Softwaretechnik / Software-Engineering

Lecture 2: Software Metrics

2017-04-27

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Topic Area Project Management: Content

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

⋮

VL 3
- **Cost Estimation**
  - Deadlines and Costs
  - Expert's Estimation
  - Algorithmic Estimation

⋮

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

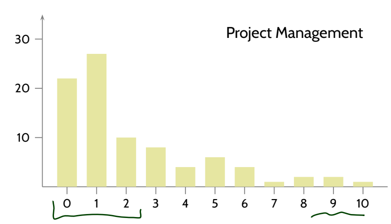
⋮

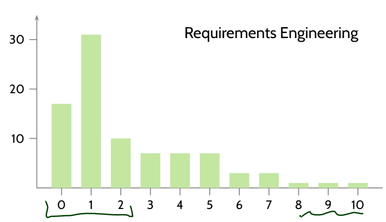
VL 5
- **Process Metrics**
  - CMMI, Spice

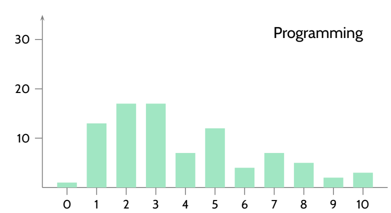
⋮

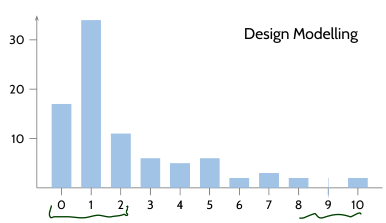
Content

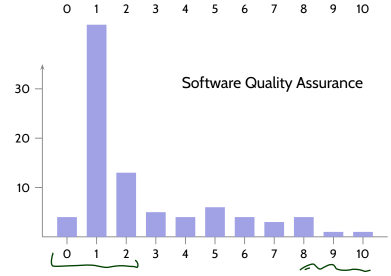
Survey: Previous Experience











Expectations

- **general**
  - ✔ work with others in a large software development team
  - ✔ communicate results to other people
  - ✔ learn how to properly document the work
  - ✔ know, how to acquire knowledge on aspects of SW Eng. on our own
  - ✔ get to know industry standards, investigate their strengths / weaknesses
  - ✔ overview, terminology, and references for own enquiries
  - ✘ know about trustful internet sources to get such information while working
  - ✔ understanding the procedure of software production, including common mishaps at each step
  - ✔ systematically analyse the steps of software development which are done "implicitly" in smaller, self-made projects
  - ✔ course is balanced with theoretical as well as practical scenarios
  - ✔ getting tools (roughly specific ideas) for attacking problems
  - ✔ have some fun, learn a lot [...] not only for the further studying or working but also for life
- **other courses**
  - (✘) Vorallem hoffe ich auf eine sinnvolle Verbindung zum Softwarepraktikum.

Introduction	L 1:	24.4., Mon
Scales, Metrics,	L 2:	27.4., Thu
	-	1.5., Mon
	T 1:	4.5., Thu
Costs,	L 3:	8.5., Mon
Development	L 4:	11.5., Thu
Process	L 5:	15.5., Mon
	T 2:	18.5., Thu
	L 6:	22.5., Mon
	-	25.5., Thu
Requirements	L 7:	29.5., Mon
Engineering	L 8:	1.6., Thu
	-	5.6., Mon
	-	8.6., Thu
	T 3:	12.6., Mon
	-	15.6., Thu
	L 9:	19.6., Mon
	L10:	22.6., Thu
Arch. & Design	L 11:	26.6., Mon
	T 4:	29.6., Thu
	L12:	3.7., Mon
Software	L13:	6.7., Thu
Modelling	L14:	10.7., Mon
	T 5:	13.7., Thu
Patterns	L15:	17.7., Mon
QA (Testing,	L16:	20.7., Thu
Formal Verif.)	L17:	24.7., Mon
Wrap-Up	L18:	27.7., Thu

