

Course "Softwareprozesse"

Software Measurement

Lutz Prechelt

Freie Universität Berlin, Institut für Informatik

- Measure, measurement
- Scale type
- Validity, reliability, precision
- Purpose matters most:
Goal-Question-Metric (GQM)
 - example purposes by CMMI process area
- Product measures
 - size, quality
- Process measures
 - process quality, e.g.
 - productivity, defect prediction

- What are measures? What is measurement?
- What are the difficulties of measurement?
- Measurement in the software process
- What would we like to measure?
- What measures are common?
- Examples of more specialized measures?

What is it about?

- Measurement is assigning a symbolic **value** to an object in order to characterize a certain **attribute** of that object

Why is it of interest?

- Measurement is useful **abstraction**: It sometimes allows talking about a complex situation in a way that is simple, yet precise and objective

What will we look at?

- Base ideas (measurement, scale type, validity, inference)
- Common mistakes
- Example metrics (product, process)

- Measurement
 - The process P of using a measure M
 - for assigning a value V to some object X
 - in order to characterize attribute A of X.
- Measure ("Maß")
 - A specific rule M for measuring A: $A_M(X) = V$
 - Each measure is on a particular scale (see below)
 - Different measures may exist for the same attribute
 - These can be equivalent (differ in scale (**measurement unit**) only)
 - non-equivalent but similar (use the same basic approach)
 - or non-compatible (differ in their approach to characterizing A)
- Metric ("Metrik")
 - Mathematics: A measure of distance between points
 - Software Engineering: A synonym for measure
 - Actually, 'metric' is more common, probably because 'measure' also means 'Maßnahme'



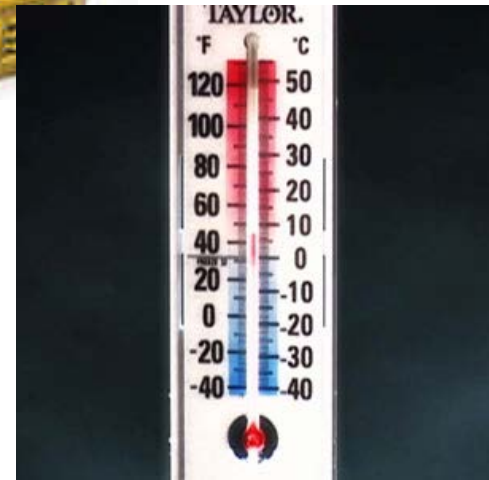
Example measurements

- $\text{NumberOfWheels}(\text{Car X}) = 4$
 - absolute scale

- $\text{Height}_{\text{feet}}(\text{Room X}) = 9.8 \text{ ft}$
 $\text{Height}_{\text{meter}}(\text{Room X}) = 3 \text{ m}$
 - ratio scale



- $\text{Temperature}_{\text{Centigrade}}(\text{Room X}) = 22^\circ \text{C}$
 $\text{Temperature}_{\text{Fahrenheit}}(\text{Room X}) = 72^\circ \text{F}$
 - difference scale



- $\text{Grade}(\text{Student X}) = \text{"very good"}$ (absolute)
 $\text{Grade}(\text{Student X}) = \text{"top group"}$ (relative)
 - ordinal scale

A word of warning: scale types

- An advantage of quantitative data is that it can be processed using mathematical operations ("inference")
 - This can allow easy summarization or can provide additional insights
- However, not all computations (inferences) are valid for all kinds of quantitative data
 - The data must be on a sufficient scale type
 - otherwise, an operation may be invalid and may make no sense
 - See the next slide

The scale types

- Nominal/categorical scale
(named values)
 - Qualitative data: The values are just names
 - Example: Design method A, design method B
 - Inference: mode
 - (the most frequent value)

- Ordinal scale
(rank scale)
 - Ordered nominal data: One value is larger than another, but we cannot characterize the size of the difference
 - Inference: comparison, median
 - Example: very good, good, OK, not so good, bad

- Difference scale
(interval scale)
 - Inference: difference, mean
 - but 0 is not equal to 'nothing'
 - Example: degrees centigrade

- Ratio scale
("Verhältnisskala")
 - We can compute ratios: 20 is twice as much as 10
 - Note: But we cannot compare apples to oranges (or meters to watts)!
 - Most physical quantities, including absolute temperature (Kelvin)

- Absolute scale
 - 1 is also special (counting)

