...

Fehlermenge
Literatur [C. Jones]

Fehler nach Ursprung	**je kLOC**
Anforderungen	5
Dokumentation	3
Design	6
Kodieren	9
Fehlerbehebung	2
Summe	**25**
Summe in agilen Projekten	**17**

Fehler nach Sprache (Java) **12,5**

Fehlermenge
Literatur [C. Jones, B. Boehm]

- Geliefert werden von
  - 1 Fehler / kLOC in militärischen System) bis
  - 8 Fehler / kLOC in MIS
- MTBF
  - 1 - 4 h bei 5 - 10 Fehlern / kLOC
  - 24 - 160 h bei 1 - 2 Fehlern / kLOC

- wieviele Fehler darf ich liefern?
- wieviel Aufwand benötige ich zum Testen?
- wann bin ich fertig?

Testaufwand
Literatur [C. Jones, B. Boehm]

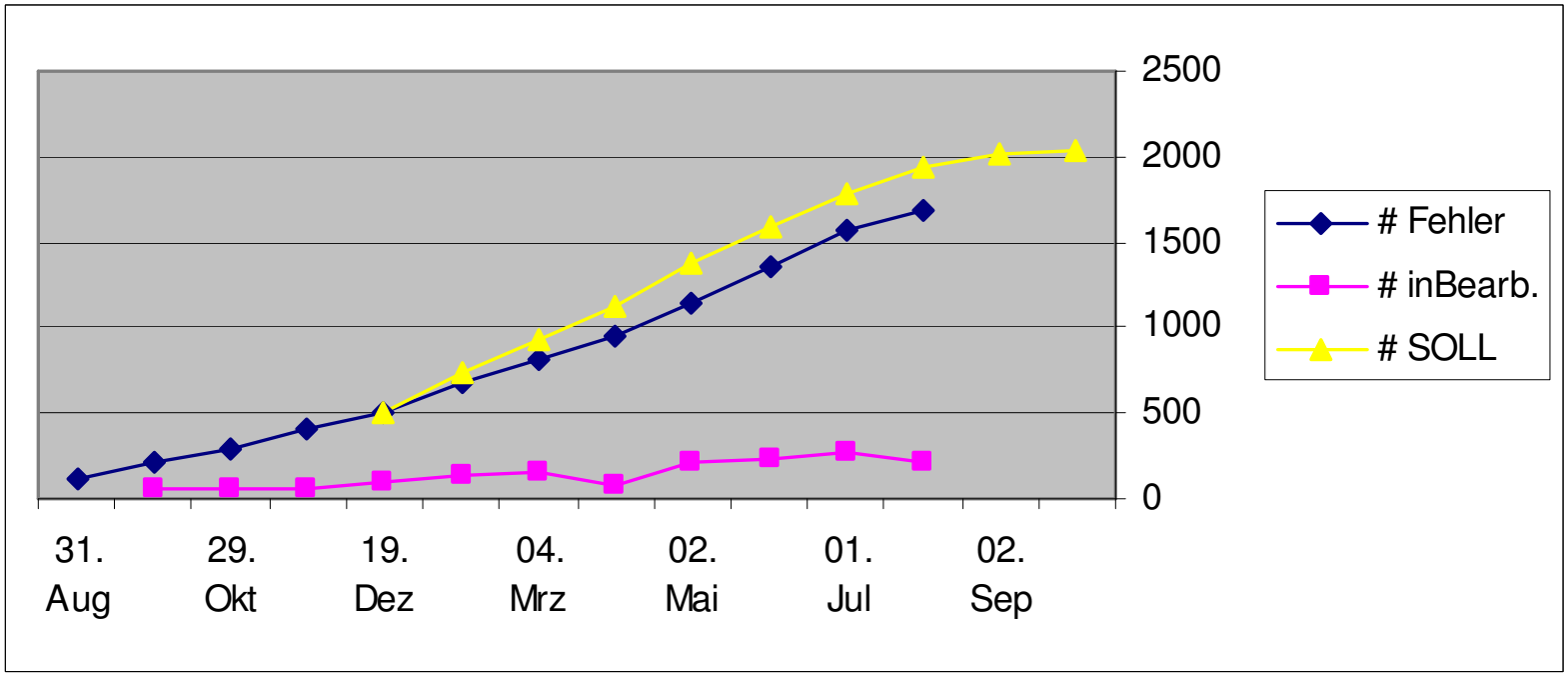
- Test ist ca. 30% bis 50% des Gesamtaufwands
- Testaufwand = A * X(pro LOC) + B * Y(pro Fehler)
- Behebungsaufwand = C * Z (pro Fehler)
- Literatur: C= 4h
- Technik Junit-Tests (oder Sunit oder ...): C = 12h
  - 2h: Analyse
  - 8h: Junit-Test schreiben und Fehler reproduzieren
  - 2h: Behebung

