2IS55 Software Evolution

Software metrics

Alexander Serebrenik

TU/e Technische Universiteit Eindhoven University of Technology

Where innovation starts

Assignments

- **Assignment 5:**
  - **Deadline: Today!**
  - **“Cathedral” / “bazaar”**

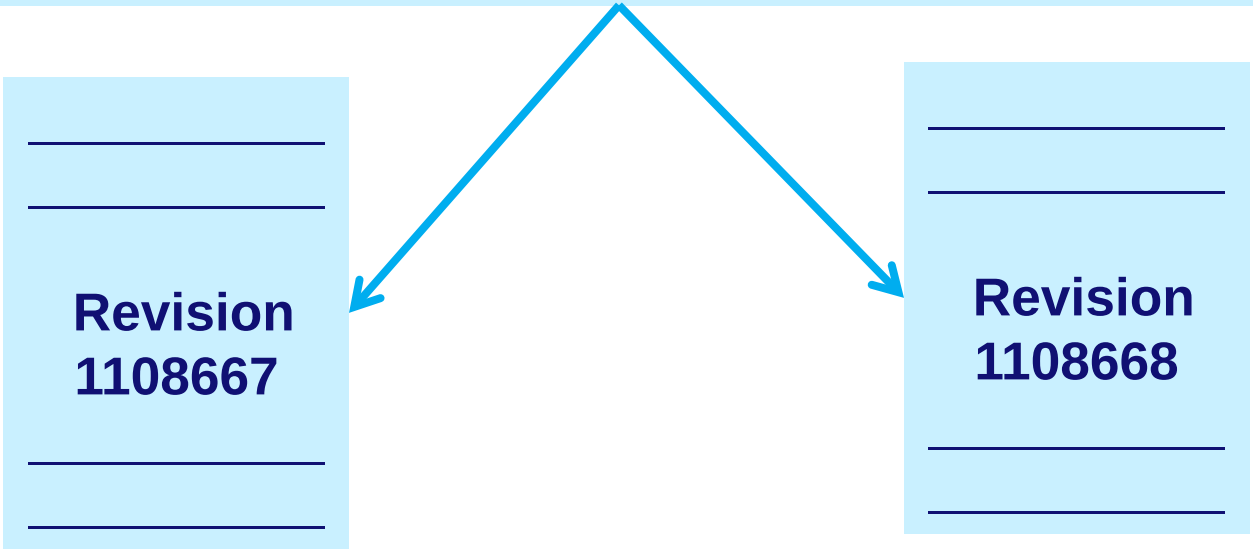
- **Assignment 6:**
  - **Deadline: May 11**
  - **Published on Peach**
  - **1-2 students**

Recap: Version control systems

- **Version control systems**
  - **Centralized vs. distributed**
  - **File versioning (CVS) vs. product versioning**
  - **Logs can be used to get insights in**
    - **How humans work?**
    - **How do the files evolve?**

Today: Version control system is not just a log…

r1108668 | tokoe | 2010-03-29 16:54:02 +0200 (ma, 29 mrt 2010)
Changed paths:
M /trunk/KDE/kdepim/kmail/kmsearchpatternedit.cpp



- **Measure each revision**
- **Get insights in the evolution**

Why do we want to measure revisions?

- **Recall the "goals-questions-<views>-metrics" approach we used for architecture reconstruction?**
  - **Goals: What problem does the measurement try to solve?**
    - **Ex.: Modifying code is experienced as difficult**
    - **Goal: Assess and improve maintainability of the code**
  - **Questions: What do we need to know to achieve the goal?**
    - **Is the code large? Complex? Appropriately modularized? Buggy? Documented?**
  - **<Views>: Which views are need to answer the questions?**
    - **Individual components, dependency structure**
  - **Metrics: How can we quantify the answers?**
    - **Main topic of the lecture**

Measure each revision…

- **Metric:**
  - **“A quantitative measure of the degree to which a system, component, or process possesses a given variable.“ --- IEEE Standard 610.12-1990**
  - **“A software metric is any type of measurement which relates to a software system, process or related documentation.” --- Ian Sommerville, Software Eng. 2006**

  - **Short: mapping of software artefacts to a well-known domain**

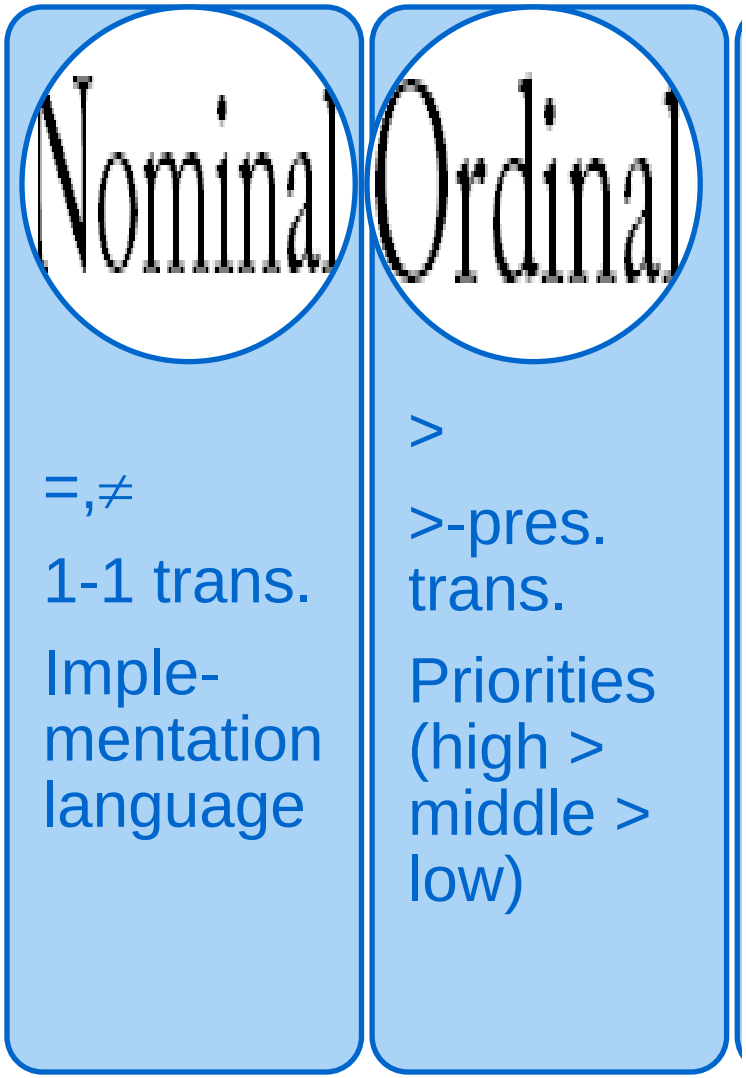
Domains and scales

=,≠

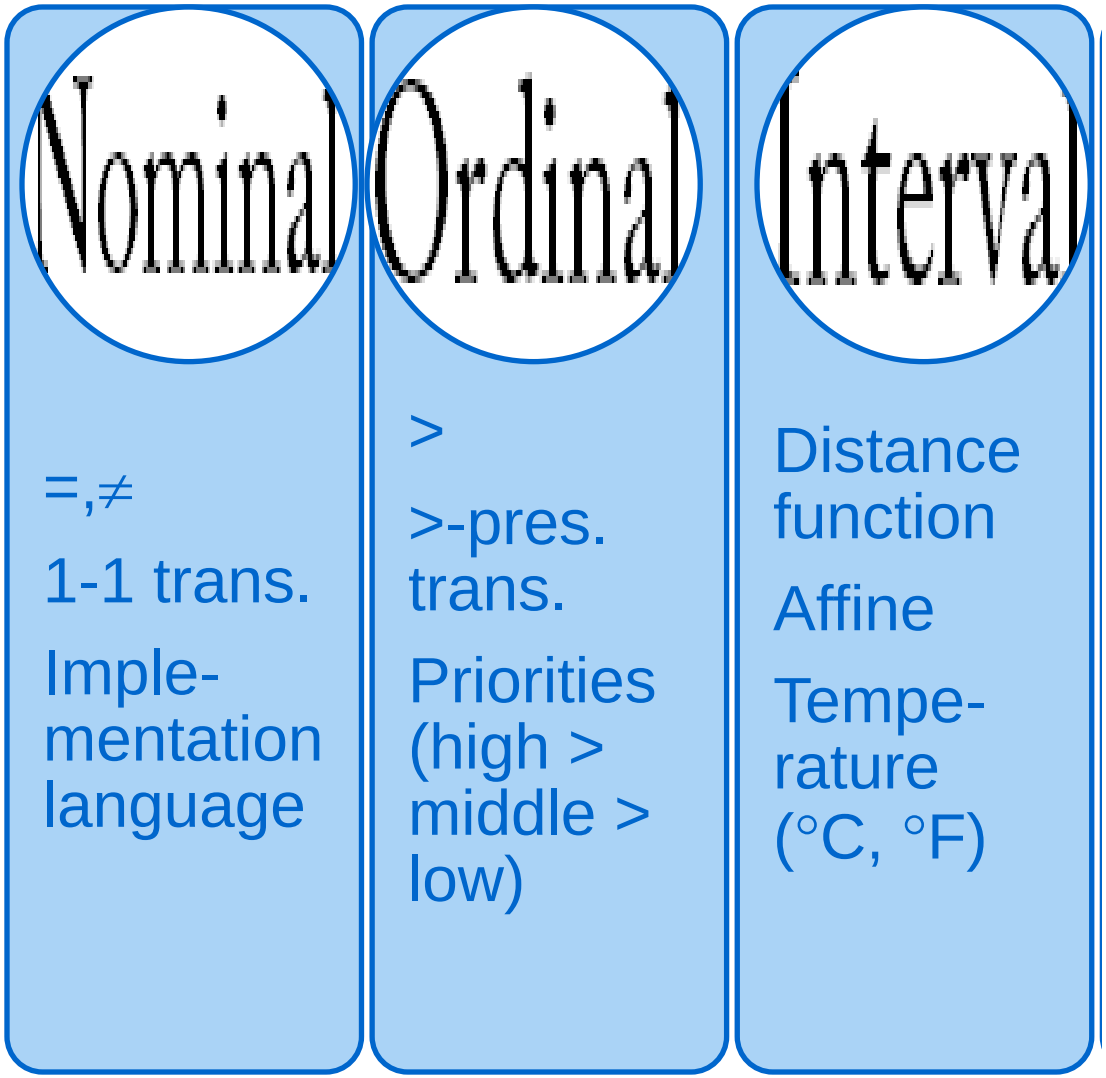
1-1 trans.

