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

Content

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

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

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:

- differentiated comparable reproducible available relevant economical plausible robust

Expert review, grading (✔) (✘)

Pseudo-metrics, derived measures

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).

Can Pseudo-Metrics be Useful?

- 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:

  valuation yield low high quality high false positive × true positive × × × × × true negative × × × × × false negative × × × × ×

- 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)

- metric computation weighted methods per class (WMC)
  \[ n \sum_{i=1}^{c_i}, n = \text{number of methods}, c_i = \text{complexity of method} \]

- depth of inheritancetree (DIT)
  graph distance in inheritance tree (multiple inheritance)

- number of children of a class (NOC)
  number of direct subclasses of the class

- coupling between object classes (CBO)
  \[ CBO(C) = |K_o \cup K_i|, K_o = \text{set of classes used by } C, K_i = \text{set of classes using } C \]

- response for a class (RFC)
  \[ RFC = |M \cup \bigcup_i R_i|, M = \text{set of methods of } C, R_i = \text{set of all methods calling method } i \]

- lack of cohesion in methods (LCOM)
  \[ \max(|P| - |Q|, 0), P = \text{methods using no common attribute}, Q = \text{methods using at least one common attribute} \]

- 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

Example problems
countermeasures

Statement

"The specification is available."

Terms may be ambiguous, conclusions are hardly possible.

Allow only certain statements, characterise them precisely.

Assessment

"The module is coded in a clever way."

Not necessarily comparable.

Only offer particular outcomes; put them on an (at least ordinal) scale.

Grading

"Readability is graded 4.0."

Subjective; grading not reproducible.

Define criteria for grades; give examples how to grade; practice on existing artefacts (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: $m = n_1 + \cdots + y_2$

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

Information Overload!?

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

Which ones should we measure?

It depends . . .

relevant plausible available differentiated economical comparable reproducible robust

One approach: Goal-Question-Metric (GQM).

Goal-Question-Metric (Basili and Weiss, 1984)

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?

- At a high level, process-metrics are focused on:
  - Predict
  - Diagnose
  - Prognose

- For example, maintainers use software attributes like:
  - McCabe / Cyclomatic Complexity
  - LOC, number of changed lines over time

- Use these measures to assess changes to improve or maintain quality:
  - For example, LOC and changed lines over time (obtained by statsvn).

And Which Metrics Should One Use?

- To describe software attributes:
  - For example, pseudo-metrics like:
    - McCabe / Cyclomatic Complexity
    - LOC, number of changed lines over time

- To assess changes:
  - For example, software attributes like:
    - McCabe / Cyclomatic Complexity
    - LOC, number of changed lines over time

- To support decision-making:
  - For example, software attributes like:
    - McCabe / Cyclomatic Complexity
    - LOC, number of changed lines over time

- To assess software attribute changes:
  - For example, software attributes like:
    - McCabe / Cyclomatic Complexity
    - LOC, number of changed lines over time

And Which Metrics Should One Use?

- To diagnose software attribute changes:
  - For example, software attributes like:
    - McCabe / Cyclomatic Complexity
    - LOC, number of changed lines over time

- To predict software attribute changes:
  - For example, software attributes like:
    - McCabe / Cyclomatic Complexity
    - LOC, number of changed lines over time

- To prescribe actions:
  - For example, software attributes like:
    - McCabe / Cyclomatic Complexity
    - LOC, number of changed lines over time

And Which Metrics Should One Use?

- To assess software attribute changes:
  - For example, software attributes like:
    - McCabe / Cyclomatic Complexity
    - LOC, number of changed lines over time

- To support decision-making:
  - For example, software attributes like:
    - McCabe / Cyclomatic Complexity
    - LOC, number of changed lines over time

- To diagnose software attribute changes:
  - For example, software attributes like:
    - McCabe / Cyclomatic Complexity
    - LOC, number of changed lines over time

And Which Metrics Should One Use?
"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: cost ('Kosten') all disadvantages of a solution benefit ('Nutzen') all benefits of a solution. (or: negative costs)

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. maximize 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</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

Ludewig and Lichter (2013)

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)

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

The "Estimation Funnel"

\[ 4 \times 2 \times 1 \times 0.5 \times 0.25 \]

effort estimated to real effort (log. scale)

Pre-Project Analysis

Design

Coding & Test

Uncertainty with estimations (following (Boehm et al., 2000), p. 10).

Visualisation:

Ludewig and Lichter (2013)

• Cost Estimation
• "(Software) Economics in a Nutshell"
• Cost Estimation
• 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.
than to directly estimate cost. Rationale: it is often easier to estimate

$\text{SCED} \cdot \text{PCAP} \cdot \text{ACAP} \cdot \ldots \cdot \text{E}$

and follow-ups,$n$.

Wheeler: $\text{COCOMO 81 for the Linux kernel (}$C\text{)}$ and size estimate is $\text{COCOMO II (}$<$\text{300 KLOC)}$.

An exceptionally large example: COCOMO 81 for the Linux kernel ($C$)

Take $\tilde{C}$ (most probably made up)

$n$ matches previous costs.

Estimate $f_{PM}$ (person-months)

Try to come up with a formula $k_{DSI} = E$ effort required

Identify (measurable) factors $P_{t}$ which influence overall cost, like size in LOC.

Constraints

Approach

Conduct new project, measure $f_{PM}$, $n = \ldots , f_{PM}$ have been measured and recorded.

$P_{t}$

Assume $\tilde{C}$ as $C$ if $f_{PM}$ take place in the past, $S_{\tilde{C}}$ would work, $\ldots , F$

$\text{DSI per PM}$

$S/E$ for a new project.

$\text{Flavours:}$

- Semi-detached
- Organic
- Constructive
- Stable
- Complex HW/

- Small
- Medium
- Large

- $1.12$
- $1.12$
- $3.2$
- $1.20$

- $6$
- $16$

- $1$
- $2$
- $3$
- $0$

- $\cdot$
- $5$
- $2$
- $4$

- $k$
- $f$
- $P$
- $t$

- $E$
- $C$
- $F$
- $S$

- $\tilde{C}$
Algorithmic Estimation: Function Points

Complexity

<table>
<thead>
<tr>
<th>Type</th>
<th>low</th>
<th>medium</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>·3·</td>
<td>·4·</td>
<td>·6·</td>
</tr>
<tr>
<td>output</td>
<td>·3·</td>
<td>·4·</td>
<td>·6·</td>
</tr>
<tr>
<td>query</td>
<td>·3·</td>
<td>·4·</td>
<td>·6·</td>
</tr>
<tr>
<td>user data</td>
<td>·7·</td>
<td>·10·</td>
<td>·15·</td>
</tr>
<tr>
<td>reference data</td>
<td>·5·</td>
<td>·7·</td>
<td>·10·</td>
</tr>
</tbody>
</table>

Unadjusted function points (UFP)

Value adjustment factor (VAF)

Adjusted function points (AFP)

\[ AFP = UFP \cdot VAF \]

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

\[ VAF = 0.65 + 1 \cdot \frac{\sum_{i=1}^{GSC} i}{100} \cdot \frac{GSC_i}{5} \]

Discussion

Ludewig and Lichter (2013) says:

- Function Point approach used in practice, in particular for commercial 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 Bachelor'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...

Software costs can be distinguished as:

- 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