Fehlerbeseitigung
Literatur

- Jede Teststufe findet effizient ca. 40% der vorh. Fehler
- große Systeme brauchen mehrere Teststufen, z.B.
  - Entwicklertest (unit test)
  - Anwendungstest (function test)
  - Systemtest (performance test)
  - Integrationstest (integration test)
  - Abnahmetest (field test, acceptance test)
- nach diesen 5 Teststufen gibt es von 12,5 noch 60% - 36% - 22% - 13% - 8% Fehler = 1 Fehler

Fehlerverfolgung
Das Basisdiagramm

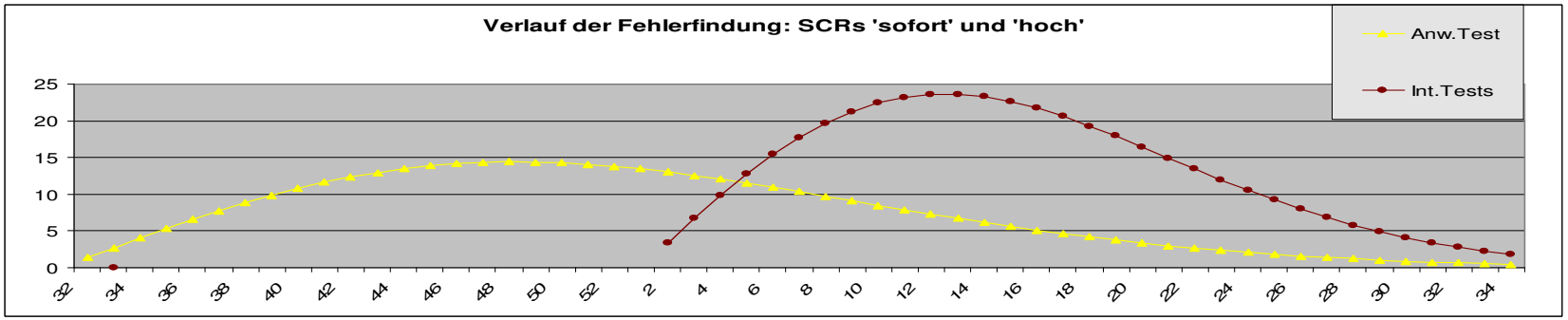
Gesamt-Soll, Gesamt-Ist, Fehler in Bearbeitung (CTA):