Expectations Cont'd

- **project management**
  - ✔ minimize risks, estimate project duration,
  - (✘) the financial part: how much money can can you demand for software?
  - (✔) how to estimate cost/time, without resorting to years of experience
  - ✔ different life stages of a software
  - ✔ become acquainted with the most common procedures of software development
  - ✔ selection of right process for a project.
  - (✘) learn how things are done in real companies
- **requirements**
  - ✔ How to communicate between customer and software team effectively
  - ✔ formalise software engineering problems
  - ✔ learn how to specify the requirements
  - (✔) how to write something based on customer's wishes, which is unambiguous (for the programmers), but understandable for the customer, such that the customers can check on their own what is meant.

Introduction	L 1:	24.4., Mon
Scales, Metrics,	L 2:	27.4., Thu
	-	1.5., Mon
	T 1:	4.5., Thu
Costs,	L 3:	8.5., Mon
Development	L 4:	11.5., Thu
Process	L 5:	15.5., Mon
	T 2:	18.5., Thu
	L 6:	22.5., Mon
	-	25.5., Thu
Requirements	L 7:	29.5., Mon
Engineering	L 8:	1.6., Thu
	-	5.6., Mon
	-	8.6., Thu
	T 3:	12.6., Mon
	-	15.6., Thu
	L 9:	19.6., Mon
	L10:	22.6., Thu
Arch. & Design	L 11:	26.6., Mon
	T 4:	29.6., Thu
	L12:	3.7., Mon
Software	L13:	6.7., Thu
Modelling	L14:	10.7., Mon
	T 5:	13.7., Thu
Patterns	L15:	17.7., Mon
QA (Testing,	L16:	20.7., Thu
Formal Verif.)	L17:	24.7., Mon
Wrap-Up	L18:	27.7., Thu

Expectations Cont'd

- **design**
  - ✔ techniques and vocabulary to express design
  - ✔ learn how to use basic and maybe some advanced techniques, models and patterns in software development
  - ✔ the modern techniques: [...] Test Driven Design, Behaviour Driven Design
  - ✔ acquire knowledge in UML
  - ✔ principles of reasonable software architectures
  - (✘) verification of architectures
  - (✔) what distinguished well-designed SW from bad-designed ones
  - ✘ how to quantify and check things like "good usability"
  - ✘ focus on software architecture
- **Implementation**
  - (✘) write reusable and maintainable code
  - (✘) knowing the adequate codes for the certain software
- **Quality Assurance**
  - (✔) Which software qualities are more important for different types of SW?
  - (✘) test code in a reusable efficient way
  - (✔) extend my basic knowledge on verification methods (unit tests etc.)
  - (✘) conduct a review

Introduction	L 1:	24.4., Mon
Scales, Metrics,	L 2:	27.4., Thu
	-	1.5., Mon
	T 1:	4.5., Thu
Costs, Development Process	L 3:	8.5., Mon
	L 4:	11.5., Thu
	L 5:	15.5., Mon
	T 2:	18.5., Thu
	L 6:	22.5., Mon
	-	25.5., Thu
Requirements Engineering	L 7:	29.5., Mon
	L 8:	1.6., Thu
	-	5.6., Mon
	-	8.6., Thu
	T 3:	12.6., Mon
	-	15.6., Thu
	L 9:	19.6., Mon
	L10:	22.6., Thu
Arch. & Design	L 11:	26.6., Mon
	T 4:	29.6., Thu
Software Modelling	L12:	3.7., Mon
	L13:	6.7., Thu
	L14:	10.7., Mon
	T 5:	13.7., Thu
Patterns	L15:	17.7., Mon
QA (Testing, Formal Verif.)	L16:	20.7., Thu
	L 17:	24.7., Mon
Wrap-Up	L18:	27.7., Thu

Content

- **Survey: Expectations on the Course**
- **Software Metrics**
  - Motivation
  - Vocabulary
  - Requirements on Useful Metrics
  - Excursion: Scales
    - Excursion Excursion: Mean, Median, Quartiles
  - Example: LOC
  - Other Properties of Metrics
    - Base Measures vs. Derived Measures
    - Subjective and Pseudo Metrics
  - Discussion