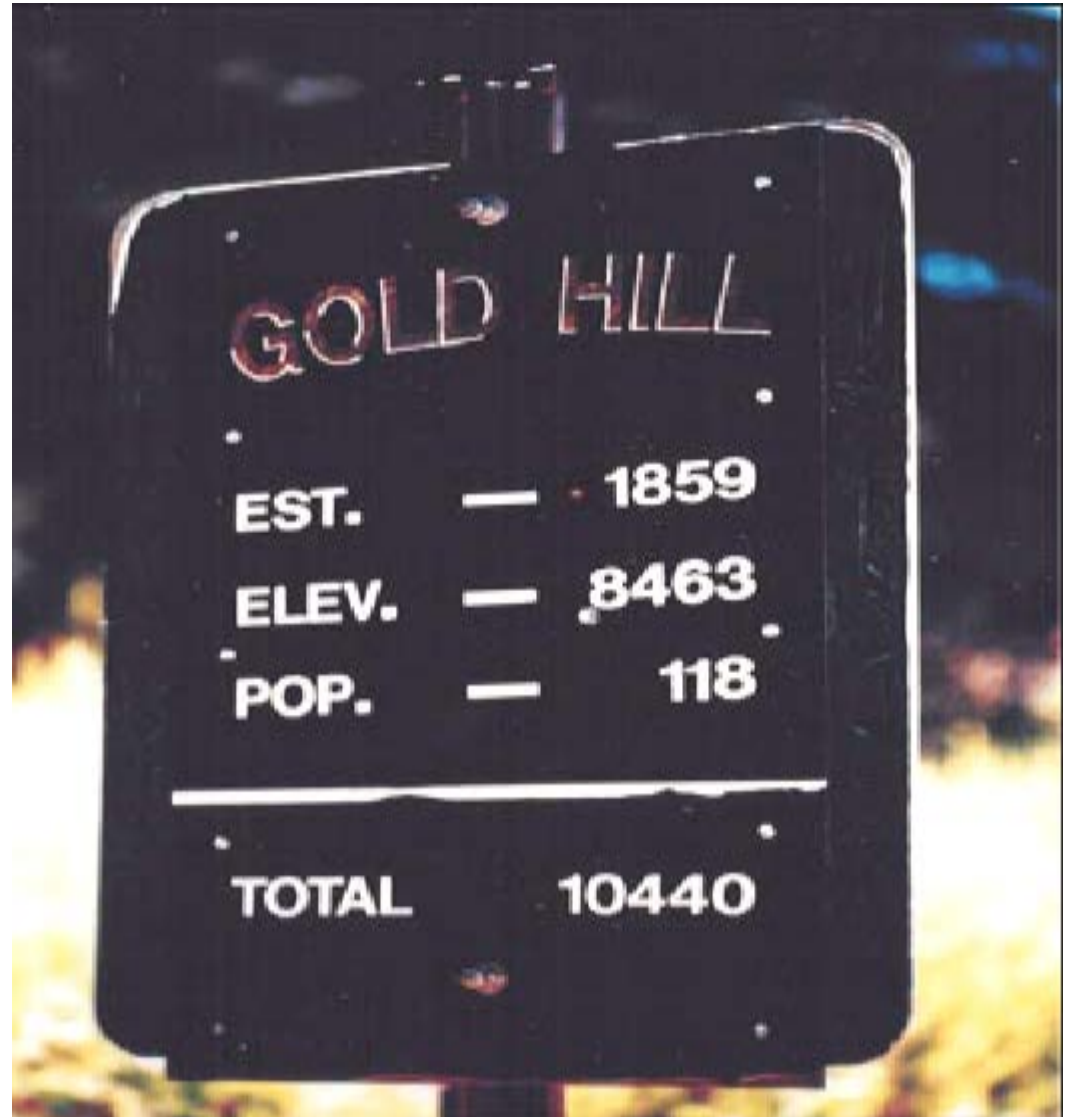
What often goes wrong with scale types

- *"Oh, 20 degrees. That's twice as warm as yesterday."*
 - A difference scale is not a ratio scale
 - 20 degrees centigrade is 293 Kelvin.
That is only 3.5% more than 283 Kelvin.
- When something qualitative is measured using an ordinal scale
 - e.g. *"How well did you like using the tool?"*
very well, well, OK, not so well, did not like it
 - Often such scales are coded with numbers: 5, 4, 3, 2, 1
 - Wrong: *"average satisfaction was 3.8"*
 - Even worse: *"average satisfaction in group B was 30% higher than in group A"*
 - This is utter nonsense! (What assumption does it imply?)
 - Assume you would have coded using 605, 604, 603, 602, 601?
Or 605, 604, 22, 0, -888?

What sometimes goes wrong with scale types

- Even if you do have a ratio scale, things may go wrong:

established
elevation
population



Exercise:
Which scales are present here?