Imple-
mentation
language

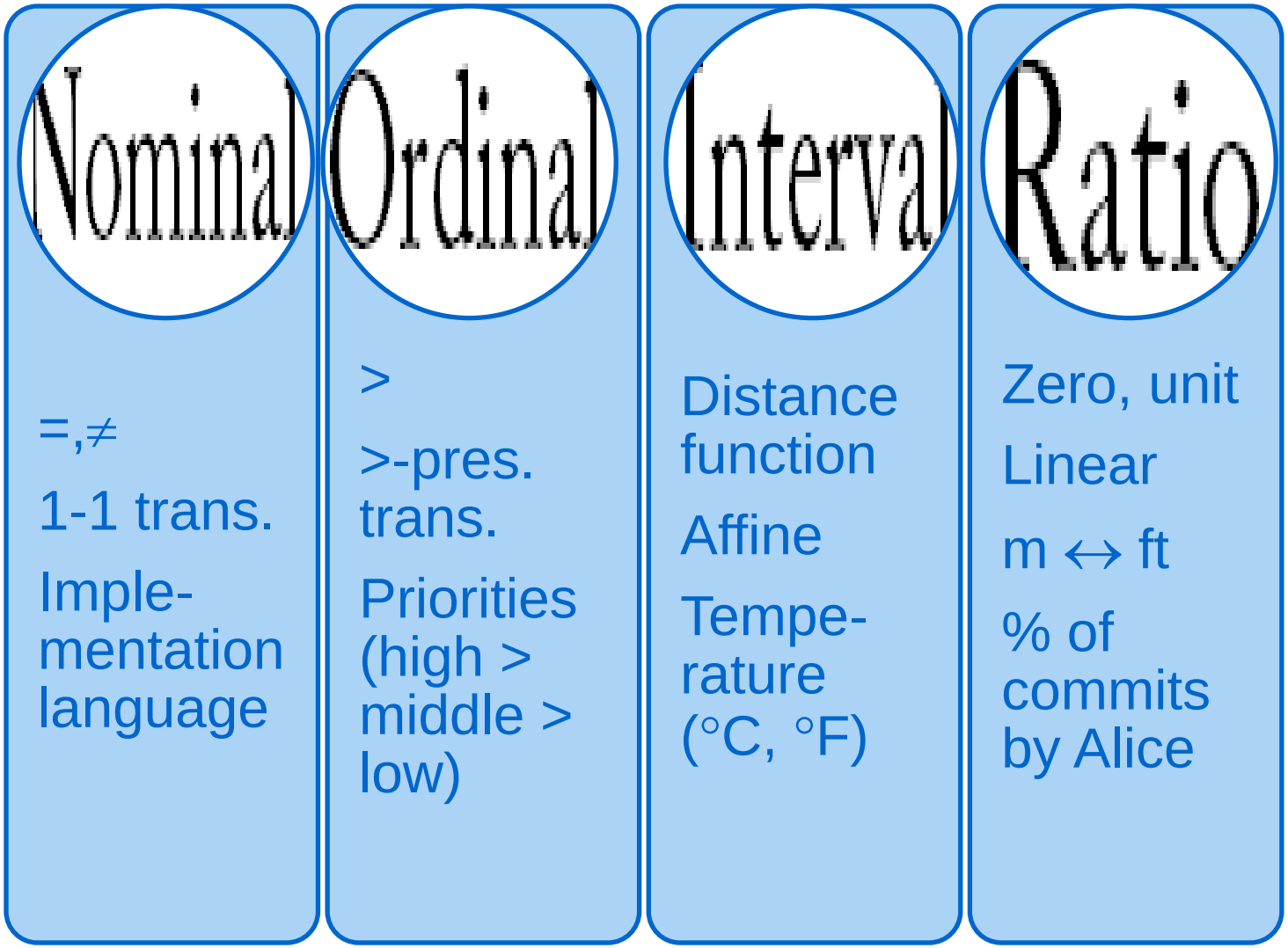
Domains and scales



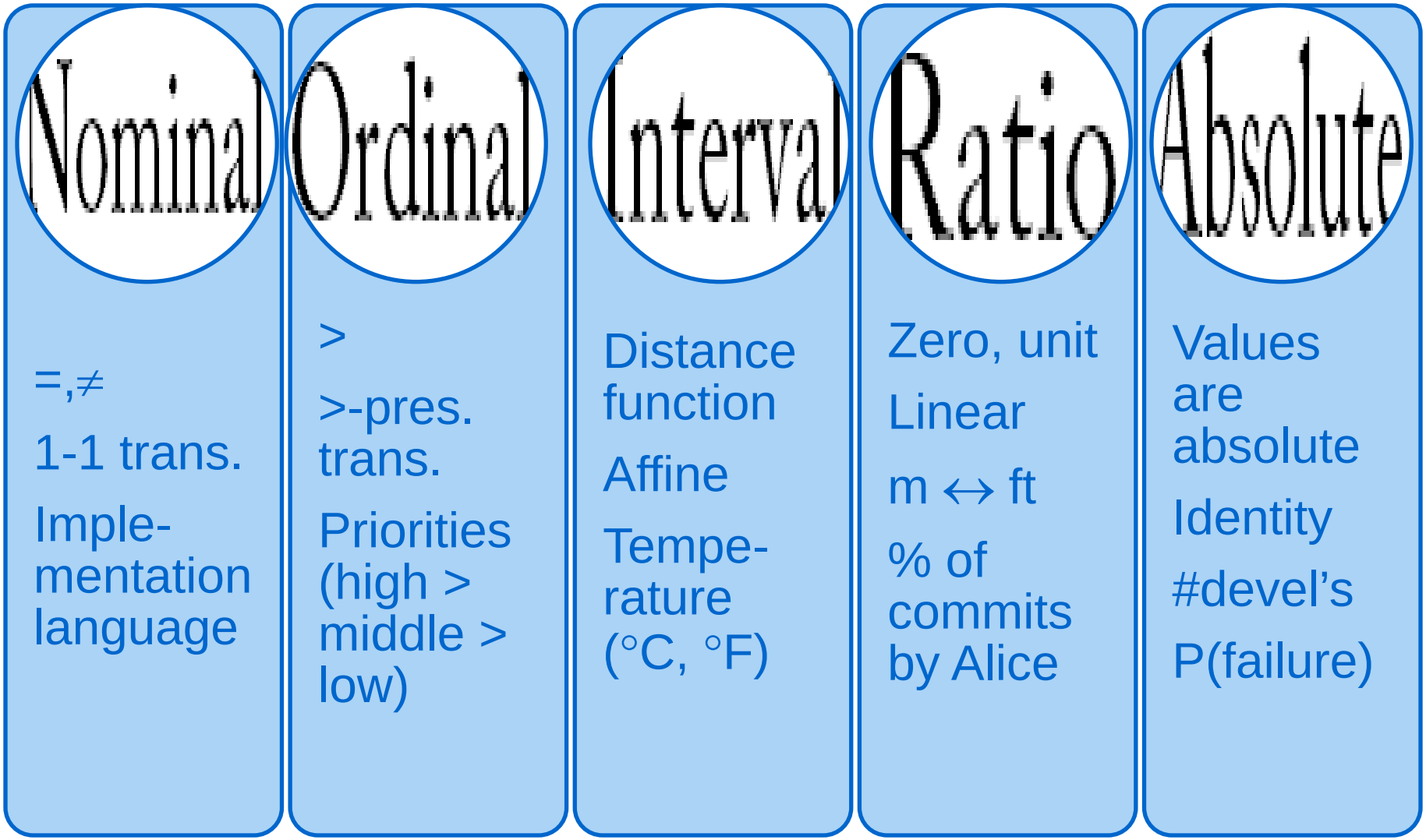
Domains and scales



Domains and scales



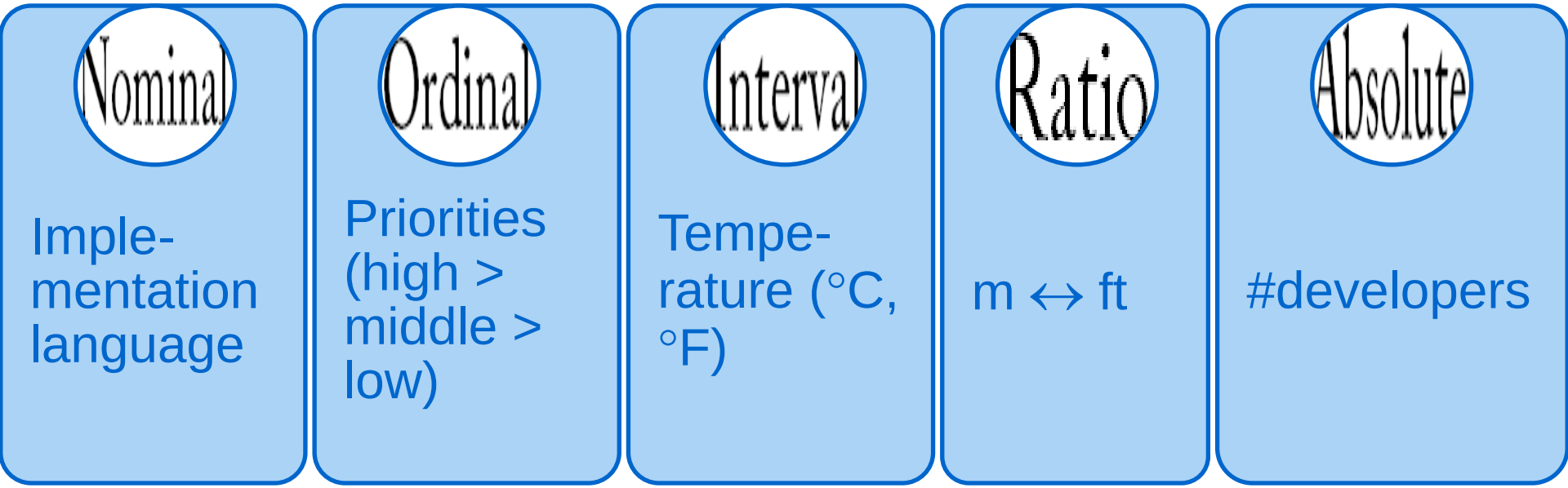
Domains and scales



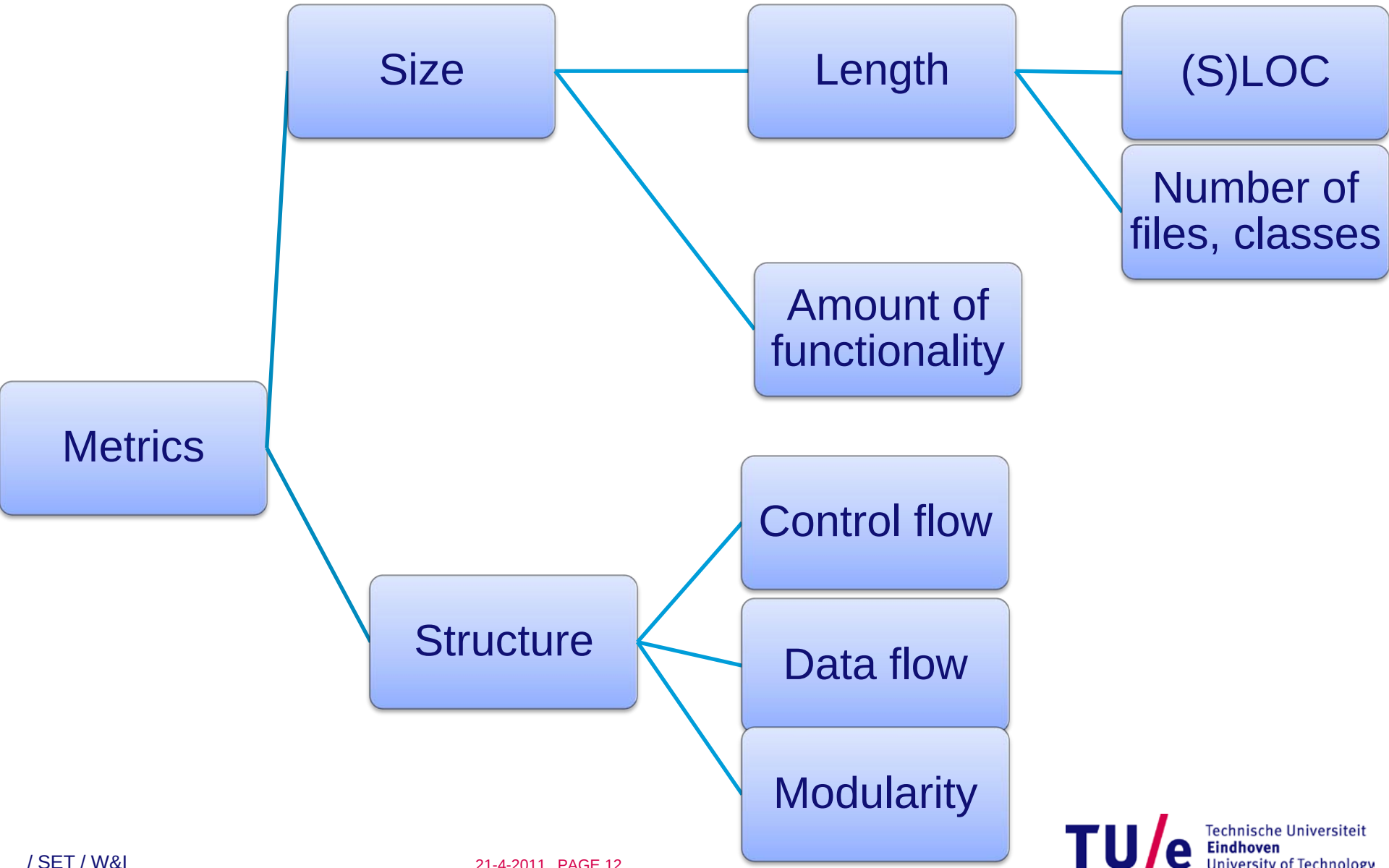
Metrics and scales

- **What metrics have we seen so far?**
  - **Size: LOC, SLOC**
  - **Code duplication: POP, RNR, …**
  - **Requirements: Flesch-Kincaid grade level**

To what scale does it belong?



Classification of metrics [à la Fenton, Pfleeger 1996]



Program length (LOC)

- **Variants:**
  - **Total**
  - **Non-blank**
  - **SLOC (source LOC): Ignore comments and blank lines**
  - **LLOC (logical LOC): Number of program statements**

```
1   for (i = 0;
2            i < 100;
3                i += 1) {
4        printf("hello");
