

# **Schätzen der Fehlerzahl in Software-Dokumenten**

**Dr. Frank Padberg  
Universität Karlsruhe  
Mai 2003**

# Motivation

- beim Testen oder bei Inspektionen werden nicht alle Fehler gefunden
- Anzahl der unentdeckten Fehler ist nie genau bekannt
- Fehlerzahl ist wichtige Größe für Qualitätssicherung und Projektmanagement

# Motivation

- beim Testen oder bei Inspektionen werden nicht alle Fehler gefunden
- Anzahl der unentdeckten Fehler ist nie genau bekannt
- Fehlerzahl ist wichtige Größe für Qualitätssicherung und Projektmanagement
- **schätze** die Fehlerzahl in einem Software-Dokument **möglichst genau!**

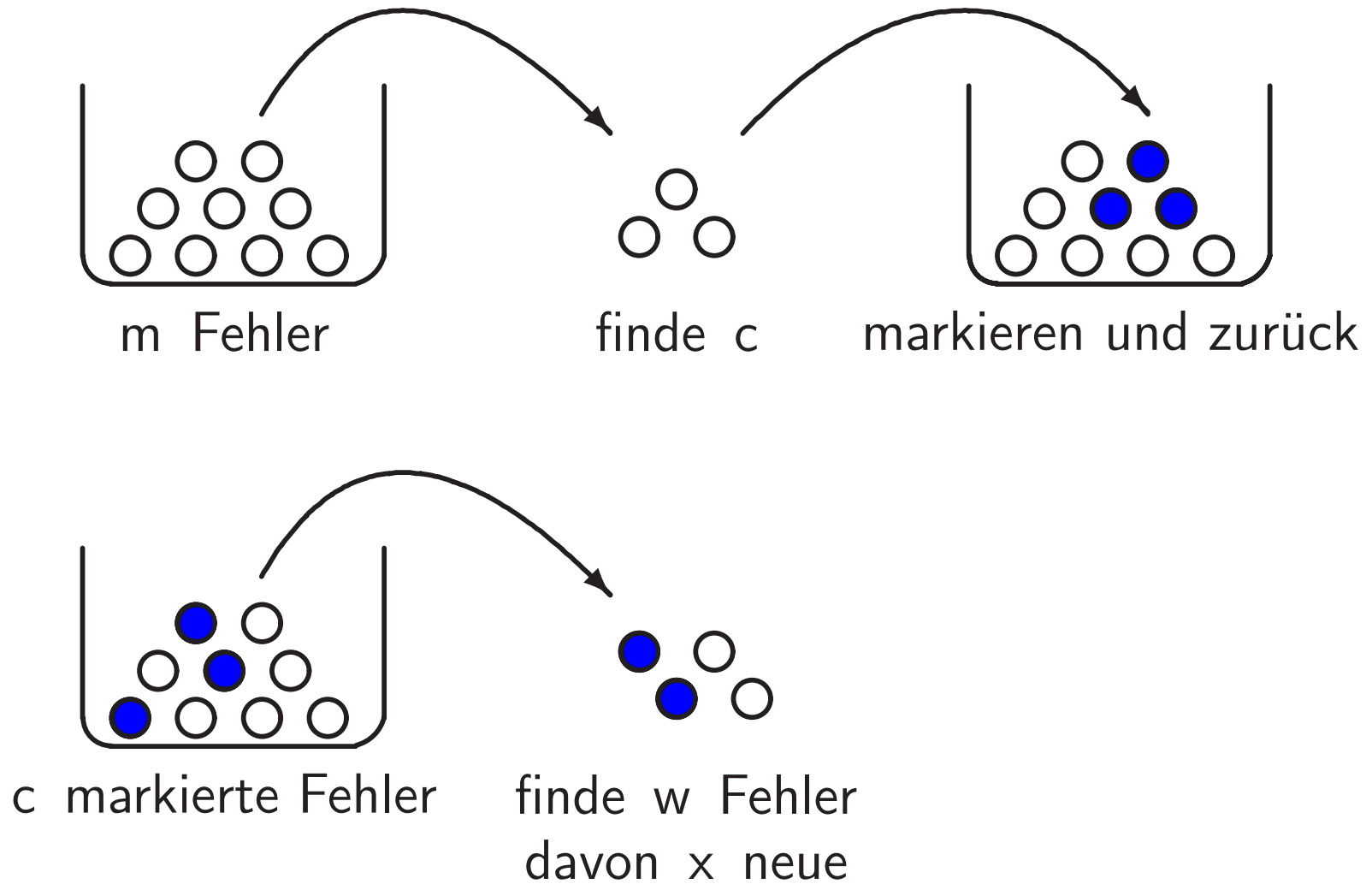
# Lösungsansatz: Testreihen für Code

- nutze Information aus den Software-Tests zum Schätzen; z.B. Anzahl der Fehler je Test
- baue stochastisches Modell für Testreihen, das durch die Fehlerzahl  $m$  parametrisiert ist
- wende statistische Techniken an, um  $m$  aus konkreten Testergebnissen zu schätzen

# Capture-Recapture-Methoden

- kommen aus der Biologie: Schätzen der Größe einer Tierpopulation
- fange Tiere, markiere sie und lasse sie wieder frei
- fange nochmal Tiere und zähle, wieviele markiert sind
- schätze die Gesamtpopulation aus dem Verhältnis zwischen markierten und nicht markierten Tieren im zweiten Fang

# Urnenmodell



# Stochastisches Modell

- hypergeometrische Verteilung:

$$p_{m,w}(x, c) = \frac{\binom{m-c}{x} \cdot \binom{c}{w-x}}{\binom{m}{w}}$$

- im Beispiel:  $c = 3$ ,  $w = 4$ ,  $x = 2$

$$p_{m,4}(2, 3) = \frac{\binom{m-3}{2} \cdot \binom{3}{4-2}}{\binom{m}{4}}$$

# Likelihood-Funktion

- wie wahrscheinlich ist die Beobachtung  $(w, x)$  wenn vorher schon  $c$  Fehler gefunden wurden?
- die Antwort hängt von  $m$  ab
- definiere die Likelihood-Funktion

