Metrics

Kristian Sandahl

LINKÖPING UNIVERSITY

Introduction

Motivation:

- Management:
  - Appraisal (What do we have?)
  - Assurance (Predict the level by process choice)
  - Control (Taking corrective action)
  - Improvement (Increase quality, lower variance)
- Research:
  - Cause-effect models

Terms:

- Metric
- Measurement

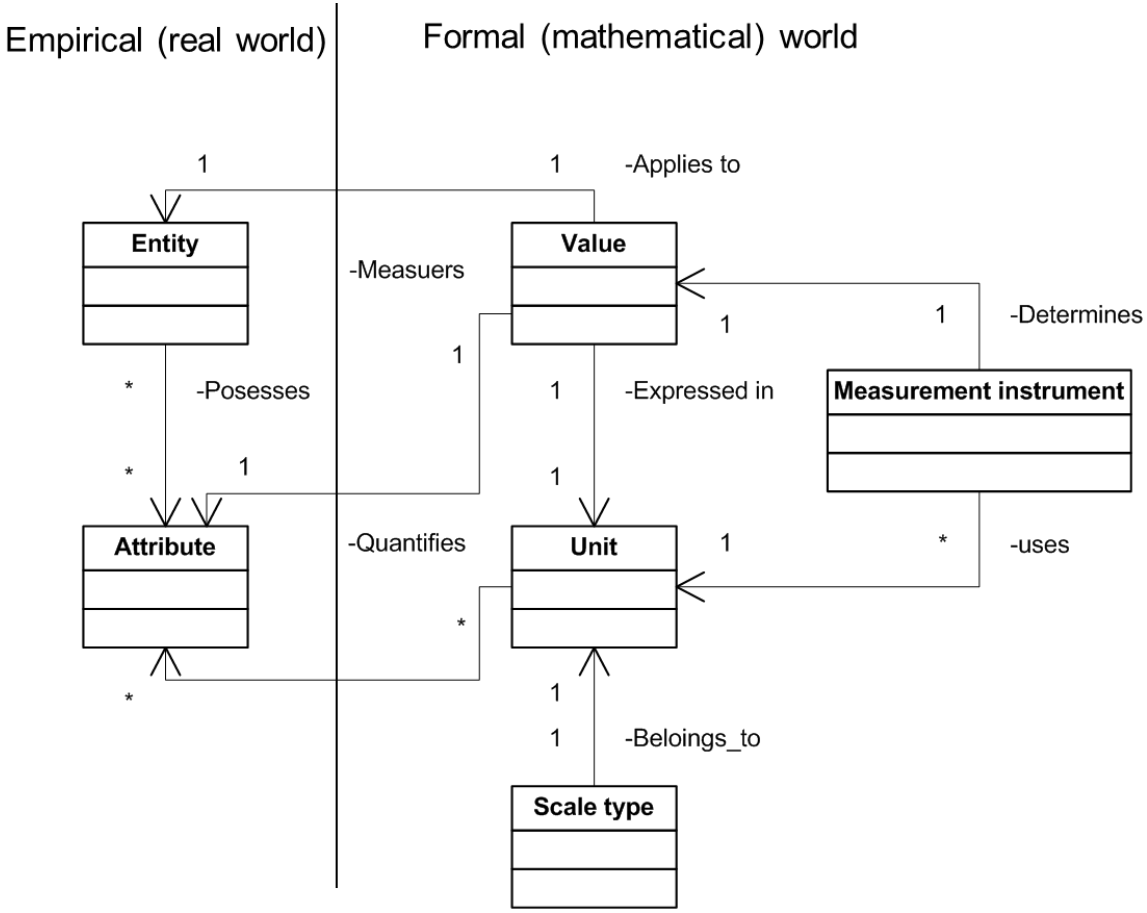
Classification

- Product metrics:
  - Observable or computed properties of the product
  - Examples: Lines of code, number of pages
- Process metrics:
  - Properties of **how** you are developing the product
  - Examples: Cycle time for a change request, number of parallel activities
- Resource metrics:
  - Properties and volumes of the instruments you are using when developing the product
  - Examples: Years of education, amount of memory in testing environment

Scales

			Examples
Nominal	= , ≠	Categories	Type of software
Ordinal	< , >	Rankings	Skill rating: high, medium, low
Interval	+ , -	Differences	%less bugs project delay
Ratio	/	Absolute zero	Lines of code

Structural model of measurement



Theoretical validation of metrics

Representational theory, based on the mapping between **attributes** of real-world **entities** – numerical **values** and **units**:

- For an attribute to be measurable, it must allow different entities to be distinguished from one another.
- A valid measure must obey the representational condition.
- Different entities can have the same attribute value.

