Softwaretechnik / Software-Engineering

Lecture 2: Software Metrics

2018-04-19

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

Albert-Ludwigs-Universität Freiburg, Germany

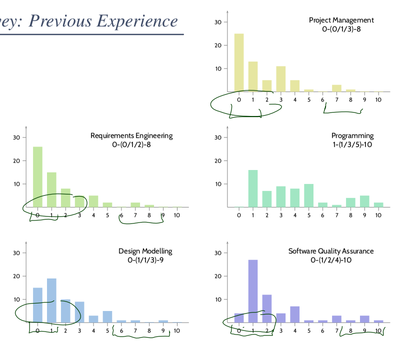
Topic Area Project Management: Content

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

Content

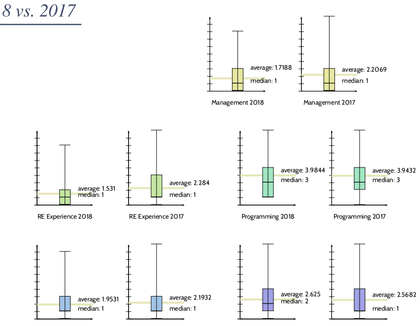
- **Survey: Expectations on the Course**
- **Software Metrics**
  - Motivation
  - Vocabulary
  - Requirements on Useful Metrics
    - Excursion: Scales
    - Some positive/negative examples
    - Example: LOC
  - Other Properties of Metrics
    - Base Measures vs. Derived Measures
    - Subjective and Pseudo Metrics
    - Example: McCabe
  - Discussion

Survey: Previous Experience

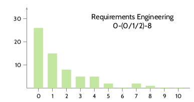


Excursion: Communicating Figures

2018 vs. 2017

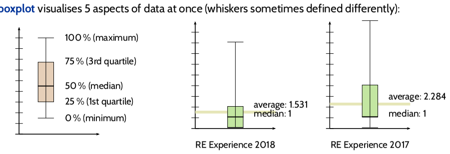


Quartiles



- **Arithmetic mean**: 1.531 (not in the scale! → later)
- **Minimum** and **maximum**: 0 and 8
- **Median**: 1 (the value such that 50% of the probands have yields below and above)
- **1st** and **3rd Quartile**: 0 and 2 (25%, 75%)

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



Back From Excursion: Communicating Figures

Content

- Survey: Expectations on the Course
- Software Metrics
  - Motivation
  - Vocabulary
  - Requirements on Useful Metrics
    - Excursion: Scales
    - Some positive/negative examples
    - Example: LOC
  - Other Properties of Metrics
    - Base Measures vs. Derived Measures
    - Subjective and Pseudo Metrics
    - Example: McCabe
  - Discussion

Expectations

- **general**
  - ✔ learn about tasks related to development of software
  - ✔ learn different aspects of working in a software producing **team** and how to cope with them
  - ✔ gain more insight with [...] challenges faced in **socio-technical systems** and practical ways to overcome those challenges
  - (✔) learn **what techniques are there** for project management, planning a software project, communicate and cooperate with partners
  - (✔) learning a **systematic** approach to developing a larger program
  - (✔) hope that course content can directly be used **in practice**
  - (✔) explicit, precise, **working** methods
  - (✔) would like to see **a lot of concrete examples** of software projects
  - (✘) concretise the role of **computer science and software engineering in business**
- **project management**
  - ✔ **overview** about modelling and organisation of a project, and how to group different problems
  - ✔ be able to plan software development towards **deadline** and **budget**.
  - ✔ plan larger projects, conduct **safely**, **timely**, **reliably**
  - ✔ skills required to take **responsibility** in managing software projects
  - ✔ how to avoid mistakes or **limit inflicted damage**