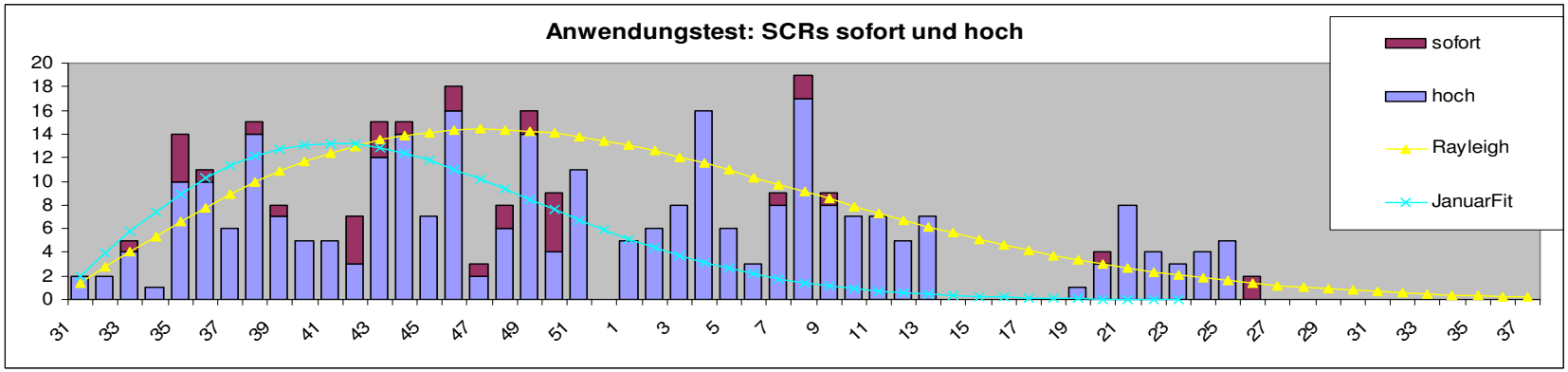


Fehlerverfolgung
Rayleighkurven für 2 Teststufen

Erlauben frühzeitige Messung (Fit) der Gesamtfehlerzahl

Daten von CTA (unten): blau = früher zu opt. Fit, gelb = real. Fit 6 Mte. vor Echtbetrieb