B. Kitchenham, S. L. Pfleeger and N. Fenton, "Towards a framework for software measurement validation," in *IEEE Transactions on Software Engineering*, vol. 21, no. 12, pp. 929-944, Dec. 1995. doi: 10.1109/32.489070

Empirical (external) validation of metrics

- Correlation between internal and external attributes
- Cause-effect models
- Statistical analysis
- Handle bias

Time sheets provide a powerful source for process improvements

Sprint 1	Nisse	Stina	Pelle
Requirement	15	10	0
Design	10	10	20
Implementation	155	210	355
Test	100	150	240
Administration	10	10	25

Well defined categories is a strength.

Halstead's software science$_{1/2}$

The measurable and countable properties are :

- n_1 = number of unique or distinct operators appearing in that implementation
- n_2 = number of unique or distinct operands appearing in that implementation
- N_1 = total usage of all of the operators appearing in that implementation
- N_2 = total usage of all of the operands appearing in that implementation
  http://yunus.hacettepe.edu.tr/~sencer/complexity.html

Halstead's software science$_{2/2}$

Equations:

- Vocabulary $n = n_1 + n_2$
- Implementation length $N = N_1 + N_2$
- Length equation: $N' = n_1 log_2 n_1 + n_2 log_2 n_2$
- Program Volume $V = N \log_2 n$
- Potential Volume $V' = (n^*_1 + n^*_2) \log_2 (n^*_1 + n^*_2)$
- Program Level $L = V' / V$
- $L' = n^*_1 n_2 / n_1 N_2$
- Elementary mental discriminations $E = V / L = V^2 / V'$
- Intelligence Content $I = L' \times V = (2n_2 / n_1 N_2) \times (N_1 + N_2) \log_2 (n_1 + n_2)$
- Time $T' = (n_1 N_2 (n_1 log_2 n_1 + n_2 log_2 n_2) log_2 n) / 2n_2 S$

Chidamber & Kemerer object-oriented metrics suite

- WMC – weighted methods per class
- DIT – depth of inheritance tree
- NOC – number of children
- CBO – coupling between object classes
- RFC – response for a class
- LCOM1 – lack of cohesion of methods

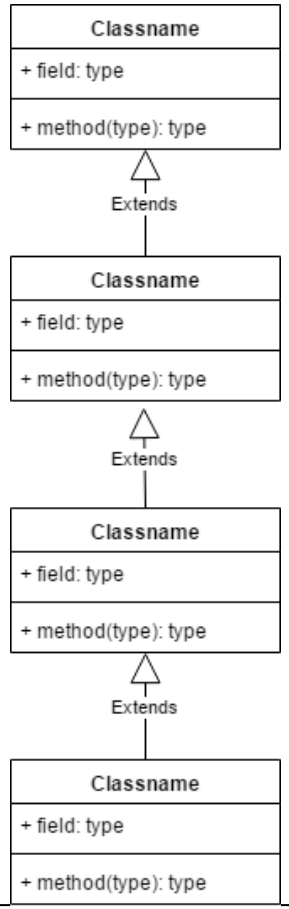
https://www.aivosto.com/project/help/pm-oo-ck.html

Weighted methods per class

- Count the number of methods per class
- Try to keep WMC low
- High WMC:
  - More faults
  - Less reuse
  - Impact of derived classes

Depth of inheritance tree

- High DIT:
  - Indicates high reuse
  - Middle classes error-prone
- Recommended max 5-8



