Softwaretechnik / Software-Engineering

Lecture 2: Software Metrics

2018-04-19

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
  • Procedure Models
  • Process Models

VL 5
• Process Metrics
  • CMMI, Spice
Survey: Expectations on the Course

Software Metrics
- Motivation
- Vocabulary
- Requirements on Useful Metrics
  - Excursion: Scales
  - Some positive/negative examples
  - Example: LOC
- Other Properties of Metrics
  - Base Measures vs. Derived Measures
  - Subjective and Pseudo Metrics
  - Example: McCabe
- Discussion
Survey: Previous Experience

- Project Management: 0-(0/1/3)-8
- Requirements Engineering: 0-(0/1/2)-8
- Programming: 1-(1/3/5)-10
- Design Modelling: 0-(1/1/3)-9
- Software Quality Assurance: 0-(1/2/4)-10
Excursion: Communicating Figures
2018 vs. 2017

**Management**
- **2018**
  - Median: 1
  - Average: 1.7188
- **2017**
  - Median: 1
  - Average: 2.069

**RE Experience**
- **2018**
  - Median: 1
  - Average: 2.284
- **2017**
  - Median: 3
  - Average: 3.9844

**Programming**
- **2018**
  - Median: 3
  - Average: 3.9432
- **2017**
  - Median: 1
  - Average: 1.9531

**Modelling**
- **2018**
  - Median: 1
  - Average: 2.1932
- **2017**
  - Median: 2
  - Average: 2.625

**QA**
- **2018**
  - Median: 1
  - Average: 2.5682
- **2017**
  - Median: 1
  - Average: 2.5682
Quartiles

- **Arithmetic mean**: 1.531 (not in the scale! → later)
- **Minimum** and **maximum**: 0 and 8
- **Median**: 1 (the value such that 50% of the probands have yields below and above)
- **1st and 3rd Quartile**: 0 and 2 (25%, 75%)

- A **boxplot** visualises 5 aspects of data at once (whiskers sometimes defined differently):

\[
\begin{align*}
100\% & \text{ (maximum)} \\
75\% & \text{ (3rd quartile)} \\
50\% & \text{ (median)} \\
25\% & \text{ (1st quartile)} \\
0\% & \text{ (minimum)}
\end{align*}
\]
Back From Excursion: Communicating Figures
- Survey: Expectations on the Course
- Software Metrics
  - Motivation
  - Vocabulary
  - Requirements on Useful Metrics
    - Excursion: Scales
    - Some positive/negative examples
    - Example: LOC
  - Other Properties of Metrics
    - Base Measures vs. Derived Measures
    - Subjective and Pseudo Metrics
    - Example: McCabe
  - Discussion
Expectations

- **general**
  - ✔ learn about tasks related to development of software
  - ✔ learn different aspects of working in a software producing team and how to cope with them
  - ✔ gain more insight with [...] challenges faced in socio-technical systems and practical ways to overcome those challenges
  - ✔ learn what techniques are there for project management, planning a software project, communicate and cooperate with partners
  - ✔ learning a systematic approach to developing a larger program
  - ✔ hope that course content can directly be used in practice
  - ✔ explicit, precise, working methods
  - ✔ would like to see a lot of concrete examples of software projects
  - ✗ concretise the role of computer science and software engineering in business

- **project management**
  - ✔ overview about modelling and organisation of a project, and how to group different problems
  - ✔ be able to plan software development towards deadline and budget
  - ✔ plan larger projects, conduct safely, timely, reliably
  - ✔ skills required to take responsibility in managing software projects
  - ✔ how to avoid mistakes or limit inflicted damage