$$L(m) = p_{m,w}(x, c)$$

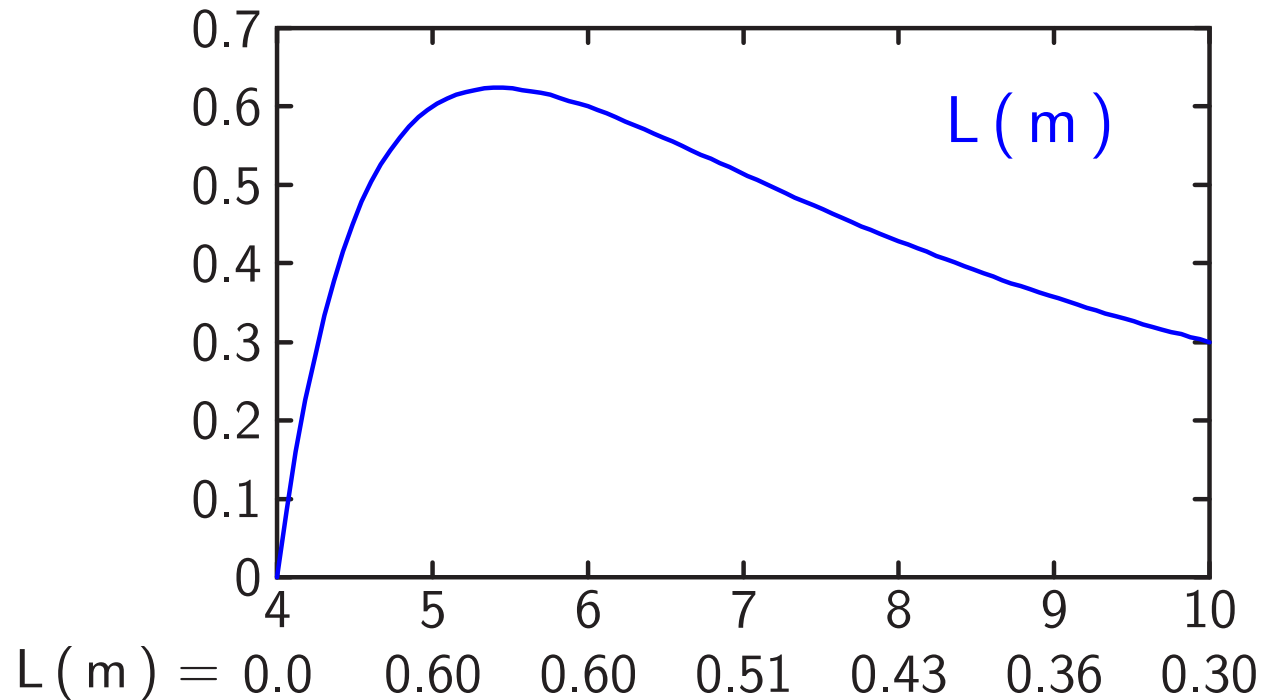
- $w$ ,  $x$  und  $c$  sind gemessen (also fest)



# Max-Likelihood-Schätzung

- **Schätzwert** ist dasjenige  $m$ , für welches die Beobachtung  $(w, x)$  am wahrscheinlichsten ist
- ein ML-Schätzwert  $\widehat{m}$  **maximiert** also die Likelihood-Funktion:  $L(\widehat{m}) \geq L(m)$
- $\widehat{m}$  muß nicht eindeutig sein

# Beispiel einer Likelihood-Funktion

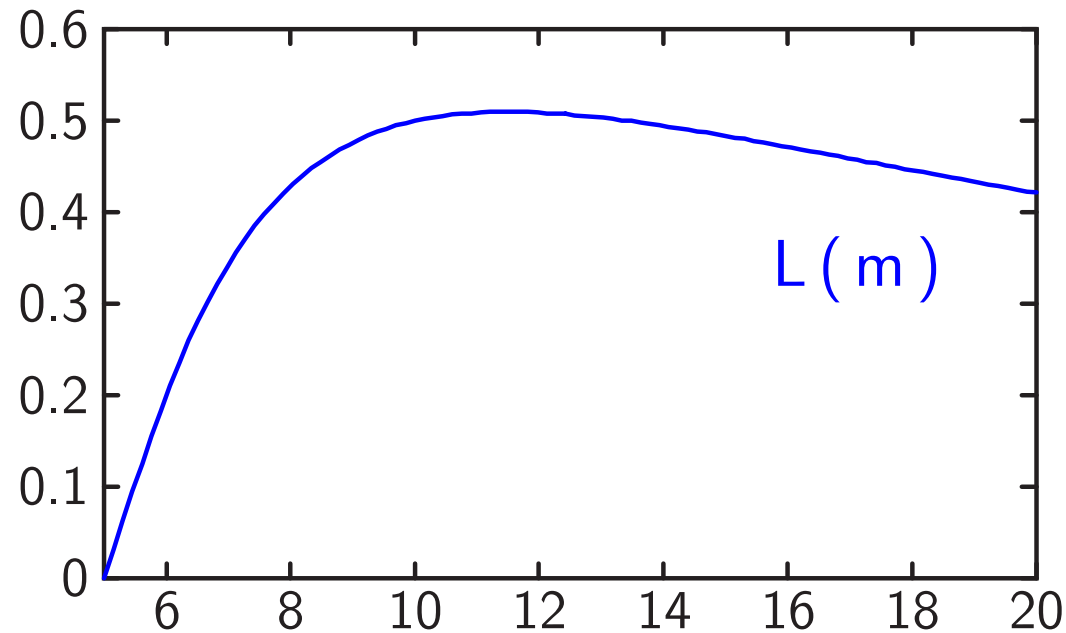


**zwei** markierte Fehler im zweiten Test

$$(c = 3, w = 4, x = 2)$$

Max-Likelihood-Schätzwert: 5 oder 6

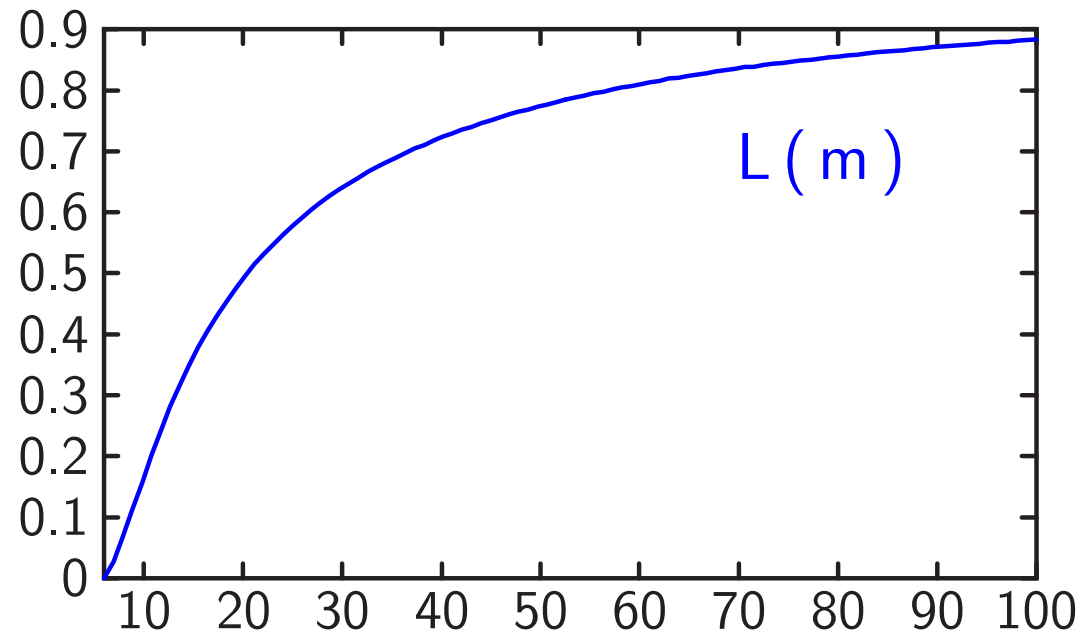
# Anderes Beispiel



nur **ein** markierter Fehler im zweiten Test  
(  $c = 3$ ,  $w = 4$ ,  $x = 3$  )

Max-Likelihood-Schätzwert: 11 oder 12

# Noch ein Beispiel



kein markierter Fehler im zweiten Test

$$(c = 3, w = 4, x = 4)$$

Max-Likelihood-Schätzwert: “unendlich”