Introduction	L 1:	16.4., Mon
Scales, Metrics,	L 2:	19.4., Thu
Costs,	L 3:	23.4., Mon
	T 1:	26.4., Thu
Development	L 4:	30.4., Mon
Process	L 5:	3.5., Thu
Requirements	L 6:	7.5., Mon
	-	10.5., Thu
Engineering	L 7:	14.5., Mon
	T 2:	17.5., Thu
	-	21.5., Mon
	-	24.5., Thu
	L 8:	28.5., Mon
	-	31.5., Thu
	L 9:	4.6., Mon
	T 3:	7.6., Thu
	L10:	11.6., Mon
Arch. & Design,	L 11:	14.6., Thu
Software-	L12:	18.6., Mon
	T 4:	21.6., Thu
Modelling,	L13:	25.6., Mon
Patterns	L14:	28.6., Thu
QA	L15:	2.7., Mon
	T 5:	5.7., Thu
(Testing, Formal	L16:	9.7., Mon
Verification)	L17:	12.7., Thu
Wrap-Up	L18:	16.7., Mon
	T 6:	19.7., Thu

Expectations Cont'd

- **project management** cont'd
  - ✔ how to **prove or test the development process**
  - (✔) learn **proper metrics** to measure progress and check quality
  - (✔) create management plan **evaluate results** of previous software projects and use evaluation results
  - (✔) useful **tools** for successful management of software projects
  - (✘) learn how project planning and management are done in **the proper way**
  - (✘) **deep knowledge** about management skills
  - ✔ don't expect being taught how to be a manager, learn conditions for success
- **requirements**
  - ✔ **communicate** expectations, needs, information, and ideas without misunderstandings **to colleagues and clients**
  - ✔ be able to **formalise** requirements ensure that they are formalised
  - ✔ investigate necessary requirements
  - (✔) learn what **well-defined requirements** look like, how they can have impact on final product

Introduction	L 1:	16.4., Mon
Scales, Metrics,	L 2:	19.4., Thu
Costs,	L 3:	23.4., Mon
	T 1:	26.4., Thu
Development	L 4:	30.4., Mon
Process	L 5:	3.5., Thu
Requirements	L 6:	7.5., Mon
	-	10.5., Thu
Engineering	L 7:	14.5., Mon
	T 2:	17.5., Thu
	-	21.5., Mon
	-	24.5., Thu
	L 8:	28.5., Mon
	-	31.5., Thu
	L 9:	4.6., Mon
	T 3:	7.6., Thu
	L10:	11.6., Mon
Arch. & Design,	L 11:	14.6., Thu
Software-	L12:	18.6., Mon
	T 4:	21.6., Thu
Modelling,	L13:	25.6., Mon
Patterns	L14:	28.6., Thu
QA	L15:	2.7., Mon
	T 5:	5.7., Thu
(Testing, Formal	L16:	9.7., Mon
Verification)	L17:	12.7., Thu
Wrap-Up	L18:	16.7., Mon
	T 6:	19.7., Thu

Expectations Cont'd

- **design**
  - ✘ get experience towards **coming up with good designs**
  - (✘) improve knowledge in [...] software design / software architecture
  - (✘) learn **how to design** a software model that fulfills all predefined requirements
  - ✔ **communicate** design specifications
  - ✔ learn **formalisms** to avoid, e.g., going for an inappropriate architecture
  - ✔ concepts/models/ideas to plan software and **verify concept before implementing** it
- **Implementation**
  - (✘) keep software clean, **modular** and expansible
  - ✘ sustainably manage a codebase
- **Quality Assurance**
  - (✘) make sure software is high quality
  - (✔) **organising** quality assurance
  - (✔) how to satisfy wishes of customers and **how to know if satisfied**
  - ✔ how to obtain software as planned and which **does what it should do**