<table>
<thead>
<tr>
<th>Introduction</th>
<th>L 1:  16.4., Mon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scales, Metrics, Costs,</td>
<td>L 2:  19.4., Thu</td>
</tr>
<tr>
<td></td>
<td>L 3:  23.4., Mon</td>
</tr>
<tr>
<td></td>
<td>T 1:  26.4., Thu</td>
</tr>
<tr>
<td>Development Process</td>
<td>L 4:  30.4., Mon</td>
</tr>
<tr>
<td>Requirements</td>
<td>L 5:  3.5., Thu</td>
</tr>
<tr>
<td></td>
<td>L 6:  7.5, Mon</td>
</tr>
<tr>
<td></td>
<td>-    10.5., Thu</td>
</tr>
<tr>
<td>Engineering</td>
<td>L 7:  14.5., Mon</td>
</tr>
<tr>
<td></td>
<td>T 2:  17.5., Thu</td>
</tr>
<tr>
<td></td>
<td>21.5., Mon</td>
</tr>
<tr>
<td></td>
<td>24.5., Thu</td>
</tr>
<tr>
<td></td>
<td>L 8:  28.5., Mon</td>
</tr>
<tr>
<td></td>
<td>-    31.5., Thu</td>
</tr>
<tr>
<td></td>
<td>L 9:  4.6., Mon</td>
</tr>
<tr>
<td></td>
<td>T 3:  7.6., Thu</td>
</tr>
<tr>
<td></td>
<td>L10: 11.6., Mon</td>
</tr>
<tr>
<td>Arch. &amp; Design, Software-</td>
<td>L11: 14.6., Thu</td>
</tr>
<tr>
<td>Modelling, Patterns</td>
<td>L12:  18.6., Mon</td>
</tr>
<tr>
<td></td>
<td>T 4:  21.6., Thu</td>
</tr>
<tr>
<td>QA</td>
<td>L13: 25.6., Mon</td>
</tr>
<tr>
<td></td>
<td>L14: 28.6., Thu</td>
</tr>
<tr>
<td></td>
<td>L15:  2.7., Mon</td>
</tr>
<tr>
<td></td>
<td>T 5:  5.7., Thu</td>
</tr>
<tr>
<td>(Testing, Formal</td>
<td>L16:  9.7., Mon</td>
</tr>
<tr>
<td>Verification)</td>
<td>L17: 12.7., Thu</td>
</tr>
<tr>
<td>Wrap-Up</td>
<td>L18: 16.7., Mon</td>
</tr>
<tr>
<td></td>
<td>T 6:  19.7., Thu</td>
</tr>
</tbody>
</table>
Expectations Cont’d

- **project management** cont’d
  - ✔ how to prove or test the development process
  - ✔ learn proper metrics to measure progress and check quality
  - ✔ create management plan evaluate results of previous software projects and use evaluation results
  - ✔ useful tools for successful management of software projects
  - ✗ learn how project planning and management are done in the proper way
  - ✗ deep knowledge about management skills
  - ✔ don’t expect being taught how to be a manager, learn conditions for success

- **requirements**
  - ✔ communicate expectations, needs, information, and ideas without misunderstandings to colleagues and clients
  - ✔ be able to formalise requirements ensure that they are formalised
  - ✔ investigate necessary requirements
  - ✔ learn what well-defined requirements look like, how they can have impact on final product
• **design**
  - ✗ get experience towards **coming up with good designs**
  - ✗ improve knowledge in [...] software design / software architecture
  - ✗ learn **how to design** a software model that fulfills all predefined requirements
  - ✔ communicate design specifications
  - ✔ learn **formalisms** to avoid, e.g., going for an inappropriate architecture
  - ✔ concepts/models/ideas to plan software and **verify concept before implementing** it

• **Implementation**
  - ✗ keep software clean, **modular** and expansible
  - ✗ sustainably manage a codebase

• **Quality Assurance**
  - ✗ make sure software is high quality
  - ✔ organising quality assurance
  - ✔ how to satisfy wishes of customers and **how to know if satisfied**
  - ✔ how to obtain software as planned and which **does what it should do**

---

<table>
<thead>
<tr>
<th>Introduction</th>
<th>L 1: 16.4., Mon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scales, Metrics,</td>
<td>L 2: 19.4., Thu</td>
</tr>
<tr>
<td>Costs,</td>
<td>L 3: 23.4., Mon</td>
</tr>
<tr>
<td>T 1: 26.4., Thu</td>
<td></td>
</tr>
<tr>
<td>Development</td>
<td>L 4: 30.4., Mon</td>
</tr>
<tr>
<td>Process</td>
<td>L 5: 3.5., Thu</td>
</tr>
<tr>
<td>Requirements</td>
<td>L 6: 7.5., Mon</td>
</tr>
<tr>
<td></td>
<td>- 10.5., Thu</td>
</tr>
<tr>
<td>Engineering</td>
<td>L 7: 14.5., Mon</td>
</tr>
<tr>
<td></td>
<td>T 2: 17.5., Thu</td>
</tr>
<tr>
<td></td>
<td>- 21.5., Mon</td>
</tr>
<tr>
<td></td>
<td>- 24.5., Thu</td>
</tr>
<tr>
<td></td>
<td>L 8: 28.5., Mon</td>
</tr>
<tr>
<td></td>
<td>- 31.5., Thu</td>
</tr>
<tr>
<td>Development</td>
<td>L 9: 4.6., Mon</td>
</tr>
<tr>
<td>Process</td>
<td>T 3: 7.6., Thu</td>
</tr>
<tr>
<td></td>
<td>L10: 11.6., Mon</td>
</tr>
<tr>
<td></td>
<td>T 4: 21.6., Thu</td>
</tr>
<tr>
<td>Arch. &amp; Design,</td>
<td>L11: 14.6., Thu</td>
</tr>
<tr>
<td>Software-</td>
<td>L12: 18.6., Mon</td>
</tr>
<tr>
<td>Modelling,</td>
<td>T 5: 21.6., Thu</td>
</tr>
<tr>
<td>Patterns</td>
<td>L13: 25.6., Mon</td>
</tr>
<tr>
<td>QA</td>
<td>L14: 28.6., Thu</td>
</tr>
<tr>
<td>(Testing, Formal</td>
<td>L15: 2.7., Mon</td>
</tr>
<tr>
<td>Verification)</td>
<td>T 5: 5.7., Thu</td>
</tr>
<tr>
<td>Wrap-Up</td>
<td>L16: 9.7., Mon</td>
</tr>
<tr>
<td></td>
<td>L17: 12.7., Thu</td>
</tr>
<tr>
<td></td>
<td>L18: 16.7., Mon</td>
</tr>
<tr>
<td></td>
<td>T 6: 19.7., Thu</td>
</tr>
</tbody>
</table>
Survey: Expectations on the Course