Software Metrics

Engineering vs. Non-Engineering

	workshop (technical product)	studio (artwork)
Mental prerequisite	the existing and available technical know-how	artist's inspiration, among others
Deadlines	can usually be planned with sufficient precision	cannot be planned due to dependency on artist's inspiration
Price	oriented on cost, thus calculable	determined by market value, not by cost
Norms and standards	exist, are known, and are usually respected	are rare and, if known, not respected
Evaluation and comparison	can be conducted using objective, quantified criteria	is only possible subjectively, results are disputed
Author	remains anonymous, often lacks emotional ties to the product	considers the artwork as part of him/herself
Warranty and liability	are clearly regulated, cannot be excluded	are not defined and in practice hardly enforceable

(Ludewig and Lichter, 2013)

Vocabulary

metric – A quantitative measure of the degree to which a system, component, or process posesses a given attribute.
See: quality metric. **IEEE 610.12** (1990)

quality metric –

(1) A quantitative measure of the degree to which an item possesses a given quality attribute.

(2) A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute. **IEEE 610.12** (1990)

Software Metrics: Motivation and Goals

Important **motivations** and **goals** for using software metrics:

- **specify** quality requirements
- **assess** the quality of products and processes
- **quantify** experience, progress, etc.
- **predict** cost/effort, etc.
- support **decisions**

Software metrics can be used:

- **prescriptive**, e.g., "all prodecures must not have more then N parameters", or
- **descriptive**, e.g., "procedure P has N parameters".

  A **descriptive** metric can be

  - **diagnostic**, e.g., "the test effort was N hours", or
  - **prognostic**, e.g., "the expected test effort is N hours".

  Note: **prescriptive** and **prognostic** are different things.

- **Examples**: support decisions by **diagnostic** measurements:

  (i) Measure CPU time spent per procedure, then "optimize" most time consuming procedure.

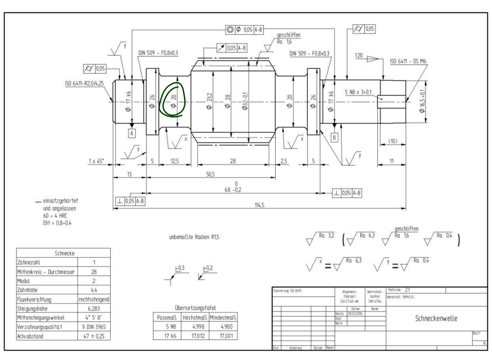
  (ii) Measure attributes which indicate architecture problems, then re-factor accordingly.

Recall: Software Quality *(ISO/IEC 9126-1:2000 (2000))*

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

Useful Metrics

- For **material goods**, useful metrics are often pretty obvious:



- Not so obvious for **immaterial goods**, like **software**.

Content

Requirements on Useful Metrics

Definition. A **software metric** is a function $m : P \to S$ which assigns to each **proband** $p \in P$ a **valuation yield** ("Bewertung") $m(p) \in S$. We call S the **scale** of m.

In order to be useful, a (software) metric should be:

differentiated	worst case: same valuation yield for all probands
comparable	ordinal scale, better: rational (or absolute) scale ($\to$ in a minute)
reproducible	multiple applications of a metric to the same proband should yield the same valuation
available	valuation yields need to be in place when needed
relevant	wrt. overall needs
economical	worst case: doing the project gives a perfect prognosis of project duration – at a high price; **irrelevant** metrics are not economical (if not available for free)
plausible	($\to$ pseudo-metric)
robust	developers cannot arbitrarily manipulate the yield; antonym: **subvertible**

Excursion: Scales

Scales and Types of Scales

Scales S are distinguished by supported **operations**:

	$=, \neq$	$<, >$ (with transitivity)	$\min$, $\max$	percen-tiles, e.g. median	Δ	propor-tion	natural 0 (zero)
nominal scale	✔	✘	✘	✘	✘	✘	✘
ordinal scale	✔	✔	✔	✔	✘	✘	✘
interval scale (with units)	✔	✔	✔	✔	✔	✘	✘
rational scale (with units)	✔	✔	✔	✔	✔	✔	✔
absolute scale	a rational scale where S comprises the key figures itself						

Examples: **Nominal Scale**

- nationality, gender, car manufacturer, geographic direction, train number, ...
- **Software engineering example**: programming language ($S = \{\texttt{Java}, \texttt{C}, \dots\}$)

→ There is no (natural) order between elements of S; the lexicographic order can be imposed ("$\texttt{C} < \texttt{Java}$"), but is not related to the measured information (thus not natural).

Scales S are distinguished by supported **operations**:

	$=, \neq$	$<, >$ (with transitivity)	$\min$, $\max$	percen-tiles, e.g. median	Δ	propor-tion	natural 0 (zero)
nominal scale	✔	✘	✘	✘	✘	✘	✘
ordinal scale	✔	✔	✔	✔	✘	✘	✘
interval scale (with units)	✔	✔	✔	✔	✔	✘	✘
rational scale (with units)	✔	✔	✔	✔	✔	✔	✔
absolute scale	a rational scale where S comprises the key figures itself						

Examples: **Ordinal Scale**

- strongly agree $>$ agree $>$ disagree $>$ strongly disagree; Chancellor $>$ Minister (administrative ranks);
- leaderboard (finishing number tells us that 1st was faster than 2nd, but not how much faster)
- types of scales, …
- **Software engineering example**: CMMI scale (maturity levels 1 to 5) ($\rightarrow$ later)

$\rightarrow$ There is a (natural) **order** between elements of M,
but no (natural) notion of **distance** or **average**.

Scales S are distinguished by supported **operations**:

	$=, \neq$	$<, >$ (with transitivity)	$\min$, $\max$	percen-tiles, e.g. median	Δ	propor-tion	natural 0 (zero)
nominal scale	✔	✘	✘	✘	✘	✘	✘
ordinal scale	✔	✔	✔	✔	✘	✘	✘
interval scale (with units)	✔	✔	✔	✔	✔	✘	✘
rational scale (with units)	✔	✔	✔	✔	✔	✔	✔
absolute scale	a rational scale where S comprises the key figures itself						

Examples: **Interval Scale**

- temperature in Fahrenheit
  - "today it is 10°F warmer than yesterday" ($\Delta(\vartheta_{\text{today}}, \vartheta_{\text{yesterday}}) =$ 10°F)
  - "100°F is twice as warm as 50°F": …? No. Note: the zero is arbitrarily chosen.
- **Software engineering example**: time of check-in in revision control system

$\rightarrow$ There is a (natural) notion of difference $\Delta : S \times S \rightarrow \mathbb{R}$, but no (natural) proportion and 0.

Scales S are distinguished by supported **operations**:

	$=, \neq$	$<, >$ (with transitivity)	$\min, \max$	percen-tiles, e.g. median	Δ	propor-tion	natural 0 (zero)
nominal scale	✔	✘	✘	✘	✘	✘	✘
ordinal scale	✔	✔	✔	✔	✘	✘	✘
interval scale (with units)	✔	✔	✔	✔	✔	✘	✘
rational scale (with units)	✔	✔	✔	✔	✔	✔	✔
absolute scale	a rational scale where S comprises the key figures itself						

Examples: **Rational Scale**

- age ("twice as old"); finishing time; weight; pressure; price; speed; distance from Freiburg…
- **Software engineering example**: runtime of a program for given inputs.

$\rightarrow$ The (natural) zero induces a meaning for proportion m_1/m_2.

