Survey: Softwarepraktikum

- not in study plan, or later
- participated earlier
- participate this semester
Content

- **Software Metrics**
  - Subjective Metrics
  - Goal-Question-Metric Approach

- **Cost Estimation**
  - “(Software) Economics in a Nutshell”
  - Cost Estimation
    - Expert’s Estimation
      - The Delphi Method
    - Algorithmic Estimation
      - COCOMO
      - Function Points

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

- **Process Metrics**
  - CMMI, Spice
# 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 (‘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 by 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>
<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>

Note: [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>Determined</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>✔️</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).
## 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>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 by 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)

---

### Subjective Metrics

<table>
<thead>
<tr>
<th></th>
<th>Example</th>
<th>Problems</th>
<th>Countermeasures</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Statement</strong></td>
<td>“The specification is available.”</td>
<td>Terms may be ambiguous, conclusions are hardly possible.</td>
<td>Allow only certain statements, characterise them precisely.</td>
</tr>
<tr>
<td><strong>Assessment</strong></td>
<td>“The module is implemented in a clever way.”</td>
<td>Not necessarily comparable.</td>
<td>Only offer particular outcomes; put them on an (at least ordinal) scale.</td>
</tr>
<tr>
<td><strong>Grading</strong></td>
<td>“Readability is graded 4.0.”</td>
<td>Subjective; grading not reproducible.</td>
<td>Define criteria for grades; give examples how to grade; practice on existing artefacts</td>
</tr>
</tbody>
</table>

(Ludewig and Lichter, 2013)
Information Overload!?

Now we have mentioned nearly 60 attributes one could measure...

Which ones should we measure?

It depends...

One approach: Goal-Question-Metric (GQM).
The three steps of GQM:
(i) **Define** the goals relevant for a project or an organisation.
(ii) **From each goal, derive questions** which need to be answered to check whether the goal is reached.
(iii) **For each question, choose** (or develop) **metrics** which contribute to finding answers.

Being **good** wrt. to a certain metric is (in general) not an asset on its own.
We usually want to optimise wrt. **goals**, not wrt. **metrics**.
In particular critical: pseudo-metrics for quality.

Software and process measurements may yield **personal data** ("personenbezogene Daten").
Their collection may be regulated by laws.

---

**Example: A Metric for Maintainability**

- **Goal:** assess **maintainability**.
- **One approach:** grade the following aspects, e.g., with scale \( S = \{0, \ldots, 10\} \).
  (Some aspects may be objective, some subjective (conduct review))

  - **Norm Conformance**
    - \( n_1 \): size of units (modules etc.)
    - \( n_2 \): labelling
    - \( n_3 \): naming of identifiers
    - \( n_4 \): design (layout)
    - \( n_5 \): separation of literals
    - \( n_6 \): style of comments

  - **Locality**
    - \( l_1 \): use of parameters
    - \( l_2 \): information hiding
    - \( l_3 \): local flow of control
    - \( l_4 \): design of interfaces

  - **Readability**
    - \( r_1 \): data types
    - \( r_2 \): structure of control flow
    - \( r_3 \): comments

  - **Testability**
    - \( t_1 \): test driver
    - \( t_2 \): test data
    - \( t_3 \): preparation for test evaluation
    - \( t_4 \): diagnostic components
    - \( t_5 \): dynamic consistency checks

  - **Typing**
    - \( y_1 \): type differentiation
    - \( y_2 \): type restriction

- **Define:** \( m = \frac{n_1 + \ldots + n_6}{20} \)
  (**with weights:** \( m_g = \frac{g_1 \cdot n_1 + \ldots + g_20 \cdot n_6}{G}, G = \sum_{i=1}^{20} g_i \)).

- **Procedure:**
  - Train reviewers on existing examples.
  - Do not over-interpret results of first applications.
  - Evaluate and adjust before putting to use, adjust regularly.

  (Ludewig and Lichter, 2013)
And Which Metrics Should One Use?

Often useful: collect some basic measures in advance (in particular if collection is cheap / automatic), e.g:

- **size**…
  - of newly created and changed code, etc.
  (automatically provided by revision control software).

- **effort**…
  - for coding, review, testing, verification, fixing, maintenance, etc.

