Softwaretechnik / Software-Engineering

Lecture 3: Metrics Cont’d & Cost Estimation

2016-04-25

Prof. Dr. Andreas Podelski, Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
Recall: Pseudo-Metrics

Some of the **most interesting aspects** of software development projects are **hard or impossible** to measure directly, e.g.:

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

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

<table>
<thead>
<tr>
<th>Expert review, grading</th>
<th>differentiated</th>
<th>comparable</th>
<th>reproducible</th>
<th>available</th>
<th>relevant</th>
<th>economical</th>
<th>plausible</th>
<th>robust</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>✔️</td>
<td>✔️</td>
<td>☒</td>
<td>✔️</td>
<td>!</td>
<td>☒</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<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).
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:

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

- **direct 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)
Subjective Metrics
# 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 coded 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)*
Example: A (Subjective) Metric for Maintainability

- **Goal:** assess maintainability.
- **One approach:** grade the following aspects, e.g., with scale $S = \{0, \ldots, 10\}$.

- **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:**
  $$m = \frac{n_1 + \cdots + y_2}{20} \quad \text{(with weights: } m_g = \frac{g_1 \cdot n_1 + \cdots + g_{20} \cdot y_2}{G}, \ G = \sum_{i=1}^{20} g_i)\text{)}$$

- **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)
**Example: A (Subjective) Metric for Maintainability**

- **Goal:** assess **maintainability**.
- **One approach:** grade the following aspects, e.g., with scale $S = \{0, \ldots, 10\}$.

**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

**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, i.e., how to compute the metric.
- (v) Develop **base measures** for all parameters of the definition.
- (vi) **Apply** and **improve** the metric.

**Define:** $m = \frac{n_1 + \cdots + y_2}{20}$ (with weights: $m_g = \frac{g_1 \cdot n_1 + \cdots + g_{20} \cdot y_2}{G}$, $G = \sum_{i=1}^{20} g_i$).

**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)
The Goal-Question-Metric Approach
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.
And Which Metrics Should One Use?
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)

**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 / automatic), e.g.:

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

- effort…
  - for coding, review, testing, verification, 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 / automatic), e.g.:

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

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

- **errors**…
  - … at least errors *found* during quality assurance, and *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).
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**

- **Goal-Question-Metric** approach:
  - it’s about the **goal**, not the metrics.

- Communicating figures: consider **percentiles**.
Topic Area Project Management: Content

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

VL 3
- Cost Estimation
  - "(Software) Economics in a Nutshell"
  - Expert's Estimation
  - Algorithmic Estimation

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

VL 5
- Process Metrics
  - CMMI, Spice
“(Software) Economics in a Nutshell”
“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.)
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</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>
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,
- ...

Software Costs in a Narrower Sense

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)

Ludewig and Lichter (2013)
Discovering Fundamental Errors Late Can Be Expensive

Relative error costs over latency according to investigations at IBM, etc.
By (Boehm, 1979); Visualisation: Ludewig and Lichter (2013).
Cost Estimation
**Why Estimate Cost?**

---

**Customer announcement (Lastenheft)** → **Customer offer (Pflichtenheft)** → **Customer software contract (incl. Pflichtenheft)** → **Developer software delivery**

---

**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.)

**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.)

---

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

- One way of getting the feature specification: **a pre-project** (may be subject of a designated contract).

- **Tricky**: one and 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)
Plan

Cost Estimation

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

Assume:
- Projects $P_1, \ldots, P_5$ took place in the past,
- Sizes $S_i$, costs $C_i$, and kinds $k_i$ ($0 = \text{blue}, 1 = \text{yellow}$) 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 $\tilde{S}_6$ and kind $\tilde{k}_6$.
(iii) Estimate $C_6$ as $\tilde{C}_6 = f(\tilde{S}_6, \tilde{k}_6)$.

