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

VL 4
- **Process Metrics**
  - CMMI, Spice
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)
Survey: Previous Experience & Expectations
Survey: Previous Experience

Project Management
0-(1/1/3)-10

Requirements Engineering
0-(1/1/3)-10

Programming
1-(3/4/6)-10

Design Modelling
0-(1/2/4)-10

Software Quality Assurance
0-(1/2/3)-10
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 requirements 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

- 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
✔ **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. : )**
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)
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**: $m_a : P_a \rightarrow S_a$
  - **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**: $m_r : P_r \rightarrow S_r$
  - **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**: $m_c : P_c \rightarrow S_c$
  - **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 (→ later)
- **reproducible**: Multiple applications to the same proband yields the same valuation
- **differentiated**: Sufficiently different valuations for sufficiently different probands
• 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 Metrics
## Example Software Metrics

<table>
<thead>
<tr>
<th>characteristic ('Merkmal')</th>
<th>positive example(s)</th>
<th>negative example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>relevant</td>
<td>expected development cost; number of errors</td>
<td>number of subclasses (NOC)</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 (→ in three minutes)</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>economical</td>
<td>number of discovered errors in code</td>
<td>highly detailed timekeeping</td>
</tr>
<tr>
<td>comparable</td>
<td>cyclomatic complexity (→ in two minutes)</td>
<td>expert’s review (in textual form)</td>
</tr>
<tr>
<td>reproducible</td>
<td>memory consumption</td>
<td>grade assigned by inspector</td>
</tr>
<tr>
<td>differentiated</td>
<td>program length in LOC (→ in a minute)</td>
<td>CMM/CMMI level below 2 (a process metric; → later)</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\textsubscript{tot}</td>
<td>number of lines in total</td>
</tr>
<tr>
<td>net program size</td>
<td>LOC\textsubscript{ne}</td>
<td>number of non-empty lines</td>
</tr>
<tr>
<td>code size</td>
<td>LOC\textsubscript{pars}</td>
<td>number of lines with not only comments and non-printable</td>
</tr>
<tr>
<td>delivered program size</td>
<td>DLOC\textsubscript{tot}, DLOC\textsubscript{ne}, DLOC\textsubscript{pars}</td>
<td>LOC of only that code which is delivered to the 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
    }
}
```
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.

```plaintext
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.
- **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)

<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 (what about 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)

- software related quality
  - process quality
    - functionality
    - reliability
    - usability
    - efficiency
    - maintainability
    - portability
  - product quality
    - suitability
    - accuracy
    - interoperability
    - security
    - maturity
    - fault tolerance
    - recoverability
    - understandability
    - learnability
    - operability
    - attractiveness
    - time behaviour
    - resource utilisation
    - analysability
    - changeability
    - stability
    - testability
    - adaptability
    - installability
    - co-existence
    - replaceability

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.

**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: 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 Metric Issues
**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.
3rd 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.
• 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**:  

<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>
<tr>
<td></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 ($S = \{ \text{Java, C, } \ldots \}$)

→ 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** $S$ can be 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>
<tr>
<td></td>
<td></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)

→ 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:

<table>
<thead>
<tr>
<th></th>
<th>$\neq$</th>
<th>$\leq, \geq$ (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

  → 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**:

<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>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: 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.

$\rightarrow$ 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**:

<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>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td><strong>ordinal scale</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td><strong>interval scale</strong> (with units)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</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>

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.
## 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), tomorrow's weather forecast</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 as LOC/h; COCOMO cost estimate; judge Software Engineer by SWT course grade</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 subvertible, plausible results, applicable to complex characteristics</td>
</tr>
<tr>
<td><strong>Disadvantages</strong></td>
<td>not always relevant, often subvertible, no interpretation</td>
<td>hard to comprehend, pseudo-objective, does not actually measure what it promises</td>
<td>assessment costly, quality of results depends on inspector</td>
</tr>
</tbody>
</table>

(Ludewig and Lichter, 2013)
**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.

<table>
<thead>
<tr>
<th></th>
<th>plausible</th>
<th>robust</th>
<th>available</th>
<th>economical</th>
<th>comparable</th>
<th>reproducible</th>
<th>differentiated</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subjective Metrics</strong></td>
<td>✔</td>
<td>✔</td>
<td>(✔)</td>
<td>(✘)</td>
<td>(✔)</td>
<td>(✘)</td>
<td>(✔)</td>
</tr>
<tr>
<td><strong>Pseudo Metrics</strong></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:

- 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)

**Risk:** people prioritise high metric valuation over (real) quality.
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, Excursión: 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).
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.

<table>
<thead>
<tr>
<th>cost ('Kosten')</th>
<th>all disadvantages of a solution.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(or: positive costs)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>benefit ('Nutzen')</th>
<th>all benefits of a solution.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(or: negative costs)</td>
<td></td>
</tr>
</tbody>
</table>

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)
• 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)
Tell Them What You’ve Told Them...

- 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 (→ tutorials).
References
References