Software Metrics
  • Motivation
  • Vocabulary
  • Requirements on Useful Metrics
    • Excursion: Scales
    • Some positive/negative examples
    • Example: LOC
  • Other Properties of Metrics
    • Base Measures vs. Derived Measures
    • Subjective and Pseudo Metrics
    • Example: McCabe
  • Discussion
Software Metrics
# 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</td>
<td>can be conducted using objective, quantified criteria</td>
<td>is only possible subjectively, results are disputed</td>
</tr>
<tr>
<td>comparison</td>
<td></td>
<td></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)
For **material goods**, it is often pretty obvious what we want to evaluate for:

![Diagram](https://commons.wikimedia.org/wiki/index.php?curid=737312)
Recall: “software that is reliable and works efficiently”

(Bauer, 1971)


### 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.
Recall: “software that is reliable and works efficiently” (Bauer, 1971)


### 6.1 Functionality
The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.

#### 6.1.1 Suitability
The capability of the software product to provide an appropriate set of functions for specified tasks and user objectives.
**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)
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.
**Example: Material Goods**

- For *material goods*, it is often pretty obvious what we want to evaluate for:

  ![Diagram of material goods evaluation](https://commons.wikimedia.org/wiki/index.php?curid=737312)

- ...and how to measure.

- And *immaterial goods*, like *software*?
Recall: “software that is reliable and works efficiently”  
(Bauer, 1971)


6.1 Functionality
The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.

6.1.1 Suitability
The capability of the software product to provide an appropriate set of functions for specified tasks and user objectives.
Content

- Survey: Expectations on the Course
- Software Metrics
  - Motivation
  - Vocabulary
  - Requirements on Useful Metrics
    - Excursion: Scales
    - Some positive/negative examples
    - Example: LOC
  - Other Properties of Metrics
    - Base Measures vs. Derived Measures
    - Subjective and Pseudo Metrics
      - Example: McCabe
  - Discussion
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.

IEEE 610.12 (1990)
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 $m$. 
Requirements on Useful Metrics

**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 \( m \).

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

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Condition</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><strong>nominal scale</strong></td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td><strong>ordinal scale</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td><strong>interval scale</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>(with units)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>rational scale</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(with units)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>absolute scale</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</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 (“$C < \text{Java}$”), but is not related to the measured information (thus not natural).
**Scales** are distinguished by supported operations:

<table>
<thead>
<tr>
<th></th>
<th>=, ≠</th>
<th>&lt;, &gt; (with transitivity)</th>
<th>min, max</th>
<th>percentiles, e.g. median</th>
<th>∆</th>
<th>proportion</th>
<th>natural 0 (zero)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>nominal scale</strong></td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td><strong>ordinal scale</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td><strong>interval scale</strong> (with units)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td><strong>rational scale</strong> (with units)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>absolute scale</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

A rational scale where the real numbers comprise 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** $S$ 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><strong>nominal scale</strong></td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td><strong>ordinal scale</strong></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td><strong>interval scale</strong></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td></td>
<td>✘</td>
</tr>
<tr>
<td>(with units)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✘</td>
</tr>
<tr>
<td><strong>rational scale</strong></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>(with units)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td><strong>absolute scale</strong></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: 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

  $\rightarrow$ 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><strong>nominal scale</strong></td>
<td>✔</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td><strong>ordinal scale</strong></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>$\times$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td><strong>interval scale</strong> (with units)</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>$\checkmark$</td>
<td>$\times$</td>
<td>$\times$</td>
</tr>
<tr>
<td><strong>rational scale</strong> (with units)</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td><strong>absolute scale</strong></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: 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.

