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:

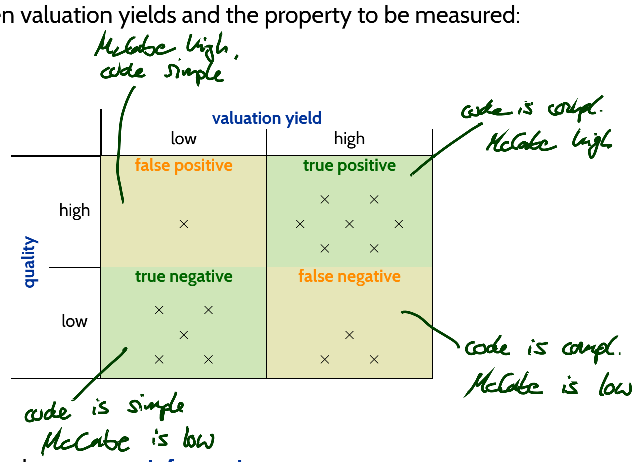
	differentiated	comparable	reproducible	available	relevant	economical	plausible	robust
Expert review, grading	(✔)	(✔)	(✘)	(✔)	✔!	(✘)	✔	✔
Pseudo-metrics, derived measures	✔	✔	✔	✔	✔!	✔	✘	✘

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:



- 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)	$\sum_{i=1}^{n} c_i$, n = number of methods, c_i = complexity of method i
depth of inheritance tree (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 = set of classes used by C, K_i = set of classes using C
response for a class (RFC)	$RFC = \|M \cup \bigcup_i R_i\|$, M set of methods of C, R_i set of all methods calling method i
lack of cohesion in methods (LCOM)	$\max(\|P\| - \|Q\|, 0)$, P = methods using no common attribute, Q = 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

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 $S = \{0, \dots, 10\}$.

- **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**: $m = \frac{n_1 + \cdots + y_2}{20}$ (**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)

Example: A (Subjective) Metric for Maintainability

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

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.

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

– 3 – 2016-04-25 – Ssubjective –

The Goal-Question-Metric Approach

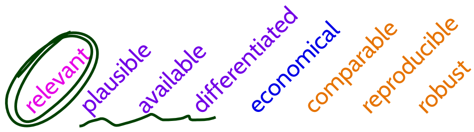
– 3 – 2016-04-25 – main –

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).

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?

- 3 - 2016-04-25 - Sgm -

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).

- 3 - 2016-04-25 - Sgm -

And Which Metrics Should One Use?

Often useful: collect some basic meas
(in particular if collection is cheap / aut

- **size**…

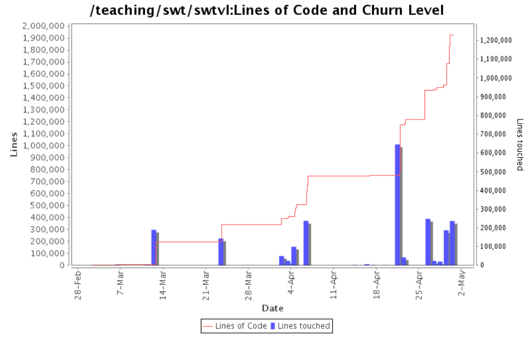
  … of newly **created** and **changed code**, e
  (automatically provided by revision co

- **effort**…

  … for **coding**, **review**, **testing**, **verificatio**

- **errors**…

  … at least errors **found** during quality ass
  (can be recorded via standardised revi



LOC and changed lines over time (obtained by `statsvn`(1).

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 meas
(in particular if collection is cheap / aut

- **size**…

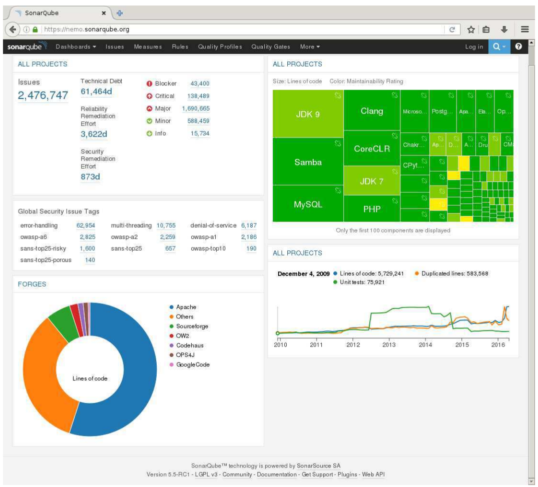
  … of newly **created** and **changed code**, e
  (automatically provided by revision co

- **effort**…

  … for **coding**, **review**, **testing**, **verificatio**

- **errors**…

  … at least errors **found** during quality ass
  (can be recorded via standardised revi



Tool support for software metrics, e.g., SonarCube.