# Verallgemeinerung auf Testreihen

Y. Tohma, R. Jacoby, Y. Murata, 1989 – 1995  
M. Yamamoto, ...

R.-H. Hou, S.-Y. Kuo, Y.-P. Chang, 1994 – 1997  
I.-Y. Chen

Hypergeometric Reliability Growth Model

# Notation

$w_k$  : Anzahl der gefundenen Fehler in Test  $k$

$x_k$  : Anzahl der neu entdeckten Fehler in Test  $k$

$c_k$  : Summe der in den ersten  $k$  Tests gefundenen verschiedenen Fehler

$$c_k = x_1 + x_2 + \dots + x_k$$

# Allgemeines Hypergeometrisches Modell

- einzelner Test  $k$

$$p_{m, w_k} (x_k, c_{k-1}) = \frac{\binom{m - c_{k-1}}{x_k} \cdot \binom{c_{k-1}}{w - x_k}}{\binom{m}{w_k}}$$

- Likelihood-Funktion für Serie von  $n$  Tests

$$L(m) = p_{m, w_1} (x_1, 0) \cdot p_{m, w_2} (x_2, c_1) \cdot \\ p_{m, w_3} (x_3, c_2) \cdot \dots \cdot p_{m, w_n} (x_n, c_{n-1})$$

# Ableiten der Likelihood-Funktion ?!

$$p_{m,w}(x, c) = \frac{(m-c)! c! w! (m-w)!}{x! (m-c-x)! (w-x)! (c-w+x)! m!}$$

$$\frac{d p_{m,w}(x, c)}{d m} = \dots \text{ viel zu kompliziert!}$$

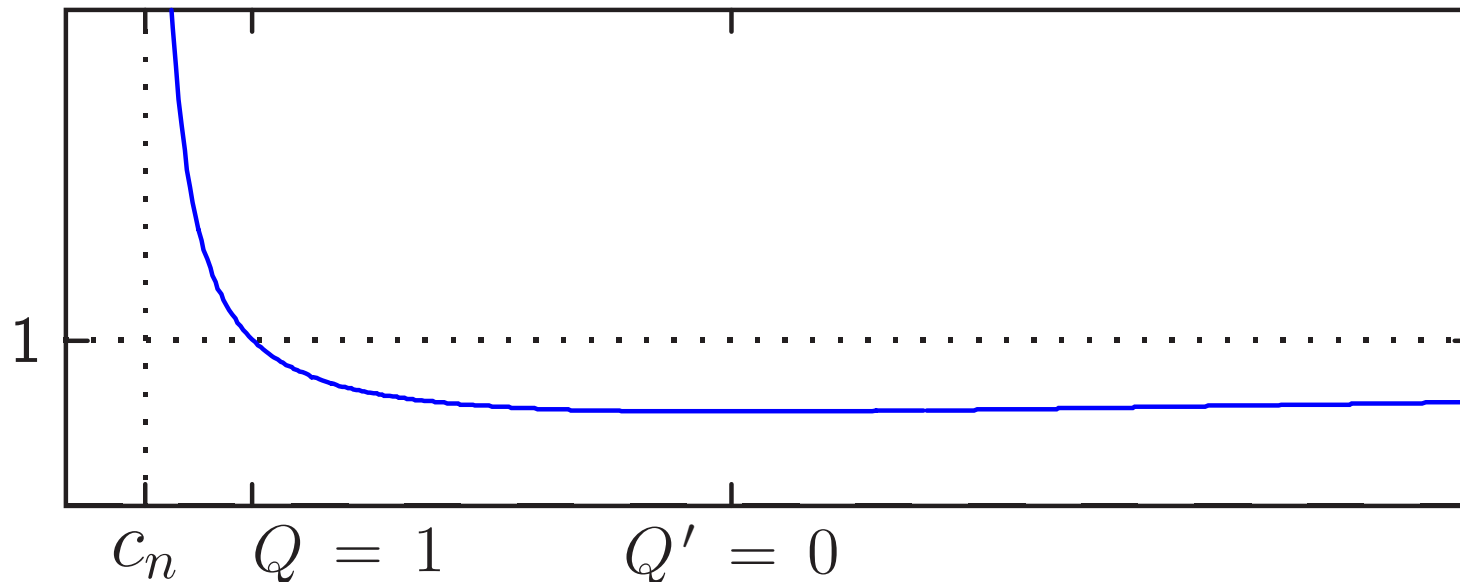


# Wachstums-Quotient

$$Q(m) = \frac{L(m)}{L(m-1)}$$
$$= \frac{(m - w_1) \cdot (m - w_2) \cdot \dots \cdot (m - w_n)}{m^{n-1} \cdot (m - c_n)}$$

- $L(m)$  wächst genau dann wenn  $Q(m) > 1$  ist
- $Q(m)$  hängt nicht von den einzelnen  $x_k$  ab, und nicht von der Reihenfolge der Tests

# Typischer Graph des Quotienten



der Graph schneidet die Gerade  $y = 1$   
genau einmal für  $x > c_n$  (Padberg 2001)

# Algorithmus

- $Q$  schneidet  $y = 1$  einmal für  $x > c_n$
- Schätzwert ist die größte ganze Zahl links vom Schnittpunkt
- probiere  $Q(c_n + 1)$ ,  $Q(c_n + 2)$ , ...
- einfach, schnell, genau, numerisch stabil