- Validity

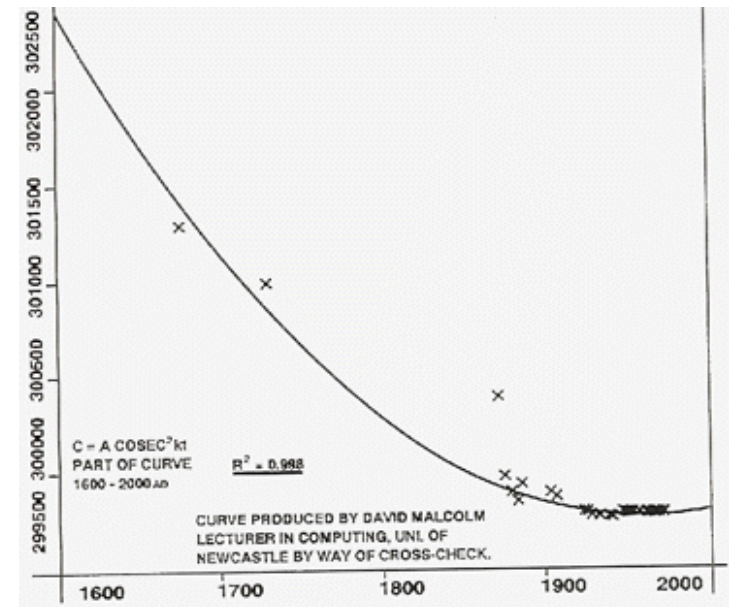
- For a measure: How well does the measure really characterize the intended attribute?
- For a measurement procedure: How well do specific measurements really approximate the true measurement value?
 - How exact is it? (Accuracy)
- See next slides

- Reliability

- How well do multiple measurements of the same object agree?
- e.g. repeated measurements of IQ

- Precision

- How fine is the maximal resolution of differences in the attribute?
- e.g. measurement of time in minutes, seconds, or milliseconds

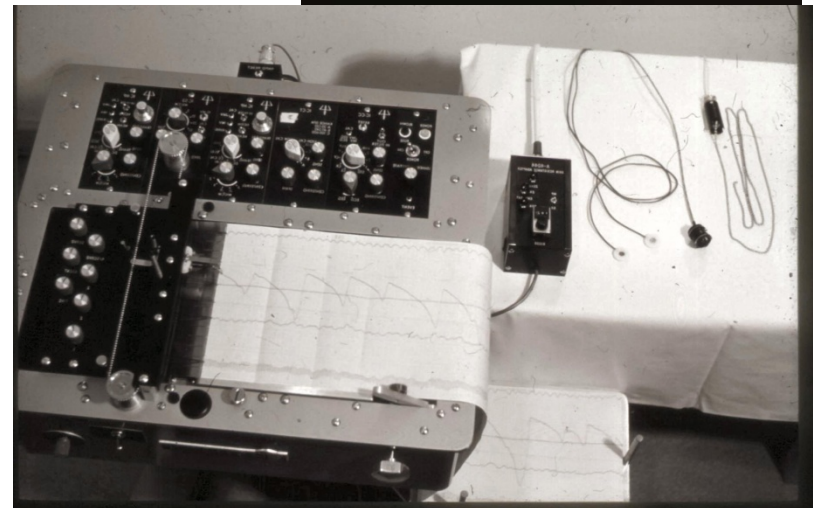
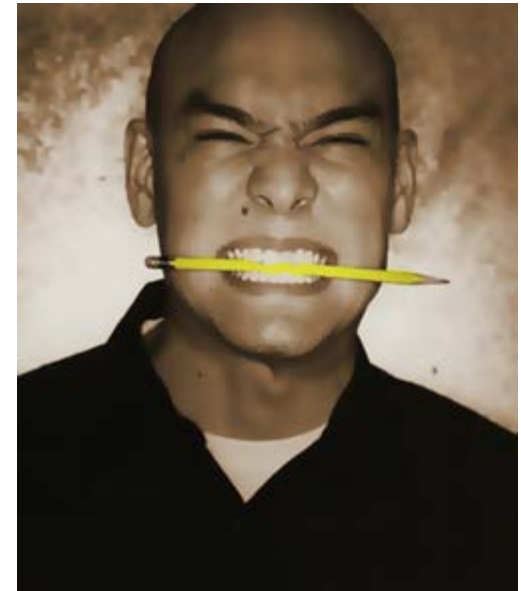


Validity of a measure

- **Simple attributes** (such as physical quantities) can be characterized completely and precisely by a single measure
 - e.g. the length
- All such measures of one attribute are equivalent and valid
 - they just use different measurement units
 - which can be converted by fixed formulae
- **Complex attributes**, however, have no single measure
 - e.g. the quality of a piece of software
- Many different measures can be invented for such attributes
- They are not equivalent
- They measure different aspects of the attribute or are different representatives (proxies) of the attribute
 - They can be more or less "good" measures of the attribute
 - This goodness is called "validity" (Validität, Gültigkeit)