(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 ($\tilde{k}_6 = 1$) and size estimate is $\tilde{S}_6 = 2.7$ then $f(\tilde{S}_6, \tilde{k}_6) = 5.16$)
Algorithmic Estimation: Principle

Approach, more general:

- Identify (measurable) factors $F_1, \ldots, F_n$ which influence overall cost, like size in LOC.
- Take a big sample of data from previous projects.
- Try to come up with a formula $f$ such that $f(F_1, \ldots, F_n)$ matches previous costs.
- Estimate values for $F_1, \ldots, F_n$ for a new project.
- Take $f(\tilde{F}_1, \ldots, \tilde{F}_n)$ as cost estimate $\tilde{C}$ for new project.
- Conduct new project, measure $F_1, \ldots, F_n$ and cost $C$.
- Adjust $f$ if $C \neq \tilde{C}$. 
Algorithmic Estimation: Principle

Approach, more general:

- Identify (measurable) factors $F_1, \ldots, F_n$ which influence overall cost, like size in LOC.
- Take a big sample of data from previous projects.
- Try to come up with a formula $f$ such that $f(F_1, \ldots, F_n)$ matches previous costs.
- Estimate values for $F_1, \ldots, F_n$ for a new project.
- Take $f(\tilde{F}_1, \ldots, \tilde{F}_n)$ as cost estimate $\tilde{C}$ for new project.
- Conduct new project, measure $F_1, \ldots, F_n$ and cost $C$.
- Adjust $f$ if $C \neq \tilde{C}$.

Note:

- The need for (expert’s) estimation does not go away: one needs to estimate $\tilde{F}_1, \ldots, \tilde{F}_n$.
- Rationale: it is often easier to estimate technical aspect than to directly estimate cost.
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): basic, intermediate, detailed
  - COCOMO II (Boehm et al., 2000)

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

- 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)
## COCOMO 81

<table>
<thead>
<tr>
<th>Size</th>
<th>Characteristics of the Type</th>
<th></th>
<th>a</th>
<th>b</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Small</strong> (＜50 KLOC)</td>
<td>Little</td>
<td>Not tight</td>
<td>Stable</td>
<td>3.2</td>
<td>1.05</td>
</tr>
<tr>
<td><strong>Medium</strong> (＜300 KLOC)</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>3.0</td>
<td>1.12</td>
</tr>
<tr>
<td><strong>Large</strong></td>
<td>Greater</td>
<td>Tight</td>
<td>Complex HW/Interfaces</td>
<td>2.8</td>
<td>1.20</td>
</tr>
</tbody>
</table>

### 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**: \( H = E/T \) [FTE (full time employee)]
- **productivity**: \( P = S/E \) [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**

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

<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</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</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</td>
<td></td>
<td></td>
<td>1</td>
<td>1.11</td>
<td>1.30</td>
<td>1.66</td>
</tr>
<tr>
<td>ACAP</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</td>
<td>1.42</td>
<td>1.17</td>
<td>1</td>
<td>0.86</td>
<td>0.70</td>
<td></td>
</tr>
<tr>
<td>LEXP</td>
<td>1.14</td>
<td>1.07</td>
<td>1</td>
<td>0.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>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</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** – adaption 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
COCOMO II: Post-Architecture

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

- **Program size:** \( S = (1 + REVL) \cdot (S_{\text{new}} + S_{\text{equiv}}) \)
  - **requirements volatility** \( REVL \):
    e.g., if new requirements make 10% of code unusable, then \( REVL = 0.1 \)
  - \( S_{\text{new}} \): estimated size minus size of re-used code,
  - \( S_{\text{equiv}} = w/q \), if writing new code takes \( q \)-times the effort of re-use.

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

<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>precedentness (experience with similar projects)</td>
<td>6.20</td>
<td>4.96</td>
<td>3.72</td>
<td>2.48</td>
<td>1.24</td>
</tr>
<tr>
<td>FLEX</td>
<td>development flexibility (development process fixed by customer)</td>
<td>5.07</td>
<td>4.05</td>
<td>3.04</td>
<td>2.03</td>
<td>1.01</td>
</tr>
<tr>
<td>RESL</td>
<td>Architecture/risk resolution (risk management, architecture size)</td>
<td>7.07</td>
<td>5.65</td>
<td>4.24</td>
<td>2.83</td>
<td>1.41</td>
</tr>
<tr>
<td>TEAM</td>
<td>Team cohesion (communication effort in team)</td>
<td>5.48</td>
<td>4.38</td>
<td>3.29</td>
<td>2.19</td>
<td>1.10</td>
</tr>
<tr>
<td>PMAT</td>
<td>Process maturity (see CMMI)</td>
<td>7.80</td>
<td>6.24</td>
<td>4.69</td>
<td>3.12</td>
<td>1.56</td>
</tr>
</tbody>
</table>
**COCOMO II: Post-Architecture Cont’d**

\[ M = RELY \cdot DATA \cdot \cdots \cdot SCED \]