# Ein größeres Beispiel: $T_{19}$

$k$	$x_k$	$w_k$
1	15	15
2	29	29
3	22	22
4	37	37
5	2	21
6	5	43
7	36	56

$k$	$x_k$	$w_k$
8	29	48
9	4	15
10	27	30
11	27	46
12	22	24
13	21	57
14	22	69

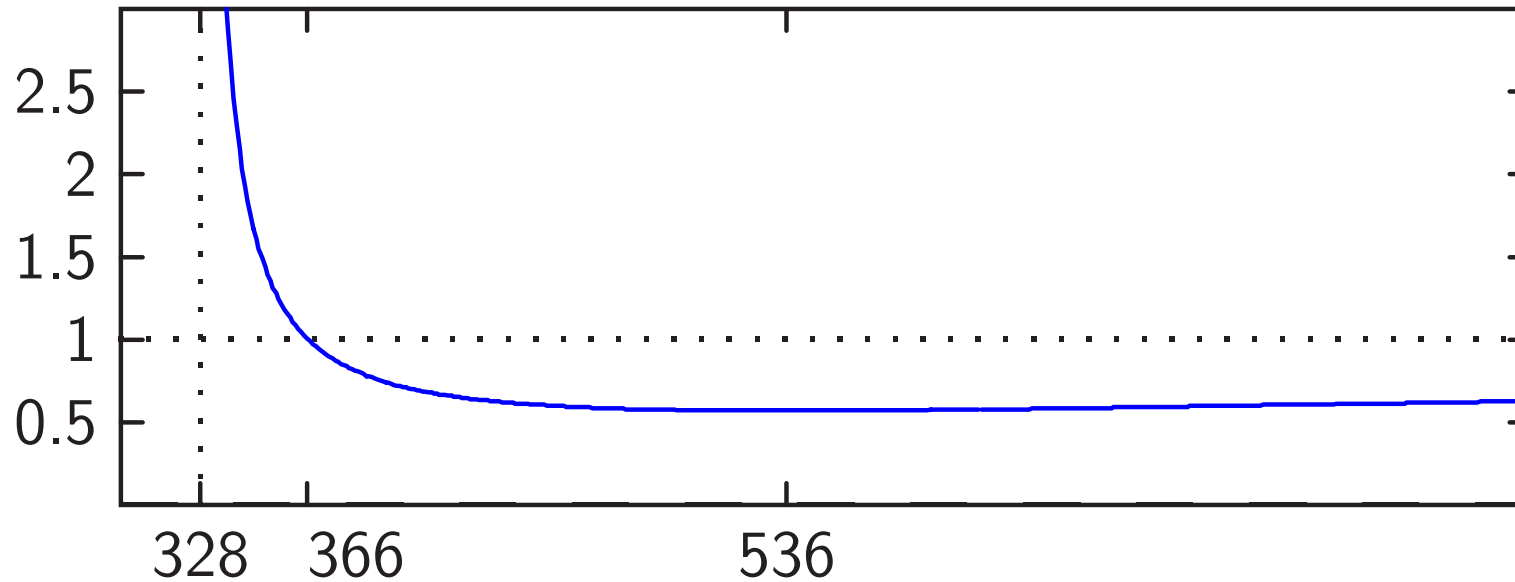
$k$	$x_k$	$w_k$
15	6	72
16	7	45
17	9	36
18	5	41
19	3	64

Testreihe der Länge  $n = 19$

# Schätzung für $T_{19}$

- Testreihe der Länge  $n = 19$
- beim Testen entdeckte Fehler:  $c_{19} = 328$
- erst später entdeckte Fehler: 30
- Gesamtzahl enthaltener Fehler: 358
- hypergeometrischer Schätzwert:  $\widehat{m} = 366$

# Graph des Quotienten für $T_{19}$



Schätzwert 366, wahrer Wert 358

# Vergleich mit anderen Schätzmodellen

Modell	Schätzwert	Fehler (%)
exponential MLE	455	+ 27.1
delayed S-shaped MLE	351	-2.0
inflection S-shaped MLE	347	-3.1
hypergeometric MLE	366	+ 2.2
hypergeometric LSE	388	+ 8.4

Max-Likelihood-Schätzer für das hypergeometrische Modell schneidet gut ab

# Eigene Veröffentlichungen zum Hypergeometrischen Modell

- *A Fast Algorithm to Compute Maximum Likelihood Estimates for the Hypergeometric Software Reliability Model*  
Asia-Pacific Conference on Quality Software APAQS (2001) 40–49,  
IEEE Computer Society Press
- *Maximum Likelihood Estimates for the Hypergeometric Software Reliability Model*  
International Journal of Reliability, Quality and Safety Engineering  
IJRQSE (2003) erscheint in Kürze, World Scientific Publishing



# Software–Inspektionen [1]

- mehrere Inspektoren (2 bis 8)
- untersuchen unabhängig voneinander dasselbe Software–Dokument
- gefundene Defekte (Fehler, Schwachstellen, Unklarheiten) werden aufgeschrieben
- gemeinsames Durchsprechen der Defekte
- Probleme identifizieren, nicht lösen

# Software–Inspektionen [2]

- anwendbar auf alle Software–Dokumente:  
Pflichtenhefte, Spezifikationen,  
Entwürfe, Code, Testfälle, ....
- jederzeit und **frühzeitig** durchführbar
- **sehr effektiv** in der industriellen Praxis
- aufwendig von Hand, also teuer
- Schätzen der Fehlerzahl?

# Lösungsansatz: Inspektion als “Testreihe”

- jeder Inspektor entspricht einem Test
- $w_k$  ist die Anzahl der Defekte, die Inspektor  $k$  gefunden hat
- $c_n$  ist die Anzahl der verschiedenen Defekte, die das Inspektionsteam insgesamt gefunden hat
- kann alles aus der 0/1-Matrix abgelesen werden

# 0/1-Matrix

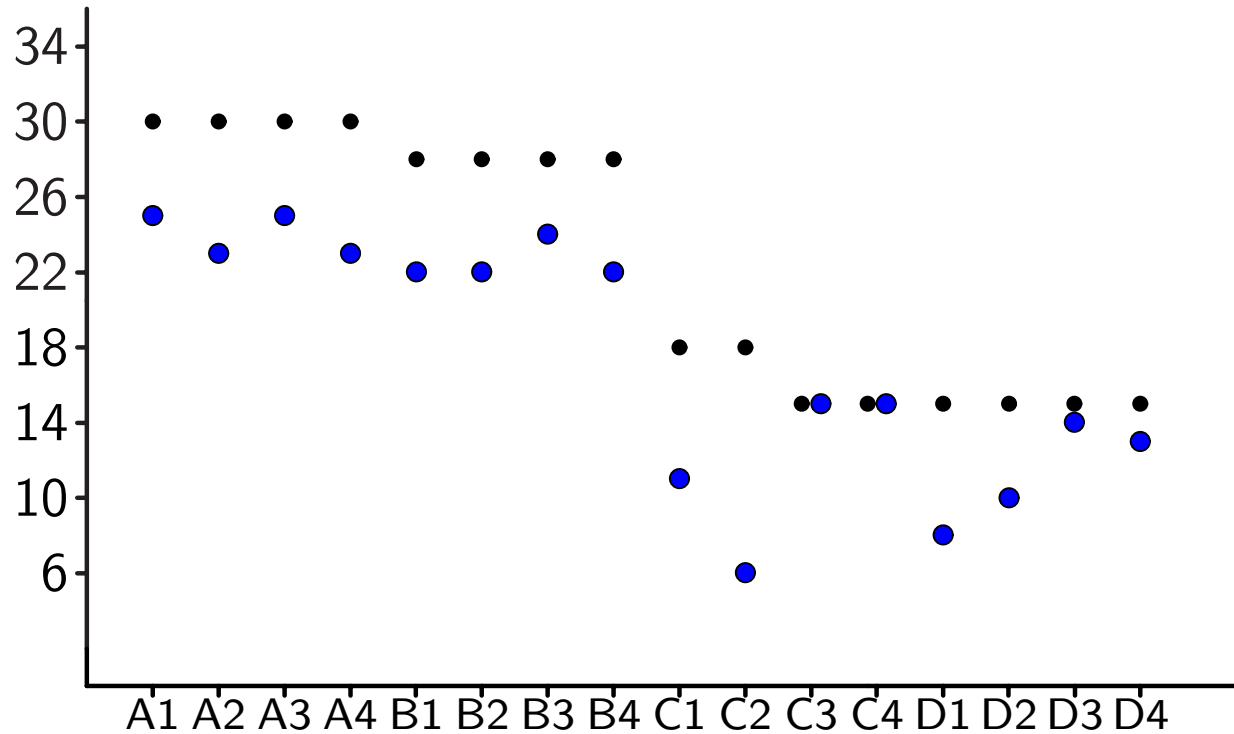
zeigt, welcher Inspektor welchen Defekt gefunden hat