5   }
6
7   /* An important loop */
```

Total LOC: 7
Non-blank LOC: 6
SLOC: 5
LLOC: 2 (for and printf)

Advantages of (S)LOC

- Related to Lehman's law of "continuous growth" (Law 6)
- Easy to calculate
  - LLOC is more difficult to determine (parser needed)
  - What happens with nested statements? for(i=0;i<10;i++)?
- Correlation with the #bugs
  - Moderate (0.4-0.5) [Rosenberg 1997, Zhang 2009]
  - Larger modules usually have more bugs
    - "Ranking ability of LOC" [Fenton and Ohlsson 2000 , Zhang 2009]
  - There are better (but more complex) ways to predict #bugs
  - Can be used to predict the development effort!

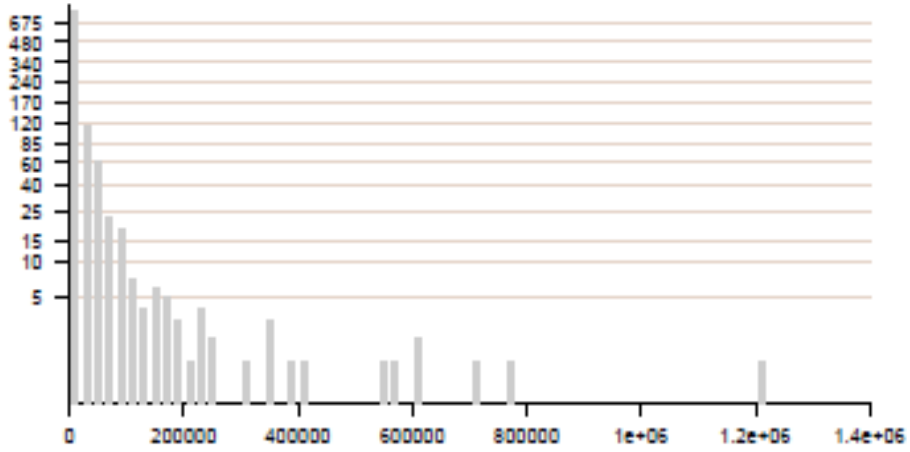
Disadvantages of (S)LOC

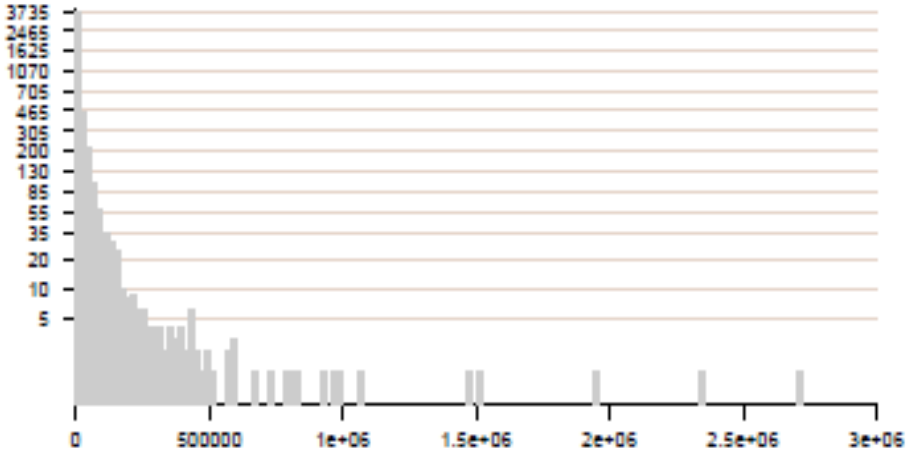
- Ignores structure of the program
  - Program code is more than just text!