Measures derived from such basic me
and buy time to take appropriate coun

- **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).

– 3 – 2016-04-25 – Sggm –

12/40

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
VL 5	– Process Models
⋮	• **Process Metrics**
	– CMMI, Spice

"(Software) Economics in a Nutshell"

Costs

"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') (or: negative costs)	all benefits of a solution.

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))

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

Costs: Economics in a Nutshell

Distinguish **current cost** ('laufende Kosten'), e.g.

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

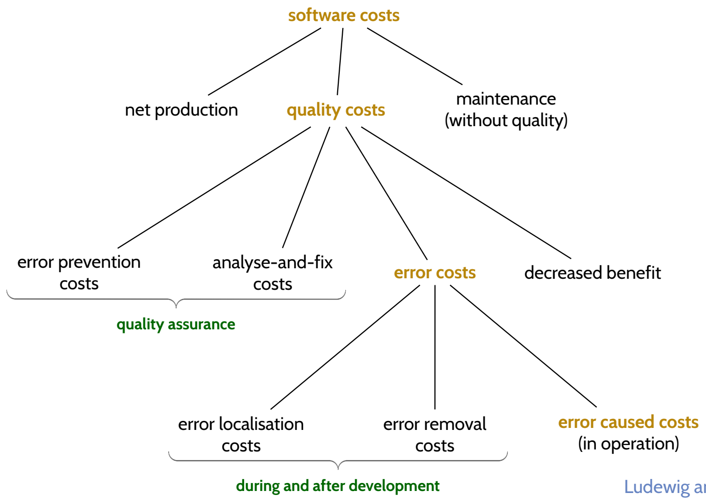
business administration

and **project-related cost** ('projektbezogene Kosten'), e.g.

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

project leader involved

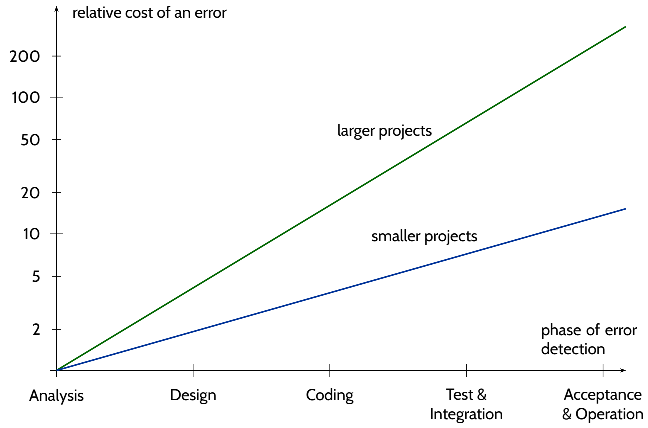
Software Costs in a Narrower Sense



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)

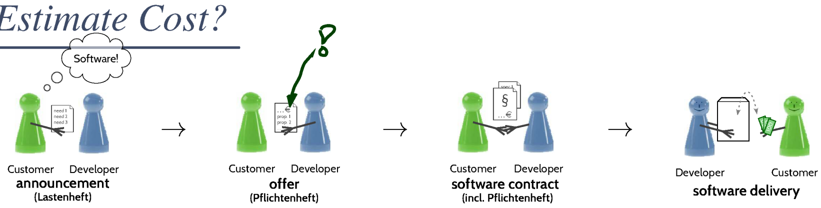
Discovering Fundamental Errors Late Can Be Expensive



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

Cost Estimation

Why Estimate Cost?



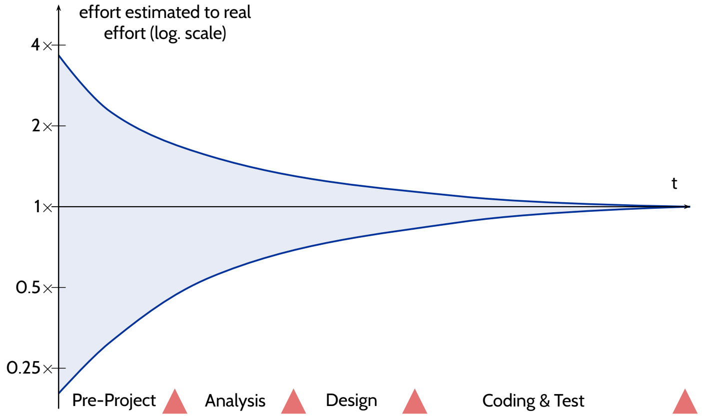
Lastenheft **(Requirements Specification)** Vom Auftraggeber festgelegte Gesamtheit der Forderungen an die Lieferungen und Leistungen eines Auftragnehmers innerhalb eines Auftrages.
(Entire demands on deliverables and services of a developer within a contracted development, created by the customer.) **DIN 69901-5 (2009)**