Validity example: Stress

- Attribute to be measured:
Current level S of stress of a person
 - (stress is not a well-defined state)
- Possible proxies:
 - Rate of heartbeat
 - Blood pressure
 - Muscle tone (Muskeltonus)
 - Skin temperature
 - Skin electrical resistance
- None of these alone is a very valid measure of stress
- However, they can be combined to form a fairly valid measure
 - This is used for instance in the Polygraph (Lügendetektor)



Validity example: Size of software

- Attribute to be measured:
The size S of a piece of software
 - (software size is not a well-defined attribute)
- Examples of possible proxies:
 - Length of source code
 - In statements? In lines? In bytes?
 - In lines, but ignoring comments, empty lines, brace-only lines, ...
 - Length of object code
 - Using which compiler? Which settings?
 - Memory requirements
 - But: Does my sort program get bigger when it sorts a longer input sequence?
 - Functionality
 - Sensible idea! But: Unit? Measured how?
- Validity depends on purpose of measurement



What would we like to measure?



1. Attributes of the product:

- Software size
- Software quality
- Software value

2. Attributes of the process:

- Effort, progress
- Productivity (efficiency)
- Process quality

4. Predictors of any of these

3. Also possible: Attributes of resources (people, teams)

- productivity
- product quality

Purposes of predictors:

- Guiding design
- Guiding scheduling
- Guiding quality assurance



Measuring software size

- Software size is of interest because it is a major factor for all process attributes relating to effort, such as
 - productivity
 - implementation/test/maintenance effort
- However, it is an elusive property.
Sensible, valid measures of size are difficult to obtain.
 - Do not confuse software size with code size!
- The most common measures are

• Function points	(at reqmts. level)	←
• Number/pages of requirements	(at reqmts. level)	
• Story points	(at reqmts. level)	←
• Number of modules/classes	(at design level)	
• Lines-of-code (LOC)	(at code level)	←

Measuring size: Function Points (FP)



- Idea:
 - Sum requirements by kind, weighted by complexity, in a standardized way. Measures size in terms of construction effort.
- Advantages:
 - Size can be determined early in the process
 - The measure is independent of language, technology, design style, superfluous functionality etc.
- Function point rules:
 - Count the number of input dialog screens ("inputs"), view/report screens ("outputs"), connections to other programs ("interfaces"), blocks of internal program state ("files")
 - Weigh each as "simple", "medium", or "complex" (via well-defined criteria), each giving a fixed number of points (not shown)
 - Sum up all these points → raw size
 - Multiply by empirical effort per point → size as expected effort

FTR's	DATA ELEMENTS		
	1-5	6-19	> 19
0-1	Low	Low	Ave
2-3	Low	Ave	High
> 3	Ave	High	High

- Function points are fairly well-defined
 - Detailed definitions exist for the requirement kinds and complexity classes
 - Even for SW changes (as opposed to new development)
 - Once the productivity (FP/personmonth) is known, FPs provide a good basis for project estimation

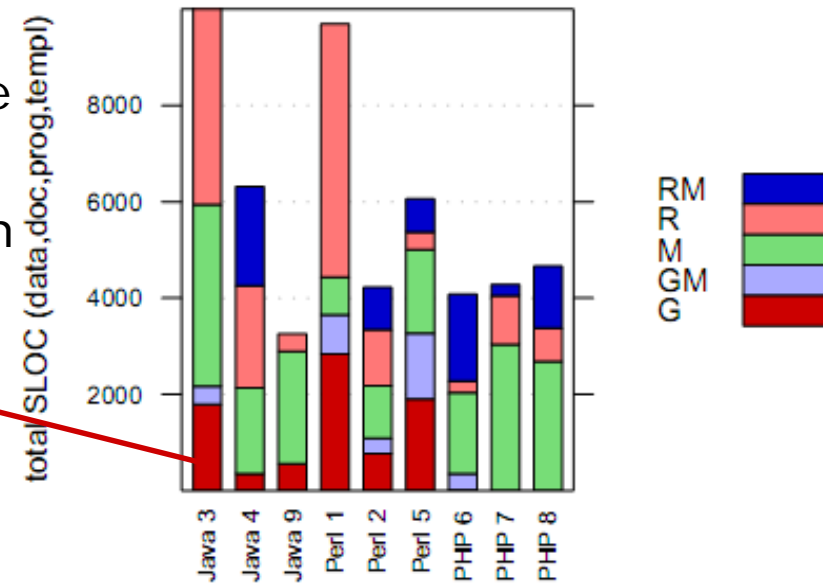
Limitations:

- Function points are valid only for certain kinds of system
 - Straightforward information systems whose internal processing is simple
 - The basic idea can be adapted to some other domains, though
- Counting FPs is a manual process
 - Except if the requirements are structured along FP counting rules