Scales S are distinguished by supported **operations**:

	$=, \neq$	$<, >$ (with transitivity)	$\min, \max$	percen-tiles, e.g. median	Δ	propor-tion	natural 0 (zero)
nominal scale	✔	✘	✘	✘	✘	✘	✘
ordinal scale	✔	✔	✔	✔	✘	✘	✘
interval scale (with units)	✔	✔	✔	✔	✔	✘	✘
rational scale (with units)	✔	✔	✔	✔	✔	✔	✔
absolute scale	a rational scale where S comprises the key figures itself						

Examples: **Absolute Scale**

- seats in a bus, number of public holidays, number of inhabitants of a country, …
- "average number of children per family: 1.203" – what is a 0.203-child?
  The absolute scale has been **used as** a rational scale (makes sense for certain purposes if done with care).
- **Software engineering example**: number of known errors.

$\rightarrow$ An absolute scale has a **median**, but in general not an average **in** the scale.

Something for the Mathematicians...

Recall:

Definition. [*Metric Space (math.)*]
Let X be a set. A function $d : X \times X \to \mathbb{R}$ is called metric on X
if and only if, for each $x, y, x \in X$,

(i) $d(x, y) \geq 0$ (non-negative)
(ii) $d(x, y) = 0 \iff x = y$ (identity of indiscernibles)
(iii) $d(x, y) = d(y, x)$ (symmetry)
(iv) $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

(X, d) is called metric space.

→ different from all scales discussed before;
a metric space requires more than a rational scale.

→ definitions of, e.g., IEEE 610.12, may use standard (math.) names for different things

Something for the Computer Scientists...

- A **function** which
  - assigns to each **algorithm** (or problem, or program)
  - a complexity class
    (worst-, average-, best-case; deterministic, non-deterministic; space, time; ...),

  can be seen as a metric (according to our earlier definition):
  - **probands** P: set of algorithms (or problems, or programs)
  - **scale** S: problem classes like $\mathcal{O}(N)$.

Example:

- Problem p: "does element E occur in unsorted, finite list L"?
- Complexity metric (worst-case; deterministic; time):
  - p is in $\mathcal{O}(N)$, $N = |L|$ (length of list).

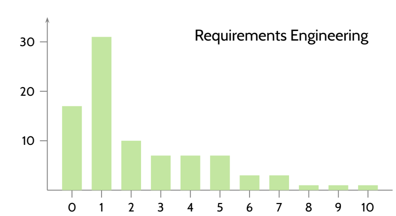
→ the McCabe metric (in a minute) is sometimes called complexity metric
(in the rough sense of "complicatedness").

→ descriptions of software metrics may use standard (comp. sc.) names for different things.

Excursion Excursion: Communicating Figures

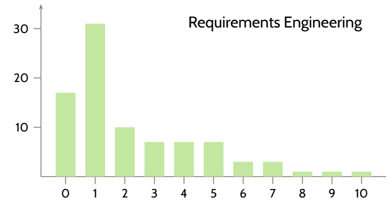
Project Management: Metrics on People

Definition. A **software metric** is a function $m : P \to S$ which assigns to each **proband** $p \in P$ a **valuation yield** ("Bewertung") $m(p) \in S$. We call S the **scale** of m.



- Here: P is the set of participants in the survey of the course "Software Engineering".
- Scale: $S = \{0, \dots, 10\}$ (ordinal scale; has $=$ and $\neq$, $<$ and $>$, $\min$ and $\max$).
- Measurement procedure: self-assessment ($\to$ subjective measure).

Reduce Information Further

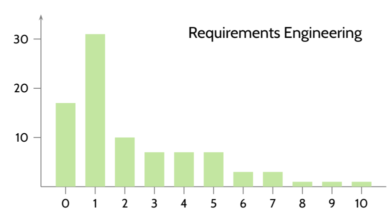


- **Arithmetic mean**: 2.284 (not in the scale!)
- **Minimum** and **maximum**: 0 and 10
- **Median**: 1 (the value such that 50% of the probands have yields below and above)