DIT

0

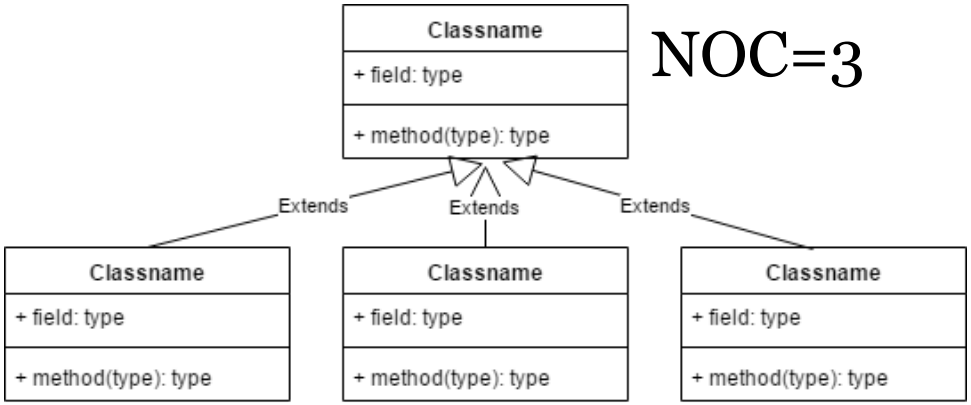
1

2

3

Number of children

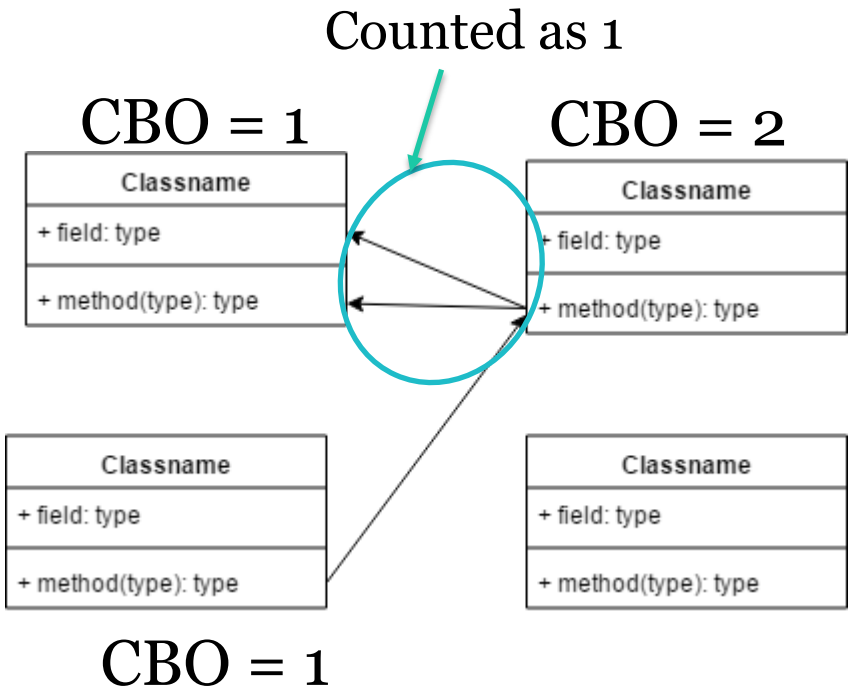
- High NOC:
  - High reuse of base class
  - Base class requires more testing
  - Misuse of sub-classing
  - Dangerous with high WMC



NOC=3

Coupling between object classes

- Limit CBO
- Low reuse
- Low maintainability
- Limit 14?



Response for a class

Let M = number of methods in a class

Let R = number of remote methods that can be called by methods in the class

RFC = M +R

High RFC:

- Low maintainability
- Low testability

RFC' includes all recursive methods in the call tree

Lack of cohesion of methods

- For each pair (m1, m2) of methods in a class:
- If m1 and m2 use a disjoint set of instance variables:
  - Increase P with 1
- If m1 and m2 use at least one common variable:
  - Increase Q with 1
- LCOM1 = {P-Q, if P>Q; 0 otherwise}
- High LCOM1 : fault prone, low testability
- Criticized measure, variants exist.

Code metrics in Visual Studio

- Lines Of Code
- Cyclomatic Complexity
- Maintainability Index = 171–5.2*ln(aveV)– 0.23*ave(g')– 16.2*ln(aveLOC)
- Depth Of Inheritance
- Class Coupling

Function Points - Background

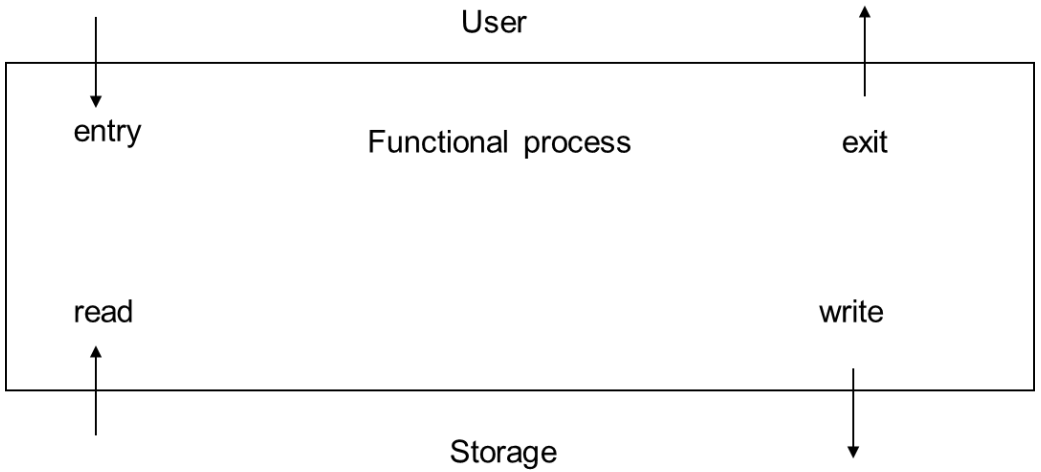
- First suggested by Albrecht 1979
- Captures complexity and size
- Language independent
- Can be used before implementation
- Used as input for estimation
- Common versions IFPUG v 4.x
- Competitor MARK II:
  - simpler to count
  - has finer granularity
  - is a continuous measure
- A “closed community”
- Traditionally used for business systems

