Softwaretechnik / Software-Engineering

Lecture 2: Software Metrics, Cost Estimation

2019-04-29

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Topic Area Project Management: Content

VL 2
- **Software Metrics**
  - Metrics, Properties of Metrics
  - Software Metrics
  - Software Metrics Issues
- **Cost Estimation**
  - (Software) Economics in a Nutshell
  - Software Cost Estimation
  - Expert's / Algorithmic Estimation

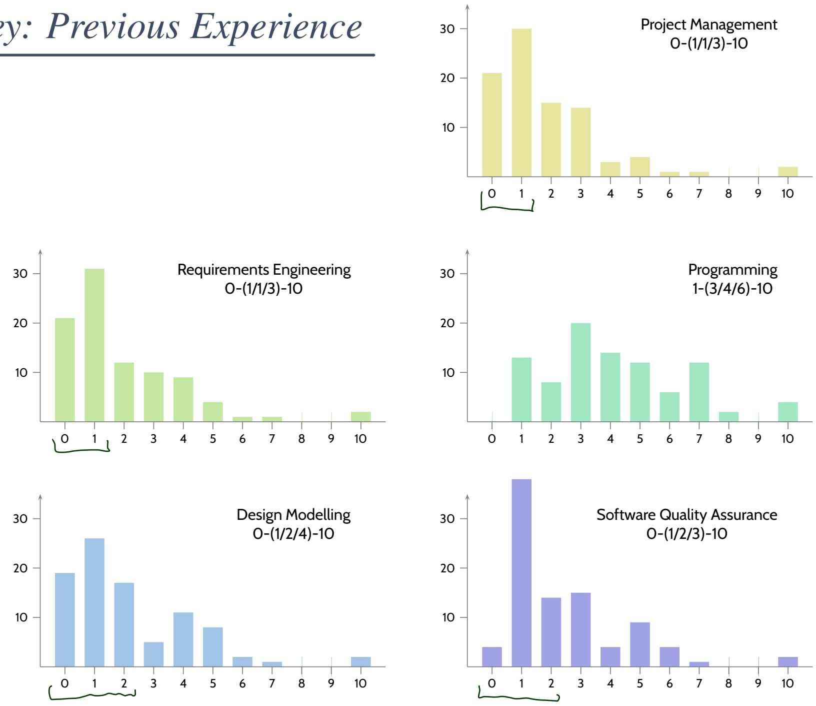
VL 3
- **Project Management**
  - Project
  - Process and Process Modelling
  - Procedure Models
  - Process Models

VL 4
- **Process Metrics**
  - CMMI, Spice

Content

Survey: Previous Experience & Expectations

Survey: Previous Experience



Expectations: What We Do Not Do (For Reasons)

- **Soft Skills**
  - ✘ Individual **time management**.
  - ✘ What do we do if a **team member** does not perform his/her tasks?
  - ✘ **dealing with** unrealistic expectations from the client
- **How to get a Good Design**
  - ✘ What does it mean: **better design**?
  - ✘ Overview over **object-oriented architecture**, Design Patterns
  - ✘ the capability to **create** a software architecture

  → Our focus: **Describe and Discuss Design Ideas**
- **Programming**
  - ✘ We want to **program** more efficient.
- **Large Examples**
  - ✘ More practical examples [...] from **larger projects**.

  → Many of our examples are **inspired** by **real** projects.

Expectations: Needs Clarification

- **Concrete problems / approaches**:

(✔) **the** state of the art of testing, project management, etc.
(✔) **ideal** planning of budget and workload
(✔) **how to find out** the customer's requiremens on a software?
(✔) learn **the** proper metrics to measure progress [...] and check product quality of the product
(✔) how to systematically **conduct** a test
(✔) **how to** decide which methods or techniques are good choices

(✘) successful completion of the Softwarepraktikum

- **Tools**

✔ Which tools can be used to develop (high quality) software?

Expectations: Yes \o/

- **Can be solved right here**:
  - what can, in general, be assumed to be self-evident ('selbstverständlich')?

- **Vocabulary**

✔ communication skills; learn **the** 'language' of the software engineering branch

- **Overview**:

✔ methodological and global view on software development

- **What not to do**

✔ avoid common errors and mistakes
✔ spotting critical points of requirements, avoid misunderstandings, etc.

- **Formal Methods**

✔ how can requirements be formalised to avoid misunderstandings
✔ ensure the feasibility of the solution
✔ how the quality of a design can be shown formally

Expectations: Yes \o/

- ✔ **UNDERSTAND** the areas of software development
- ✔ In the end, you have to **organise yourself**, nobody else can do that for you.

  We find it important to get **stimuli to think about** the importance of project management, quality assurance etc. and get examples **to see** that it is really important.

  The rest, we think, we can figure out on our own.

  "Dann kann man den Rest denke ich auch alleine schaffen."
- ✔ In a nutshell, We expect to **get prepared** for the future,

  and above all, to **have a good time**. :)

– 2 – 2019-04-29 – Sexpectations –

Content

– 2 – 2019-04-29 – Scontent –

Metrics

Vocabulary

metric — A quantitative measure of the degree to which a system, component, or process possesses a given attribute. See: quality metric. **IEEE 610.12 (1990)**

quality metric —

(1) A quantitative measure of the degree to which an item possesses a given quality attribute.