Defekt	Inspektor				
	A	B	C	D	Meet
5:40	1				
6:29			1	1	
9:32		1			
4:17					1

# Empirischer Datensatz

- 16 Inspektionen aus kontrollierten Experimenten bei NASA SEL ( Basili e.a. 1994/95 )
- jeweils 6 bis 8 Inspektoren
- Spezifikationen als untersuchte Dokumente
- wahre Zahl der enthaltenen Fehler genau bekannt
- Datensatz ist “ Standard-Benchmark“

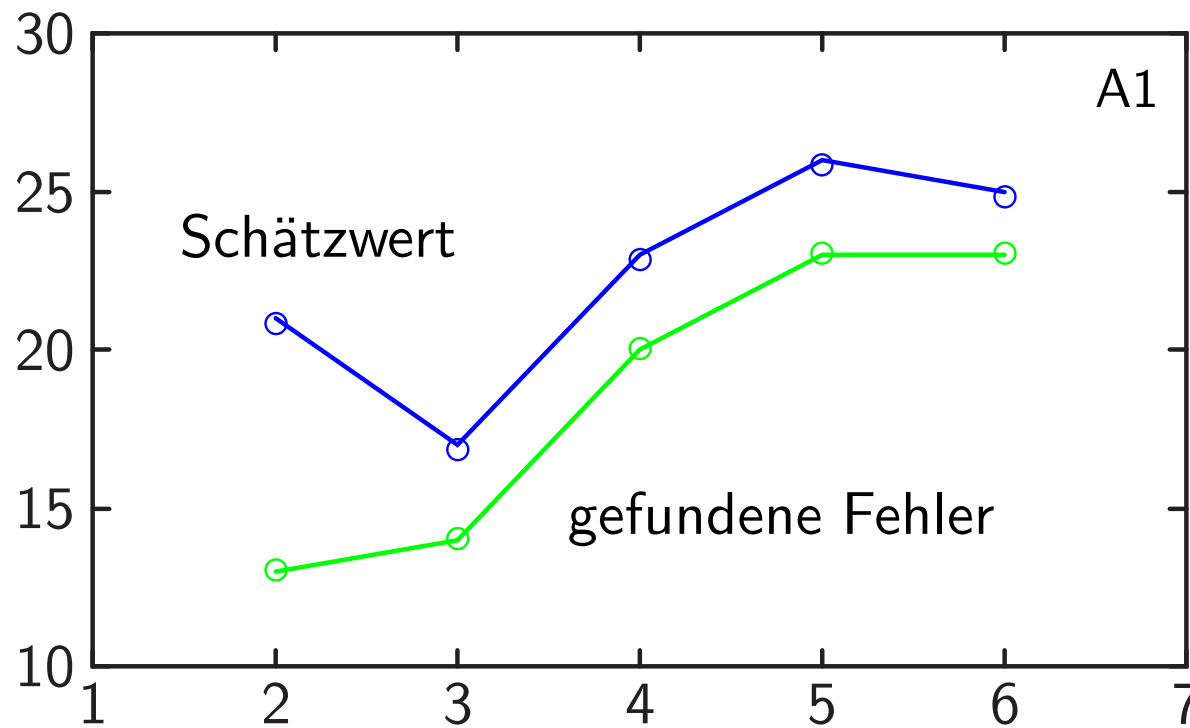
# Schätzung mit Hypergeometrischem Modell



durchschnittlicher Fehler von 24 Prozent;

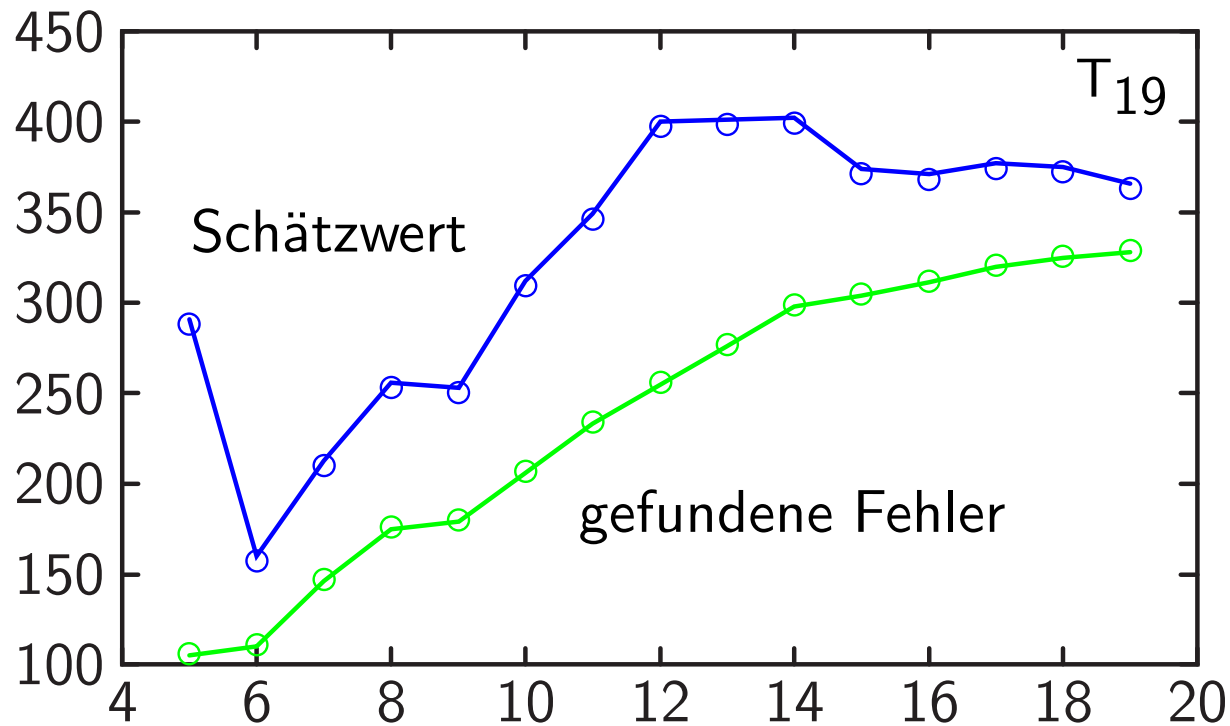
maximaler Fehler von -67 Prozent

# Schätzwert versus Anzahl der Inspektoren



Werte schwanken mit Anzahl der Inspektoren;  
Schätzwert deutlich zu niedrig (25 statt 30)

# Schätzwert versus Länge der Testreihe



Wert "stabilisiert" sich bei längerer Testreihe;  
deutliche Schwankung bei den ersten paar Tests



# Warum Capture–Recapture fehlschlägt

- Mathematik: “Testreihe“ ist zu kurz
- nur das Ergebnis dieser Inspektion geht in die Schätzung ein
- mit anderen Worten: **kein Lernen** aus der Erfahrung

# Neuer Ansatz

- nutze Wissen aus früheren Inspektionen zum Schätzen der Fehlerzahl im neuen Dokument

# Neuer Ansatz

- nutze Wissen aus **früheren** Inspektionen zum Schätzen der Fehlerzahl im neuen Dokument
- “**lerne**“ Zusammenhang zwischen **Merkmale**n einer Inspektion und der **Zahl der** tatsächlich im Dokument enthaltenen **Fehler**

