# Softwaretechnik / Software-Engineering

## Lecture 2: Software Metrics

*2016-04-21*

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

Albert-Ludwigs-Universität Freiburg, Germany



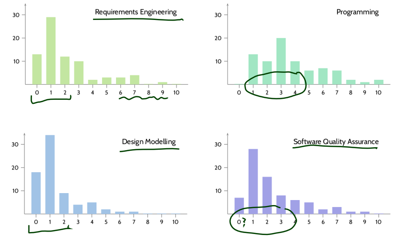
## Is Software Development Always Successful? No.

Ariane 5, V88 · Therac-25 · Toll Collect

- self-driving car, 2016; wrong strategy in traffic situation; crash, no injury
- game distribution platform, 2015; unintentional `rm -rf /`; damage not quantified
- car, 2015; security issue, remote exploit; 1.4 Mio. cars recalled
- car, 2014; unintended acceleration, stack overflows; people injured and killed
- photocopier, 2013; unintentional lossy compression; no damage known
- tiltrotor aircraft, 2000; hydraulic failure not handled; 4 killed
- credit card failures, 2000; incompatibility of new EMV chip; parties ruined
- spacecraft lander, 1998; landing gear operation in flight; 100s Mio. $
- war vessel, 1997; uncontrolled ship by division by 0; no damage
- plane landing, 1993; environment assumptions problem; 2 killed, 54 injured
- ambulance management, 1992; management issues, poor QA; 46 killed
- missile defense, 1991; integer overflow; 28 killed
- telephone infrastructure, 1990; erronously entered mode; 9h no phones, 75 + 100 Mio. $
- defense system, 1979; random bits, false rocket attack announced; no harm
- weather balloons, 1971; poor protocol design; 72 weather-balloons and data lost
- ...



## Survey: Previous Experience






## Expectations

- none, because mandatory course
- **overall**
  - ✔ well-structured lectures
  - (✔) praxis oriented
  - ✘ practical knowledge about planning, designing and testing software
  - ✔ improve skills in scientific work
  - (✔) more about scientific methods
- **other courses**
  - ✘ more on how courses are linked together
  - ✘ skills we need to organise SoPra
  - ✔ maybe transfer knowledge in SoPra
- **"real world"**
  - ✔ vocabulary and methods in professional software development
  - ✔ learn how things work in a company, to easier integrate into teams, e.g., communication
- **kinds of software**
  - ✔ embedded systems and software
  - ✘ how to combine HW and SW parts

| | | |
|---|---|---|
| Introduction | L 1: | 18.4., Mon |
| Scales, Metrics, Costs | L 2: | 21.4., Thu |
| | L 3: | 25.4., Mon |
| | T 1: | 28.4., Thu |
| Development | L 4: | 2.5., Mon |
| | - | 5.5., Thu |
| Process | L 5: | 9.5., Mon |
| | L 6: | 12.5., Thu |
| | - | 16.5., Mon |
| | - | 19.5., Thu |
| | T 2: | 23.5., Mon |
| | - | 26.5., Thu |
| Requirements Engineering | L 7: | 30.5., Mon |
| | L 8: | 2.6., Thu |
| | L 9: | 6.6., Mon |
| | T 3: | 9.6., Thu |
| Architecture & Design | L10: | 13.6., Mon |
| | L 11: | 16.6., Thu |
| | L 12: | 20.6., Mon |
| | T 4: | 23.6., Thu |
| Software Mondelling | L 13: | 27.6., Mon |
| | L 14: | 30.6., Thu |
| | L 15: | 4.7., Mon |
| | T 5: | 7.7., Thu |
| Quality Assurance (Testing, Formal Verification) | L 16: | 11.7., Mon |
| | L 17: | 14.7., Thu |
| | L 18: | 18.7., Mon |
| Wrap-Up | L 19: | 21.7., Thu |