See the pdf in Course Documents on Lisam

COSMIC-FFP

(COmmon Software Measurement International Consortium Full Function Point)

- An ISO-approved method for calculating FP for embedded, real-time systems
- Partitions the system in Functional User Requirements (FUR)



Example: Change customer data in a warehouse of items

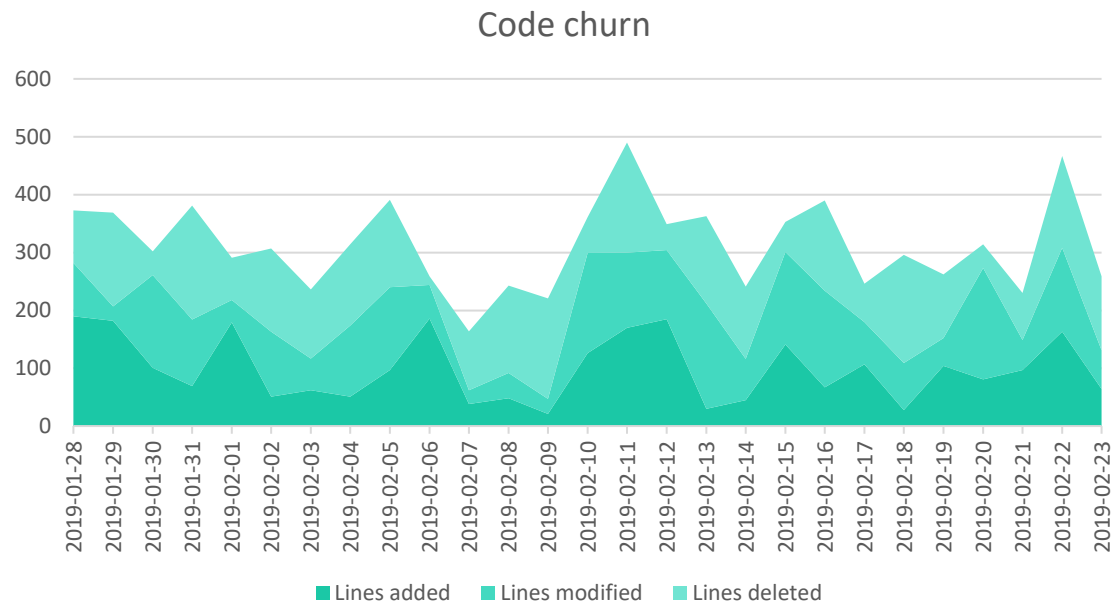
User entry	Entry	1
Retrieve customer data	Read	1
Display error message	Exit	1
Display customer data	Exit	1
Enter changed data	Entry	1
Retrieve item data	Read	1
Store item data	Write	1
Store modified data	Write	1
Total Cfsu		8

Connections to other methods

- Mapping to UML – Use cases as Sequence diagrams, count messages
- Cfsu = C_1 + C_2 FP, for less than 100 Cfsu
- C_2 1.1-1.2
- C_1 varies

- Are FP valid?

Change-based metrics: Code churn



Measure usability?

Relevance	Efficiency	Learnability	Attitude
• number of good and bad features recalled by users	• time to complete a task	• ratio of successes to failures (over time)	• percent of favorable/unfavorable user comments
• number of available commands not invoked by users	• percent of task completed	• time spent in errors	• number of good and bad features recalled by users
• number of available commands invoked by users	• percent of task completed per unit time (speed metric)	• percent or number of errors	• number of users preferring the system
• number of times user needs to work around a problem	• time spent in errors	• number of commands used	• number of times user loses control of the system
• percent of task completed	• number of commands used	• frequency of help and documentation use	• number of times the user is disrupted from a work task
	• frequency of help and documentation use	• time spent using help or documentation	
	• time spent using help or documentation	• number of repetitions of failed commands	

System Usability Scale (SUS)