# Neuer Ansatz

- nutze Wissen aus **früheren** Inspektionen zum Schätzen der Fehlerzahl im neuen Dokument
- “**lerne**“ Zusammenhang zwischen **Merkmale**n einer Inspektion und der **Zahl der** tatsächlich im Dokument enthaltenen **Fehler**
- Schätzen der Fehlerzahl als **Regressionsaufgabe**
- Padberg e.a. 2002

# Kandidaten für Merkmale

- abgeleitet aus der 0/1-Matrix
- TDD, AVE, MIN, MAX, STD
- Beispiel Inspektion A1:

( 9 , 7 , 6 , 13 , 9 , 6 ) und 23 ergibt

TDD	AVE	MIN	MAX	STD
23	8.3	6	13	2.4

# Eingabedaten für Lineare Regression

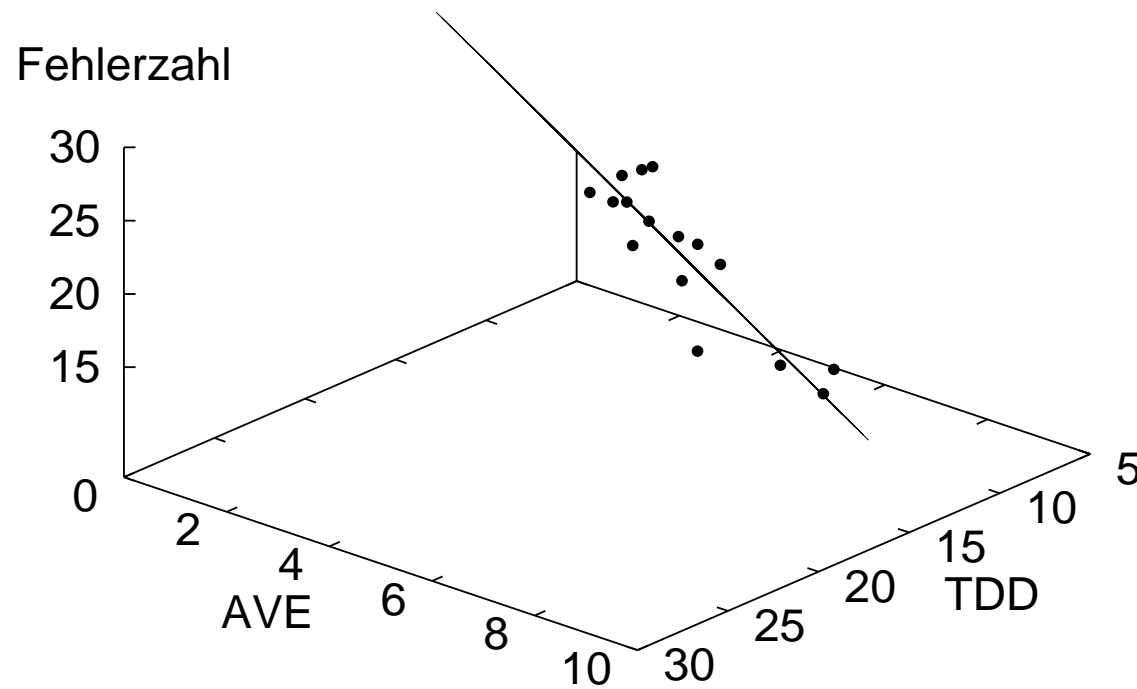
- Korrelationsanalyse ergibt Rangfolge

TDD > AVE > MIN > MAX > STD

- einige Stützpunkte:

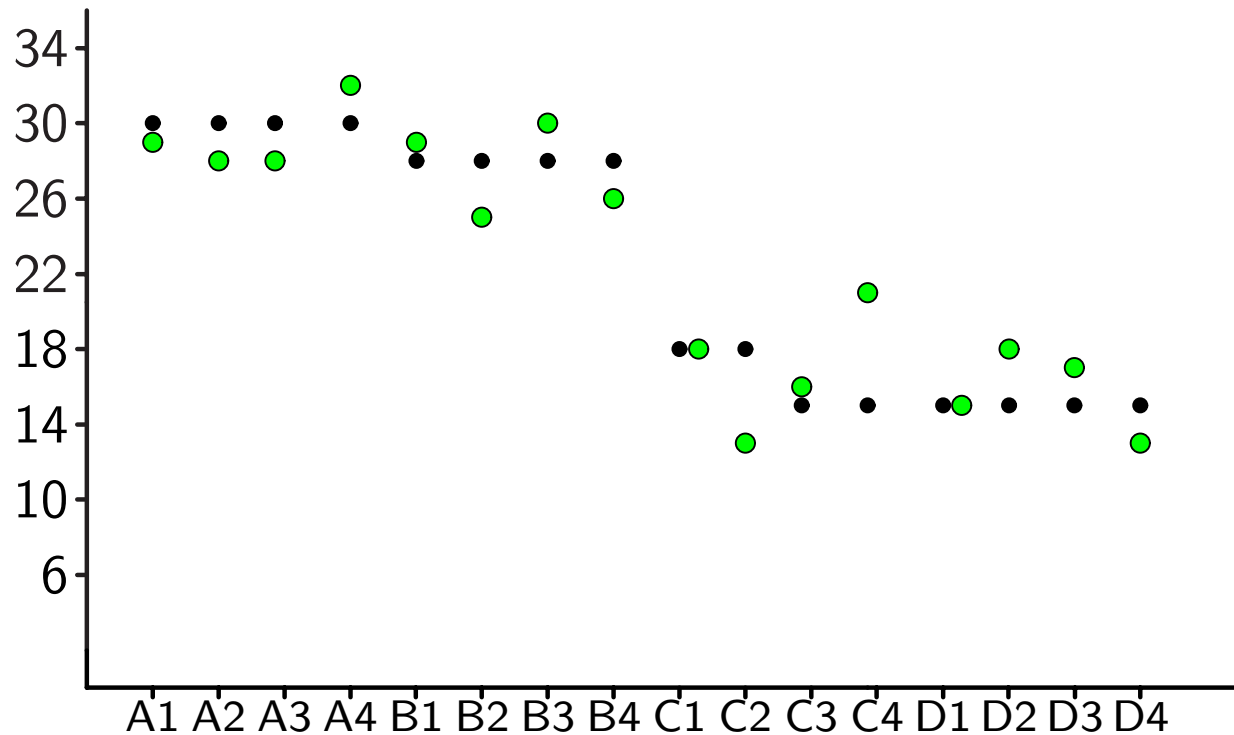
Inspektion	TDD	AVE	Zielwert
A1	23	8.3	30
B1	20	6.0	28
C1	10	3.2	18
D1	6	1.3	15