- Difficult to compare modules in different languages or written by different developers
  - Some languages are more verbose due to
    - Presence/absence of “built-in” functionality
    - Structural verbosity (e.g., .h in C)
  - Some developers are paid per LOC!
  - Hand-written vs. generated code

(S)LOC distribution

Robles et al. 2006





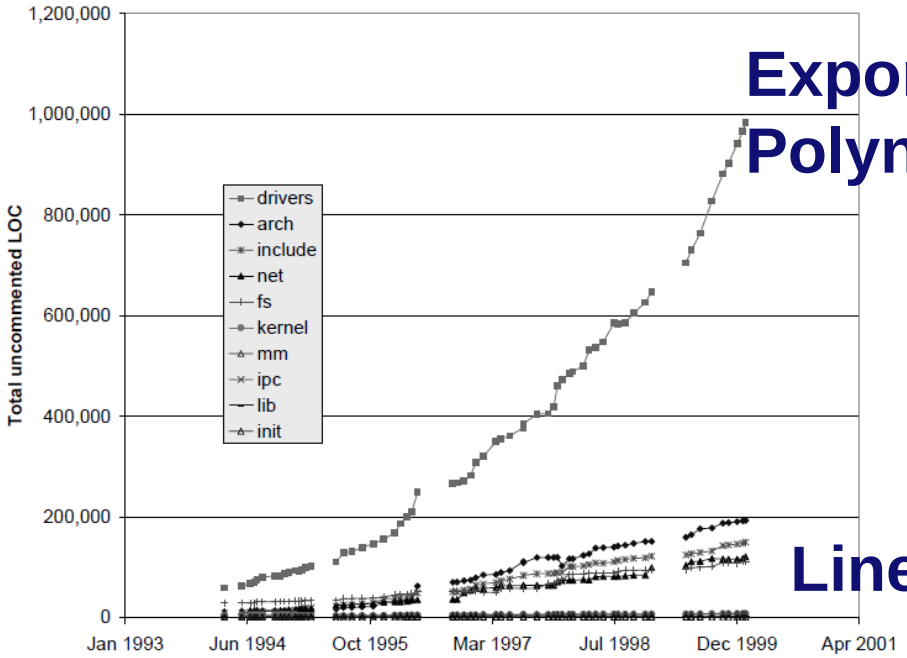
- **Distribution of SLOC in Debian 2.0 (left) and 3.0 (right)**
- **Controversy: log-normal or double Pareto?**
  - **Importance: knowing distribution one can estimate the probability to obtain files of a given size**
  - **Hence, to estimate size of the entire system**
  - **And the effort required (size ⇒ effort)**

What do we know about evolution of SLOC?

- **Related to Lehman's 6:**
  - **The functional capability <…> must be continually enhanced to maintain user satisfaction over system lifetime.**
  - **Earlier versions: "size".**

- **Also related to Lehman's 5:**
  - **In general, the incremental growth (growth rate trend) of E-type systems is constrained by the need to maintain familiarity.**
  - **Lehman interpreted this as linear growth**

What do we know about evolution of SLOC?



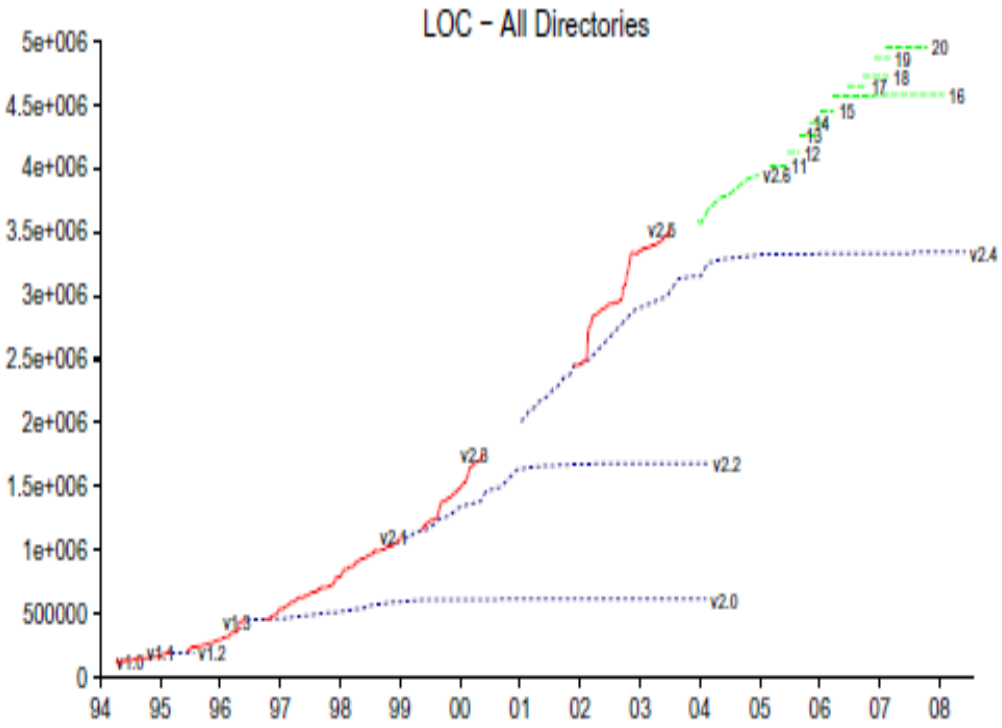
Exponential?

Polynomial?

Linear

- **Godfrey and Tu: superlinear growth is typical for OS**
  - **Does this remind you of Cathedral and Bazaar?**
  - **Koch 2007: Quadratic growth is better for larger OS projects (study of 8621 OS projects on SourceForge)**

LOC in Linux kernel



Superlinear up to 2.5, linear for 2.6

Scacchi – mix of superlinear and sublinear

Israeli, Feitelson:

- **Linux kernel**
- **Multiple versions and variants**
  - **Production (blue dashed)**
  - **Development (red)**
  - **Current 2.6 (green)**

(S)LOC: Summary

- Different variants: LOC, SLOC, LLOC
- Advantages:
  - Easy to compute, moderately correlates with #bugs
  - Can be used to estimate the development effort (more details in two weeks)
- Disadvantages
  - Different programming languages and developers
  - Hand-written vs. generated code
- Distribution “exponential-like”
- Evolution:
  - Linear
  - Linux (other OS?): Superlinear
  - Mix

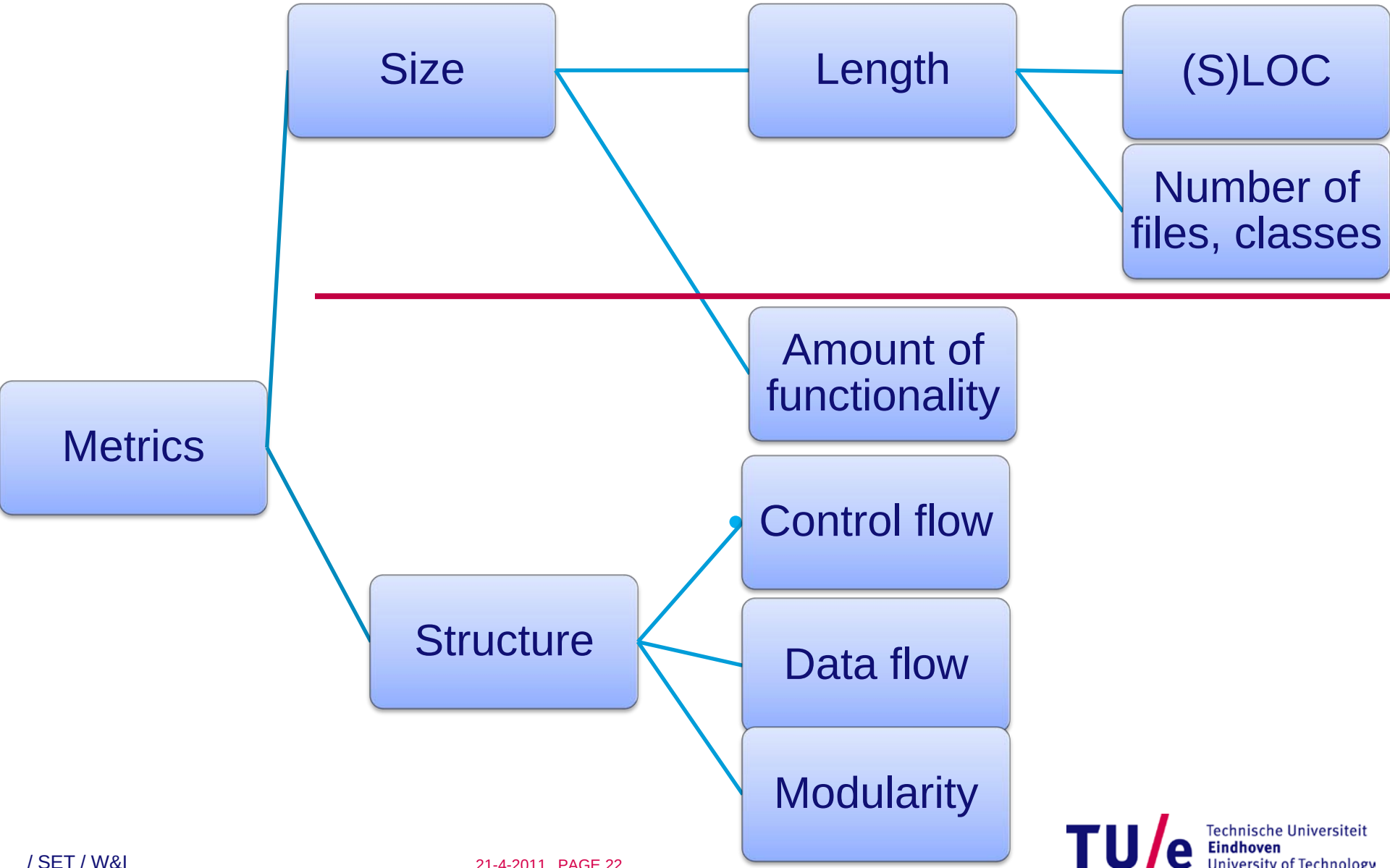
Length: #components

- Number of files, classes, packages
- Intuitive: “number of volumes in an encyclopaedia”