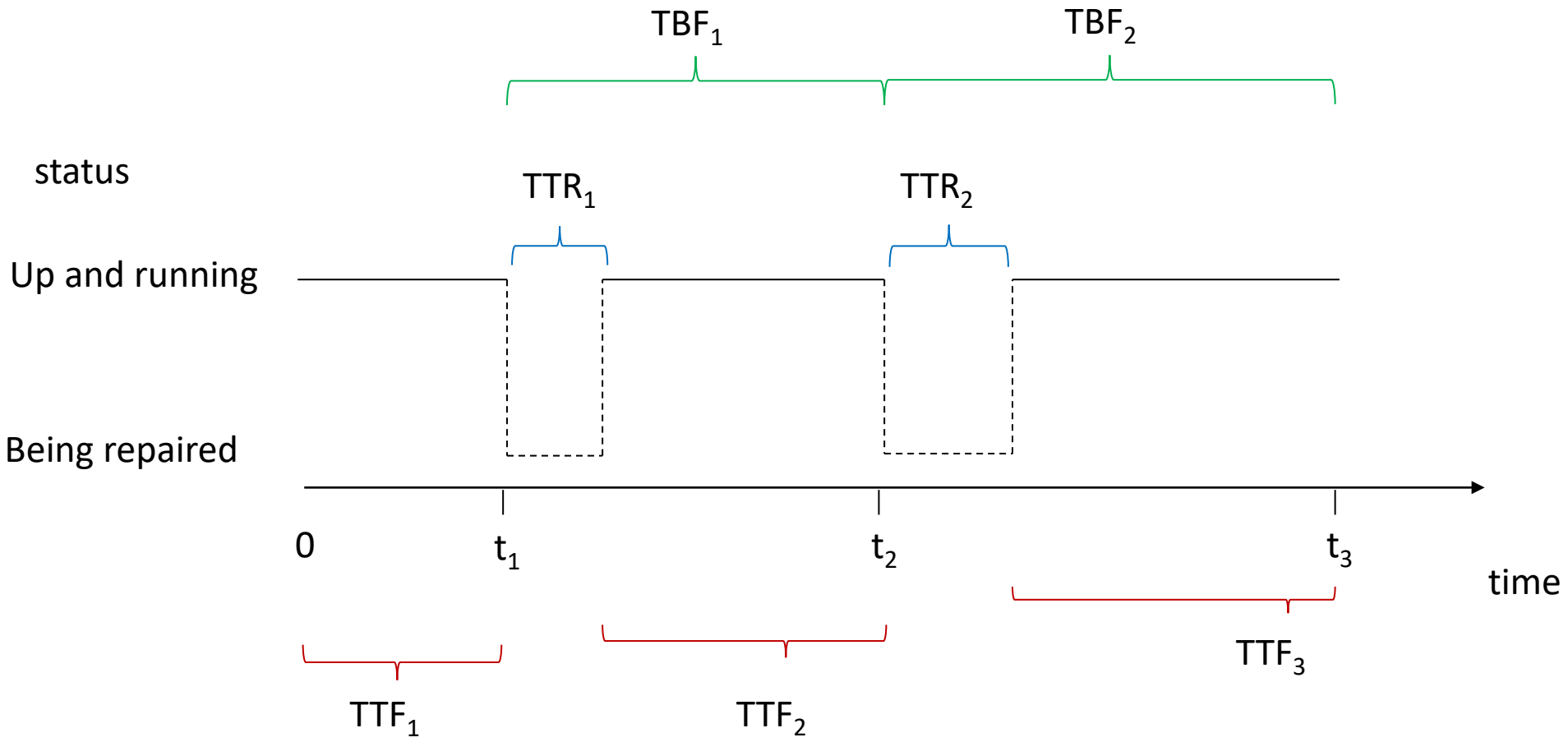
1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

Strongly disagree				Strongly agree
○ 1	○ 2	○ 3	○ 4	○ 5

For odd question numbers score = answer – 1
For even question numbers score = 5 – answer
SUS score = 2.5 $\sum$ score [0,100]

SUS score 68 is considered average

Simplified model with repair time



Reliability growth model

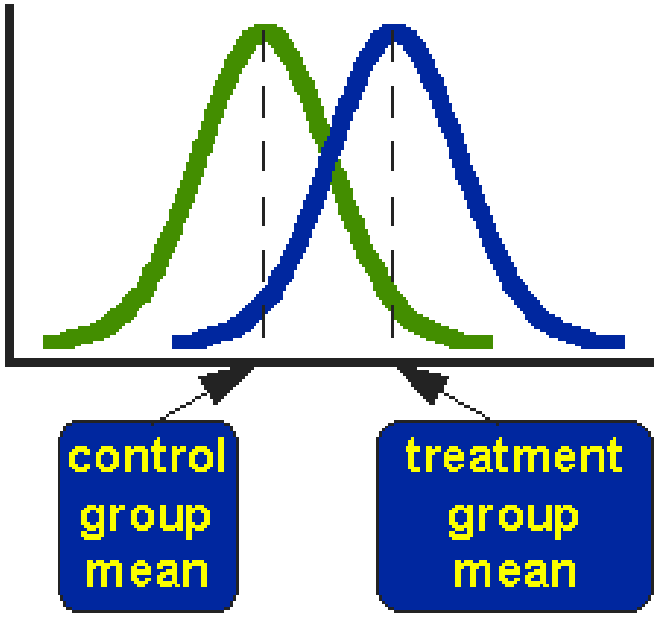
- The probability that the software executes with no failures during a specified time interval
- MTTF = Mean Time To Failure
- Approximation: MTTF/(1+MTTF)
- Example
- Easier to manage: Failure intensity, [failures / hours of execution time]
- Another approximation: $\lambda = (1-R)/t$
- Example