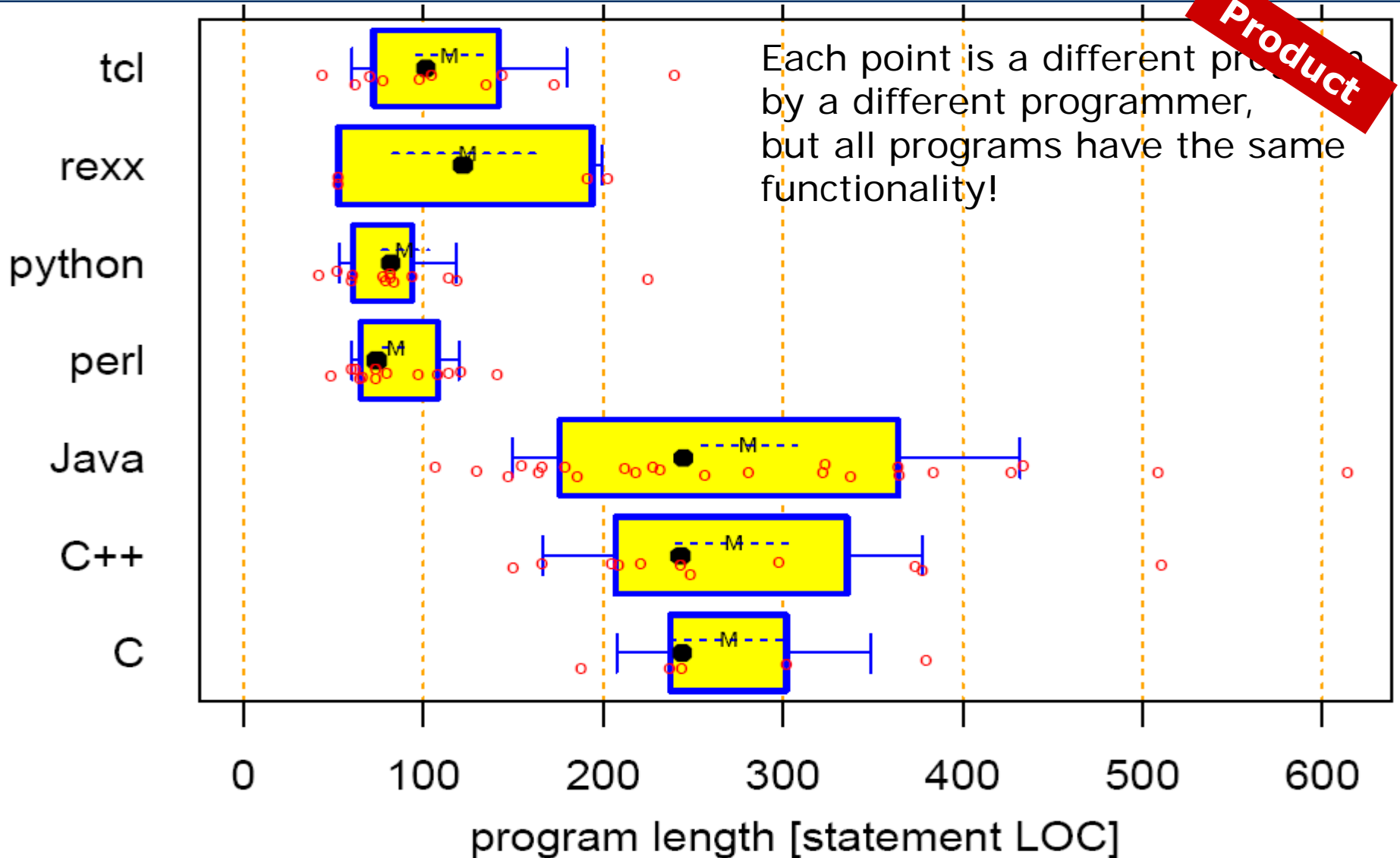
Measuring size: Lines-of-Code (LOC)

Product

- Idea:
 - Length of source code is a reasonable representative of size
- Advantages:
 - Can be measured automatically
- Common definition:
 - Count all lines containing program code (declaration/statement)
 - i.e., ignore empty lines and comment-only lines
 - Names: LOC (lines of code), KLOC (1000 LOC), SLOC (source LOC), NCSLOC (non-comment SLOC)
- What to watch out for:
 - Depends on programming language
 - Depends on formatting style
 - Heavily depends on program design
 - Should comments count or not?
 - Have code generators been used?
 - Has code been cloned?