(2) A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute. **IEEE 610.12 (1990)**

Definition. A **metric**[1] is a function

$$m : P \to S$$

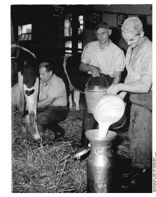
that assigns to each **proband** $p \in P$ a **valuation** ("Bewertung") $m(p) \in S$. We call S the **scale** of m.

[1]: in mathematics, a metric is something different (would be too easy otherwise…).

Metric Examples from Other Engineering Disciplines

- **Agricultural Engineering**: $m_a : P_a \to S_a$
  - **probands** P_a: milk samples;
    scale $S_a = [0, 100] \times [0, 100]$: percentage of fat and protein
  - "can be interpreted as (the degree of) [...] quality":
    milk sample p_a has acceptable quality
    if $m_a(p_a) \geq (4.0, 3.4)$ (4 % fat, 3.4 % protein)

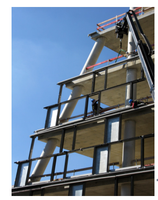
    (higher values are better: dairy may pay extra)



- **Railway Engineering**: $m_r : P_r \to S_r$
  - **probands** P_r: trams,
    scale $S_r = \mathbb{R}_0^+$: braking distance from 70 km/h to 0 km/h in m
  - "can be interpreted as (the degree of) [...] quality":
    tram p_r has acceptable evasive braking quality if $m_r(p_r) \leq 69$ (BOStrab)

    fun fact: a tram brake manufacturer may view $m_r(p_r) = 68$ as of lower overall quality

- **Construction Engineering**: $m_c : P_c \to S_c$
  - **probands** P_c: walls (length up to 3 m),
    scale $S_c = \mathbb{R}$: deviation from nominal in mm
  - "can be interpreted as (the degree of) [...] quality":
    wall p_c has acceptable dimension if $|m_c(p_c)| \leq 12$ (DIN 18202)



Common Uses of (Software) Metrics

- **Specify Product Properties**

  Example: The code should be written in MISRA-C.
  Metric: Number of MISRA-C violations (should be 0).

- **Assess Product Properties** / **Support Decisions**

  Example: The system is responsive for 100 concurrent users.
  Metric: average milliseconds between event and response
  (measure for 100 concurrent users; note: 'up to 100 users' is a different property).

- **Project Management**

  Example: Do not have too many open bug reports.
  Metric: Number of open bug reports (if above threshold, fix bugs before writing new code).

- **Predict** / **Estimate** / **Forecast**

  Example: Effort estimation for new project.
  Metric: Effort (in person-months); collect data from previous projects.

- **Research** / **State & Investigate Hypotheses**

  Example: The SWT course audience is not homogeneous regarding previous experience.

Common Uses of (Software) Metrics

- **Specify Product Properties**
- **Assess Product Properties** / **Support Decisions**
- **Project Management**
- **Predict** / **Estimate** / **Forecast**
- **Research** / **State & Investigate Hypotheses**

In other terms: Metrics can be used

- **prescriptive**, i.e. stating a **need** or **demand** on not yet existing software.

  Example: "The system to be developed needs to have a response time below 100 ms."
  (In order for the customers to accept and pay.)

- **descriptive**, i.e. stating a **diagnosed** or **prognosed** property of existing software.

  Examples:

  - **diagnostic** / **measured**:
    "The system has a response time of 50 ms." (Hence we meet the customers' needs.)
  - **prognostic** / **predicted**:
    "There are N open bug reports; if these bugs are all 'as usual', we expect to have all closed in M days."

- Note: **prescriptive** and **prognostic** are different things.

Desirable Properties of (Software) Metrics

In Order to be Useful, a Metric Should be …

- **relevant** wrt. overall goals and needs
- **plausible**: Good evidence that proband's valuations and quality are related
- **robust**: The valuation of a proband cannot be arbitrarily manipulated;
  antonym / opposite: subvertible
- **available**: Valuations need to be in place when needed
- **economical**: Cost of measuring needs to be in a good relation to gain

  Note: irrelevant metrics are not economical (if not available for free).

- **comparable**: Some scales have incomparable values ($\rightarrow$ later)
- **reproducible**: Multiple applications to the same proband yields the same valuation
- **differentiated**: Sufficiently different valuations for sufficiently different probands

Content

Software Metrics

Example Software Metrics

characteristic ('Merkmal')	positive example(s)	negative example(s)
relevant	expected development cost; number of errors	number of subclasses (NOC)
plausible	cost estimation following COCOMO (to a certain amount)	cyclomatic complexity of a program with pointer operations
robust	grading by experts	almost all pseudo-metrics (→ in three minutes)
available	number of developers	number of errors in the code (not only known ones)
economical	number of discovered errors in code	highly detailed timekeeping
comparable	cyclomatic complexity (→ in two minutes)	expert's review (in textual form)
reproducible	memory consumption	grade assigned by inspector
differentiated	program length in LOC (→ in a minute)	CMM/CMMI level below 2 (a process metric; → later)