Similar pattern: Availability and Maintainability

- Measure Mean Time To Repair (MTTR) and Mean Time To Failure (MTTF)
- Availability, A:
- A = MTTF/(MTTF+MTTR)

- Measure Mean Time To Repair (MTTR)
- Maintainability, M:
- M = 1/(1 + MTTR)

Comparing means



Under certain conditions: Student's t-test

Significance level: nomally 5%

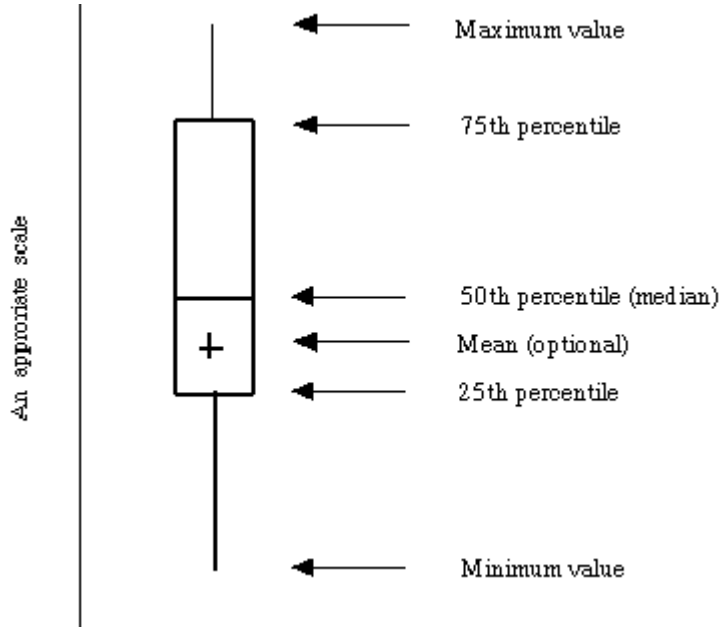
Comparing distributions

- Are the testers' methods the same?
- Under certain conditions: use the Chi-square test
- For 2x2 contingency tables other methods apply, for instance Cohen's Kappa

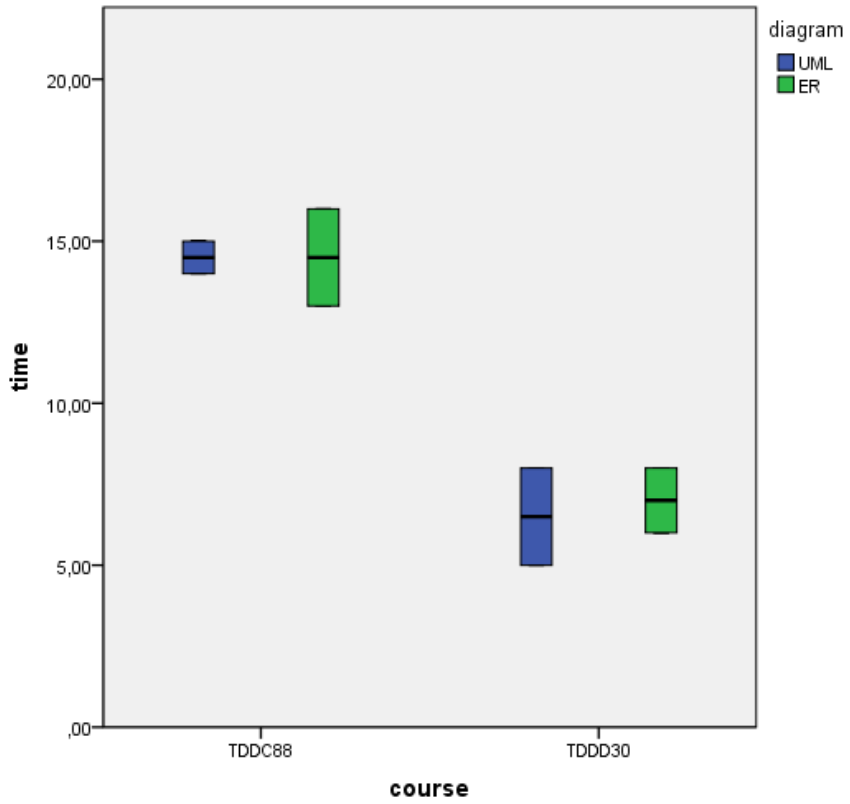
Comparing severity ratings

Severity	Tester 1	Tester 2
Catastrophic	4	2
Severe	9	6
Moderate	53	27
Minor	105	58