LOC: Dependence on language and program design



Measuring size: Story points

- Story points are often used in the iteration planning session in agile development
- Story points apply to an individual Story
 - Story: small piece of functionality visible to an end user
 - Determined by team discussion → consensus estimate
- Story points represent **expected implementation effort**
 - Fibonacci values are often used: 1, 2, 3, 5, 8, 13, ...
 - A unit of measurement may or may not be attached
 - If present, it is typically ideal work hours.
- Purposes:
 - Comparing cost/benefit ratio of one story against another
 - Filling the iteration without overfilling it



Comparing size measures

- So which is the best size measure?
 - Unclear!
 - Why unclear? Because their purposes differ
- Purposes:
 - Function points:
 - Estimating implementation effort, measuring productivity via amount of functionality
 - Lines of code:
 - Estimating maintenance effort (does not work for: functionality, productivity)
 - Story points:
 - Estimating implementation effort, determining implementation priorities
- So we should ask for measurement goals, not measures

- All measurement must be made for a specific, well-understood purpose
 - i.e., in order to reach a process-related goal
- Otherwise, the measurements will probably not be useful
 - and perhaps even misleading
- The process must support the measurement
- Otherwise
 - measurement cost will often be high
 - data quality may be low (mistakes, lack of precision, gaps)
- An approach for designing and arranging goal-directed, well-supported measurement programmes is called **GQM**
 - see next slide

Goal-Question-Metric (GQM)

- Source: Victor Basili, David Weiss: "*A Methodology for Collecting Valid Software Engineering Data*", IEEE Transactions on Software Engineering 10(6), Nov. 1984.
- Step 1(**G**): Establish the goals of the data collection
 - e.g. "Understand our software changes and change process"
- Step 2(**Q**): Formulate questions of interest
 - e.g. "What is the distribution of change reasons?",
"What is the distribution of changes over system components?",
"How large are individual changes?",
"How much effort is needed for individual changes?"
- Step 3(**M**): Define concrete measurements (metrics)
 - categorial ones, e.g. change reason classes, subtask types
 - quantitative ones, e.g. number of files modified, person hours

Goal-Question-Metric, GQM (2)



- Step 4(**M**): Implement the measurement
 - Manual measurement: develop a data collection form
 - Automated measurement: implement data collection software
 - Introduce the process change that includes the measurement
- Step 5(**M**): Collect and validate data
 - Validate as soon as possible
 - Initially all data, later samples of data
 - Validate even automatically collected data
- Step 6(**Q**): Analyze data
 - Analyze measurement data in order to answer questions
- And then(**G**):
 - Progress towards your goal,
 - perhaps modify measurements

Important goals/purposes for measurement



By CMMI key process area:

- PP: Project planning
 - effort estimates
- PMC: Project Monitoring & Control
 - measures of progress
- RSKM: Risk Management
 - rough quantification of some key risks
- OPP: Organizational Process Performance
 - measure how well we can produce software and how fast
- QPM: Quantitative Project Mgmt
 - all of the above, plus
 - measure product quality (internal, external)
 - measure process quality (initial quality, QA steps, rework)

Measure quality

Measure productivity

Measuring product quality



- Measuring quality is the holy grail of SW measurement
- Quality is not one attribute but many
 - e.g. efficiency, modifiability, readability, correctness, robustness, usability, learnability
- All of them can only (if at all!) be measured after specifying context and assumptions
 - e.g. the workload or the expected changes
- Some of them can then be measured
 - e.g. efficiency by benchmarking
 - again several aspects: runtime, throughput, memory consumption, network traffic, energy consumption, etc.
 - e.g. learnability by a beginner experiment
 - measure task completion time, correctness, and success rates of beginners after receiving a well-defined training or instruction
- others are by-and-large impractical to measure
 - e.g. modifiability: How to generalize from one change to all?

Measure quality

Measuring product quality: Quantify where possible

Product

- Quality attributes are very important for software project agreements/contracts
- Therefore, all quality attributes that can be operationalized should really be defined and measured: "Quantify quality"
 - "Poor quantification is better than none" ([Tom Gilb](#))
 - at least it can be; because it allows for systematic progress
 - Definitions could be part of the contract
 - Measurements can be part of the acceptance test
 - They will often involve human beings and subjectivity, but that is acceptable if statistical averaging is used
 - e.g. test learnability by measuring a group of 20 random representative users
 - See course "Empirische Bewertung in Informatik"
- Definitions will change as you learn how to measure better

Measure quality



Tom Gilb

Note: Quality and complexity attribute scales



There are a number of important attributes of software for which no good ratio scales are known

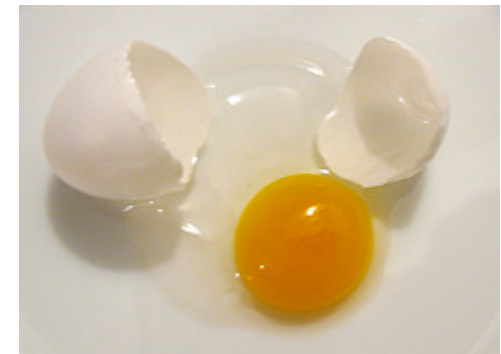
In particular:

- Quality
 - And all quality attributes such as comprehensibility, usability, portability, maintainability, etc.
- Complexity
- These are frequent places of scale type mis-use
 - So watch out what you are doing
 - e.g. reducing the number of defects by half is not doubling the correctness
 - correctness is bounded by 1 and does probably not have a sensible ratio scale (as opposed to reliability, which has one)

Measure product quality: defects

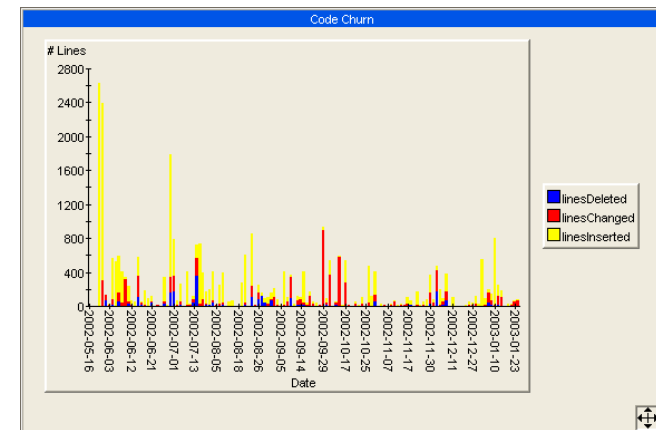
Product

- Most quality measures are costly to evaluate
 - They require much additional effort
- Hence, most projects take a much simplified approach: They count defects only
 - Hence they only measure correctness and measure it indirectly:
 - Some defects lead to frequent failures, others to rare ones or none;
 - some failures have high impact, others are negligible
 - e.g. bookkeeping: rounding errors vs. double booking
- Defects can be counted in various situations:
 - during inspections
 - during testing
 - after delivery ("field defects")
- Defects can be classified in various ways
 - See separate lecture



Measuring process quality

- For process improvement, we are not only interested in product quality, but also in the quality of the process that produced it
- Common measures of process quality:
 - Productivity (see subsequent slides)
 - Inspection yield
 - What fraction of all defects present in a document is found by an inspection?
 - Defect insertion
 - How often do new code or defect corrections introduce new defects?
 - Schedule accuracy/predictability
 - Tasks: How much too long how often?
 - When does replanning occur (early/late)?
 - Churn
 - What fraction of documents (e.g. code files) is changed often? How often? Why?
 - etc.



Measuring productivity

- Increasing productivity is one of the main goals of software engineering
- In general terms, productivity is output per input
 - For SW engineering:
Amount (or value) of SW produced per effort invested
- **Effort** can be measured reasonably well
 - (include all tasks; don't forget non-technical staff)
- We have discussed measuring the **size** of SW before:
This is a vague concept!
- **Value** would be a superior replacement for size
 - but is difficult to measure as well
 - see lecture "Essentials of SE"

Example: Hugely different levels of (LOC-)productivity

- 29 projects from Hewlett Packard (median size 70 KLOC)
 - M. Cusumano et al., IEEE Software, September 2003

Variable	Description	Mean	Median	Standard deviation	Minimum	Maximum
Defect rate	Average no. of customer-reported defects per month per million lines of new code over first 12 months	18.8	7.1	23.1	0.0	80.0
Productivity	New LOC developed per person-day	26.4	17.6	24.0	0.7	85.0
Functional specification	Percentage of functional specification that was complete before team started coding	55%	55%	32%	0%	100%
Design specification	Percentage of detailed design specification complete before team started coding	20%	10%	26%	0%	80%
Design review	Binary: 1 if design reviews were performed during development, 0 if not	0.79	1	0.41	0	1
Code review	Binary: 1 if the number of people who typically reviewed another person's code was one or more, 0 if none	0.52	1	0.51	0	1
Subcycles	Binary: 1 if development was divided into separate subcycles, 0 if not	0.76	1	0.43	0	1
Early prototype	Percentage of final product's functionality contained in the first prototype	38%	40%	24%	0%	90%
Daily builds	Binary: 1 if design changes were integrated into code base and compiled daily; 0 if not	0.32	0	0.48	0	1
Regression test	Binary: 1 if someone ran an integration or regression test when checking code into the build; 0 if not	0.55	1	0.51	0	1

Example (cont'd): Productivity study results

Results of the study:

- Several of the development practices investigated allow to predict (somewhat) the productivity or product quality:

- System SW has much higher defect rates
- Automated tests upon each check-in correlate with greatly improved resulting quality, as do design reviews
- Daily builds correlate with greatly increased productivity

- (Note: The study only indicates correlation, not causation!)