## Expectations Cont'd

- **software development**
  - ✔ understand how software development practically works
  - ✔ developing, maintaining software at bigger scale
  - ✔ aspects of software development
- **software project management**
  - ✔ learn what is important to plan
  - ✔ how to structure the process of a project
  - ✔ how to keep control of project, measure success
  - ✘ which projects need full-time project manager
  - ✘ which kind of documentation is really necessary
  - ✘ want to get better in leading a team; how to lead team of engineers
- **cost estimation**
  - ✔ how to estimate time and effort
  - (✘) formal methods for better planning of projects
  - ✘ tools which help planning
- **quality**
  - ✔ learn ways how to judge quality based on the requirements
  - ✔ avoid mistakes during software development
  - ✔ make better programs, or make programs more efficiently



## Expectations Cont'd

- **requirements**
  - ✔ formal ways to specify requirements
  - ✔ learn techniques to reduce misunderstandings
  - ✔ understand types of requirements
  - (✔) learn how requirements are to be stated
  - (✔) how to create requirements/specification document
- **design**
  - ✔ techniques for design
  - ✔ predict potential risks and crucial design errors
  - (✘) come up with good design, learn how to design
  - (✘) practical knowledge on application of design patterns
  - ✘ how to structure, compose components, how to define interfaces
  - ✘ standards for keeping parts of project compatible
  - ✘ how to guarantee a particular reliability
- **Implementation**
  - (✔) modular programming, better documentation of big projects
  - ✘ more of computers and programming, write faster better programs
  - ✘ strengths and weaknesses of standards, training in their application
  - ✘ improve coding skills
  - ✘ how to increase (software) performance

## Expectations Cont'd

- **code quality assurance**
  - ✔ methods for testing to guarantee high level of quality
  - (✔) how to conduct most exhaustive test as possible in reasonable time
  - ✔ formal methods like program verification
  - ✘ learn about practical implementation of these tools
- **extra information**
  - "will work as teacher"
  - "want to work on medical software"
  - "want to work in automotive industry"
  - "worked as software-engineer"

| | | |
|---|---|---|
| Introduction | L 1: | 18.4., Mon |
| Scales, Metrics, Costs | L 2: | 21.4., Thu |
| | L 3: | 25.4., Mon |
| | T 1: | 28.4., Thu |
| Development | L 4: | 2.5., Mon |
| | - | 5.5., Thu |
| Process | L 5: | 9.5., Mon |
| | L 6: | 12.5., Thu |
| | - | 16.5., Mon |
| | - | 19.5., Thu |
| | T 2: | 23.5., Mon |
| | - | 26.5., Thu |
| Requirements Engineering | L 7: | 30.5., Mon |
| | L 8: | 2.6., Thu |
| | L 9: | 6.6., Mon |
| | T 3: | 9.6., Thu |
| Architecture & Design | L10: | 13.6., Mon |
| | L 11: | 16.6., Thu |
| | L 12: | 20.6., Mon |
| | T 4: | 23.6., Thu |
| Software Mondelling | L 13: | 27.6., Mon |
| | L 14: | 30.6., Thu |
| | L 15: | 4.7., Mon |
| | T 5: | 7.7., Thu |
| Quality Assurance (Testing, Formal Verification) | L16: | 11.7., Mon |
| | L 17: | 14.7., Thu |
| | L 18: | 18.7., Mon |
| Wrap-Up | L19: | 21.7., Thu |




## 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

- **Software Metrics**
  - Motivation
  - Vocabulary
  - Requirements on Useful Metrics
  - Excursion: Scales
  - Example: LOC
  - Other Properties of Metrics
  - Subjective and Pseudo Metrics
  - Discussion
- **Cost Estimation**
  - Deadlines and Costs
  - Expert's Estimation
  - Algorithmic Estimation



## Software Metrics



## Engineering vs. Non-Engineering

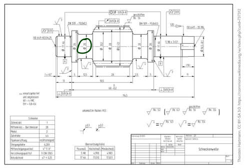
| | workshop (technical product) | studio (artwork) |
|---|---|---|
| Mental prerequisite | the existing and available technical know-how | artist's inspiration, among others |
| Deadlines | can usually be planned with sufficient precision | cannot be planned due to dependency on artist's inspiration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

## Motivation

- Goal: **specify**, and **systematically compare** and **improve** industrial products.
- Approach: **precisely** describe and **assess** the products (and the process of creation).