The box plot


An appropriate scale
Maximum value
75th percentile
50th percentile (median)
Mean (optional)
25th percentile
Minimum value

Comparing variance



Linear regression

ANOVA[b]

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	120,250	2	60,125	25,860	,002[a]
	Residual	11,625	5	2,325		
	Total	131,875	7			

a. Predictors: (Constant), course, diagram
b. Dependent Variable: time

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	14,375	,934		15,395	,000
	diagram	,250	1,078	,031	,232	,826
	course	-7,750	1,078	-,954	-7,188	,001

a. Dependent Variable: time

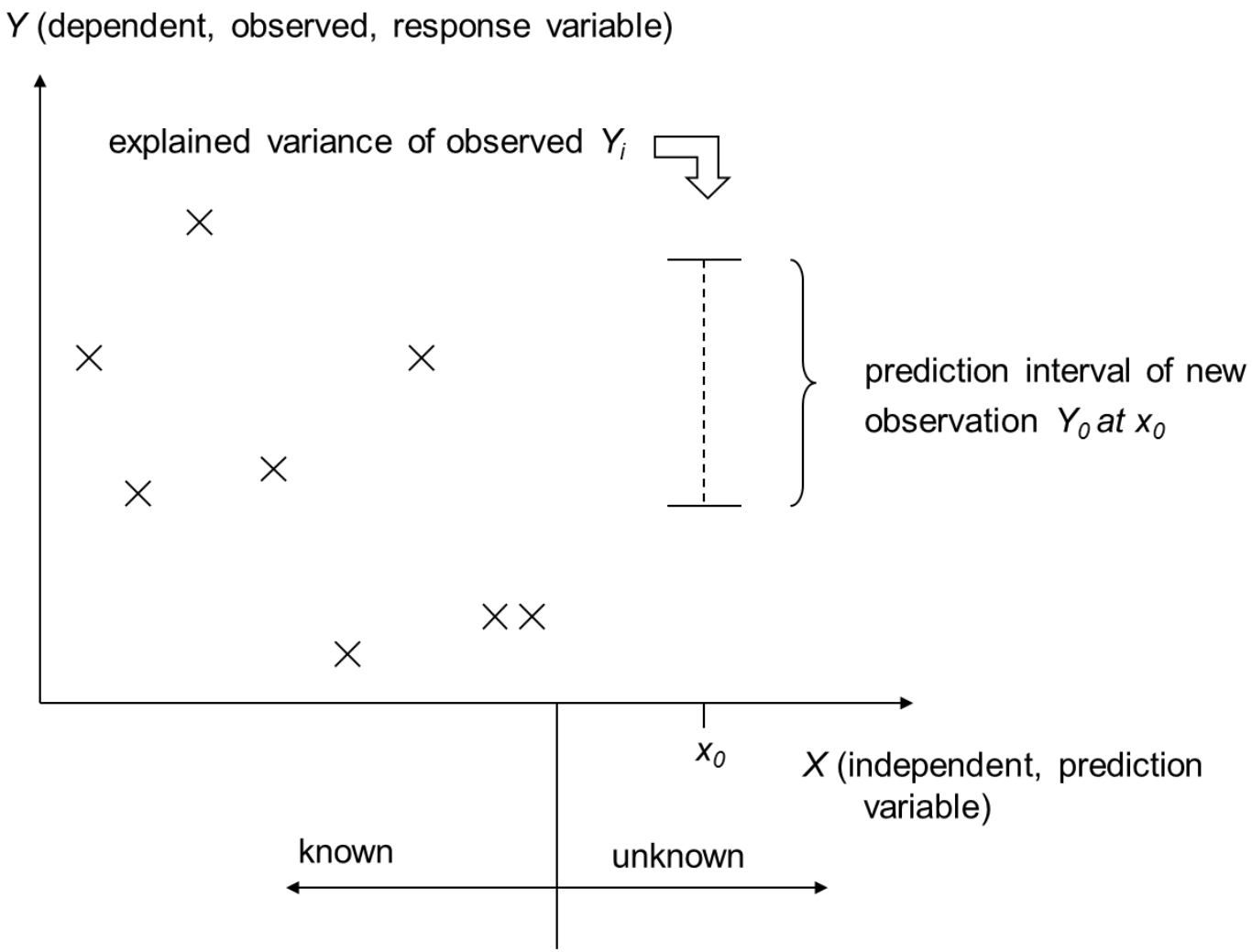
Prediction metrics

Prediction of:

- Resources
- Calendar time
- Quality (or lack of quality)
- Change impact
- Process performance