(Ludewig and Lichter, 2013)

Example: Lines of Code (LOC)

```java
/* https://de.wikipedia.org/wiki/
 * Liste_von_Hallo-Welt-Programmen/
 * Höhere_Programmiersprachen#Java */

class Hallo {

  public static void
  main( String[] args ) {
    System.out.print(
      "Hallo Welt!" ); // no newline
  }
}
```

dimension	unit	measurement procedure	
program size	LOC_{tot}	number of lines in total	↪ 12
net program size	LOC_{ne}	number of non-empty lines	↪ 10
code size	LOC_{pars}	number of lines with not only comments and non-printable	↪ 7
delivered program size	$DLOC_{tot}$, $DLOC_{ne}$, $DLOC_{pars}$	LOC of only that code which is delivered to the customer	

(Ludewig and Lichter, 2013)

relevant	
plausible	
robust	
available	✓
economical	✓
comparable	✓
reproducible	✓
differentiated	✓

McCabe Complexity

complexity —

(1) The degree to which a system or component has a design or implementation that is difficult to understand and verify. Contrast with: simplicity.

(2) Pertaining to any of a set of structure-based metrics that measure the attribute in (1).

IEEE 610.12 (1990)

Definition. [*Cyclomatic Number [graph theory]*]
Let $G = (V, E)$ be a graph comprising **vertices** V and **edges** E.
The **cyclomatic number** of G is defined as

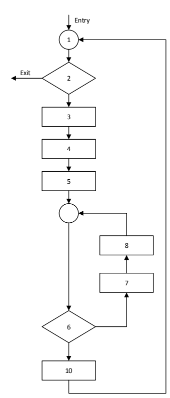
$$v(G) = |E| - |V| + 1.$$

Intuition: minimum number of edges to be removed to make G cycle free.

McCabe Complexity Cont'd

Definition. [*Cyclomatic Complexity [McCabe, 1976]*]
Let $G = (V, E)$ be the **Control Flow Graph** of program P.
Then the **cyclomatic complexity** of P is defined as $v(P) = |E| - |V| + p$ where p is the number of entry or exit points.

```
void insertionSort(int[] array) {
  for (int i = 2; i < array.length; i++) {
    tmp = array[i];
    array[0] = tmp;
    int j = i;
    while (j > 0 && tmp < array[j-1]) {
      array[j] = array[j-1];
      j--;
    }
    array[j] = tmp;
  }
}
```



Number of edges:	$\lvert E\rvert = 11$
Number of nodes:	$\lvert V\rvert = 6 + 2 + 2 = 10$
External connections:	$p = 2$

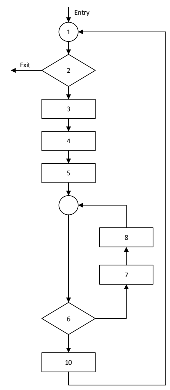
$\rightarrow v(P) = 11 - 10 + 2 = 3$

McCabe Complexity Cont'd

Definition. [*Cyclomatic Complexity [McCabe, 1976]*]
Let $G = (V, E)$ be the **Control Flow Graph** of program P.
Then the **cyclomatic complexity** of P is defined as $v(P) = |E| - |V| + p$ where p is the number of entry or exit points.

- **Intuition**: number of paths, number of decision points.
- **easy to compute**;
  Interval scale (not absolute, no zero due to $p > 0$);
- Somewhat **independent** from programming language.
- **Plausibility**:
  - \+ **loops and conditions** are harder to understand than sequencing.
  - – doesn't consider **data**.
- **Prescriptive** use:

  "For each procedure, either limit cyclomatic complexity to **[agreed-upon limit]** or provide written explanation of why limit exceeded."



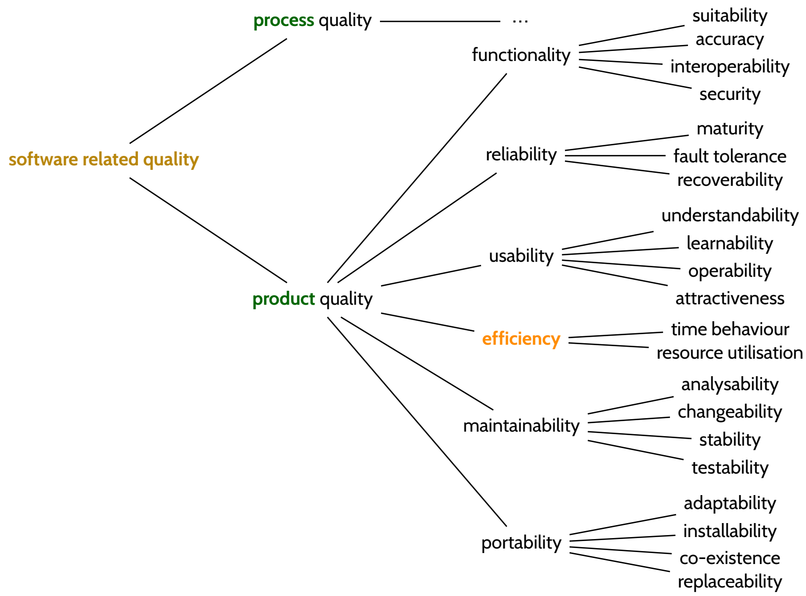
Code Metrics for OO Programs (Chidamber and Kemerer, 1994)