$\rightarrow$ An absolute scale has a median, but in general not an average in the scale.
**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.).
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>(→ pseudo-metric)</td>
</tr>
<tr>
<td>robust</td>
<td>developers cannot arbitrarily manipulate the yield; antonym: subvertible</td>
</tr>
</tbody>
</table>
Content

- **Survey**: Expectations on the Course

- **Software Metrics**
  - Motivation
  - Vocabulary
  - Requirements on Useful Metrics
    - Excursion: Scales
    - Some positive/negative examples
    - Example: LOC
  - Other Properties of Metrics
    - Base Measures vs. Derived Measures
    - Subjective and Pseudo Metrics
      - Example: McCabe
  - Discussion
### Some Positive and Negative 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 (→ in a minute)</td>
<td>CMM/CMMI level below 2</td>
</tr>
<tr>
<td>comparable</td>
<td>cyclomatic complexity (→ in two minutes)</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)
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
    }
}
```
- Survey: Expectations on the Course
- Software Metrics
  - Motivation
  - Vocabulary
  - Requirements on Useful Metrics
    - Excursion: Scales
    - Some positive/negative examples
    - Example: LOC
  - Other Properties of Metrics
    - Base Measures vs. Derived Measures
    - Subjective and Pseudo Metrics
    - Example: McCabe
  - Discussion
Other Properties of Metrics

**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, possibly standardised</td>
<td>computation (based on measurements or assessment)</td>
<td>review by inspector, verbal or by given scale</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 (&quot;bad weather&quot;)</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 by COCOMO</td>
<td>usability; severeness 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>
<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>
</tbody>
</table>

(Ludewig and Lichter, 2013)
Pseudo-Metrics
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>✔</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
<td>✔ !</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pseudo-metrics, derived measures</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>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔ !</td>
<td>✘</td>
<td>✔</td>
<td>✘</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).
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).

  \[
  \begin{align*}
  x &:= y + z; \\
  \end{align*}
  \]

- Team may write \[
  \begin{align*}
  x &:= y + z;
  \end{align*}
  \]
  \(\rightarrow\) 5-time productivity increase, but real efficiency actually decreased.

\(\rightarrow\) not (at all) plausible.

\(\rightarrow\) clearly pseudo.
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>Quality</th>
<th>Valuation Yield</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>High</td>
</tr>
<tr>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>High</td>
</tr>
</tbody>
</table>

Usefulness may strongly depend on context information:
- If LOC was (or could be made) non-subvertible (→ tutorials), then LOC/day could be a useful measure for, e.g., project progress.
**Content**

- **Survey**: Expectations on the Course

- **Software Metrics**
  - Motivation
  - Vocabulary
  - Requirements on Useful Metrics
    - Excursion: Scales
    - Some positive/negative examples
    - Example: LOC
  - Other Properties of Metrics
    - Base Measures vs. Derived Measures
    - Subjective and Pseudo Metrics
      - Example: McCabe
  - Discussion
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.
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: $|E| = 11$
Number of nodes: $|V| = 6 + 2 + 2 = 10$
External connections: $p = 2$

$\rightarrow v(P) = 11 - 10 + 2 = 3$
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 $v(P) = |E| - |V| + p$ 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.”
### Code Metrics for OO Programs (Chidamber and Kemerer, 1994)

<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, \text{ } n = \text{ number of methods, } c_i = \text{ 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>[ \text{CBO}(C) =</td>
</tr>
<tr>
<td>response for a class (RFC)</td>
<td>[ \text{RFC} =</td>
</tr>
<tr>
<td>lack of cohesion in methods (LCOM)</td>
<td>[ \text{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)
Tell Them What You’ve Told Them...

- **Software metrics** are defined in terms of **scales**.
- Use software metrics to
  - specify, assess, quantify, predict, support decisions
  - prescribe / describe (diagnose / prognose).
- Whether a **software metric** is **useful** depends...
- Not every **software attribute** is directly measurable:
  - derived measures,
  - subjective metrics, and
  - pseudo metrics...
  - have to be **used with care** – do we measure what we want to measure?
- **Metric examples**:
  - LOC, McCabe / Cyclomatic Complexity,
  - more than 50 more metrics **named**
References