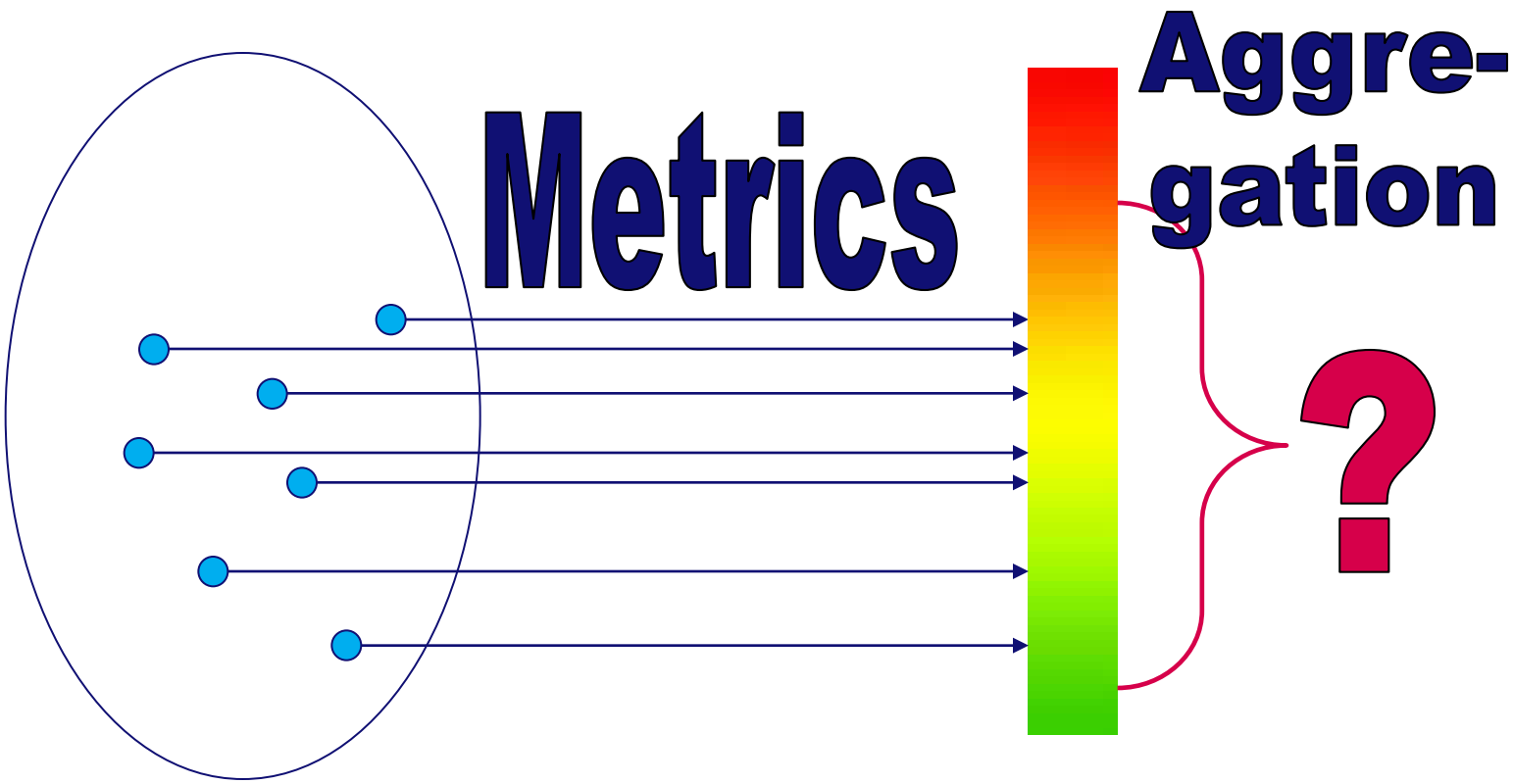
- Variants:
  - All files, classes, packages
  - No empty/library/third-party files, classes, packages
  - No nested/inner classes
  - No or only some auxiliary files (makefiles, header files)

- Correlation with the #post-release defects [Nagappan, Ball, Zeller 2006]
  - significant for modules A, B, C (strength:0.5-0.7), insignificant for modules D, E
  - for each module correlation with some other metrics!

So far


Size
Length
(S)LOC
Number of files, classes
Amount of functionality
Metrics
Control flow
Structure
Data flow
Modularity

Metrics for higher-level objects as aggregation of metrics for low-level objects



Aggregation techniques

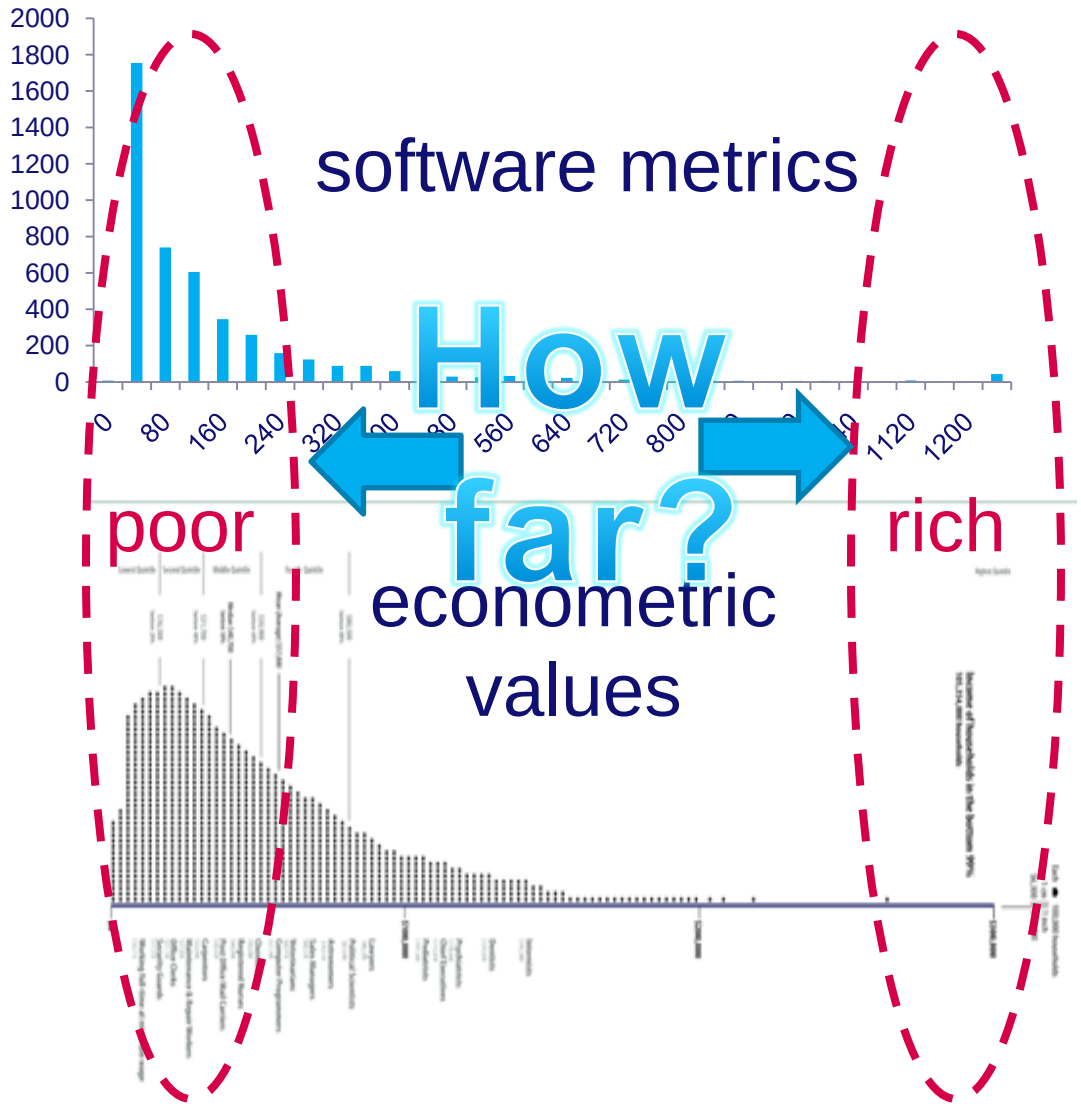
- **Metrics-independent**
  - **Applicable for any metrics to be aggregated**
    - **Traditional: sum, mean, median...**
    - **Econometric: inequality indices**

- **Metrics-dependent**
  - **Produces more precise results**
  - **BUT: needs to be redone for any new metrics**
  - **Based on fitting probability distributions**

Metrics independent: Coefficient of variation

- **Coefficient of variation: $C = \sigma/\mu$**
  - **Allows to compare distributions with different means**
  - **Sometimes used to assess stability of the metrics**
    - **Metrics is stable for $C < 0.3$**
    - **Unreliable for small samples**
    - **Evolution should be studied…**