Model	Defect rate	Productivity
Constant	16.36	34.9****
Systems	14.87**	
Early prototype†	0.48***	-0.42***
Daily builds		16.89**
Regression test	-12.64**	
Design review	-19.65**	
R-squared (adjusted)	74.6%	52.8%
F-ratio	15	11.1
Df	15	16

A product+process measurement goal: Defect prediction

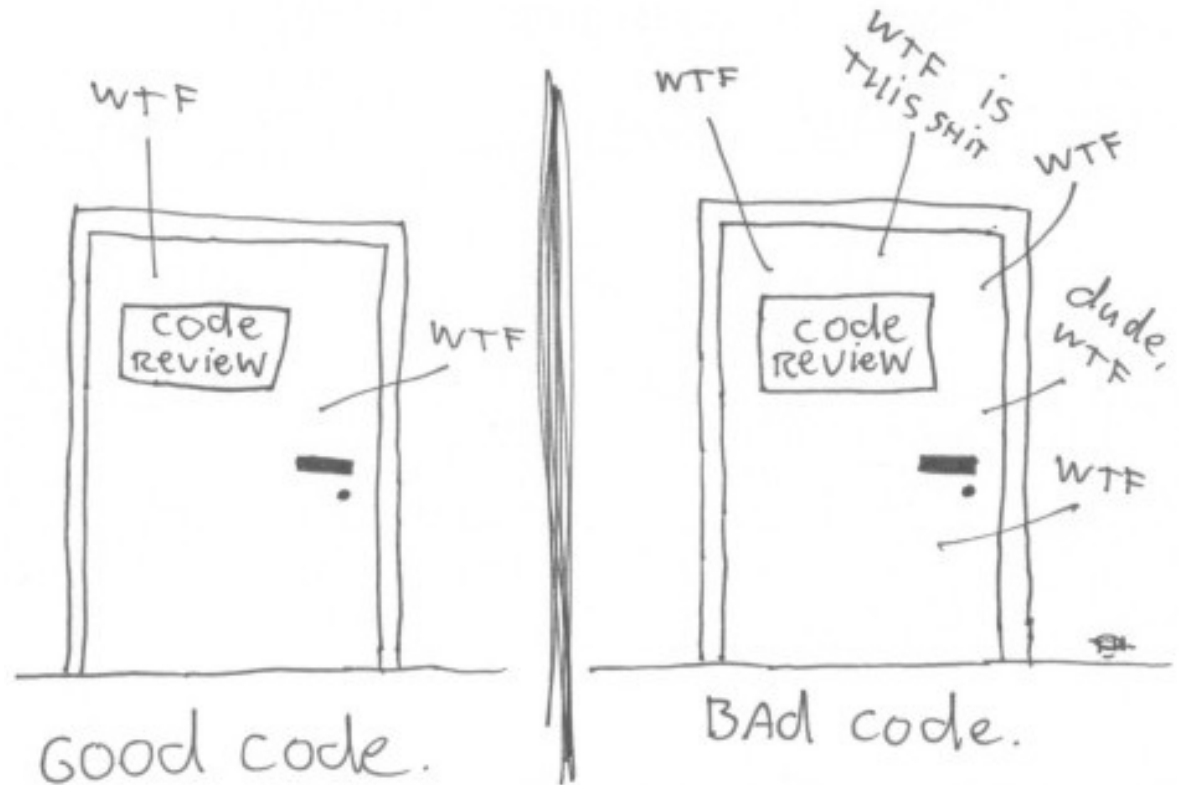


- Goal: Predict which modules will contain a defect
 - so as to best allocate the available QA effort
- There is a lot of literature about this problem.
- Foyzur Rahman, Prem Devanbu: *"How, and Why, Process Metrics Are Better"*, ICSE 2013. for defect prediction
 - Code metrics: size (on various levels), fan-in, fan-out, cohesion, inheritance, ...
 - Process metrics: #commits, #developers, #lines added/del, change scattering, owner's experience, committer's exp, ...
- Result:
 - Prediction models based on code metrics are highly project-specific → Cannot bring in prior knowledge well.
 - **Prediction models based on code metrics stagnate: They are not release-specific. But the defects are.**

- Measurement is assigning values to attributes of objects
 - There are different scale types for the values: Beware!
- Quality attributes of measures: validity, reliability, precision
 - For most of the interesting attributes of SW and SW processes, it is very difficult to define highly valid measures!
- Measurement **purpose** matters. Use Goal-Question-Metric.
 - compare the very different measures of software size
- Product quality and process quality are difficult to measure
 - but even crude quantifications can be valuable
- Process measures are often more useful than product measures
 - e.g. for defect prediction

Thank you!

The ONLY VALID MEASUREMENT OF code QUALITY: WTFs/MINUTE



(c) 2008 Focus Shift