- Often confounded with the decision process

Historical data



Methods for building prediction models

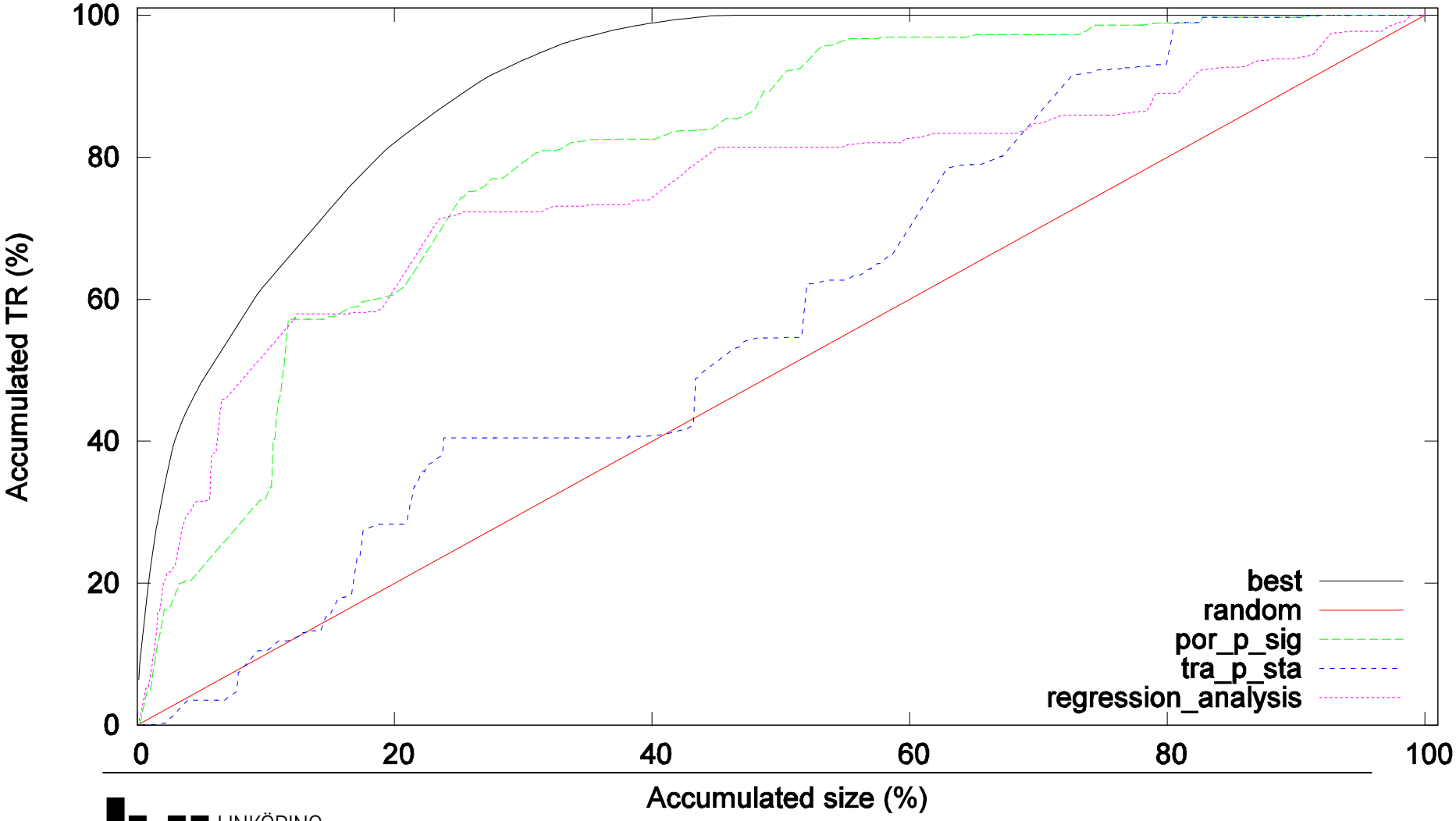
- Statistical
  - Parametric
    - Make assumptions about distribution of the variables
    - Good tools for automation
    - Linear regression, Variance analysis, …
  - Non-parametric, robust
    - No assumptions about distribution
    - Less powerful, low degree of automation
    - Rank-sum methods, Pareto diagrams, …
- Causal models
  - Link elements with semantic links or numerical equations
  - Simulation models, connectionism models, genetic models, …
- Judgemental
  - Organise human expertise
  - Delphi method, pair-wise comparison, Lichtenberg method

The Lichtenbeg method process

- Staff the analysis group
- Describe the work to be estimated
- Define general constraints and assumptions
- Define the structure
- Individual judgement of MIN, MAX, LIKLEY
- Calculate common result (MIN+MAX+3*LIKELY)/5
- Find workpackages with large variance
- Sub-divide them and rework

- 5-20 participants
- Never influence each others judgements
- MIN and MAX should be extreme – 1% of the cases

Example of a pareto diagram



Metrics and experimentation/ Kristian Sandahl

www.liu.se

LINKÖPING UNIVERSITY