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

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)

- 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

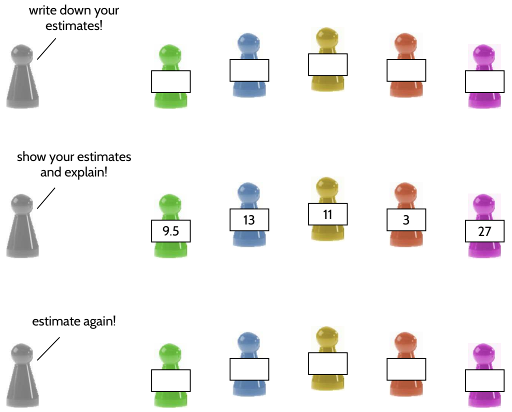
- **Cost Estimation**
  - "(Software) Economics in a Nutshell" ✓
  - Cost Estimation
    - Expert's Estimation
      - The Delphi Method
    - Algorithmic Estimation
      - COCOMO
      - Function Points

Expert's Estimation

Expert's Estimation

One approach: the **Delphi** method.

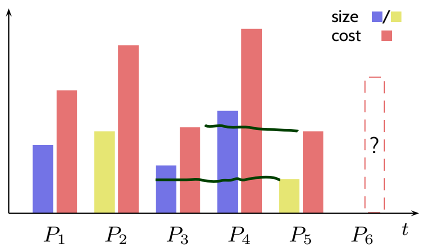
- Step 1:
- Step 2:
- Step 3:



- 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 ($0 =$ **blue**, $1 =$ **yellow**) have been measured and recorded.

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

Approach:

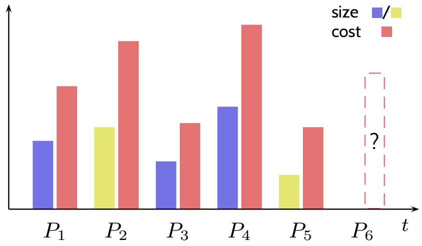
(i) Try to find a function f such that $f(S_i, k_i) = C_i$, for $1 \leq i \leq 5$.

(ii) Estimate size $\tilde{S}_6$ and kind $\tilde{k}_6$. ← estimate

(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 $f(\tilde{S}_6, \tilde{k}_6) = 5.16$)

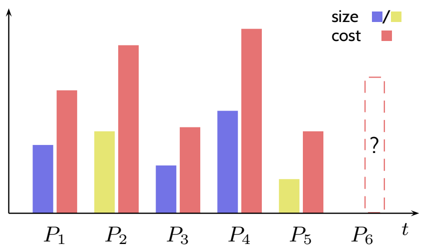
Algorithmic Estimation: Principle



Approach, more general:

- Identify (measurable) factors $F_1, \dots, F_n$ 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 $f(F_1, \dots, F_n)$ matches previous costs.
- **Estimate** values for $F_1, \dots, F_n$ for a new project.
- Take $f(\tilde{F}_1, \dots, \tilde{F}_n)$ as cost estimate $\tilde{C}$ for new project.
- Conduct new project, measure $F_1, \dots, F_n$ and cost C.
- Adjust f if $C \neq \tilde{C}$.

Algorithmic Estimation: Principle



Approach, more general:

- Identify (measurable) factors $F_1, \dots, F_n$ 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 $f(F_1, \dots, F_n)$ matches previous costs.
- **Estimate** values for $F_1, \dots, F_n$ for a new project.
- Take $f(\tilde{F}_1, \dots, \tilde{F}_n)$ as cost estimate $\tilde{C}$ for new project.
- Conduct new project, measure $F_1, \dots, F_n$ 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 $\tilde{F}_1, \dots, \tilde{F}_n$.
- Rationale: it is often easier to estimate technical aspect than to directly estimate cost.

