Softwaretechnik / Software-Engineering

Lecture 2: Software Metrics

2017-04-27

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany

Topic Area Project Management: Content

- VL 2
  - Software Metrics
    - Properties of Metrics
    - Scales
    - Examples

- VL 3
  - Cost Estimation
    - Deadlines and Costs
    - Expert’s Estimation
    - Algorithmic Estimation

- VL 4
  - Project Management
    - Project
    - Process and Process Modelling

- VL 5
  - Process Metrics
    - CMMI, Spice
Survey: Previous Experience

1. Project Management
2. Requirements Engineering
3. Programming
4. Design Modelling
5. Software Quality Assurance
## Expectations

- **general**
  - ✔ work with others in a large software development team
  - ✔ communicate results to other people
  - ✔ learn how to properly document the work
  - ✔ know, how to acquire knowledge on aspects of SW Eng. on our own
  - ✔ get to know industry standards, investigate their strengths / weaknesses
  - ✔ overview, terminology, and references for own enquiries
  - ✘ know about trustful internet sources to get such information while working
  - ✔ understand the procedure of software production, including common mishaps at each step
  - ✔ systematically analyse the steps of software development which are done “implicitly” in smaller, self-made projects
  - ✔ course is balanced with theoretical as well as practical scenarios
  - ✔ getting tools (roughly specific ideas) for attacking problems
  - ✔ have some fun, learn a lot [...] not only for the further studying or working but also for life

- **other courses**
  (✘) Vor allem hoffe ich auf eine sinnvolle Verbindung zum Softwarepraktikum.

## Expectations Cont’d

- **project management**
  - ✔ minimize risks, estimate project duration,
  - ✔ how to estimate cost/time, without resorting to years of experience
  - ✔ different life stages of a software
  - ✔ become acquainted with the most common procedures of software development
  - ✔ selection of right process for a project
  - ✔ learn how things are done in real companies

- **requirements**
  - ✔ How to communicate between customer and software team effectively
  - ✔ formalise software engineering problems
  - ✔ learn how to specify the requirements
  - ✔ how to write something based on customer’s wishes, which is unambiguous (for the programmers), but understandable for the customer, such that the customers can check on their own what is meant.
**Expectations Cont’d**

- **design**
  - ✔ techniques and vocabulary to express design
  - ✔ learn how to use basic and maybe some advanced techniques, models and patterns in software development
  - ✔ the modern techniques: [...] Test Driven Design, Behaviour Driven Design
  - ✔ acquire knowledge in UML
  - ✔ principles of reasonable software architectures
  - (✘) verification of architectures
  - (✔) what distinguished well-designed SW from bad-designed ones
  - (✘) how to quantify and check things like “good usability”
  - (✘) focus on software architecture

- **Implementation**
  - (✘) write reusable and maintainable code
  - (✘) knowing the adequate codes for the certain software

- **Quality Assurance**
  - (✔) Which software qualities are more important for different types of SW?
  - (✘) test code in a reusable efficient way
  - (✔) extend my basic knowledge on verification methods (unit tests etc.)
  - (✘) conduct a review

---

**Content**

- **Survey: Expectations on the Course**
- **Software Metrics**
  - Motivation
  - Vocabulary
  - Requirements on Useful Metrics
  - Excursion: Scales
    - Excursion Excursion: Mean, Median, Quartiles
  - Example: LOC
  - Other Properties of Metrics
    - Base Measures vs. Derived Measures
    - Subjective and Pseudo Metrics
  - Discussion
## Engineering vs. Non-Engineering