Introduction	L 1:	16.4., Mon
Scales, Metrics,	L 2:	19.4., Thu
Costs,	L 3:	23.4., Mon
	T 1:	26.4., Thu
Development	L 4:	30.4., Mon
Process	L 5:	3.5., Thu
Requirements	L 6:	7.5., Mon
	-	10.5., Thu
Engineering	L 7:	14.5., Mon
	T 2:	17.5., Thu
	-	21.5., Mon
	-	24.5., Thu
	L 8:	28.5., Mon
	-	31.5., Thu
	L 9:	4.6., Mon
	T 3:	7.6., Thu
	L10:	11.6., Mon
Arch. & Design,	L 11:	14.6., Thu
Software-	L12:	18.6., Mon
	T 4:	21.6., Thu
Modelling,	L13:	25.6., Mon
Patterns	L14:	28.6., Thu
QA	L15:	2.7., Mon
	T 5:	5.7., Thu
(Testing, Formal	L16:	9.7., Mon
Verification)	L17:	12.7., Thu
Wrap-Up	L18:	16.7., Mon
	T 6:	19.7., Thu

Content

- **Survey: Expectations on the Course**
- **Software Metrics**
  - Motivation
  - Vocabulary
  - Requirements on Useful Metrics
    - Excursion: Scales
    - Some positive/negative examples
    - Example: LOC
  - Other Properties of Metrics
    - Base Measures vs. Derived Measures
    - Subjective and Pseudo Metrics
    - Example: McCabe
  - Discussion

Software Metrics

Engineering vs. Non-Engineering

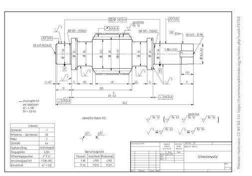
	workshop (technical product)	studio (artwork)
Mental prerequisite	the existing and available technical know-how	artist's inspiration, among others
Deadlines	can usually be planned with sufficient precision	cannot be planned due to dependency on artist's inspiration
Price	oriented on cost, thus calculable	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)

8/45

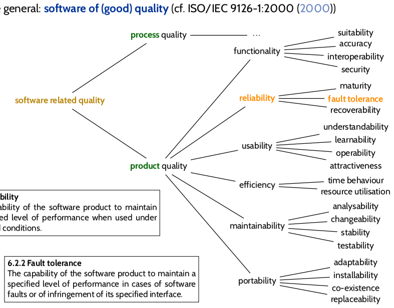
Example: Material Goods

- For **material goods**, it is often pretty obvious what we want to evaluate for:



Recall: "software that is reliable and works efficiently" (Bauer, 1971)

More general: **software of (good) quality** (cf. ISO/IEC 9126-1:2000 (2000))

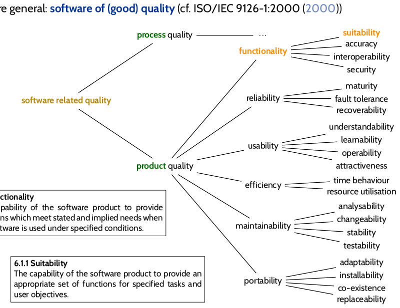


6.2 Reliability
The capability of the software product to maintain a specified level of performance when used under specified conditions.

6.2.2 Fault tolerance
The capability of the software product to maintain a specified level of performance in cases of software faults or of infringement of its specified interface.

Recall: "software that is reliable and works efficiently" (Bauer, 1971)

More general: **software of (good) quality** (cf. ISO/IEC 9126-1:2000 (2000))



6.1 Functionality
The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.

6.1.1 Suitability
The capability of the software product to provide an appropriate set of functions for specified tasks and user objectives.

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:

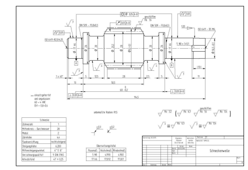
- **prescriptive**, e.g., "all prodecures must not have more then N parameters", or
- **descriptive**, e.g., "procedure P has N parameters".
  A **descriptive** metric can be
  - **diagnostic**, e.g., "the test effort was N hours", or
  - **prognostic**, e.g., "the expected test effort is N hours".

  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.