<table>
<thead>
<tr>
<th>group</th>
<th>factor</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Product factors</strong></td>
<td>RELY</td>
<td>required software reliability</td>
</tr>
<tr>
<td></td>
<td>DATA</td>
<td>size of database</td>
</tr>
<tr>
<td></td>
<td>CPLX</td>
<td>complexity of system</td>
</tr>
<tr>
<td></td>
<td>RUSE</td>
<td>degree of development of reusable components</td>
</tr>
<tr>
<td></td>
<td>DOCU</td>
<td>amount of required documentation</td>
</tr>
<tr>
<td><strong>Platform factors</strong></td>
<td>TIME</td>
<td>execution time constraint</td>
</tr>
<tr>
<td></td>
<td>STOR</td>
<td>memory consumption constraint</td>
</tr>
<tr>
<td></td>
<td>PVOL</td>
<td>stability of development environment</td>
</tr>
<tr>
<td><strong>Team factors</strong></td>
<td>ACAP</td>
<td>analyst capability</td>
</tr>
<tr>
<td></td>
<td>PCAP</td>
<td>programmer capability</td>
</tr>
<tr>
<td></td>
<td>PCON</td>
<td>continuity of involved personnel</td>
</tr>
<tr>
<td></td>
<td>APEX</td>
<td>experience with application domain</td>
</tr>
<tr>
<td></td>
<td>PLEX</td>
<td>experience with development environment</td>
</tr>
<tr>
<td></td>
<td>LTEX</td>
<td>experience with programming language(s) and tools</td>
</tr>
<tr>
<td><strong>Project factors</strong></td>
<td>TOOL</td>
<td>use of software tools</td>
</tr>
<tr>
<td></td>
<td>SITE</td>
<td>degree of distributedness</td>
</tr>
<tr>
<td></td>
<td>SCED</td>
<td>required development schedule</td>
</tr>
</tbody>
</table>

(Also in COCOMO 81, new in COCOMO II)
Function Points
### Algorithmic Estimation: Function Points

<table>
<thead>
<tr>
<th>Type</th>
<th>Complexity</th>
<th>Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>low: 0.3 =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>medium: 0.4 =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>high: 0.6 =</td>
<td></td>
</tr>
<tr>
<td>output</td>
<td>low: 0.4 =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>medium: 0.5 =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>high: 0.7 =</td>
<td></td>
</tr>
<tr>
<td>query</td>
<td>low: 0.3 =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>medium: 0.4 =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>high: 0.6 =</td>
<td></td>
</tr>
<tr>
<td>user data</td>
<td>low: 0.7 =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>medium: 1.0 =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>high: 1.5 =</td>
<td></td>
</tr>
<tr>
<td>reference data</td>
<td>low: 0.5 =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>medium: 0.7 =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>high: 1.0 =</td>
<td></td>
</tr>
</tbody>
</table>

Unadjusted function points (UFP)

Value adjustment factor (VAF)

Adjusted function points (AFP = UFP \cdot VAF)

\[
VAF = 0.65 + \frac{1}{100} \cdot \sum_{i=1}^{14} GSC_i, \\
0 \leq GSC_i \leq 5.
\]
## Algorithmic Estimation: Function Points

<table>
<thead>
<tr>
<th>Type</th>
<th>Complexity</th>
<th>Unadjusted function points (UFP)</th>
<th>Value adjustment factor (VAF)</th>
<th>Adjusted function points (AFP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>low</td>
<td>0.3 =</td>
<td></td>
<td>UFP · VAF</td>
</tr>
<tr>
<td>output</td>
<td>medium</td>
<td>0.4 =</td>
<td></td>
<td></td>
</tr>
<tr>
<td>query</td>
<td>low</td>
<td>0.3 =</td>
<td></td>
<td></td>
</tr>
<tr>
<td>user data</td>
<td>high</td>
<td>0.7 =</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reference data</td>
<td>high</td>
<td>0.5 =</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IBM and VW curve for the conversion from AFPs to PM according to (Noth and Kretzschmar, 1984) and (Knöll and Busse, 1991).

$$VAF = 0.65 + \frac{1}{100} \sum_{i=1}^{14} GSC_i,$$

$$0 \leq GSC_i \leq 5.$$
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...

- 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**.
References
References