# Regressionsebene



einige Datenpunkte weichen deutlich von der Ebene ab

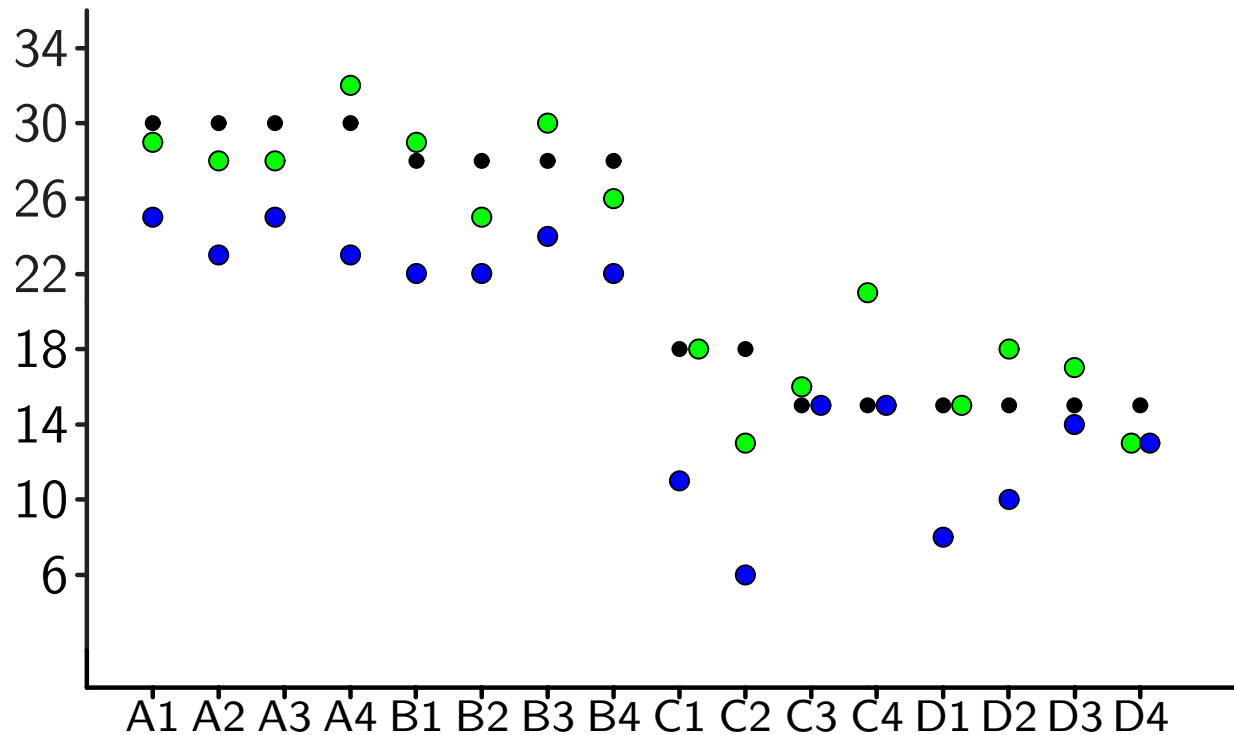
# Schätzwerte mit Linearer Regression



Jackknife-Validierung ( "leave one out" )  
durchschnittlicher Fehler von 11 Prozent;  
maximaler Fehler von 40 Prozent



# Schätzwerte mit Linearer Regression

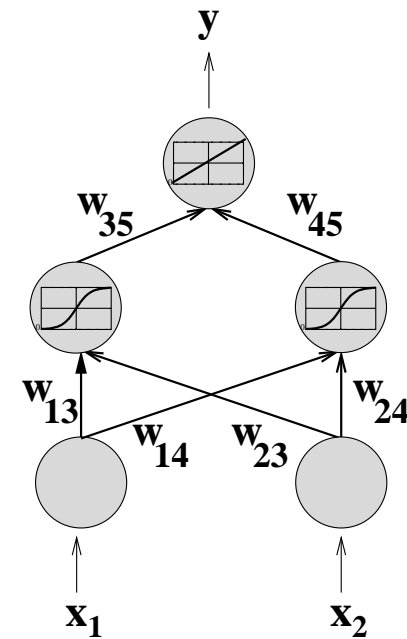
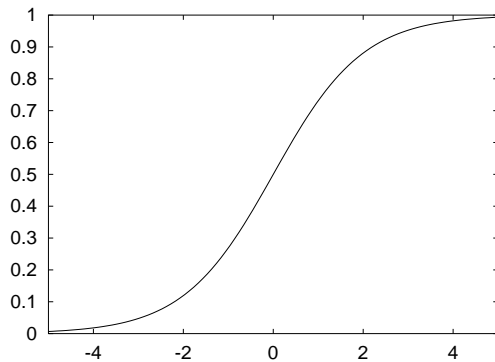


Jackknife-Validierung ( "leave one out" )  
durchschnittlicher Fehler von 11 Prozent;  
maximaler Fehler von 40 Prozent

# Nicht-Lineare Regression: Neuronale Netze

$$\text{logist}(x) = \frac{1}{1 + e^{-x}}$$

$$s_i = \text{logist}\left(\sum_j w_{ji} \cdot s_j\right)$$



# Eingabedaten für Nicht-Lineare Regression

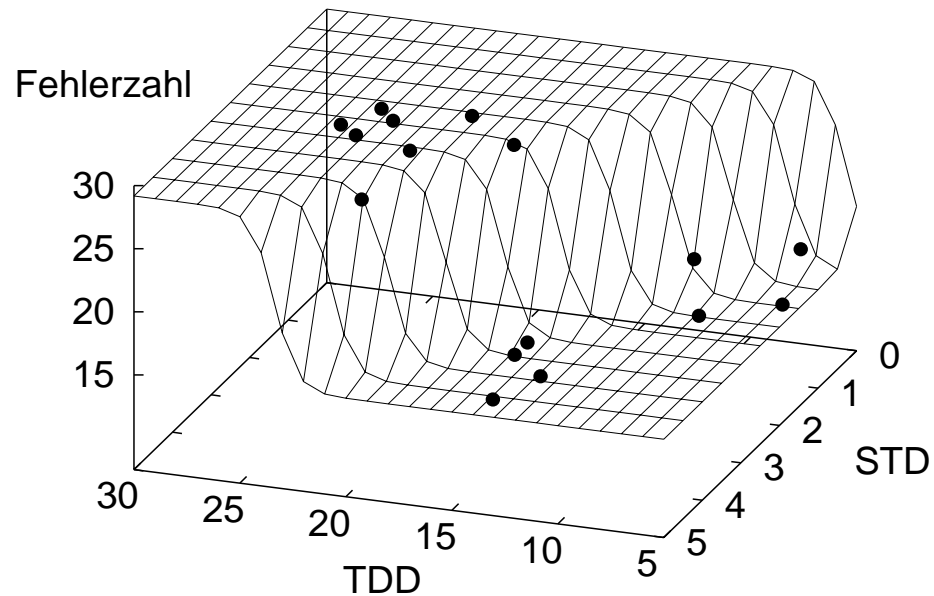
- Informationsgehalt ergibt Rangfolge

TDD > STD > MAX > MIN > AVE

- einige Trainingspunkte:

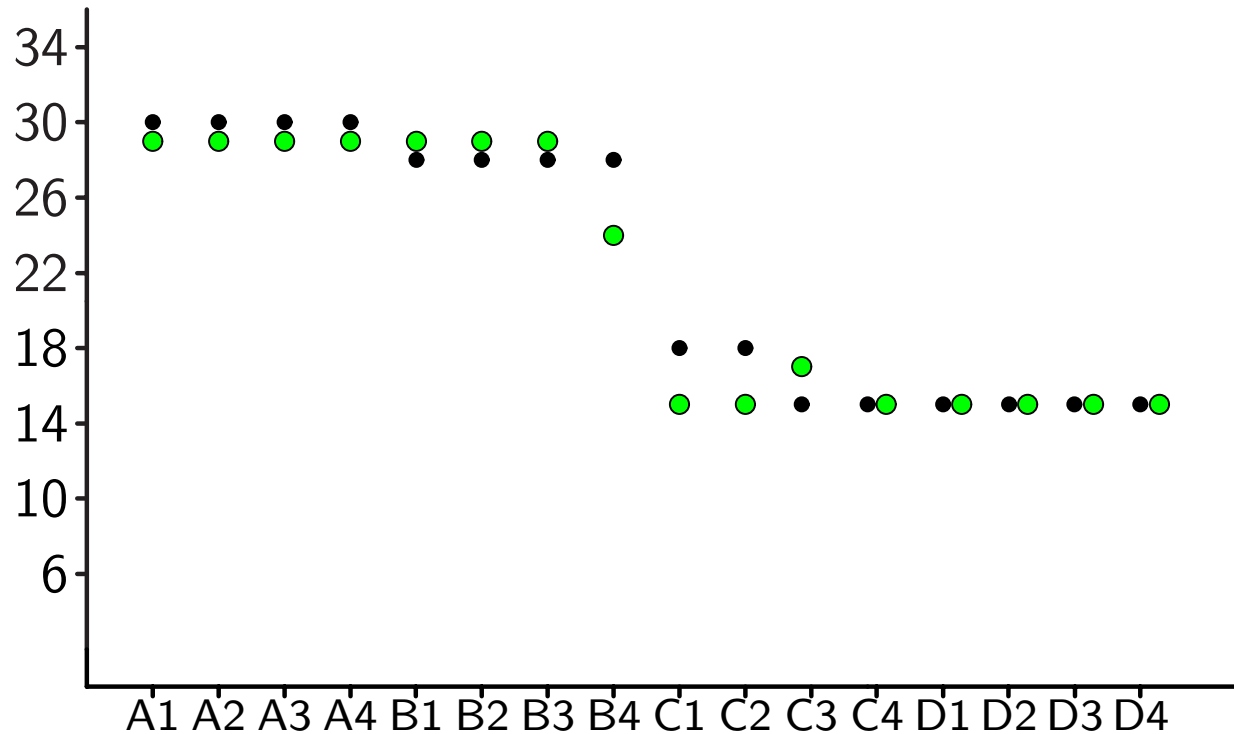
Inspektion	TDD	STD	Zielwert
A1	23	2.4	30
B1	20	1.7	28
C1	10	1.5	18
D1	6	1.4	15

# Nicht-Lineare Regressionsfläche



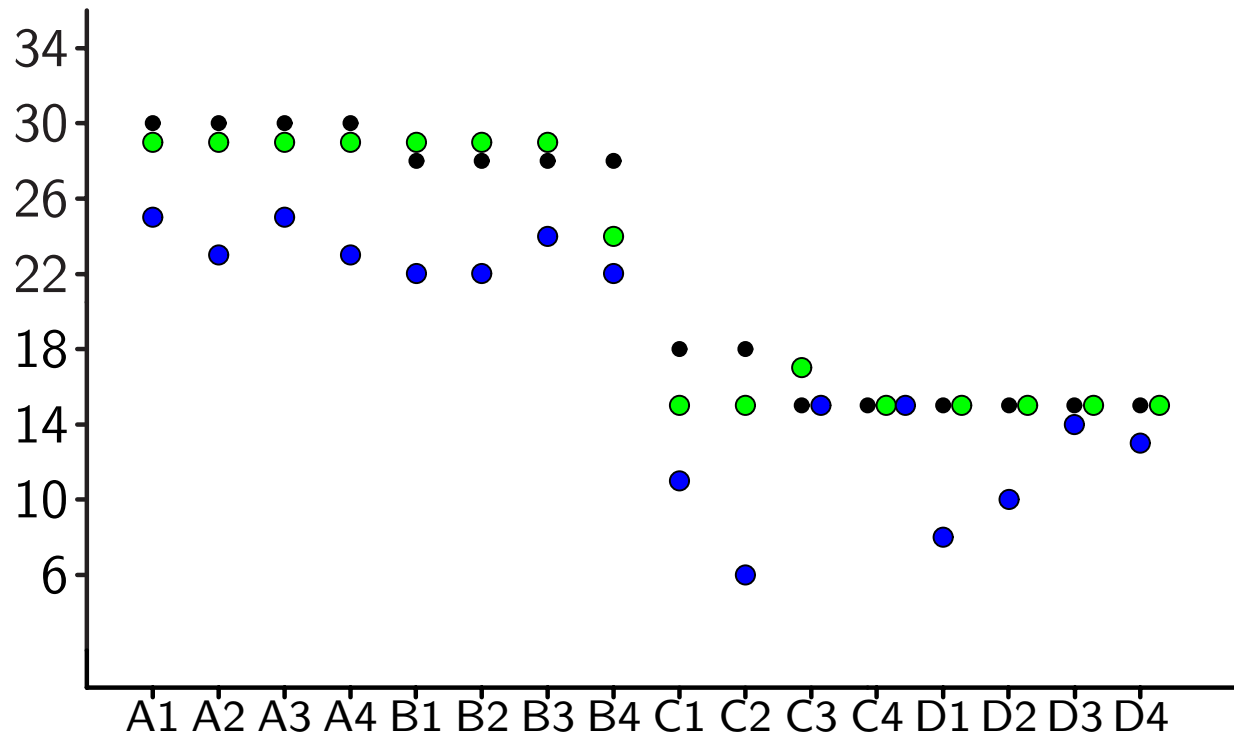
Netz mit zwei versteckten Neuronen;  
Fläche paßt sich den Datenpunkten gut an

# Schätzwerte mit Neuronalen Netzen



Jackknife-Validierung ( "leave one out" )  
durchschnittlicher Fehler von nur 6 Prozent;  
maximaler Fehler von -17 Prozent

# Schätzwerte mit Neuronalen Netzen



Jackknife-Validierung ( "leave one out" )  
durchschnittlicher Fehler von nur 6 Prozent;  
maximaler Fehler von -17 Prozent

# Vergleich mit anderen Schätzverfahren

Methode	mittlerer abs. Fehler	maximaler Fehler
Capture–Recapture	24 %	- 67 %
Detection Profile	36 %	+ 113 %
Lineare Regression	11 %	+ 40 %
Interval Estimates	7 %	+ 14 %
Neuronale Netze	6 %	- 17 %

der Regressionsansatz ist vielversprechend;  
weitere Validierung mit empirischen Daten ist nötig

# Eigene Veröffentlichungen

## zum Fehlerschätzen nach Inspektionen

- *Empirical Interval Estimates for the Defect Content After an Inspection*  
International Conference on Software Engineering ICSE (2002)  
58–68, IEEE Computer Society Press
- *Applying Machine Learning to Solve an Estimation Problem in Software Inspections*  
International Conference on Artificial Neural Networks ICANN  
(2002) 516–521, Springer LNCS 2415  
(mit T. Ragg und R. Schoknecht)



**Vielen Dank!**