<table>
<thead>
<tr>
<th></th>
<th>workshop (technical product)</th>
<th>studio (artwork)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mental prerequisite</td>
<td>the existing and available technical know-how</td>
<td>artist’s inspiration, among others</td>
</tr>
<tr>
<td>Deadlines</td>
<td>can usually be planned with sufficient precision</td>
<td>cannot be planned due to dependency on artist’s inspiration</td>
</tr>
<tr>
<td>Price</td>
<td>oriented on cost, thus calculable</td>
<td>determined by market value, not by cost</td>
</tr>
<tr>
<td>Norms and standards</td>
<td>exist, are known, and are usually respected</td>
<td>are rare and, if known, not respected</td>
</tr>
<tr>
<td>Evaluation and comparison</td>
<td>can be conducted using objective, quantified</td>
<td>is only possible subjectively. Results are disputed</td>
</tr>
<tr>
<td>Author</td>
<td>remains anonymous, often lacks emotional ties to the product</td>
<td>considers the artwork as part of him/herself</td>
</tr>
<tr>
<td>Warranty and liability</td>
<td>are clearly regulated, cannot be excluded</td>
<td>are not defined and in practice hardly enforceable</td>
</tr>
</tbody>
</table>

Ludewig and Lichter, 2013
**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)

---

**Software Metrics: Motivation and Goals**

Important motivations and goals for using software metrics:

- specify quality requirements
- assess the quality of products and processes
- quantify experience, progress, etc.
- predict cost/effort, etc.
- support decisions

Software metrics can be used:

- prescriptive, e.g., "all procedures must not have more than \( N \) parameters", or
- descriptive, e.g., "procedure \( P \) has \( N \) parameters"

A descriptive metric can be

- diagnostic, e.g., "the test effort was \( N \) hours", or
- prognostic, e.g., "the expected test effort is \( N \) hours".

Note: prescriptive and prognostic are different things.

- Examples: support decisions by diagnostic measurements:
  
  (i) Measure CPU time spent per procedure, then "optimize" most time consuming procedure.
  
  (ii) Measure attributes which indicate architecture problems, then re-factor accordingly.

**Useful Metrics**

- For **material goods**, useful metrics are often pretty obvious:

- Not so obvious for **immaterial goods**, like software.
Requirements on Useful Metrics

Definition. A software metric is a function \( m : P \to S \) which assigns to each proband \( p \in P \) a valuation yield \( (\text{'}Bewertung\text{'}) m(p) \in S \). We call \( S \) the scale of \( m \).

In order to be useful, a (software) metric should be:

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>differentiated</td>
<td>worst case: same valuation yield for all probands</td>
</tr>
<tr>
<td>comparable</td>
<td>ordinal scale, better: rational (or absolute) scale ( \rightarrow ) in a minute</td>
</tr>
<tr>
<td>reproducible</td>
<td>multiple applications of a metric to the same proband should yield the same valuation</td>
</tr>
<tr>
<td>available</td>
<td>valuation yields need to be in place when needed</td>
</tr>
<tr>
<td>relevant</td>
<td>wrt. overall needs</td>
</tr>
<tr>
<td>economical</td>
<td>worst case: doing the project gives a perfect prognosis of project duration – at a high price; irrelevant metrics are not economical (if not available for free)</td>
</tr>
<tr>
<td>plausible</td>
<td>( \rightarrow ) pseudo-metric</td>
</tr>
<tr>
<td>robust</td>
<td>developers cannot arbitrarily manipulate the yield; antonym: subvertible</td>
</tr>
</tbody>
</table>
Excursion: Scales

Scales and Types of Scales

Scales \( S \) are distinguished by supported operations:

<table>
<thead>
<tr>
<th></th>
<th>( =, \neq )</th>
<th>(&lt;, &gt;) (with transitivity)</th>
<th>min. max</th>
<th>percentiles, e.g. median</th>
<th>(\Delta)</th>
<th>proportion</th>
<th>natural 0 (zero)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nominal scale</td>
<td>✔</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>ordinal scale</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>interval scale (with units)</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>rational scale (with units)</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>absolute scale</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

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 \( S = \{ \text{Java, C}, \ldots \} \)

→ There is no (natural) order between elements of \( S \); the lexicographic order can be imposed ("C < Java"), but is not related to the measured information (thus not natural).
Scales $S$ are distinguished by supported operations:

<table>
<thead>
<tr>
<th>Scales</th>
<th>$=$, $\neq$</th>
<th>$&lt;$, $&gt;$ (with transitivity)</th>
<th>min, max</th>
<th>percentiles, e.g. median</th>
<th>$\Delta$</th>
<th>proportion</th>
<th>natural 0 (zero)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nominal scale</td>
<td>✔</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ordinal scale</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>interval scale  (with units)</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>rational scale  (with units)</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>absolute scale</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

Examples: Ordinal Scale

- strongly agree $\succ$ agree $\succ$ disagree $\succ$ strongly disagree: Chancellor $\succ$ 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 $S$ are distinguished by supported operations:

Examples: Interval Scale

- temperature in Fahrenheit
  - “today it is $10^\circ F$ warmer than yesterday” ($\Delta(\vartheta_{\text{today}}, \vartheta_{\text{yesterday}}) = 10^\circ F$)
  - “$100^\circ F$ is twice as warm as $50^\circ F$”…? 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 $\Delta : S \times S \rightarrow \mathbb{R}$, but no (natural) proportion and 0.
Scales and Types of Scales

Scales $S$ are distinguished by supported operations:

<table>
<thead>
<tr>
<th>Nominal scale</th>
<th>Ordinal scale</th>
<th>Interval scale</th>
<th>Rational scale</th>
<th>Absolute scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>$=, \neq$</td>
<td>$=, \neq$</td>
<td>$=, \neq$</td>
<td>$=, \neq$</td>
<td>$=, \neq$</td>
</tr>
<tr>
<td>$&lt;, &gt;$ (with transitivity)</td>
<td>$&lt;, &gt;$ (with transitivity)</td>
<td>$&lt;, &gt;$ (with transitivity)</td>
<td>$&lt;, &gt;$ (with transitivity)</td>
<td>$&lt;, &gt;$ (with transitivity)</td>
</tr>
<tr>
<td>$\min$, $\max$</td>
<td>$\min$, $\max$</td>
<td>$\min$, $\max$</td>
<td>$\min$, $\max$</td>
<td>$\min$, $\max$</td>
</tr>
<tr>
<td>$\text{percentiles, e.g. median}$</td>
<td>$\text{percentiles, e.g. median}$</td>
<td>$\text{percentiles, e.g. median}$</td>
<td>$\text{percentiles, e.g. median}$</td>
<td>$\text{percentiles, e.g. median}$</td>
</tr>
<tr>
<td>$\Delta$</td>
<td>$\Delta$</td>
<td>$\Delta$</td>
<td>$\Delta$</td>
<td>$\Delta$</td>
</tr>
<tr>
<td>Proportion</td>
<td>Proportion</td>
<td>Proportion</td>
<td>Proportion</td>
<td>Proportion</td>
</tr>
<tr>
<td>Natural zero (zero)</td>
<td>Natural zero (zero)</td>
<td>Natural zero (zero)</td>
<td>Natural zero (zero)</td>
<td>Natural zero (zero)</td>
</tr>
</tbody>
</table>

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 $S$ are distinguished by supported operations:

<table>
<thead>
<tr>
<th>Nominal scale</th>
<th>Ordinal scale</th>
<th>Interval scale</th>
<th>Rational scale</th>
<th>Absolute scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>$=, \neq$</td>
<td>$=, \neq$</td>
<td>$=, \neq$</td>
<td>$=, \neq$</td>
<td>$=, \neq$</td>
</tr>
<tr>
<td>$&lt;, &gt;$ (with transitivity)</td>
<td>$&lt;, &gt;$ (with transitivity)</td>
<td>$&lt;, &gt;$ (with transitivity)</td>
<td>$&lt;, &gt;$ (with transitivity)</td>
<td>$&lt;, &gt;$ (with transitivity)</td>
</tr>
<tr>
<td>$\min$, $\max$</td>
<td>$\min$, $\max$</td>
<td>$\min$, $\max$</td>
<td>$\min$, $\max$</td>
<td>$\min$, $\max$</td>
</tr>
<tr>
<td>$\text{percentiles, e.g. median}$</td>
<td>$\text{percentiles, e.g. median}$</td>
<td>$\text{percentiles, e.g. median}$</td>
<td>$\text{percentiles, e.g. median}$</td>
<td>$\text{percentiles, e.g. median}$</td>
</tr>
<tr>
<td>$\Delta$</td>
<td>$\Delta$</td>
<td>$\Delta$</td>
<td>$\Delta$</td>
<td>$\Delta$</td>
</tr>
<tr>
<td>Proportion</td>
<td>Proportion</td>
<td>Proportion</td>
<td>Proportion</td>
<td>Proportion</td>
</tr>
<tr>
<td>Natural zero (zero)</td>
<td>Natural zero (zero)</td>
<td>Natural zero (zero)</td>
<td>Natural zero (zero)</td>
<td>Natural zero (zero)</td>
</tr>
</tbody>
</table>

