Is Software Development Always Successful? No.

- self-driving car, 2016: wrong strategy in traffic situation; crash, no injury
- game distribution platform, 2015: unintentional rm -rf /; damage not quantified
- car, 2015: security issue, remote exploit; 1.4 Mio. cars recalled
- car, 2014: unintended acceleration, stack overflows: people injured and killed
- photocopier, 2015: unintentional lossy compression: no damage known
- tiltrotor aircraft, 2000: hydraulic failure not handled: 4 killed
- credit card failures, 2000: incompatibility of new EMV chip: parties ruined
- war vessel, 1997: uncontrolled ship by division by 0; no damage
- plane landing, 1993: environment assumptions problem; 2 killed, 54 injured
- ambulance management, 1992: management issues; poor QA: 46 killed
- missile defense, 1991: integer overflow; 28 killed
- telephone infrastructure, 1990: erroneously entered mode; 9h no phones, 75 + 100 Mio. $
- defense system, 1979: random bits, false rocket attack announced; no harm
- weather balloons, 1971: poor protocol design; 72 weather-balloons and data lost
- ...
Survey: Previous Experience

Expectations

- none, because mandatory course
- overall
  - ✔ well-structured lectures
  - (✔) praxis oriented
  - ✗ practical knowledge about planning, designing and testing software
  - ✔ improve skills in scientific work
  - (✔) more about scientific methods
- other courses
  - ✗ more on how courses are linked together
  - ✗ skills we need to organise SoPra
  - ✔ maybe transfer knowledge in SoPra
- "real world"
  - ✔ vocabulary and methods in professional software development
  - ✔ learn how things work in a company, to easier integrate into teams, e.g., communication
- kinds of software
  - ✔ embedded systems and software
  - ✗ how to combine HW and SW parts

Introduction
L 1: 18.4., Mon

Scales, Metrics, Costs
L 2: 21.4., Thu
L 3: 25.4., Mon
T 1: 28.4., Thu

Development
L 4: 2.5., Mon
L 5: 5.5., Thu

Process
L 6: 9.5., Mon
L 7: 13.5., Mon
L 8: 16.5, Mon
L 9: 19.5, Thu
T 2: 23.5, Mon

Requirements Engineering
L 10: 2.5., Mon
L 11: 5.5., Thu

Architecture & Design
L 12: 9.5., Mon
L 13: 13.6., Mon
L 14: 16.6., Thu
L 15: 19.6., Thu
T 3: 23.6., Thu

Software Modelling
L 16: 27.6., Mon
L 17: 30.6., Mon

Quality Assurance (Testing, Formal Verification)
L 18: 7.7., Thu
L 19: 11.7, Mon

Wrap-Up
T 4: 14.7., Thu
L 20: 18.7, Mon
L 21: 21.7, Thu
Expectations Cont’d

- software development
  ✔ understand how software development practically works
  ✔ developing, maintaining software at bigger scale
  ✔ aspects of software development

- software project management
  ✔ learn what is important to plan
  ✔ how to structure the process of a project
  ✔ how to keep control of project, measure success
  X which projects need full-time project manager
  X which kind of documentation is really necessary
  X want to get better in leading a team; how to lead team of engineers

- cost estimation
  ✔ how to estimate time and effort
  (X) formal methods for better planning of projects
  X tools which help planning

- quality
  ✔ learn ways how to judge quality based on the requirements
  ✔ avoid mistakes during software development
  ✔ make better programs, or make programs more efficiently

Expectations Cont’d

- requirements
  ✔ formal ways to specify requirements
  ✔ learn techniques to reduce misunderstandings
  ✔ understand types of requirements
  (X) learn how requirements are to be stated
  (X) how to create requirements/specification document

- design
  ✔ techniques for design
  ✔ predict potential risks and crucial design errors
  (X) come up with good design, learn how to design
  (X) practical knowledge on application of design patterns
  X how to structure, compose components, how to define interfaces
  X standards for keeping parts of project compatible
  X how to guarantee a particular reliability

- Implementation
  (X) modular programming, better documentation of big projects
  X more of computers and programming, write faster better programs
  X strengths and weaknesses of standards, training in their application
  X improve coding skills
  X how to increase (software) performance
Expectations Cont’d

• code quality assurance
  ✓ methods for testing to guarantee high level of quality
  ✓ formal methods like program verification
  ❌ learn about practical implementation of these tools

• extra information
  • “will work as teacher”
  • “want to work on medical software”
  • “want to work in automotive industry”
  • “worked as software-engineer”