$\underbrace{3, 7, 1, 1, 2}$

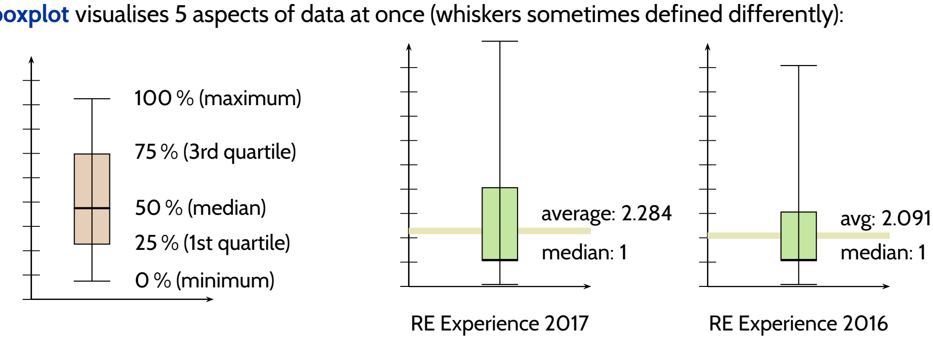
1, 1, 2, 3, 7

Reduce Information Further

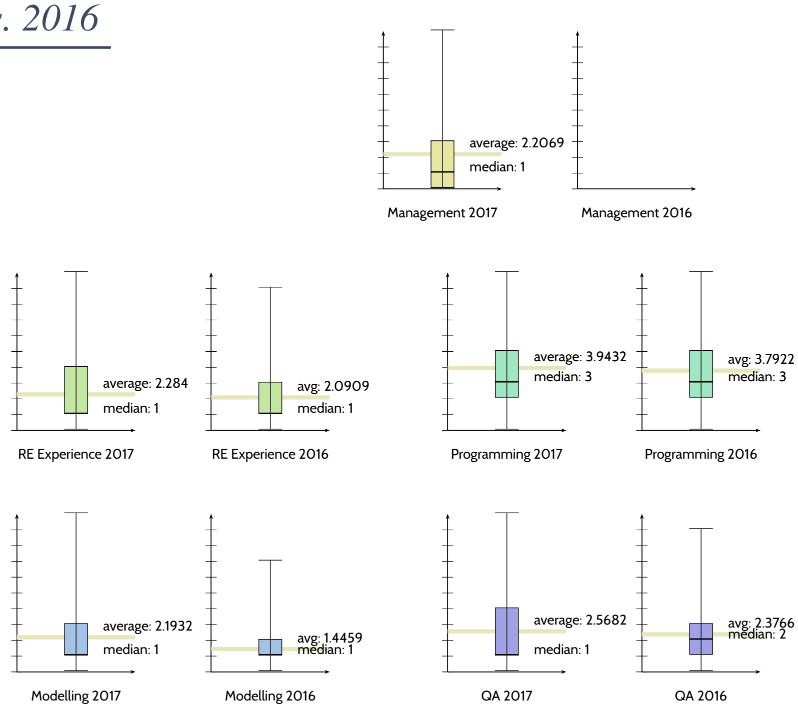


- **Arithmetic mean**: 2.284 (not in the scale!)
- **Minimum** and **maximum**: 0 and 10
- **Median**: 1 (the value such that 50% of the probands have yields below and above)
- **1st** and **3rd Quartile**: 1 and 4 (25%, ~~50~~ 75%)

- a **boxplot** visualises 5 aspects of data at once (whiskers sometimes defined differently):



2017 vs. 2016



Back From Excursion: Scales

Content

Requirements on Useful Metrics

In order to be useful, a (software) metric should be:

differentiated	worst case: same valuation yield for all probands
comparable	ordinal scale, better: rational (or absolute) scale
reproducible	multiple applications of a metric to the same proband should yield the same valuation
available	valuation yields need to be in place when needed
relevant	wrt. overall needs
economical	worst case: doing the project gives a perfect prognosis of project duration – at a high price; **irrelevant** metrics are not economical (if not available for free)
plausible	(⟶ pseudo-metric)
robust	developers cannot arbitrarily manipulate the yield; antonym: **subvertible**