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.
**Something for the Mathematicians...**

Recall:

Definition. [Metric Space (math.)]

Let $X$ be a set. A function $d : X \times X \to \mathbb{R}$ is called **metric** on $X$ if and only if, for each $x, y, z \in X$,

1. $d(x, y) \geq 0$ (non-negative)
2. $d(x, y) = 0 \iff x = y$ (identity of indiscernibles)
3. $d(x, y) = d(y, x)$ (symmetry)
4. $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

$(X, d)$ is called **metric space**.

→ different from all scales discussed before; a metric space requires more than a rational scale.

→ definitions of, e.g., IEEE 610.12, may use standard (math.) names for different things

**Something for the Computer Scientists...**

- **A function** which
  - assigns to each **algorithm** (or problem, or program)
  - a **complexity class**
    (worst-, average-, best-case; deterministic, non-deterministic; space, time; ...),
  - can be seen as a metric (according to our earlier definition):
    - **probands** $P$: set of algorithms (or problems, or programs)
    - **scale** $S$: problem classes like $O(N)$.

**Example:**

- Problem $p$: “does element $E$ occur in unsorted, finite list $L$?”
- Complexity metric (worst-case; deterministic; time):
  - $p$ is in $O(N)$, $N = |L|$ (length of list).

→ the McCabe metric (in a minute) is sometimes called complexity metric
  (in the rough sense of “complicatedness”).

→ descriptions of software metrics may use standard (comp. sc.) names for different things.
**Project Management: Metrics on People**

**Definition.** A **software metric** is a function \( m : P \rightarrow S \) which assigns to each **proband** \( p \in P \) a **valuation yield** (“Bewertung”) \( m(p) \in S \). We call \( S \) the **scale** of \( m \).

- Here: \( P \) is the set of participants in the survey of the course “Software Engineering”.
- Scale: \( S = \{0, \ldots, 10\} \) (ordinal scale; has \( = \) and \( \neq \), \( < \) and \( > \), \( \min \) and \( \max \)).
- **Arithmetic mean**: 2.284 (not in the scale!)
- **Minimum and maximum**: 0 and 10
- **Median**: 1 (the value such that 50% of the probands have yields below and above)
- **1st and 3rd Quartile**: 1 and 4

A boxplot visualises 5 aspects of data at once (whiskers sometimes defined differently):
2017 vs. 2016

- Management 2017
  - Median: 1
  - Average: 2.2069

- Management 2016
  - Median: 1
  - Average: 2.284

- RE Experience 2017
  - Median: 1
  - Average: 2.0909

- RE Experience 2016
  - Median: 1
  - Average: 3.9432

- Programming 2017
  - Median: 3
  - Average: 3.7922

- Programming 2016
  - Median: 3
  - Average: 2.1932

- Modelling 2017
  - Median: 1
  - Average: 1.4459

- Modelling 2016
  - Median: 1
  - Average: 2.5682

- QA 2017
  - Median: 1
  - Average: 2.3766

- QA 2016
  - Median: 1
  - Average: 2.3766

Back From Excursion: Scales
Requirements on Useful Metrics

In order to be useful, a (software) metric should be:

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>differentiated</td>
<td>worst case: same valuation yield for all probands</td>
</tr>
<tr>
<td>comparable</td>
<td>ordinal scale, better: rational (or absolute) scale</td>
</tr>
<tr>
<td>reproducible</td>
<td>multiple applications of a metric to the same proband should yield the same valuation</td>
</tr>
<tr>
<td>available</td>
<td>valuation yields need to be in place when needed</td>
</tr>
<tr>
<td>relevant</td>
<td>wrt. overall needs</td>
</tr>
<tr>
<td>economical</td>
<td>worst case: doing the project gives a perfect prognosis of project duration – at a high price; irrelevant metrics are not economical (if not available for free)</td>
</tr>
<tr>
<td>plausible</td>
<td>[\rightarrow] pseudo-metric</td>
</tr>
<tr>
<td>robust</td>
<td>developers cannot arbitrarily manipulate the yield; antonym: subvertible</td>
</tr>
</tbody>
</table>
Example: Lines of Code (LOC)

<table>
<thead>
<tr>
<th>dimension</th>
<th>unit</th>
<th>measurement procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>program size</td>
<td>LOC_tot</td>
<td>number of lines in total</td>
</tr>
<tr>
<td>net program size</td>
<td>LOC_ne</td>
<td>number of non-empty lines</td>
</tr>
<tr>
<td>code size</td>
<td>LOC_pars</td>
<td>number of lines with not only comments and non-printable</td>
</tr>
<tr>
<td>delivered program</td>
<td>DLOC_tot</td>
<td>like LOC, only code (as source or compiled) given to customer</td>
</tr>
<tr>
<td>program size</td>
<td>DLOC_ne</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DLOC_pars</td>
<td></td>
</tr>
</tbody>
</table>

([Ludewig and Lichter, 2013])

More Examples

<table>
<thead>
<tr>
<th>characteristic ('Merkmal')</th>
<th>positive example</th>
<th>negative example</th>
</tr>
</thead>
<tbody>
<tr>
<td>differentiated</td>
<td>program length in LOC</td>
<td>CMM/CMMI level below 2</td>
</tr>
<tr>
<td>comparable</td>
<td>cyclomatic complexity</td>
<td>review (text)</td>
</tr>
<tr>
<td>reproducible</td>
<td>memory consumption</td>
<td>grade assigned by inspector</td>
</tr>
<tr>
<td>available</td>
<td>number of developers</td>
<td>number of errors in the code (not only known ones)</td>
</tr>
<tr>
<td>relevant</td>
<td>expected development cost; number of errors</td>
<td>number of subclasses (NOC)</td>
</tr>
<tr>
<td>economical</td>
<td>number of discovered errors in code</td>
<td>highly detailed timekeeping</td>
</tr>
<tr>
<td>plausible</td>
<td>cost estimation following COCOMO</td>
<td>cyclomatic complexity of a program with pointer operations</td>
</tr>
<tr>
<td></td>
<td>(to a certain amount)</td>
<td></td>
</tr>
<tr>
<td>robust</td>
<td>grading by experts</td>
<td>almost all pseudo-metrics</td>
</tr>
</tbody>
</table>

([Ludewig and Lichter, 2013])
**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)

**Examples:**
- lines of code, hours spent on testing, …

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

**Examples:**
- average/median lines of code, productivity (lines per hour), …
<table>
<thead>
<tr>
<th>Procedure</th>
<th>objective metric</th>
<th>pseudo metric</th>
<th>subjective metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measurement, counting, possibly standardised</td>
<td>Computation (based on measurements or assessment)</td>
<td>Review by inspector, verbal or by given scale</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Advantages</th>
<th>objective metric</th>
<th>pseudo metric</th>
<th>subjective metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exact, reproducible, can be obtained automatically</td>
<td>Yields relevant, directly usable statement on not directly visible characteristics</td>
<td>Not subvertable, plausible results, applicable to complex characteristics</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Disadvantages</th>
<th>objective metric</th>
<th>pseudo metric</th>
<th>subjective metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not always relevant, often subvertable, no interpretation</td>
<td>Hard to comprehend, pseudo-objective</td>
<td>Assessment costly, quality of results depends on inspector</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example, general</th>
<th>objective metric</th>
<th>pseudo metric</th>
<th>subjective metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body height, air pressure</td>
<td>Body mass index (BMI), weather forecast for the next day</td>
<td>Health condition, weather condition (“bad weather”)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Example in Software Engineering</th>
<th>objective metric</th>
<th>pseudo metric</th>
<th>subjective metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size in LOC or NCSI; number of (known) bugs</td>
<td>Productivity; cost estimation by COCOMO</td>
<td>Usability; severeness of an error</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usually used for</th>
<th>objective metric</th>
<th>pseudo metric</th>
<th>subjective metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection of simple base measures</td>
<td>Predictions (cost estimation); overall assessments</td>
<td>Quality assessment; error weighting</td>
<td></td>
</tr>
</tbody>
</table>