metric	computation
weighted methods per class (WMC)	$\sum_{i=1}^{n} c_i$, n = number of methods, c_i = complexity of method i
depth of inheritance tree (DIT)	graph distance in inheritance tree (what about multiple inheritance?)
number of children of a class (NOC)	number of direct subclasses of the class
coupling between object classes (CBO)	$CBO(C) = \lvert K_o \cup K_i \rvert$, K_o = set of classes used by C, K_i = set of classes using C
response for a class (RFC)	$RFC = \lvert M \cup \bigcup_{m \in M} R_m \rvert$, M = set of methods of C, R_m = set of all methods calling method m
lack of cohesion in methods (LCOM)	$\max(\lvert P \rvert - \lvert Q \rvert, 0)$, P = methods using no common attribute, Q = methods using at least one common attribute

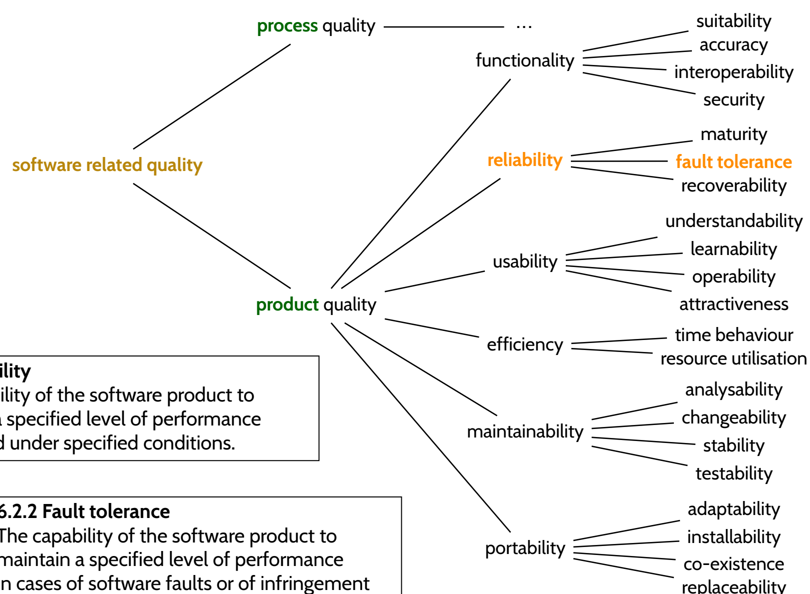
- **objective metrics**: DIT, NOC, CBO; **pseudo-metrics**: WMC, RFC, LCOM

  ... there seems to be agreement that it is far more important to focus on empirical validation (or refutation) of the proposed metrics than to propose new ones, ... (Kan, 2003)

Aspects of Software Quality *(cf. ISO/IEC 9126-1:2000 (2000))*



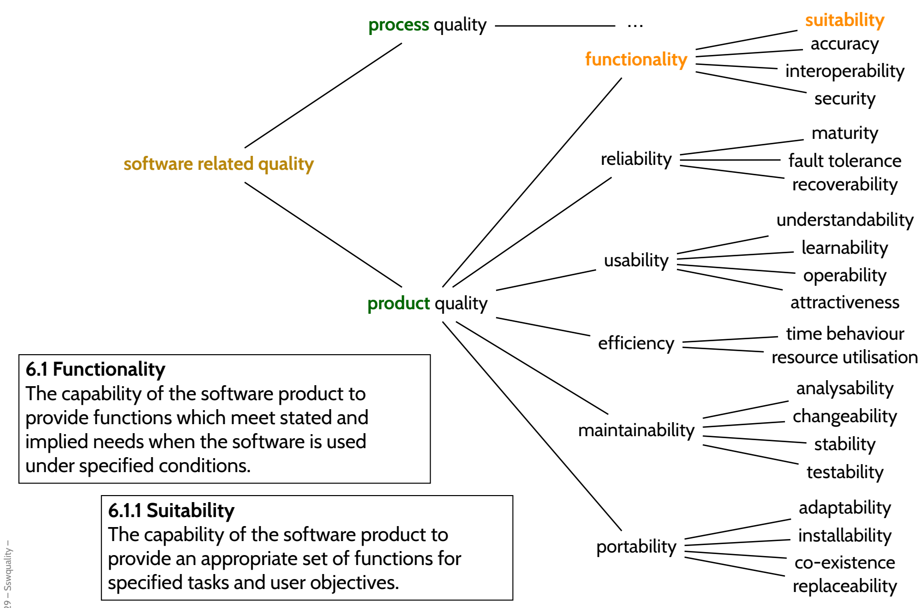
Aspects of Software Quality *(cf. ISO/IEC 9126-1:2000 (2000))*



6.2 Reliability
The capability of the software product to maintain a specified level of performance when used under specified conditions.