Example: Lines of Code (LOC)

```
/* https://de.wikipedia.org/wiki/
 * Liste_von_Hallo-Welt-Programmen/
 * H%C3%B6here_Programmiersprachen#Java */

class Hallo {
␣
␣public static void
␣main( String[] args ) {
␣   System.out.print(
␣      "Hallo␣Welt!" ); // no newline
␣}
}
```

dimension	unit	measurement procedure
program size	LOC_{tot}	number of lines in total
net program size	LOC_{ne}	number of non-empty lines
code size	LOC_{pars}	number of lines with not only comments and non-printable
delivered program size	DLOC_{tot}, DLOC_{ne}, DLOC_{pars}	like LOC, only code (as source or compiled) given to customer

(Ludewig and Lichter, 2013)

differentiated	✓
comparable	✓
reproducible	✓
available	✓
relevant	?
economical	(✓)
plausible	!
robust	?

More Examples

characteristic ('Merkmal')	positive example	negative example
differentiated	program length in LOC	CMM/CMMI level below 2
comparable	cyclomatic complexity	review (text)
reproducible	memory consumption	grade assigned by inspector
available	number of developers	number of errors in the code (not only known ones)
relevant	expected development cost; number of errors	number of subclasses (NOC)
economical	number of discovered errors in code	highly detailed timekeeping
plausible	cost estimation following COCOMO (to a certain amount)	cyclomatic complexity of a program with pointer operations
robust	grading by experts	almost all pseudo-metrics

(Ludewig and Lichter, 2013)

Other Properties of Metrics

Kinds of Metrics: ISO/IEC 15939:2011

base measure – measure defined in terms of an attribute and the method for quantifying it. **ISO/IEC 15939 (2011)**

Examples:

- lines of code, hours spent on testing, …
-

derived measure – measure that is defined as a function of two or more values of base measures. **ISO/IEC 15939 (2011)**

Examples:

- average/median lines of code, productivity (lines per hour), …
-

Kinds of Metrics: by Measurement Procedure

	objective metric	pseudo metric	subjective metric
Procedure	measurement, counting, possibly standardised	computation (based on measurements or assessment)	review by inspector, verbal or by given scale
Advantages	exact, reproducible, can be obtained automatically	yields relevant, directly usable statement on not directly visible characteristics	not subvertable, plausible results, applicable to complex characteristics
Disadvantages	not always relevant, often subvertable, no interpretation	hard to comprehend, pseudo-objective	assessment costly, quality of results depends on inspector
Example, general	body height, air pressure	body mass index (BMI), weather forecast for the next day	health condition, weather condition ("bad weather")
Example in Software Engineering	size in LOC or NCSI; number of (known) bugs	productivity; cost estimation by COCOMO	usability; severeness of an error
Usually used for	collection of simple base measures	predictions (cost estimation); overall assessments	quality assessment; error weighting

(Ludewig and Lichter, 2013)

Pseudo-Metrics

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:

	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** $\rightarrow$ we really measure average LOC per module.
- measure **maintainability** in average LOC per module: derived, **pseudo**
  $\rightarrow$ we don't really **measure** maintainability; average-LOC is only **interpreted** as maintainability.
  Not robust if easily subvertible (see exercises).

Pseudo-Metrics Example

Example: **productivity** (derived).

- Team T develops software S with LOC $N = 817$ in $t = 310$h.
- Define **productivity** as $p = N/t$, here: ca. 2.64 LOC/h.
- Pseudo-metric: measure **performance**, **efficiency**, **quality**, … of teams by **productivity** (as defined above).
- team may write

```
x
:=
y
+
z;
```

instead of

```
x := y + z;
```

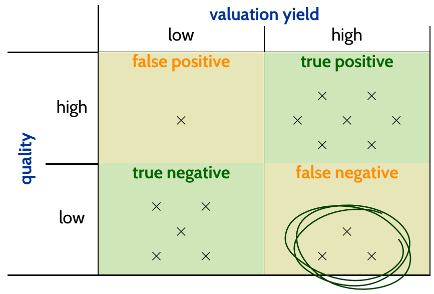
  $\rightarrow$ 5-time productivity increase, but real efficiency actually decreased.

  $\rightarrow$ not (at all) plausible.