- **errors**…
  - at least errors found during quality assurance, and errors reported by customer
  (can be recorded via standardised revision control messages)

---

**Example: A Metric for Maintainability**

- Goal: assess **maintainability**.
- One approach: **grade** the following:
  (Some aspects may be objective, some subjective)

  - **Norm Conformance**
    - $n_1$: size of units (modules etc.)
    - $n_2$: labelling
    - $n_3$: naming of identifiers
    - $n_4$: design (layout)
    - $n_5$: separation of literals
    - $n_6$: style of comments

  - **Locality**
    - $l_1$: use of parameters, i.e., how to compute the metric
    - $l_2$: interface, flow
    - $l_3$: design

  - **Readability**
    - $r_1$: data types
    - $r_2$: structure of control flow
    - $r_3$: comments

  - **Typing**
    - $t_1$: type differentiation
    - $t_2$: type restriction

  - **Development of a pseudo-metrics:**
    - (i) Identify aspect to be represented.
    - (ii) Devise a model of the aspect.
    - (iii) Fix a scale for the metric.
    - (iv) Develop a definition of the pseudo-metric.
    - (v) Develop base measures for all parameters of the definition.
    - (vi) Apply and improve the metric

**Define:** $m = \frac{n_1 + \ldots + n_6}{20}$  
(with weights: $m_g = \frac{g_1 \cdot n_1 + \ldots + g_6 \cdot n_6}{G}$, $G = \sum_{i=1}^{20} g_i$)

**Procedure:**
- Train reviewers on existing examples.
- Do not over-interpret results of first applications.
- Evaluate and adjust before putting to use, adjust regularly.

(Ludwig and Lichter, 2013)
Often useful: collect some basic measures in advance (in particular if collection is cheap / automatic), e.g.:

- **size**...
  - of newly created and changed code, etc.
    (automatically provided by revision control software).

- **effort**...
  - for coding, review, testing, verification, fixing, maintenance, etc.

- **errors**...
  - at least errors found during quality assurance, and errors reported by customer
    (can be recorded via standardised revision control messages).

Measures derived from such basic measures may indicate problems ahead early enough and buy time to take appropriate counter-measures. E.g., track

- error rate per release, error density (errors per LOC),
- average effort for error detection and correction,
- etc.

over time. In case of unusual values: investigate further (maybe using additional metrics).
And Which Metrics Should One Use?

Often useful: collect some basic measures (in particular if collection is cheap / automated), e.g.:

- **size**... of newly created and changed code, etc. (automatically provided by revision control software).
- **effort**... for coding, review, testing, verification, fixing, maintenance, etc.
- **errors**... at least errors found during quality assurance, and errors reported by customer (can be recorded via standardised revision control messages).

*Measures derived* from such basic measures may indicate problems ahead and buy time to take appropriate counter-measures. E.g.,

- error rate per release, error density (errors per LOC),
- average effort for error detection and correction,
- etc.

over time. In case of **unusual values**: investigate further (maybe using additional metrics).

---

**Content**

- **Software Metrics**
  - Subjective Metrics
  - Goal-Question-Metric Approach

- **Cost Estimation**
  - "(Software) Economics in a Nutshell"
  - Cost Estimation
    - Expert’s Estimation
      - The Delphi Method
    - Algorithmic Estimation
      - COCOMO
      - Function Points
Content

- **Software Metrics**
  - Subjective Metrics
  - Goal-Question-Metric Approach

- **Cost Estimation**
  - "(Software) Economics in a Nutshell"
  - Cost Estimation
    - Expert's Estimation
      - The Delphi Method
    - Algorithmic Estimation
      - COCOMO
      - Function Points
  - Algorithmic Estimation
Costs

“Next to 'Software,' ‘Costs’ is one of the terms occurring most often in this book.”
Ludewig and Lichter (2013)

A first approximation:

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

Note: costs / benefits can be subjective – and not necessarily quantifiable in terms of money...

Super-ordinate goal of many projects:
• Minimize overall costs, i.e. maximise difference between benefits and costs.
  (Equivalent: minimize sum of positive and negative costs.)
Costs vs. Benefits: A Closer Look

The benefit of a software is determined by the advantages achievable using the software; it is influenced by:

- the degree of coincidence between product and requirements,
- additional services, comfort, flexibility etc.