- This is common practice for **material goods**:

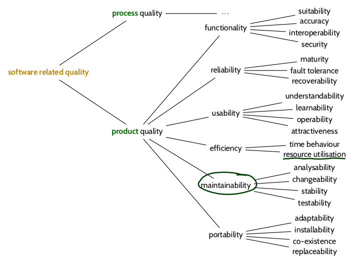


- Not so obvious (and common) for **immaterial goods**, like **software**.
  It should be common: **objective measures** are central to engineering approaches.



## Why "no so obvious" for software?

- Recall, e.g., **quality** (ISO/IEC 9126-1:2000 (2000)):






## 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 time spent per procedure, then "optimize" most time consuming procedure.
  (ii) Measure attributes which indicate architecture problems, then re-factor accordingly.



## 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 $S$ **scale**.

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;<br>**irrelevant** metrics are not economical (if not available for free) |
| ▷ **plausible** | (→ pseudo-metric) |
| **robust** | developers cannot arbitrarily manipulate the yield;<br>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 | $\Delta$ | 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 | | | | | | |



## *Scales and Types of Scales*

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

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

## 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 $x, y, z \in X$,

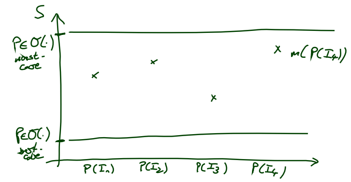
- (i) $d(x, y) \geq 0$ (non-negative)
- (ii) $d(x, y) = 0 \iff x = y$ (identity of indiscernibles)
- (iii) $d(x, y) = d(y, x)$ (symmetry)
- (iv) $d(x, z) \leq d(x, y) + d(y, z)$ (triangle inequality)

$(X, d)$ 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 Comp. Scientist



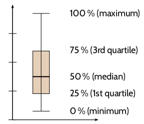


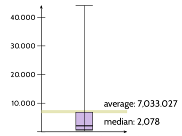

## Median and Box-Plots

| | $M_1$ | $M_2$ | $M_3$ | $M_4$ | $M_5$ |
|---|---|---|---|---|---|
| LOC | 127 | 213 | 152 | 139 | 13297 |

- **arithmetic average**: $2785.6$
- **median**: 127, 139, **152**, 213, 13297

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



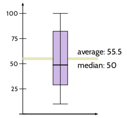

LOC of 2015's lecture's `*.tex` files



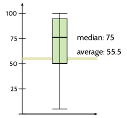
## Example: Project Management

$m$: commits took place at $n$-th day of project.

Team A:
$10, 20, 30, 40, 50, 60, 70, 80, 90, 100$




Team B:
$5, 50, 60, 75, 80, 85, 95, 100$




Team B: "Oh, this SoPra was so stressful… Could we have done something about that?"



## 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** |

## 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
  }
}
```

$LOC_{tot} = 12$
$LOC_{ne} = 11$
$LOC_{pars} = 7$

| | |
|---|---|
| **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, poss. normed | 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 following 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 **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**.



## *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*

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

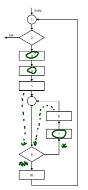
**Intuition**: minimum number of edges to be removed to make $G$ cycle free.



## *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**.

```
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: $p = 2$

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

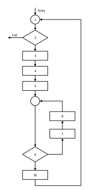


## *McCabe Complexity Cont'd*

**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."






## *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.

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.