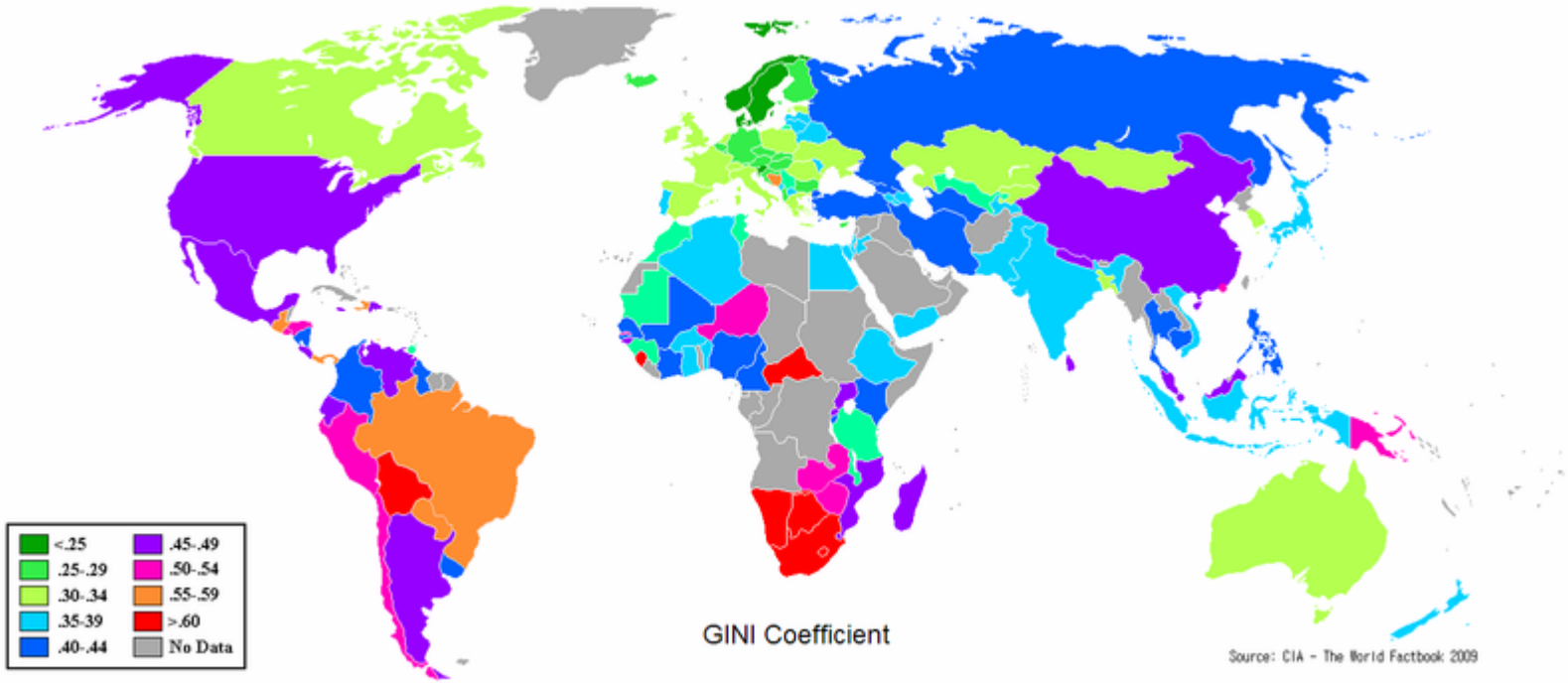
Metrics are like money



- prog. lang.
- domain
- …

- region
- education
- gender
- …

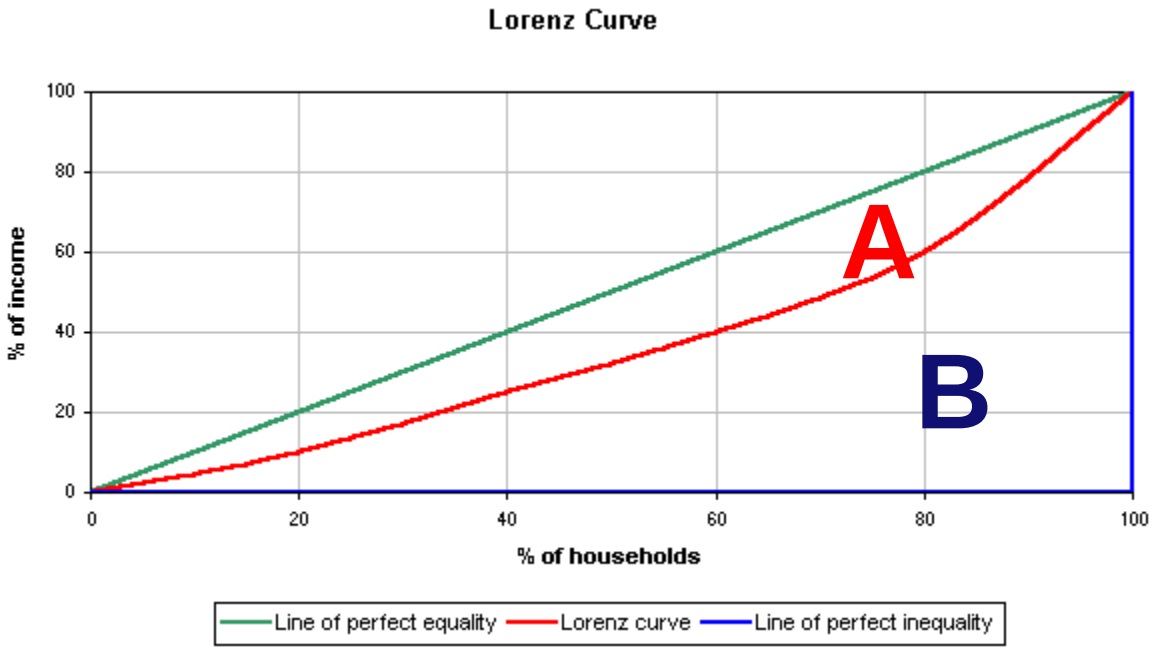
Popular technique: Gini coefficient



- **Gini coefficient measure of economic inequality**
- **Ranges on [0; 1]**
- **High values indicate high inequality**

Gini coefficient: Formally

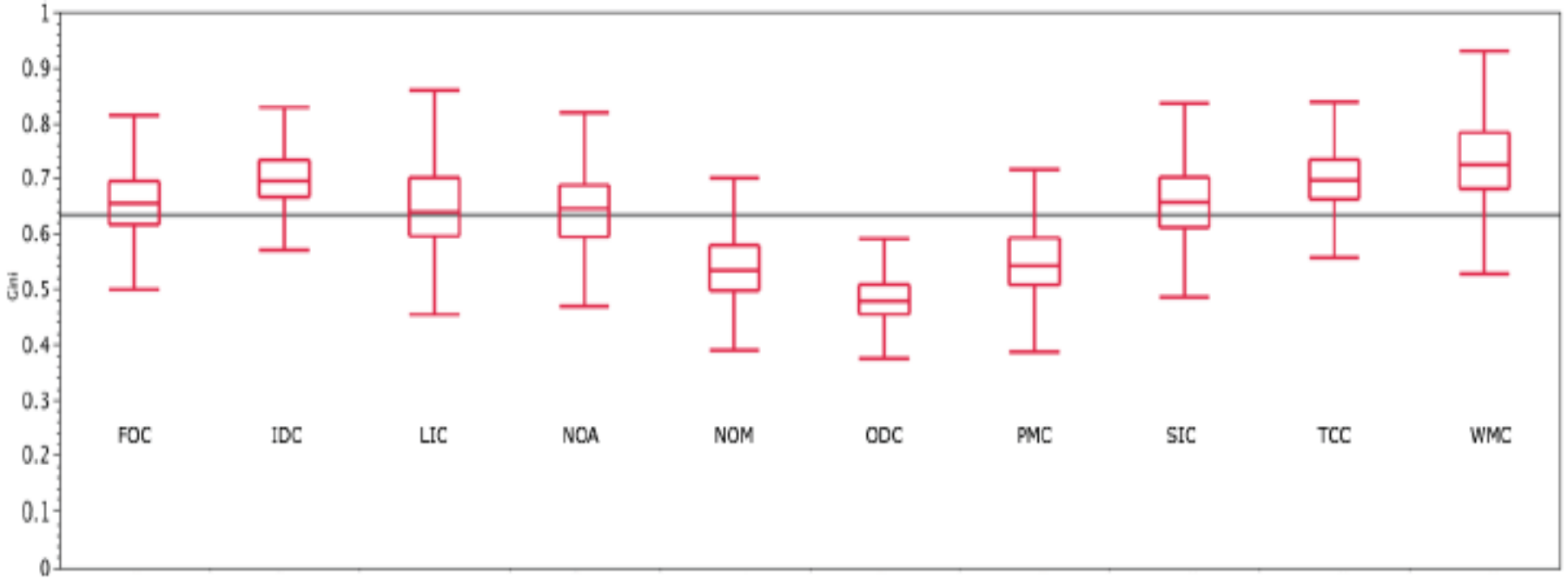
- **Lorenz curve:**
  - **% of income shared by the lower % of the population**



- **Gini = A/(A+B)**
- **Since A+B = 0.5**

  Gini = 2A

Gini and software metrics [Vasa et al. 2009]

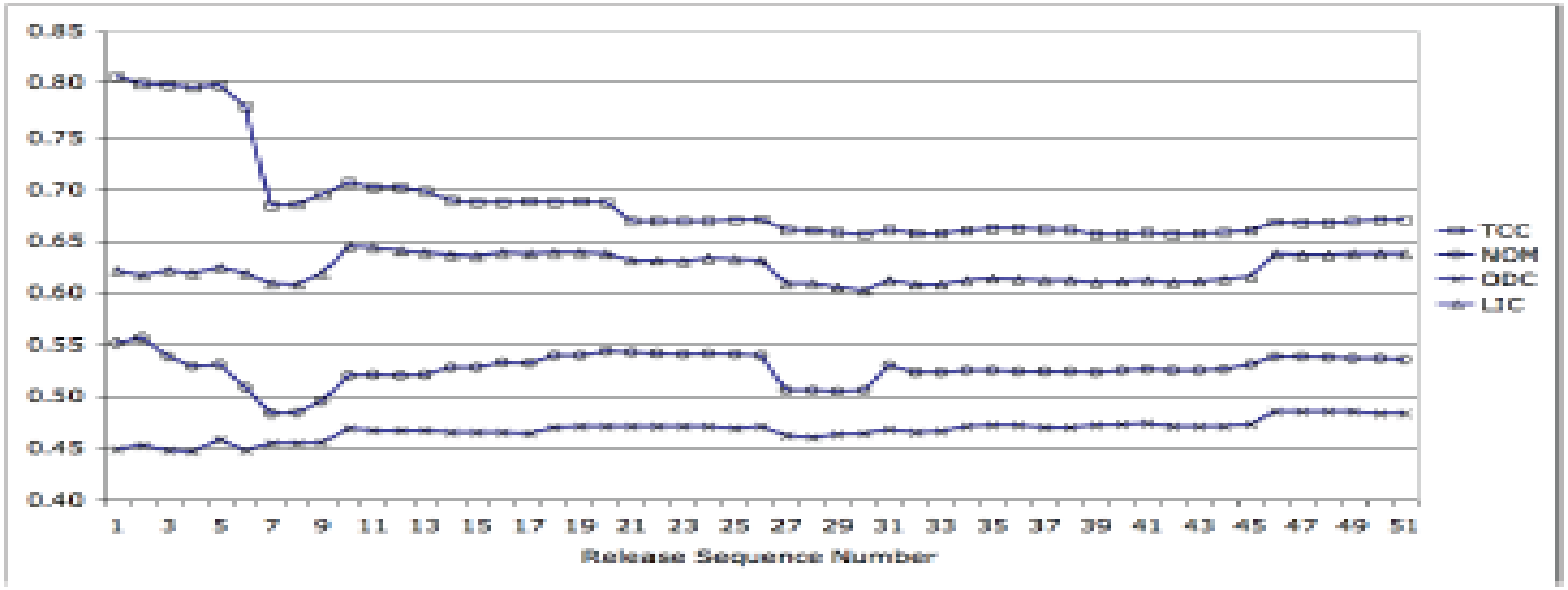


- **For most of the metrics on the benchmark systems: 0.45 ≤ Gini ≤ 0.75**
- **Higher Gini/WMC: presence of generated code or code, structured in a way similar to the generated code (parsers)**

Gini and metrics: Exceptions

System	Metrics	Increase		Explanation
JabRef	WMC	0.75	0.91	Machine generated parser introduced
Checkstyle	Fan-in (classes)	0.44	0.80	Plug-in based architecture introduced.
Jasper-Reports	#Public methods	0.58	0.69	Introduction of a set of new base classes.
WebWork	Fan-out	0.51	0.62	A large utility class and multiple cases of copy-and paste introduced.