6.2.2 Fault tolerance
The capability of the software product to maintain a specified level of performance in cases of software faults or of infringement of its specified interface.

Aspects of Software Quality *(cf. ISO/IEC 9126-1:2000 (2000))*



– 2 – 2019-04-29 – Sswquality –

Content

- **Survey: Previous Experience and Expectations**
- **Software Metrics**
  - **Metrics**
    - Vocabulary, Examples from Other Disciplines
    - Common Uses of (Software) Metrics
    - Desirable Properties of (Software) Metrics
  - **Software Metrics**
    - Properties of Some Software Metrics
    - Examples: LOC, McCabe
  - **Software Metrics Issues**
    - Base vs. Derived Measures, Excursion: Scales
    - Objective, Subjective, Pseudo
    - Practical Software Metrics
- **Cost Estimation**
  - (Software) Economics in a Nutshell
  - Software Cost Estimation
    - Expert's Estimation (Delphi Method)
    - Algorithmic Estimation (COCOMO, Function Points)

– 2 – 2019-04-29 – Scontent –

Software Metric Issues

Kinds of Metrics: ISO/IEC 15939:2011

base measure — measure defined in terms of an attribute and the method for quantifying it.
ISO/IEC 15939 (2011)

derived measure — measure that is defined as a function of two or more values of base measures. **ISO/IEC 15939 (2011)**

Examples:

- lines of code,
- hours spent on testing,
- execution time,
- …

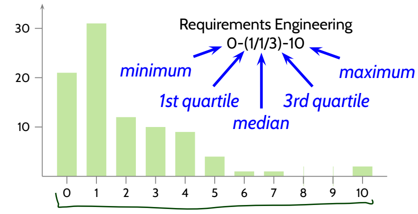
Examples:

- average or median of lines of code,
- productivity (in lines per hour),
- …

- Derived measures are **easier to get wrong**, i.e., to not measure the intended property.

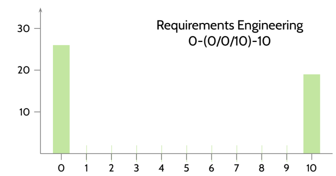
  → be **extra careful** with derived metrics/measures

Issues with Scales I: People Like Aggregated Data



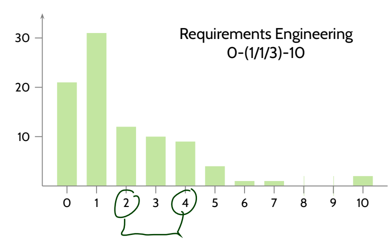
- **1st quartile**: 25 % of the values are below-or-equal
- **2nd quartile** or **median**: 50 % of the values are below-or-equal
- **3nd quartile**: 75 % of the values are below-or-equal

- **Issue**: There are scales on which **quartiles are not defined**. For example: program of studies.
- **Issue**: How would data with arithmetic average, mean 3.725 look like? For example, like this:



→ when aggregating data **with defined quartiles and mean**, aggregate carefully.

Issues with Scales II: People Like to Compare Data



- How much better **exactly** is response '4' compared to response '2'?
- **We cannot tell!** The scale is only ordinal.

Scales and Types of Scales

Scales S can be distinguished by supported **operations**:

	$=, \neq$	$<, >$ (with transitivity)	$\min$, $\max$	percentiles, e.g. median	Δ	proportion	natural 0 (zero)
nominal scale	✔	✘	✘	✘	✘	✘	✘
ordinal scale	✔	✔	✔	✔	✘	✘	✘
interval scale (with units)	✔	✔	✔	✔	✔	✘	✘
rational scale (with units)	✔	✔	✔	✔	✔	✔	✔
absolute scale	a rational scale where S comprises the key figures itself						

Examples: **Nominal Scale**

- nationality, gender, car manufacturer, geographic direction, train number, ...
- **Software engineering example**: programming language ($S = \{\texttt{Java}, \texttt{C}, \dots\}$)

→ There is no (natural) order between elements of S; the lexicographic order can be imposed ("$\texttt{C} < \texttt{Java}$"), but is not related to the measured information (thus not natural).

Scales and Types of Scales

Scales S can be distinguished by supported **operations**:

	$=, \neq$	$<, >$ (with transitivity)	$\min$, $\max$	percentiles, e.g. median	Δ	proportion	natural 0 (zero)
nominal scale	✔	✘	✘	✘	✘	✘	✘
ordinal scale	✔	✔	✔	✔	✘	✘	✘
interval scale (with units)	✔	✔	✔	✔	✔	✘	✘
rational scale (with units)	✔	✔	✔	✔	✔	✔	✔
absolute scale	a rational scale where S comprises the key figures itself						

Examples: **Ordinal Scale**

- strongly agree $>$ agree $>$ disagree $>$ strongly disagree; Chancellor $>$ Minister (administrative ranks);
- leaderboard (finishing number tells us that 1st was faster than 2nd, but not how much faster)
- types of scales, ...
- **Software engineering example**: CMMI scale (maturity levels 1 to 5) (→ later)

→ There is a (natural) **order** between elements of M, but no (natural) notion of **distance** or **average**.