Some other examples of cost/benefit pairs: (inspired by Jones (1990))

<table>
<thead>
<tr>
<th>Costs</th>
<th>Possible Benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Labor during development (e.g., develop new test machinery)</td>
<td>Use of result (e.g., faster testing)</td>
</tr>
<tr>
<td>New equipment (purchase, maintenance, depreciation)</td>
<td>Better equipment (maintenance; maybe revenue from selling old)</td>
</tr>
<tr>
<td>New software purchases</td>
<td>(Other) use of new software</td>
</tr>
<tr>
<td>Conversion from old system to new</td>
<td>Improvement of system, maybe easier maintenance</td>
</tr>
<tr>
<td>Increased data gathering</td>
<td>Increased control</td>
</tr>
<tr>
<td>Training for employees</td>
<td>Increased productivity</td>
</tr>
</tbody>
</table>

Costs: Economics in a Nutshell

Distinguish current cost (‘laufende Kosten’), e.g.

- wages,
- (business) management, marketing,
- rooms,
- computers, networks, software as part of infrastructure,
- ...

and project-related cost (‘projektbezogene Kosten’), e.g.

- additional temporary personnel,
- contract costs,
- expenses,
- hardware and software as part of product or system,
- ...

Business administration

Project leader involved
Software Costs in a Narrower Sense

- software costs
  - net production
  - quality costs
    - error prevention costs
    - analyse-and-fix costs
  - maintenance (without quality)
  - error costs
    - error localisation costs
    - error removal costs
    - decreased benefit
  - error caused costs (in operation)

Quality assurance: during and after development

Ludewig and Lichter (2013)

Cost Estimation

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)

commons.wikimedia.org (CC-by-sa 3.0)
### Why Estimate Cost?

- Developer can help with writing the requirements specification, in particular if customer is lacking technical background.

  - **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 Realisierungsanforderungen 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]

  - One way of getting the feature specification: a pre-project (may be subject of a designated contract). The same content can serve both purposes; then only the title defines the purpose.
The "Estimation Funnel"

Uncertainty with estimations (following (Boehm et al., 2000), p. 10).
Visualisation: Ludewig and Lichter (2013)

Expert's Estimation
**Expert’s Estimation**

**One approach:** the Delphi method.

- **Step 1:** write down your estimates!
  
- **Step 2:** show your estimates and explain!
  
- **Step 3:** estimate again!
  
- Then take the median, for example.

**Algorithmic Estimation**
Assume:
- Projects $P_1, \ldots, P_5$ took place in the past.
- Sizes $S_i$, costs $C_i$, and kinds $k_i$ ($0 = \text{blue-ish}, 1 = \text{yellow-ish}$) have been measured and recorded.

Question: What is the cost of the new project $P_6$?

Approach:
(i) Try to find a function $f$ such that $f(S_i, k_i) = C_i$, for $1 \leq i \leq 5$.
(ii) Estimate size $\hat{S}_6$ and kind $\hat{k}_6$.
(iii) Estimate cost $C_6$ as $\hat{C}_6 = f(\hat{S}_6, \hat{k}_6)$.

(In the artificial example above, $f(S, k) = S \cdot 1.8 + k \cdot 0.3$ would work, i.e.
if $P_6$ is of kind yellow (thus $\hat{k}_6 = 1$) and size estimate is $\hat{S}_6 = 2.7$ then estimate $C_6$ as $f(\hat{S}_6, \hat{k}_6) = 5.16$.)

Approach, more general:
(i) Identify (measurable) factors $F_1, \ldots, F_n$ which influence overall cost, like size in LOC.
(ii) Take a big sample of data from previous projects.
(iii) Try to come up with a formula $f$ such that $f(F_1, \ldots, F_n)$ matches previous costs.
(iv) Estimate values for $F_1, \ldots, F_n$ for a new project.
(v) Take $f(\tilde{F}_1, \ldots, \tilde{F}_n)$ as cost estimate $\tilde{C}$ for the new project.
(vi) Conduct new project, measure $F_1, \ldots, F_n$ and cost $C$.
(vii) Adjust $f$ if $C$ is too different from $\tilde{C}$.

Note:
- The need for (expert’s) estimation does not go away: one needs to estimate $F_1, \ldots, F_n$.
- Rationale: it is often easier to estimate technical aspects than to estimate cost directly.
Algorithmic Estimation: COCOMO

- **Constructive Cost Model:**
  Formulae which fit a huge set of archived project data (from the late 70’s).

- **Flavours:**
  - COCOMO 81 (Boehm, 1981): variants **basic, intermediate, detailed**
  - COCOMO II (Boehm et al., 2000)

- All flavours are based on estimated program size $S$ measured in DSI (Delivered Source Instructions) or kDSI (1000 DSI).