Gini and evolution: Spring



- **Rather stable: programmers accumulate competence and tend to solve similar problems by similar means**
- **Similar for other econometric techniques: Theil, Hoover, Atkinson, ...**

Aggregation techniques

- **Metrics-independent**
  - **Applicable for any metrics to be aggregated**
  - **Are the results also metrics-independent?**
  - **Based on econometrics**

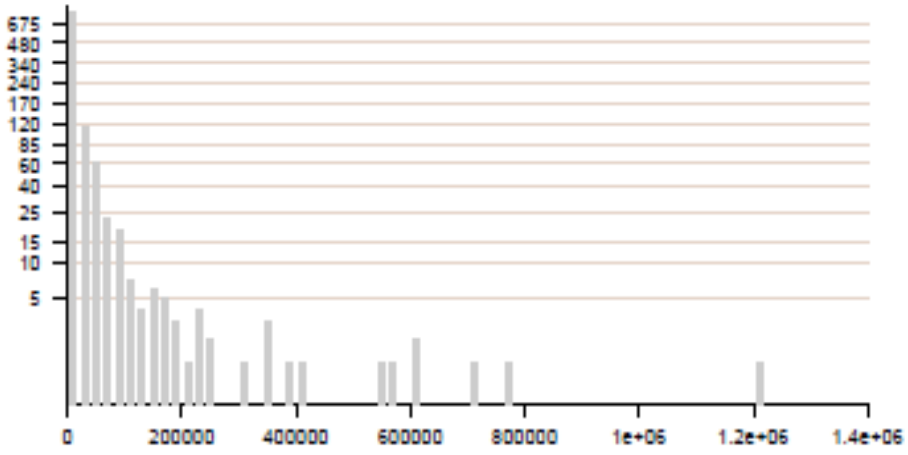
- **Metrics-dependent**
  - **Produces more precise results**
  - **BUT: needs to be redone for any new metrics**
  - **Based on fitting probability distributions**

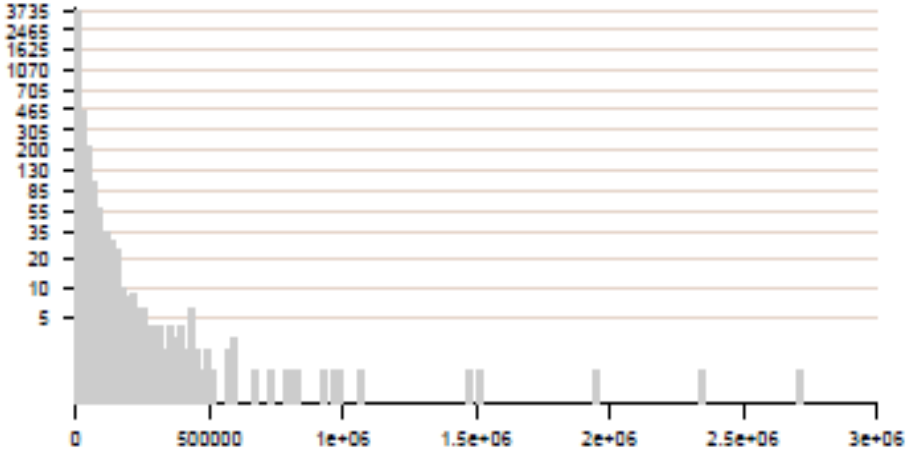
Metrics-dependent aggregation: Statistical fitting

1. Collect the metrics values for the lower-level elements
2. Present a histogram
3. Fit a (theoretical) probability distribution to describe the sample distribution
   a) Select a family of theoretical distributions
   b) Fit the parameters of the probability distribution
   c) Assess the goodness of fit
4. If a theoretical distribution can be fitted, use the fitted parameters as the aggregated value

Step 1: Histograms

- **We have seen quite a number of them already!**

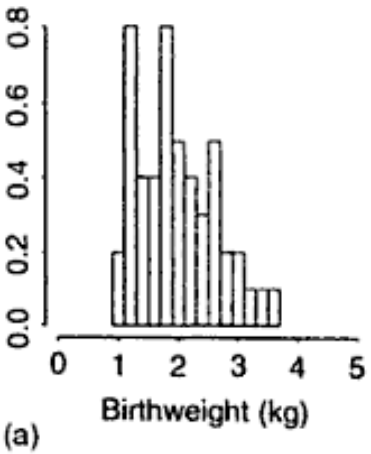


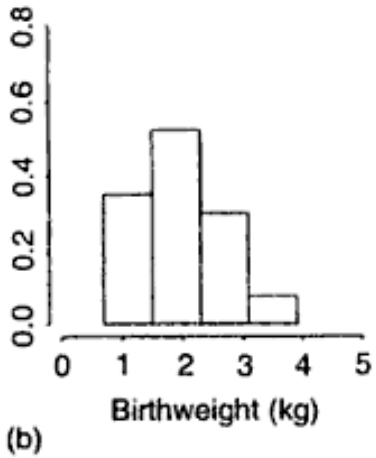


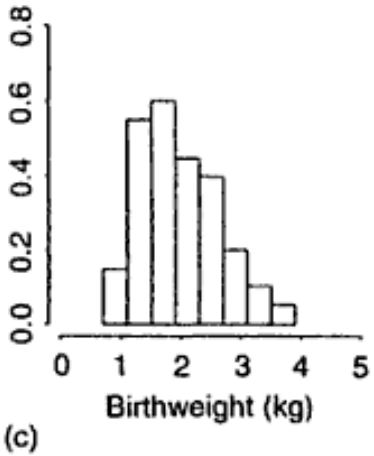
Robles et al. 2006: LOC in Debian 2.0 (left) and 3.0 (right)

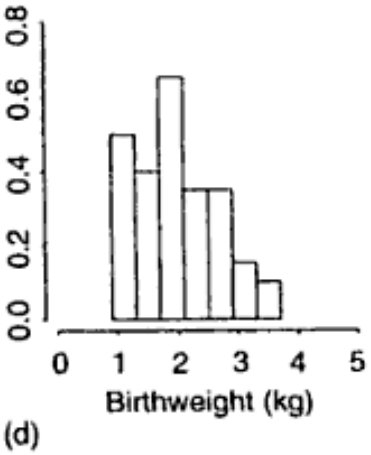
Histograms are not without problems

- **Data: 50 birth weights of children with a severe idiopathic respiratory syndrome**









- **The same data leads to four different "distributions"**
- **What can affect the way histogram looks like?**
  - **Bin width**
  - **Position of the bin's edges**

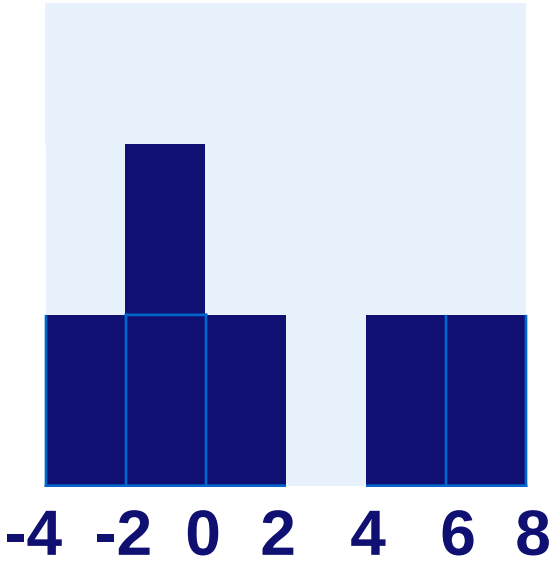
Kernel density estimators

- **Advantages**
  - **Statistically more sound (no dependency on the end-points of the bins)**
  - **Produces smooth curves**

- **Disadvantages**
  - **Statistically more complex**
  - **Parameter tuning might be a challenge**

Kernel density estimates: Intuition

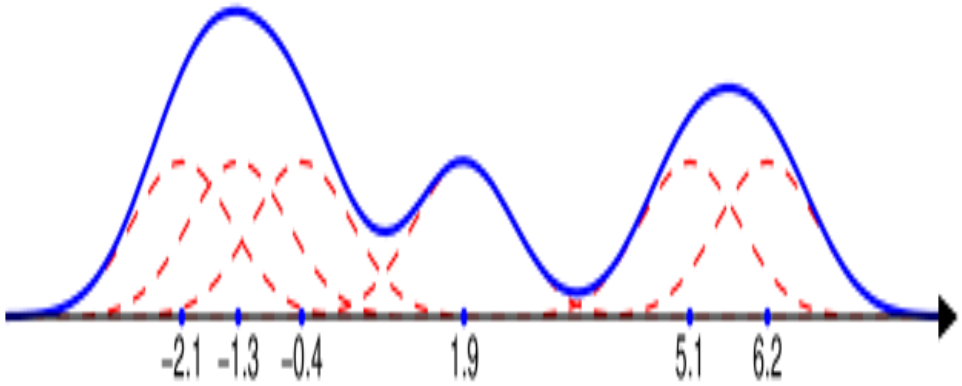
- **Data: -2.1, -1.3, -0.4, 1.9, 5.1, 6.2**



Histogram: every value is a rectangle.