Scales and Types of Scales

Scales S can be distinguished by supported **operations**:

	$=, \neq$	$<, >$ (with transitivity)	$\min$, $\max$	percen-tiles, e.g. median	Δ	propor-tion	natural 0 (zero)
nominal scale	✔	✘	✘	✘	✘	✘	✘
ordinal scale	✔	✔	✔	✔	✘	✘	✘
interval scale (with units)	✔	✔	✔	✔	✔	✘	✘
rational scale (with units)	✔	✔	✔	✔	✔	✔	✔
absolute scale	a rational scale where S comprises the key figures itself						

Examples: **Interval Scale**

- temperature in Fahrenheit
  - "today it is 10°F warmer than yesterday" ($\Delta(\vartheta_{\text{today}}, \vartheta_{\text{yesterday}}) = 10$°F)
  - "100°F is twice as warm as 50°F": …? No. Note: the zero is arbitrarily chosen.
- **Software engineering example**: time of check-in in revision control system

→ There is a (natural) notion of difference $\Delta : S \times S \to \mathbb{R}$, but no (natural) proportion and 0.

Scales and Types of Scales

Scales S can be distinguished by supported **operations**:

	$=, \neq$	$<, >$ (with transitivity)	$\min$, $\max$	percen-tiles, e.g. median	Δ	propor-tion	natural 0 (zero)
nominal scale	✔	✘	✘	✘	✘	✘	✘
ordinal scale	✔	✔	✔	✔	✘	✘	✘
interval scale (with units)	✔	✔	✔	✔	✔	✘	✘
rational scale (with units)	✔	✔	✔	✔	✔	✔	✔
absolute scale	a rational scale where S comprises the key figures itself						

Examples: **Rational Scale**

- age ("twice as old"); finishing time; weight; pressure; price; speed; distance from Freiburg…
- **Software engineering example**: runtime of a program for given inputs.

→ The (natural) zero induces a meaning for proportion m_1/m_2.

Scales and Types of Scales

Scales S can be distinguished by supported **operations**:

	$=, \neq$	$<, >$ (with transitivity)	min, max	percen-tiles, e.g. median	Δ	propor-tion	natural 0 (zero)
nominal scale	✔	✘	✘	✘	✘	✘	✘
ordinal scale	✔	✔	✔	✔	✘	✘	✘
interval scale (with units)	✔	✔	✔	✔	✔	✘	✘
rational scale (with units)	✔	✔	✔	✔	✔	✔	✔
absolute scale	a rational scale where S comprises the key figures itself						

Examples: Absolute Scale

- seats in a bus, number of public holidays, number of inhabitants of a country, ...
- "average number of children per family: 1.203" – what is a 0.203-child?
  The absolute scale has been **used as** a rational scale (makes sense for certain purposes if done with care).
- **Software engineering example**: number of known errors.

→ An absolute scale has a **median**, but in general not an average **in** the scale.

– 2 – 2019-04-29 – Scales –

Kinds of Metrics: by Measurement Procedure

	objective metric	pseudo metric	subjective metric
Procedure	measurement, counting, possibly standardised	computation (based on measurements or assessment)	review by inspector, verbal or by given scale
Example, general	body height, air pressure	body mass index (BMI), tomorrow's weather forecast	health condition, weather condition ("bad weather")
Example in Software Engineering	size in LOC or NCSI; number of (known) bugs	productivity as LOC/h; COCOMO cost estimate; ***judge Software Engineer by SWT course grade***	usability; severeness of an error
Usually used for	collection of simple base measures	predictions (cost estimation); overall assessments	quality assessment; error weighting
Advantages	exact, reproducible, can be obtained automatically	yields relevant, directly usable statement on not directly visible characteristics	not subvertible, plausible results, applicable to complex characteristics
Disadvantages	not always relevant, often subvertible, no interpretation	hard to comprehend, pseudo-objective, ***does not actually measure what it promises***	assessment costly, quality of results depends on inspector

(Ludewig and Lichter, 2013)

– 2 – 2019-04-29 – Sobsubpseudo –

Pseudo-Metrics

For many of the **most relevant aspects** of software development projects, such as:

- how **maintainable** is the software?
- how much **effort** is needed until completion?
- does the product have good **usability**?
- **documentation** sufficient and well usable?

(today) **we do not have** good objective metrics.