  $\rightarrow$ clearly **pseudo**.

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 ($\rightarrow$ tutorials)), then **productivity** could be useful measure for, e.g., team performance.

McCabe Complexity

complexity –

(1) The degree to which a system or component has a design or implementation that is difficult to understand and verify. Contrast with: simplicity.

(2) Pertaining to any of a set of structure-based metrics that measure the attribute in (1).

IEEE 610.12 (1990)

Definition. [*Cyclomatic Number [graph theory]*]
Let $G = (V, E)$ be a graph comprising **vertices** V and **edges** E.
The **cyclomatic number** of G is defined as number of edges

$$v(G) = |E| - |V| + 1.$$

Intuition: minimum number of edges to be removed to make G cycle free.

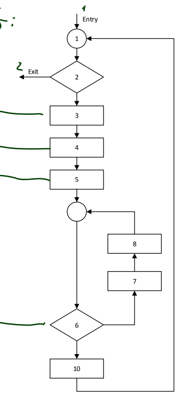
McCabe Complexity Cont'd

Definition. [*Cyclomatic Complexity [McCabe, 1976]*]
Let $G = (V, E)$ be the **Control Flow Graph** of program P.
Then the **cyclomatic complexity** of P is defined as $v(P) = |E| - |V| + p$ where p is the number of entry or exit points.

P:

```
void insertionSort(int[] array) {
  for (int i = 2; i < array.length; i++) {
    tmp = array[i];
    array[0] = tmp;
    int j = i;
    while (j > 0 && tmp < array[j-1]) {
      array[j] = array[j-1];
      j--;
    }
    array[j] = tmp;
  }
}
```

G:



Number of edges: $|E| = 11$
Number of nodes: $|V| = 6 + 2 + 2 = 10$
External connections: $p = 2$

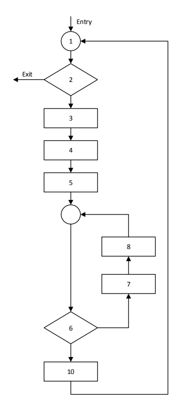
$\rightarrow v(P) = 11 - 10 + 2 = 3$

McCabe Complexity Cont'd

Definition. [*Cyclomatic Complexity [McCabe, 1976]*]
Let $G = (V, E)$ be the **Control Flow Graph** of program P.
Then the **cyclomatic complexity** of P is defined as $v(P) = |E| - |V| + p$ where p is the number of entry or exit points.

- **Intuition**: number of paths, number of decision points.
- **Interval scale** (not absolute, no zero due to $p > 0$); **easy to compute**
- Somewhat **independent** from programming language.
- **Plausibility**:
  - \+ **loops and conditions** are harder to understand than sequencing.
  - – doesn't consider **data**.
- **Prescriptive** use:

  > "For each procedure, either limit cyclomatic complexity to [agreed-upon limit] or provide written explanation of why limit exceeded."



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)	$CBO(C) = \lvert K_o \cup K_i \rvert$, $K_o =$ set of classes used by C, $K_i =$ set of classes using C
response for a class (RFC)	$RFC = \lvert M \cup \bigcup_i R_i \rvert$, M set of methods of C, R_i set of all methods calling method i
lack of cohesion in methods (LCOM)	$\max(\lvert P \rvert - \lvert Q \rvert, 0)$, $P =$ methods using no common attribute, $Q =$ methods using at least one common attribute

- **objective metrics**: DIT, NOC, CBO; **pseudo-metrics**: WMC, RFC, LCOM

> *... there seems to be agreement that it is far more important to focus on empirical validation (or refutation) of the proposed metrics than to propose new ones, ...* (Kan, 2003)

Content

References

References

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

IEEE (1990). *IEEE Standard Glossary of Software Engineering Terminology*. Std 610.12-1990.

ISO/IEC (2011). *Information technology - Software engineering - Software measurement process*. 15939:2011.

ISO/IEC FDIS (2000). *Information technology - Software product quality - Part 1: Quality model*. 9126-1:2000(E).

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

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