<table>
<thead>
<tr>
<th>Introduction</th>
<th>L 1: 18.4, Mon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scales, Metrics, Costs</td>
<td>L 2: 21.4, Thu</td>
</tr>
<tr>
<td>Development</td>
<td>L 3: 23.4, Mon</td>
</tr>
<tr>
<td>T 1: 28.4, Thu</td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td>L 4: 2.5, Mon</td>
</tr>
<tr>
<td>L 5: 5.5, Thu</td>
<td></td>
</tr>
<tr>
<td>L 6: 9.5, Mon</td>
<td></td>
</tr>
<tr>
<td>L 7: 13.5, Thu</td>
<td></td>
</tr>
<tr>
<td>T 2: 19.5, Thu</td>
<td></td>
</tr>
<tr>
<td>T 3: 23.5, Mon</td>
<td></td>
</tr>
<tr>
<td>T 4: 26.5, Thu</td>
<td></td>
</tr>
<tr>
<td>Requirements Engineering</td>
<td>L 7: 30.5, Mon</td>
</tr>
<tr>
<td>L 8: 2.6, Thu</td>
<td></td>
</tr>
<tr>
<td>L 9: 6.6, Mon</td>
<td></td>
</tr>
<tr>
<td>T 3: 9.6, Thu</td>
<td></td>
</tr>
<tr>
<td>Architecture &amp; Design</td>
<td>L 10: 13.6, Mon</td>
</tr>
<tr>
<td>L 11: 16.6, Thu</td>
<td></td>
</tr>
<tr>
<td>L 12: 20.6, Mon</td>
<td></td>
</tr>
<tr>
<td>T 4: 23.6, Thu</td>
<td></td>
</tr>
<tr>
<td>Software Modelling</td>
<td>L 13: 27.6, Mon</td>
</tr>
<tr>
<td>L 14: 30.6, Thu</td>
<td></td>
</tr>
<tr>
<td>L 15: 4.7, Mon</td>
<td></td>
</tr>
<tr>
<td>T 5: 7.7, Thu</td>
<td></td>
</tr>
<tr>
<td>Quality Assurance (Testing, Formal Verification)</td>
<td>L 16: 11.7, Mon</td>
</tr>
<tr>
<td>L 17: 14.7, Thu</td>
<td></td>
</tr>
<tr>
<td>L 18: 18.7, Mon</td>
<td></td>
</tr>
<tr>
<td>T 5: 21.7, Thu</td>
<td></td>
</tr>
<tr>
<td>Wrap-Up</td>
<td>L 19: 21.7, Thu</td>
</tr>
</tbody>
</table>
### Content

- **Software Metrics**
  - Motivation
  - Vocabulary
  - Requirements on Useful Metrics
  - Excursion: Scales
  - Example: LOC
  - Other Properties of Metrics
  - Subjective and Pseudo Metrics
  - Discussion

- **Cost Estimation**
  - Deadlines and Costs
  - Expert’s Estimation
  - Algorithmic Estimation
## 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 criteria</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)
Motivation

- Goal: specify, and systematically compare and improve industrial products.
- Approach: precisely describe and assess the products (and the process of creation).
- This is common practice for material goods:

![Image of a caliper measuring something](https://commons.wikimedia.org/wiki/smetricintro)

- Not so obvious (and common) for immaterial goods, like software.
- It should be common: objective measures are central to engineering approaches.

Why “no so obvious” for software?

- Recall, e.g., quality (ISO/IEC 9126-1:2000 (2000)):
**Vocabulary**

**metric** – A quantitative measure of the degree to which a system, component, or process possesses a given attribute.

See: quality metric.

**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.

**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 time spent per procedure, then “optimize” most time consuming procedure.
  
  (ii) Measure attributes which indicate architecture problems, then re-factor accordingly.
Requirements on Useful Metrics

Definition. A software metric is a function $\mathit{m} : P \rightarrow S$ which assigns to each proband $p \in P$ a valuation yield ("Bewertung") $\mathit{m}(p) \in S$. We call $S$ scale.

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 ($\rightarrow$ 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>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>✗</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>a rational scale where $S$ comprises the key figures itself</td>
</tr>
</tbody>
</table>

Examples: Nominal Scale

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

→ 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 and Types of Scales

Scales $S$ are distinguished by supported operations:

<table>
<thead>
<tr>
<th></th>
<th>$\neq$, $\neq$</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>X</td>
<td>✔</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>a rational scale where $S$ comprises the key figures itself</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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)

$\rightarrow$ There is a (natural) order between elements of $M$, but no (natural) notion of distance or average.