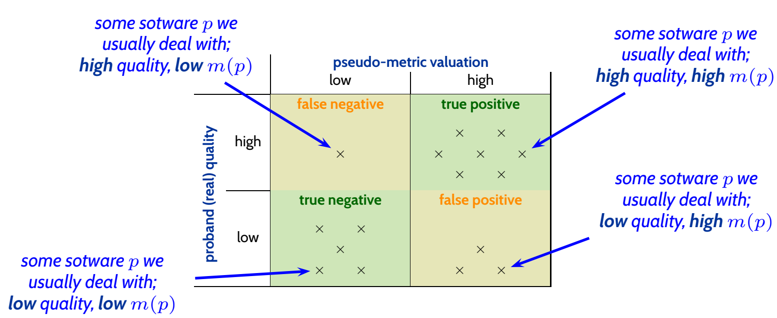
Two choices left: **subjective** or **pseudo** metrics.

	plausible	robust	available	economical	comparable	reproducible	differentiated
Subjective Metrics	✔	✔	(✔)	(✘)	(✔)	(✘)	(✔)
Pseudo Metrics	(✘)	(✘)	✔	✔	✔	✔	✔

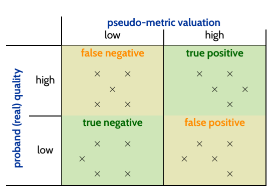
Note: Not every derived measure is a pseudo-metric:

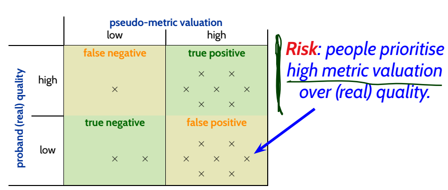
- measure **average LOC per module**:
  derived, **not pseudo** (we really measure average LOC per module).
- measure **maintainability** in average LOC per module:
  derived, **pseudo** (we do not really **measure** maintainability; average-LOC is only **interpreted** as such.)
  Plus: Not robust if easily subvertible (see exercises).

Useful and Non-Useful Pseudo-Metrics



- Useful: a pseudo-metric m with **good correlation** between proband quality and metric valuation **for our usual probands**!
- Not Useful: pretty random (left); too many false positives (right)





Content

- **Survey**: **Previous Experience** and **Expectations**
- **Software Metrics**
  - **Metrics**
    - Vocabulary, Examples from Other Disciplines
    - Common Uses of (Software) Metrics
    - Desirable Properties of (Software) Metrics
  - **Software Metrics**
    - Properties of Some Software Metrics
    - Examples: LOC, McCabe
  - **Software Metrics Issues**
    - Base vs. Derived Measures, Excursion: Scales
    - Objective, Subjective, Pseudo
    - Practical Software Metrics
- **Cost Estimation**
  - (Software) Economics in a Nutshell
  - Software Cost Estimation
    - Expert's Estimation (Delphi Method)
    - Algorithmic Estimation (COCOMO, Function Points)

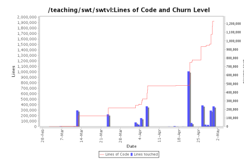
Practical Software Metrics

Which Metrics Should We Use?

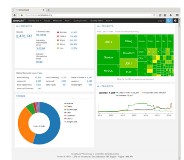
- **Approach**:
  Understand what we **need to know**, then choose / develop metrics that measure that.
  For example, **Goal-Question-Metric** (GQM) (Basili and Weiss, 1984):
  (i) **Identify** the **goals** relevant for project or organisation.
  (ii) From each goal, **derive questions**
  that need to be answered to see whether the goal is reached.
  (iii) For each question, **choose** (or develop) **metrics** that contribute to finding answers.

- **Often useful**:
  Collect some basic measures **continuously**
  (in particular if collection is cheap), e.g.:
  - **size** of ...newly created and changed code, etc.
  - **effort** for ...coding, review, testing, verification, fixing, maintenance, etc.
  - **number of errors** ...found during quality assurance, corrected, reported by customer, etc.

  Know usual valuations and **keep an eye** on current measures over time:
  Unusual values may **indicate problems**;
  investigate further (possibly with other metrics).



LOC and changed lines over time (obtained by `statsvn`(1)).



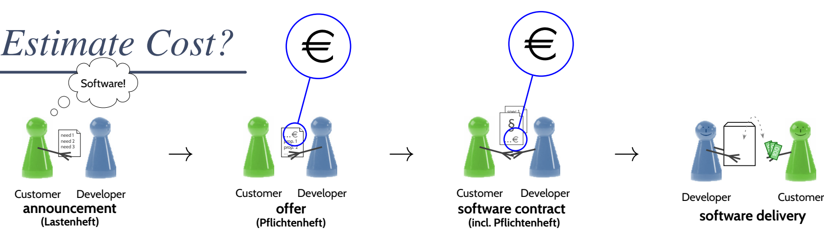
Tool support for software metrics, e.g., SonarCube.

Content

- **Survey**: **Previous Experience** and **Expectations**
- **Software Metrics**
  - **Metrics**
    - Vocabulary, Examples from Other Disciplines
    - Common Uses of (Software) Metrics
    - Desirable Properties of (Software) Metrics
  - **Software Metrics**
    - Properties of Some Software Metrics
    - Examples: LOC, McCabe
  - **Software Metrics Issues**
    - Base vs. Derived Measures, Excursion: Scales
    - Objective, Subjective, Pseudo
    - Practical Software Metrics
- **Cost Estimation**
  - (Software) Economics in a Nutshell
  - Software Cost Estimation
    - Expert's Estimation (Delphi Method)
    - Algorithmic Estimation (COCOMO, Function Points)

(Software) Economics in a Nutshell

Why Estimate Cost?