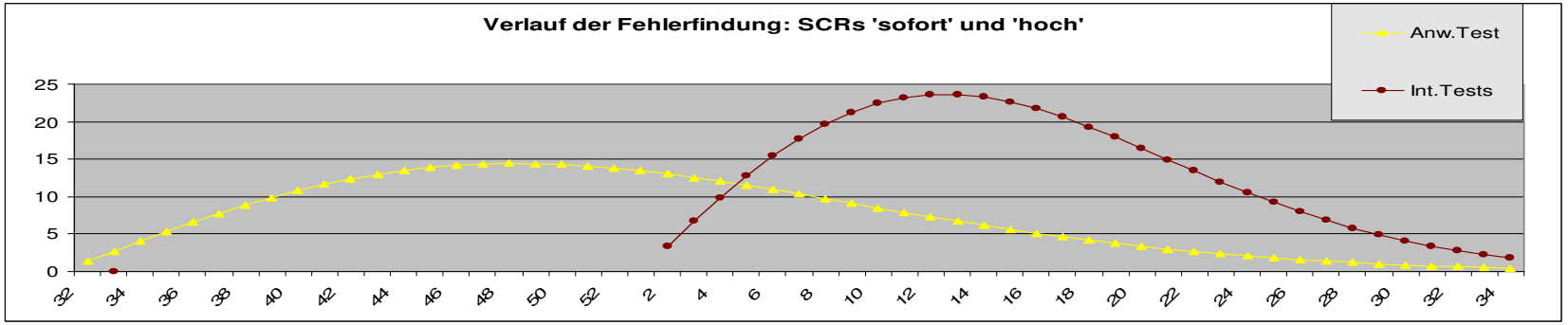


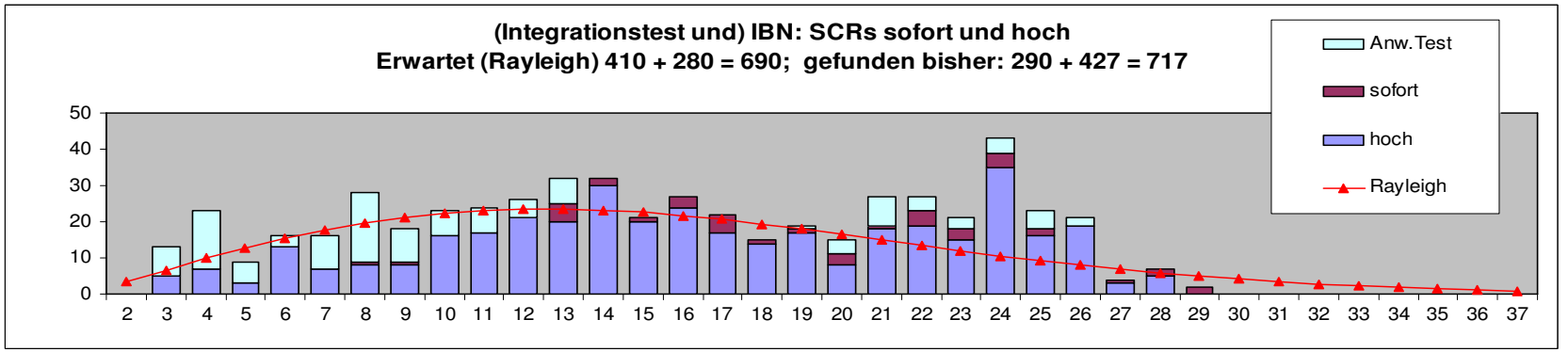


Fehlerverfolgung
Rayleighkurven für 2 Teststufen

Erlauben frühzeitige Messung (Fit) der Gesamtfehlerzahl

Hier angewandt auf die 2 höchsten Fehlerklassen von 5 (gemessener Anteil 42%)





Fehlerverfolgung im CTA-Projekt gelernt

- 12,5 (/kLOC Java) * 600 kLOC = 7500 Fehler nach Progr. erwartet
  - davon 60% oder 4500 im Test erwartet
- Fit der Gesamtfehlerzahl auf die gemessenen Rayleigh-Kurven
  - im Oktober 2001 und im Februar 2002
- **2050 Fehler wurden / werden in den Test geliefert**
  - d.s. 45% des Literaturwertes
- ~~7,5 Fehler / kLOC~~ **3,5 Fehler / kLOC nach JUnit**
- **Ende Feb. 02 wurde der Termin (26.6.02) für den Beginn des Echtbetriebs (erste Schiffsabfertigung) verläßlich prognostiziert**

Empfehlungen

- Angebote
  - Bieten Sie bei wenig Zeit mit einer Makroschätzung an!
  - Validieren Sie jede Mikroschätzung mit einer Makroschätzung
- Messen in Projekten
  - Setzen Sie ein Messprogramm auf, mindestens LOC (/ Klasse)
  - WP (oder FP)
  - Änderungsrate (Qualität)
  - Fehlermengen (Aufwand, Qualität)
- Nachbereitung
  - Qualität: aus den letzten Messungen
  - Produktivität: Aufwand / LOC und Aufwand / WP (oder FP)

Literaturhinweise Weiterbrowsen

- Software-Produktivität und Controlling
  - Quantitative Software Management (Larry Putnam) http://www.qsm.com
  - Software Productivity Research (Capers Jones) http://www.spr.com (LanguageTable nur noch gegen $ 75,--)
- Testen
  - www.junit.org (www.xunit.org)
- Funktionspunkte
  - International Function Point UsersGroup http://www.ifpug.org
  - Deutschsprachige Anwendergruppe für Software-Metrik und Aufwandschätzung http://www.dasma.de
  - GI Fachgruppe SMB (2.1.10) SW-Messung und -Bewertung http://ivs.cs.uni-magdeburg.de/sw-eng/us/giak

Literaturhinweise Weiterlesen

- Praxis:
  - Controlling Software Projects, Management, Measurement & Estimation, Tom de Marco, Prentice Hall, 1982
- Theorie
  - “On the Anatomy of Development Projects” von P.V. Norden, IRE Transactions on Engineering Management, PGEM, EM-7 (60) 34
  - “A General Empirical Solution to the Macro SW Sizing and Estimating Problem” von L.H.Putnam, IEEE Transactions Software Engr., July 1978, 345
  - “Cost Estimation for SW Development” von B. Londeix, Addison Wesley, 1987

Literaturhinweise Weiterlesen

- “Der Termin” von Tom deMarco, Hanser 1998
- “SW Engineering Economics” von Barry Boehm, Prentice Hall 1981
- “The Mythical Man Month” von F. Brooks, Addison Wesley, 1975
- “Applied Software Measurement” und “Estimating Software Costs” von C. Jones, McGraw-Hill, 1996 und 1998

Literaturhinweise
Weiterlesen