- Factors like security requirements or experience of the project team are mapped to values for parameters of the formulae.

- COCOMO examples:
  - textbooks like Ludewig and Lichter (2013) (most probably made up)
  - an exceptionally large example: COCOMO 81 for the Linux kernel (Wheeler, 2006) (and follow-ups)
Basic COCOMO:

- **effort required:** \( E = a \cdot \left( \frac{S}{kDSI} \right)^b \) [PM (person-months)]
- **time to develop:** \( T = c \cdot E^d \) [months]
- **headcount:** \( H = \frac{E}{T} \) [FTE (full time employee)]
- **productivity:** \( P = \frac{S}{E} \) [DSI per PM] (← use to check for plausibility)

Intermediate COCOMO:

\[
E = M \cdot a \cdot \left( \frac{S}{kDSI} \right)^b \quad \text{[person-months]}
\]

\[
M = RELY \cdot CPLX \cdot TIME \cdot ACAP \cdot PCAP \cdot LEXP \cdot TOOL \cdot SCED
\]

### COCOMO 81: Some Cost Drivers

\[
M = RELY \cdot CPLX \cdot TIME \cdot ACAP \cdot PCAP \cdot LEXP \cdot TOOL \cdot SCED
\]

<table>
<thead>
<tr>
<th>factor</th>
<th>very low</th>
<th>low</th>
<th>normal</th>
<th>high</th>
<th>very high</th>
<th>extra high</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELY  required software reliability</td>
<td>0.75</td>
<td>0.88</td>
<td>1</td>
<td>1.15</td>
<td>1.40</td>
<td></td>
</tr>
<tr>
<td>CPLX  product complexity</td>
<td>0.70</td>
<td>0.85</td>
<td>1</td>
<td>1.15</td>
<td>1.30</td>
<td>1.65</td>
</tr>
<tr>
<td>TIME  execution time constraint</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1.11</td>
<td>1.30</td>
<td>1.66</td>
</tr>
<tr>
<td>ACAP  analyst capability</td>
<td>1.46</td>
<td>1.19</td>
<td>1</td>
<td>0.86</td>
<td>0.71</td>
<td></td>
</tr>
<tr>
<td>PCAP  programmer capability</td>
<td>1.42</td>
<td>1.17</td>
<td>1</td>
<td>0.86</td>
<td>0.7</td>
<td></td>
</tr>
<tr>
<td>LEXP  programming language experience</td>
<td>1.14</td>
<td>1.07</td>
<td>1</td>
<td>0.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOOL  use of software tools</td>
<td>1.24</td>
<td>1.10</td>
<td>1</td>
<td>0.91</td>
<td>0.83</td>
<td></td>
</tr>
<tr>
<td>SCED  required development schedule</td>
<td>1.23</td>
<td>1.08</td>
<td>1</td>
<td>1.04</td>
<td>1.10</td>
<td></td>
</tr>
</tbody>
</table>

- **Note:** what, e.g., “extra high” TIME means, may depend on project context. (Consider data from previous projects.)
COCOMO II (Boehm et al., 2000)

Consists of

- **Application Composition Model** - project work is configuring components, rather than programming
- **Early Design Model** - adaptation of Function Point approach (in a minute); does not need completed architecture design
- **Post-Architecture Model** - improvement of COCOMO 81; needs completed architecture design, and size of components estimatable

\[ E = 2.94 \cdot S^X \cdot M \]

**Program size:**
\[ S = (1 + \text{REVL}) \cdot (S_{new} + S_{equiv}) \]

- requirements volatility \( \text{REVL} \): e.g., if new requirements make 10% of code unusable, then \( \text{REVL} = 0.1 \)
- \( S_{new} \): estimated size minus size \( w \) of re-used code.
- \( S_{equiv} = w/q \), if writing new code takes \( q \)-times the effort of re-use.