Algorithmic Estimation: COCOMO

- **Co**nstructive **Co**st **Mo**del:
  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

Characteristics of the Type				a	b	Software Project Type
Size	Innovation	Deadlines/ Constraints	Dev. Environment			
Small (<50 KLOC)	Little	Not tight	Stable	3.2	1.05	Organic
Medium (<300 KLOC)	Medium	Medium	Medium	3.0	1.12	Semi-detached
Large	Greater	Tight	Complex HW/ Interfaces	2.8	1.20	Embedded

Basic COCOMO:

- **effort required**: $E = a \cdot (S/kDSI)^b$ [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:

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

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

COCOMO 81: Some Cost Drivers

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

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

- **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**: $S = (1 + REVL) \cdot (S_{new} + S_{equiv})$
  - **requirements volatility** $REVL$:
    e.g., if new requirements make 10% of code unusable, then $REVL = 0.1$
  - 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**:
  $X = \delta + \omega, \quad \omega = 0.91, \quad \delta = \frac{1}{100} \cdot (PREC + FLEX + RESL + TEAM + PMAT)$

factor		very low	low	normal	high	very high	extra high
PREC	**precedentness** (experience with similar projects)	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	**development flexibility** (development process fixed by customer)	5.07	4.05	3.04	2.03	1.01	0.00
RESL	**Architecture/risk resolution** (risk management, architecture size)	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	**Team cohesion** (communication effort in team)	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	**Process maturity** (see CMMI)	7.80	6.24	4.69	3.12	1.56	0.00

COCOMO II: Post-Architecture Cont'd

$$M = RELY \cdot DATA \cdot \cdots \cdot SCED$$

group	factor	description
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)

Function Points

Algorithmic Estimation: Function Points

classify data into...

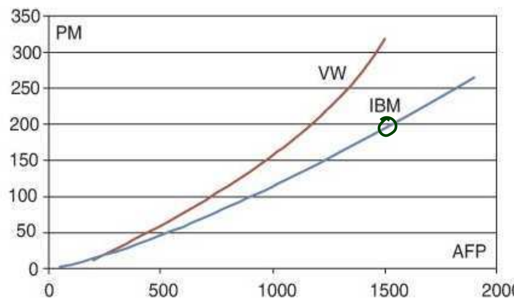
Type	Complexity low	Complexity medium	Complexity high	Sum
input	____ $\cdot 3 =$	____ $\cdot 4 =$	____ $\cdot 6 =$	
output	____ $\cdot 4 =$	____ $\cdot 5 =$	____ $\cdot 7 =$	
query	____ $\cdot 3 =$	____ $\cdot 4 =$	____ $\cdot 6 =$	
user data	____ $\cdot 7 =$	____ $\cdot 10 =$	____ $\cdot 15 =$	
reference data	____ $\cdot 5 =$	____ $\cdot 7 =$	____ $\cdot 10 =$	
Unadjusted function points	UFP			
Value adjustment factor	VAF			
Adjusted function points	AFP = UFP · VAF			

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

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

Algorithmic Estimation: Function Points

Type	low	me[…]		
input	____ $\cdot 3 =$	____		
output	____ $\cdot 4 =$	____		
query	____ $\cdot 3 =$	____		
user data	____ $\cdot 7 =$	____		
reference data	____ $\cdot 5 =$	____ $\cdot 7 =$	____ $\cdot 10 =$	
Unadjusted function points		UFP		
Value adjustment factor		VAF		
Adjusted function points		AFP = UFP · VAF		



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

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

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

Discussion

Ludewig and Lichter (2013) says:

- **Function Point** approach used in practice, in particular for **commercial software** (business software?).
  business secrets
- **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

Basili, V. R. and Weiss, D. M. (1984). A methodology for collecting valid software engineering data. *IEEE Transactions of Software Engineering*, 10(6):728–738.

Bauer, F. L. (1971). Software engineering. In *IFIP Congress (1)*, pages 530–538.

Boehm, B. W. (1979). Guidelines for verifying and validating software requirements and design specifications. In *EURO IFIP 79*, pages 711–719. Elsevier North-Holland.

Boehm, B. W. (1981). *Software Engineering Economics*. Prentice-Hall.

Boehm, B. W., Horowitz, E., Madachy, R., Reifer, D., Clark, B. K., Steece, B., Brown, A. W., Chulani, S., and Abts, C. (2000). *Software Cost Estimation with COCOMO II*. Prentice-Hall.

Chidamber, S. R. and Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493.

DIN (2009). *Projektmanagement; Projektmanagementsysteme*. DIN 69901-5.

Jones, G. W. (1990). *Software Engineering*. John Wiley & Sons.

Kan, S. H. (2003). *Metrics and models in Software Quality Engineering*. Addison-Wesley, 2nd edition.

Knöll, H.-D. and Busse, J. (1991). *Aufwandsschätzung von Software-Projekten in der Praxis: Methoden, Werkzeugeinsatz, Fallbeispiele*. Number 8 in Reihe Angewandte Informatik. BI Wissenschaftsverlag.

Ludewig, J. and Lichter, H. (2013). *Software Engineering*. dpunkt.verlag, 3. edition.

Noth, T. and Kretzschmar, M. (1984). *Aufwandsschätzung von DV-Projekten, Darstellung und Praxisvergleich der wichtigsten Verfahren*. Springer-Verlag.