Example: Material Goods

- For **material goods**, it is often pretty obvious what we want to evaluate for:

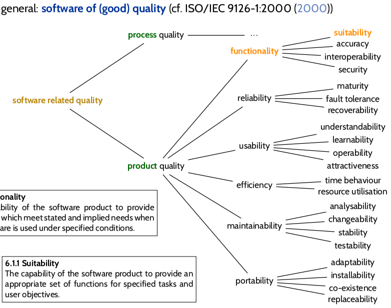


- ...and **how to measure**.
- And **immaterial goods**, like **software**?

Recall: "software that is reliable and works efficiently"

(Bauer, 1971)

More general: **software of (good) quality** (cf. ISO/IEC 9126-1:2000 (2000))



6.1 Functionality
The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.

6.1.1 Suitability
The capability of the software product to provide an appropriate set of functions for specified tasks and user objectives.

Content

Recall: 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)

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.

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 (→ 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	(→ 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)	$\min, \max$	percentiles, e.g. median	Δ	proportion	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 and Types of Scales

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

	$=, \neq$	$<, >$ (with transitivity)	$\min, \max$	percentiles, e.g. median	Δ	proportion	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) (→ later)

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

Scales and Types of Scales

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

	$=, \neq$	$<, >$ (with transitivity)	$\min, \max$	percentiles, e.g. median	Δ	proportion	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" ($\Delta(\vartheta_{\text{today}}, \vartheta_{\text{yesterday}}) = 10°\text{F}$)
  - "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

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

Scales and Types of Scales

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

	$=, \neq$	$<, >$ (with transitivity)	$\min, \max$	percentiles, e.g. median	Δ	proportion	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.

→ The (natural) zero induces a meaning for proportion m_1/m_2.

Scales and Types of Scales

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

	$=, \neq$	$<, >$ (with transitivity)	$\min, \max$	percentiles, e.g. median	Δ	proportion	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						

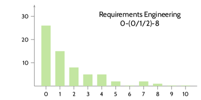
Examples: Absolute Scale

- 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.

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

Another Example: 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".
- Measurement procedure: self-assessment (→ subjective measure).
- Scale: $S = \{0, \dots, 10\}$ (ordinal scale; has $=$ and $\neq$, $<$ and $>$, $\min$ and $\max$).

Back From Excursion: Scales

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**

Content

- **Survey: Expectations on the Course**
- **Software Metrics**
  - Motivation
  - Vocabulary
  - Requirements on Useful Metrics
    - Excursion: Scales
    - Some positive/negative examples
    - Example: LOC
  - Other Properties of Metrics
    - Base Measures vs. Derived Measures
    - Subjective and Pseudo Metrics
    - Example: McCabe
  - Discussion

Some Positive and Negative Examples

characteristic ('Merkmal')	positive example	negative example
differentiated	program length in LOC (→ in a minute)	CMM/CMMI level below 2
comparable	cyclomatic complexity (→ in two minutes)	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)

Example: Lines of Code (LOC)

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)

```
/* 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
  }
}
```

differentiated	✓
comparable	✓
reproducible	✓
available	✓
relevant	?
economical	✓
plausible	?
robust	?

Content

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
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
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

(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** → 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 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

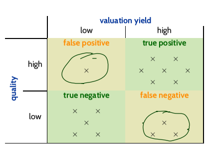
```
x
:=
y
+
z;
```

instead of `x := y + z;`

  → 5-time productivity increase, but real efficiency actually decreased.
  → not (at all) plausible.
  → 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:



- Usefulness may strongly depend on **context information**:
  - If LOC was (or could be made) non-subvertible (→ tutorials), then LOC/day **could be** a useful measure for, e.g., project progress.

Content

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

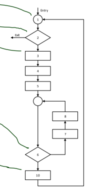
$$v(G) = |E| - |V| + 1.$$

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.

```
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;
  }
}
```



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

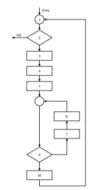
$\rightarrow v(P) = 11 - 10 + 2 = 3$

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."



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

- **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)

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

References

References

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

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.