**Scaling factors:**
\[ X = \delta + \omega, \quad \omega = 0.91, \quad \delta = \frac{1}{100} \cdot (\text{PREC} + \text{FLEX} + \text{RESL} + \text{TEAM} + \text{PMAT}) \]

<table>
<thead>
<tr>
<th>factor</th>
<th>very low</th>
<th>low</th>
<th>normal</th>
<th>high</th>
<th>very high</th>
<th>extra high</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREC</td>
<td>6.20</td>
<td>4.96</td>
<td>3.72</td>
<td>2.48</td>
<td>1.24</td>
<td>0.00</td>
</tr>
<tr>
<td>FLEX</td>
<td>5.07</td>
<td>4.05</td>
<td>3.04</td>
<td>2.03</td>
<td>1.01</td>
<td>0.00</td>
</tr>
<tr>
<td>RESL</td>
<td>7.07</td>
<td>5.65</td>
<td>4.24</td>
<td>2.83</td>
<td>1.41</td>
<td>0.00</td>
</tr>
<tr>
<td>TEAM</td>
<td>5.48</td>
<td>4.38</td>
<td>3.29</td>
<td>2.19</td>
<td>1.10</td>
<td>0.00</td>
</tr>
<tr>
<td>PMAT</td>
<td>7.80</td>
<td>6.24</td>
<td>4.69</td>
<td>3.12</td>
<td>1.56</td>
<td>0.00</td>
</tr>
</tbody>
</table>
### Product factors
- **RELY**: required software reliability
- **DATA**: size of database
- **CPLX**: complexity of system
- **RUSE**: degree of development of reusable components
- **DOCU**: amount of required documentation

### Platform factors
- **TIME**: execution time constraint
- **STOR**: memory consumption constraint
- **PVOL**: stability of development environment

### Team factors
- **ACAP**: analyst capability
- **PCAP**: programmer capability
- **PCON**: continuity of involved personnel
- **APEX**: experience with application domain
- **PLEX**: experience with development environment
- **LTEX**: experience with programming language(s) and tools

### Project factors
- **TOOL**: use of software tools
- **SITE**: degree of distributedness
- **SCED**: required development schedule

(Also in COCOMO 81, new in COCOMO II)

### Algorithmic Estimation: Function Points

\[ M = \text{RELY} \times \text{DATA} \times \cdots \times \text{SCED} \]
<table>
<thead>
<tr>
<th>Type</th>
<th>Complexity</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>low</td>
<td>medium</td>
</tr>
<tr>
<td>input</td>
<td>.3</td>
<td>.4</td>
</tr>
<tr>
<td>output</td>
<td>.4</td>
<td>.5</td>
</tr>
<tr>
<td>query</td>
<td>.3</td>
<td>.4</td>
</tr>
<tr>
<td>user data</td>
<td>.7</td>
<td>.10</td>
</tr>
<tr>
<td>reference data</td>
<td>.5</td>
<td>.7</td>
</tr>
</tbody>
</table>

Unadjusted function points (UFP)

Value adjustment factor (VAF)

Adjusted function points (AFP) = UFP · VAF

\[
VAF = 0.65 + \frac{1}{100} \sum_{i=1}^{14} GSC_i, \quad 0 \leq GSC_i \leq 5.
\]
Discussion

Ludewig and Lichter (2013) says:

- **Function Point** approach used in practice, in particular for commercial software (business software?).
- **COCOMO** tends to **overestimate** in this domain; needs to be adjusted by corresponding factors.

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

“Estimate, document, estimate better.” (Ludewig and Lichter, 2013)

**Suggestion**: start to explicate your experience now.

- **Take notes on your projects**:
  (e.g., Softwarepraktikum, Bachelor Projekt, Master Bacherlor's Thesis, Master Projekt, Master's Thesis, …)
- **timestamps, size of program created, number of errors found, number of pages written, …**
- **Try to identify factors**: what hindered productivity, what boosted productivity, …
- **Which detours and mistakes** were **avoidable** in hindsight? How?
Tell Them What You’ve Told Them...

- **Goal-Question-Metric** approach:
  - Define goals, derive questions, choose metrics.
  - Evaluate and adjust.
  
  Recall: It’s about the **goal**, not the metrics.

- For **software costs**, we can distinguish
  - net production, quality costs, maintenance.

  **Software engineering** is about being **economic** in all three aspects.

- Why estimate?
  - **Requirements specification** (‘Lastenheft’)
  - **Feature specification** (‘Pflichtenheft’)

  The latter (plus budget) is usually part of **software contracts**.

- **Approaches**:
  - **Expert’s Estimation**
  - **Algorithmic Estimation**: COCOMO, Function Points
    → estimate cost **indirectly**, by estimating more **technical aspects**.

  In the end, it’s **experience** (and experience (and experience)).

---

**References**
References