Lastenheft (Requirements Specification) Vom Auftraggeber festgelegte Gesamtheit der Forderungen an die Lieferungen und Leistungen eines Auftragnehmers innerhalb eines Auftrages.

(Entire demands on deliverables and services of a developer within a contracted development, created by the customer.) **DIN 69901-5 (2009)**

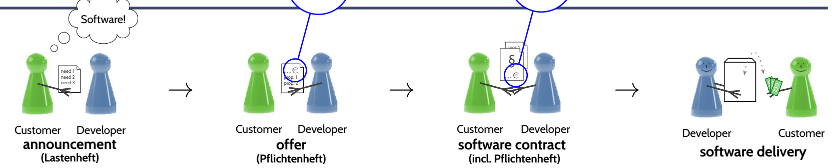
Pflichtenheft (Feature Specification) Vom Auftragnehmer erarbeitete Realisierungsvorgaben aufgrund der Umsetzung des vom Auftraggeber vorgegebenen Lastenhefts.

(Specification of how to realise a given requirements specification, created by the developer.)

DIN 69901-5 (2009)

Speaking of Lastenheft / Pflichtenheft:

- If customer is, e.g., lacking technical background or time.
  developer side can **help with writing** the requirements specification.
- Creating the feature specification can be **a project** on its own
  (may be subject of a designated contract, then needs cost estimation...).
- **Tricky** / **Confusing**: one and the same content can serve both purposes;
  then only the title defines the purpose.
  In other words: Lastenheft / Pflichtenheft is not a property of content.



- Usual developer side's view:
  Maximise profit, i.e. maximise difference between **benefit** and **cost**.

cost ('Kosten') (or: positive costs)	all disadvantages of a solution.
benefit ('Nutzen') (or: negative costs)	all benefits of a solution.

Note: cost / benefit may be subjective — and not necessarily quantifiable in terms of money...

> **Software Engineering** — the establishment and use of sound engineering principles to obtain economically software that is reliable and works efficiently on real machines.
> **F. L. Bauer (1971)**

"Next to '**Software**', '**Cost**' is one of the terms occurring most often in this book."
Ludewig and Lichter (2013)

Software Cost Estimation

In the end, it's **experience**, **experience**, **experience**:

> "Estimate, document, estimate better." (Ludewig and Lichter, 2013)

Content

- Software Metrics
  - A (software) **metric**
    - Is a **quantitative** measure on software data.
    - Has **valuations** that can be **interpreted** as degree-of-quality.
    - Can be used **prescriptive** or **descriptive** (**diagnostic** or **prognostic**).
    - Measuring material goods (milk, trams, walls, etc.) is often **much easier** than measuring immaterial goods (like software).
  - **Look out for relevant**, **plausible**, **robust**, **economical** metrics.
  - **Be careful with** derived measures and pseudo-metrics.
- Cost Estimation
  - F. L. Bauer: "**[...] obtain software economically [...]**"
  - It's about **experience** (and based on data obtained with metrics), and often a **well-kept business secret**.
  - Distinguish **Expert's** and **Algorithmic** Cost Estimation.
  - Algorithmic Cost Estimations "just" **shift** the estimation.
  - Cost estimation is **everywhere** ($\rightarrow$ tutorials).

– 2 – 2019-04-29 – Sttwytt –

References

– 2 – 2019-04-29 – main –

Basili, V. R. and Weiss, D. M. (1984). A methodology for collecting valid software engineering data. *IEEE Transactions of Software Engineering*, 10(6):728–738.

Bauer, F. L. (1971). Software engineering. In *IFIP Congress (1)*, pages 530–538.

Boehm, B. W. (1981). *Software Engineering Economics*. Prentice-Hall.

Boehm, B. W., Horowitz, E., Madachy, R., Reifer, D., Clark, B. K., Steece, B., Brown, A. W., Chulani, S., and Abts, C. (2000). *Software Cost Estimation with COCOMO II*. Prentice-Hall.

Chidamber, S. R. and Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493.

DIN (2009). *Projektmanagement; Projektmanagementsysteme*. DIN 69901-5.

IEEE (1990). *IEEE Standard Glossary of Software Engineering Terminology*. Std 610.12-1990.

ISO/IEC (2011). *Information technology – Software engineering – Software measurement process*. 15939:2011.

ISO/IEC FDIS (2000). *Information technology – Software product quality – Part 1: Quality model*. 9126-1:2000(E).

Kan, S. H. (2003). *Metrics and models in Software Quality Engineering*. Addison-Wesley, 2nd edition.

Knöll, H.-D. and Busse, J. (1991). *Aufwandsschätzung von Software-Projekten in der Praxis: Methoden, Werkzeugeinsatz, Fallbeispiele*. Number 8 in Reihe Angewandte Informatik. BI Wissenschaftsverlag.

Ludewig, J. and Lichter, H. (2013). *Software Engineering*. dpunkt.verlag, 3. edition.

Noth, T. and Kretzschmar, M. (1984). *Aufwandsschätzung von DV-Projekten, Darstellung und Praxisvergleich der wichtigsten Verfahren*. Springer-Verlag.

Wheeler, D. A. (2006). Linux kernel 2.6: It's worth more!