Shape is a "sum" of the rectangles.

What if each value will be a "bump" that can be added together to create a smooth curve?

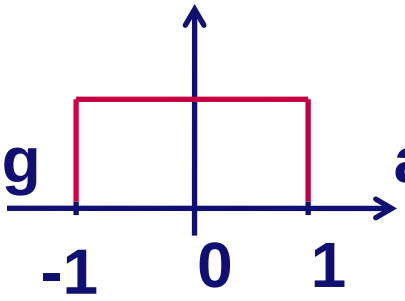


Kernel density estimation: Formally

$$f(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h} \right)$$

Where

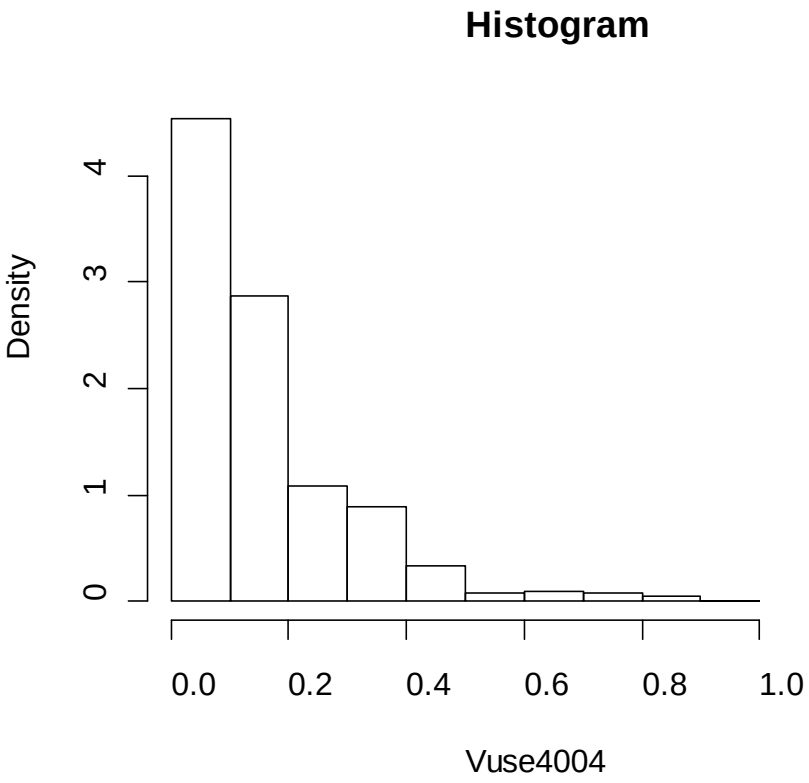
- **n – number of observations**
- **h – a smoothing parameter, the "bandwidth"**
- **K – a weighting function, the "kernel"**

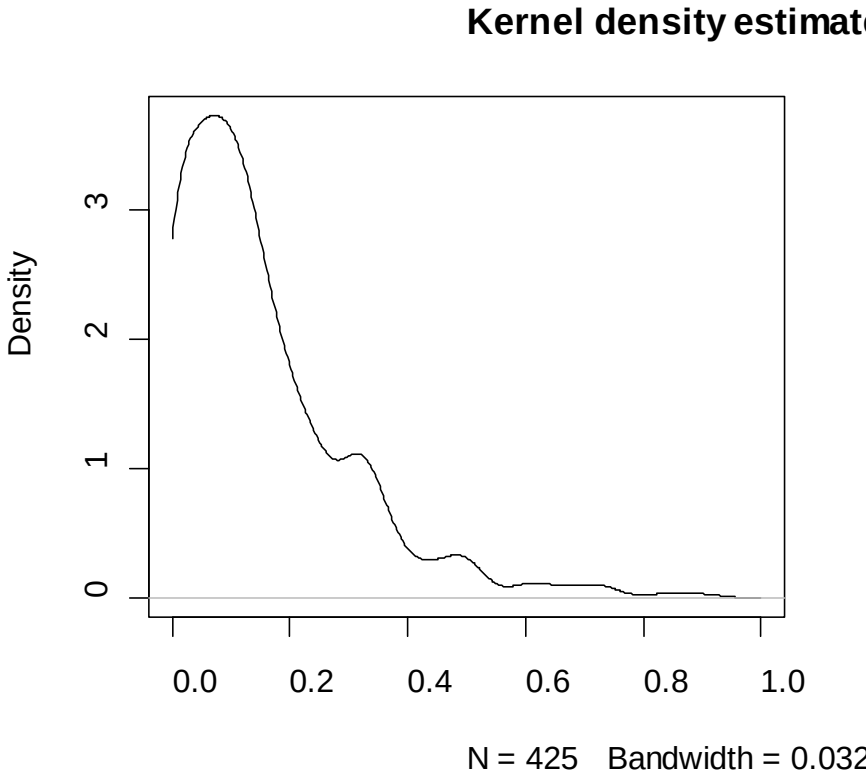
Histogram can be obtained using 

as K

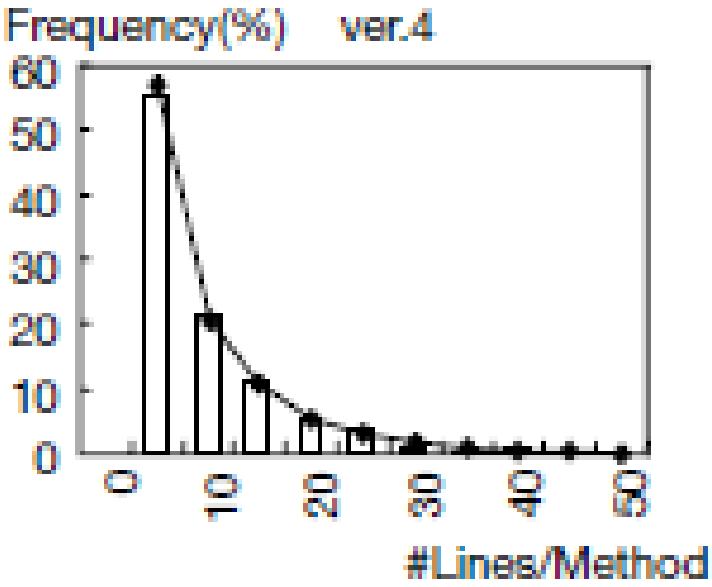
Once K is chosen one can determine the optimal h.

Histogram vs. Kernel density estimate





Step 2: fitting a distribution



Tamai, Nakatani. Negative binomial distribution

- **Family of distributions is chosen based on shape**
- **If the parameters fitting is not good enough try a different one!**

Step 3c. Goodness of fit: Pearson χ^2 test

- **The test statistic**

$$X^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

where

- **O – observed frequency of the result i**
- **E – expected frequency of the result i**

- **Compare X^2 with the theoretical χ^2 distribution for the given number of degrees of freedom: $P(\chi^2 > X^2)$**
  - **Degrees of freedom = number of observations – number of fitted parameters**
  - **Comparison is done based on table values**
  - **If the $P(\chi^2 > X^2) <$ threshold – the fit is good**
  - **Common thresholds are 0.1, 0.05 and 0.01**

Recapitulation: Statistical fitting

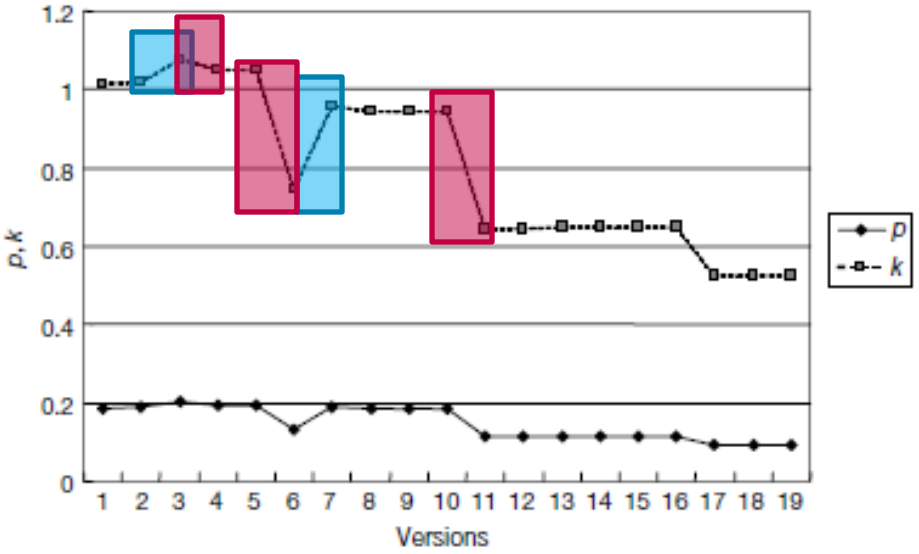
1. Collect the metrics values for the lower-level elements
2. Present a histogram
3. Fit a (theoretical) probability distribution to describe the sample distribution
   a) Select a family of theoretical distributions
   b) Fit the parameters of the probability distribution
   c) Assess the goodness of fit
4. If a theoretical distribution can be fitted, use the fitted parameters as the aggregated value

What about the evolution of the aggregated values?

- **Geometry library: Jun, subsystem “Geometry”**
- **Tamai, Nakatani: Negative binomial distribution**

$$f(x) = \binom{x-1}{k-1} p^k (1-p)^{x-k}$$

- **p, k – distribution parameters**
- $\binom{x-1}{k-1}$ **- binomial coefficient extended to the reals**



- **Increase – functionality enhancement**
- **Decrease – refactoring**

Conclusions

- Software metrics are used to assess maintainability and evolution
- Size
  - LOC, SLOC, LLOC
  - Amount of functionality (FP, size of the API)

- Next time
  - Halstead: volume V, effort E, time T and #bugs B
  - McCabe's cyclomatic complexity: v(G)
  - Combined metrics: Maintainability index
  - OO and more!