Scales and Types of Scales

Scales $S$ are distinguished by supported operations:

<table>
<thead>
<tr>
<th></th>
<th>$\neq$, $\neq$</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>X</td>
<td>✔</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>a rational scale where $S$ comprises the key figures itself</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Examples: Interval Scale

- temperature in Fahrenheit
- "today it is 10°F warmer than yesterday" ($\Delta(\theta_{\text{today}}, \theta_{\text{yesterday}}) = 10°F$)
- "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

$\rightarrow$ 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></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>

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></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>

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.
Recall:

Definition. [Metric Space (math.):]
Let $X$ be a set. A function $d : X \times X \to \mathbb{R}$ is called a metric on $X$ if and only if, for each $x, y, z \in X$,

(i) $d(x, y) \geq 0$ (non-negative)
(ii) $d(x, y) = 0 \iff x = y$ (identity of indiscernibles)
(iii) $d(x, y) = d(y, x)$ (symmetry)
(iv) $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

$(X, d)$ is called a 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
**Median and Box-Plots**

<table>
<thead>
<tr>
<th>LOC</th>
<th>M1</th>
<th>M2</th>
<th>M3</th>
<th>M4</th>
<th>M5</th>
</tr>
</thead>
<tbody>
<tr>
<td>127</td>
<td>213</td>
<td>152</td>
<td>139</td>
<td>13297</td>
<td></td>
</tr>
</tbody>
</table>

- arithmetic average: 2785.6
- median: 127, 139, 152, 213, 13297

A boxplot visualises 5 aspects of data at once (whiskers sometimes defined differently, with “outliers”):

![Boxplot Example](image)

**Example: Project Management**

$m$: commits took place at $n$-th day of project.

**Team A:**
10, 20, 30, 40, 50, 60, 70, 80, 90, 100

**Team B:**
5, 50, 60, 75, 80, 85, 95, 100

Team B: “Oh, this SoPra was so stressful... Could we have done something about that?”
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&lt;sub&gt;tot&lt;/sub&gt;</td>
<td>number of lines in total</td>
</tr>
<tr>
<td>net program size</td>
<td>LOC&lt;sub&gt;ne&lt;/sub&gt;</td>
<td>number of non-empty lines</td>
</tr>
<tr>
<td>code size</td>
<td>LOC&lt;sub&gt;pars&lt;/sub&gt;</td>
<td>number of lines with not only comments and non-printable</td>
</tr>
<tr>
<td>delivered program size</td>
<td>DLOC&lt;sub&gt;tot&lt;/sub&gt;, DLOC&lt;sub&gt;ne&lt;/sub&gt;, DLOC&lt;sub&gt;pars&lt;/sub&gt;</td>
<td>like LOC, only code (as source or compiled) given to customer</td>
</tr>
</tbody>
</table>

(Ludewig and Lichter, 2013)

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

\[
\text{LOC tot} = 12 \\
\text{LOC ne} = 11 \\
\text{LOC pars} = 7
\]

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

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 (to a certain amount)</td>
<td>cyclomatic complexity of a program with pointer operations</td>
</tr>
<tr>
<td>robust</td>
<td>grading by experts</td>
<td>almost all pseudo-metrics</td>
</tr>
</tbody>
</table>

(Ludewig and Lichter, 2013)

**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), ...
## Kinds of Metrics: by Measurement Procedure

<table>
<thead>
<tr>
<th></th>
<th>Objective Metric</th>
<th>Pseudo Metric</th>
<th>Subjective Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Procedure</strong></td>
<td>measurement, counting, poss. normed</td>
<td>computation (based on measurements or assessment)</td>
<td>review by inspector, verbal or by given scale</td>
</tr>
<tr>
<td><strong>Advantages</strong></td>
<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>
</tr>
<tr>
<td><strong>Disadvantages</strong></td>
<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>
</tr>
<tr>
<td><strong>Example, general</strong></td>
<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>
</tr>
<tr>
<td><strong>Example in Software Engineering</strong></td>
<td>size in LOC or NCSI; number of (known) bugs</td>
<td>productivity; cost estimation following COCOMO</td>
<td>usability; severity of an error</td>
</tr>
<tr>
<td><strong>Usually used for</strong></td>
<td>collection of simple base measures</td>
<td>predictions (cost estimation); overall assessments</td>
<td>quality assessment; error weighting</td>
</tr>
</tbody>
</table>

(Ludewig and Lichter, 2013)

### Pseudo-Metrics

- size in LOC or NCSI
- number of (known) bugs
- productivity; cost estimation following COCOMO
- usability; severity of an error
- predictions (cost estimation); overall assessments
- quality assessment; error weighting
Pseudo-Metrics

Some of the most interesting aspects of software development projects are 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>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td></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 \( p = \frac{N}{t} \), here: ca. 2.64 LOC/h.
- Pseudo-metric: measure performance, efficiency, quality, … of teams by productivity (as defined above).

- team may write \( \begin{bmatrix} z \\ y \\ x \end{bmatrix} := \begin{bmatrix} y \\ z \end{bmatrix} \) instead of \( x := y + z; \)

→ 5-time productivity increase, but real efficiency actually decreased.  
→ not (at all) plausible.  
→ clearly pseudo.
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
\[
v(P) = |E| - |V| + p \text{ where } p \text{ is the number of entry or exit points.}
\]

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

Number of edges: \(|E| = 11\)
Number of nodes: \(|V| = 6 + 2 + 2 = 10\)
External connections: \(p = 2\)
\[ v(P) = 11 - 10 + 2 = 3 \]
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

\[
\nu(P) = |E| - |V| + p
\]

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.”

References
References