(Ludewig and Lichter, 2013)
Some of the most interesting aspects of software development projects are (today) hard or impossible to measure directly, e.g.:

- how maintainable is the software?
- how much effort is needed until completion?
- how is the productivity of my software people?
- do all modules do appropriate error handling?
- is the documentation sufficient and well usable?

Due to high relevance, people want to measure despite the difficulty in measuring. Two main approaches:

<table>
<thead>
<tr>
<th>Expert review, grading</th>
<th>differentiated</th>
<th>comparable</th>
<th>reproducible</th>
<th>available</th>
<th>relevant</th>
<th>economical</th>
<th>plausible</th>
<th>robust</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pseudo-metrics, derived measures</td>
<td>✓</td>
<td>✓</td>
<td>(x)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

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

- average LOC per module: derived, not pseudo → we really measure average LOC per module.
- measure maintainability in average LOC per module: derived, pseudo → we don't really measure maintainability; average-LOC is only interpreted as maintainability.
  Not robust if easily subvertible (see exercises).

**Pseudo-Metrics Example**

Example: productivity (derived).

- Team $T$ develops software $S$ with LOC $N = 817$ in $t = 310$ h.
- Define productivity as $p = N/t$, here: ca. 2.64 LOC/h.
- Pseudo-metric: measure performance, efficiency, quality, ... of teams by productivity (as defined above).

- team may write $\begin{align*}
    x & := y \\
    y & := z
\end{align*}$ instead of $\begin{align*}
    x & := y + z;
\end{align*}$

→ 5-time productivity increase, but real efficiency actually decreased.
→ not (at all) plausible.
→ clearly pseudo.
Can Pseudo-Metrics be Useful?

- Pseudo-metrics can be useful if there is a (good) correlation (with few false positives and few false negatives) between valuation yields and the property to be measured:

<table>
<thead>
<tr>
<th>high</th>
<th>low</th>
<th>valuation yield</th>
</tr>
</thead>
<tbody>
<tr>
<td>true positive</td>
<td>true negative</td>
<td>high</td>
</tr>
<tr>
<td>false positive</td>
<td>false negative</td>
<td>low</td>
</tr>
</tbody>
</table>

- This may strongly depend on context information:
  - If LOC was (or could be made non-subvertible (→ tutorials)), then productivity could be useful measure for, e.g., team performance.

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

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

• Interval scale (not absolute, no zero due to $p > 0$); easy to compute

• 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.”
<table>
<thead>
<tr>
<th>metric</th>
<th>computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>weighted methods per class (WMC)</td>
<td>( \sum_{i=1}^{n} c_i ), ( n ) = number of methods, ( c_i ) = complexity of method ( i )</td>
</tr>
<tr>
<td>depth of inheritance tree (DIT)</td>
<td>graph distance in inheritance tree (multiple inheritance ?)</td>
</tr>
<tr>
<td>number of children of a class (NOC)</td>
<td>number of direct subclasses of the class</td>
</tr>
<tr>
<td>coupling between object classes (CBO)</td>
<td>( CBO(C) =</td>
</tr>
<tr>
<td>response for a class (RFC)</td>
<td>( RFC =</td>
</tr>
<tr>
<td>lack of cohesion in methods (LCOM)</td>
<td>( \max(</td>
</tr>
</tbody>
</table>

- **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)
References


