INF5180: Software Product- and Process Improvement in Systems Development

Part 06:

Measurement-based Improvement

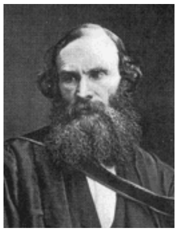
UNIVERSITETET I OSLO

Dr. Dietmar Pfahl

email: dietmarp@ifi.uio.no

Spring 2010

Why Do Measurement?



Lord Kelvin
(1824-1907)

- **"In physical science the first essential step in the direction of learning any subject is to find principles of numerical reckoning and practicable methods for measuring some quality connected with it."** [Popular Lectures and Addresses, vol. 1, "Electrical Units of Measurement", 1883-05-03]
- **"I often say that when you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the state of Science, whatever the matter may be."** [Popular Lectures and Addresses, vol. 1, "Electrical Units of Measurement", 1883-05-03]
- **"If you can not measure it, you can not improve it."**
- **"To measure is to know."**

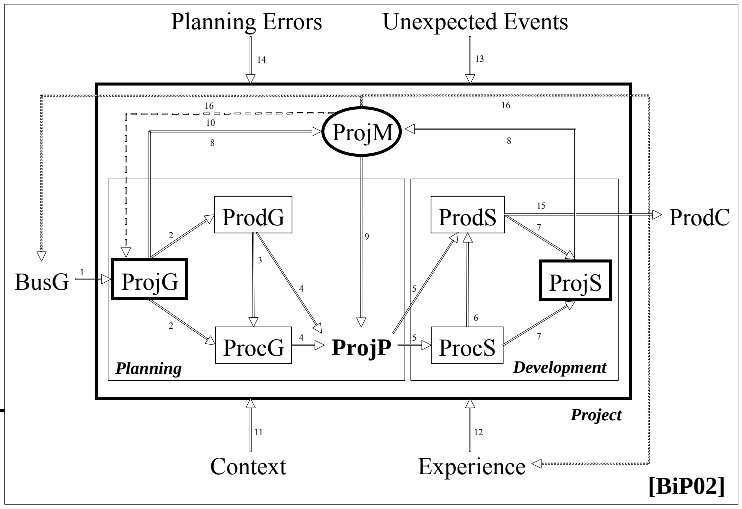
Software Measurement:

Why is it essential for SPI?

Systems Model of Project Management and SPI

- **SPI = Software Process Improvement**

  G = Goal
  P = Plan
  S = State
  C = Customer
  M = Manager
  Bus = Business
  Proj = Project
  Prod = Product
  Proc = Process



[BiP02]

Question 1: Where does the sprint backlog in agile projects fit into this picture?

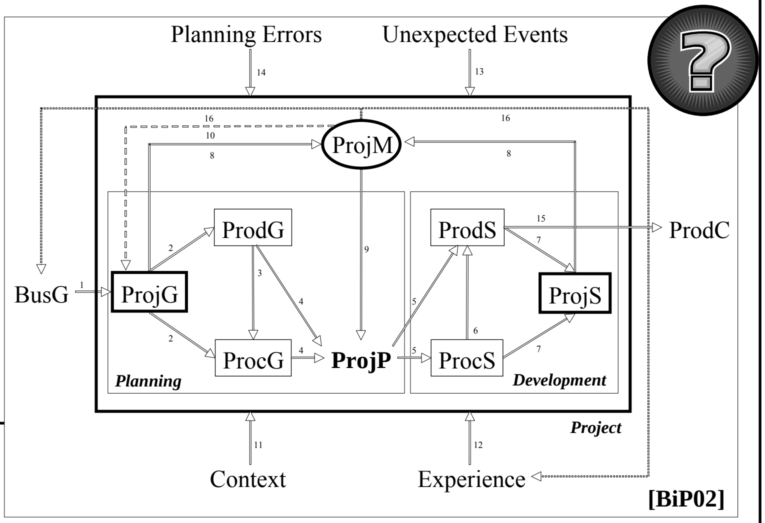
Question 2: In an agile project, what could be interpreted as a process goal?

Question 3: Where does the burndown chart of agile projects fit into this picture (system model)?

Systems Model of Project Management and SPI

- **SPI = Software Process Improvement**

G = Goal
P = Plan
S = State
C = Customer
M = Manager
Bus = Business
Proj = Project
Prod = Product
Proc = Process



[BiP02]

Why Measure in SPI?

- **To generate objective information that results in objective knowledge**
  - **From: “I think that the number of defects in our software has decreased in recent years”**
  - **To: “The number of defects per 1000 lines of code found in acceptance test have been reduced from 3 to 1”**

- **To be able to identify causal relationships and learn from experience**
  - **Experiments can, e.g., show that new practices (e.g., pair programming) have a positive effect on quality and make quality more predictable**

- **To be able to validate that goals have been achieved (targets met)**
  - **Measurability of quality related requirements forces customer to give the requirements as precisely as possible. Requirements that are not “falsifiable” are often ambiguous/unclear.**

Software Measurement: Why is it difficult?

Measurement: Characterization

- Relevant objects (entities) may be described, identified, categorized, ordered, and compared in terms of their key properties (attributes)
- Measurement is a means of assessing these properties:
  - with known reliability
  - with known systematic bias, if any
  - efficiently
  - in a manner that is useful for decision making

Software Measurement Challenges

- **Measuring physical properties:**

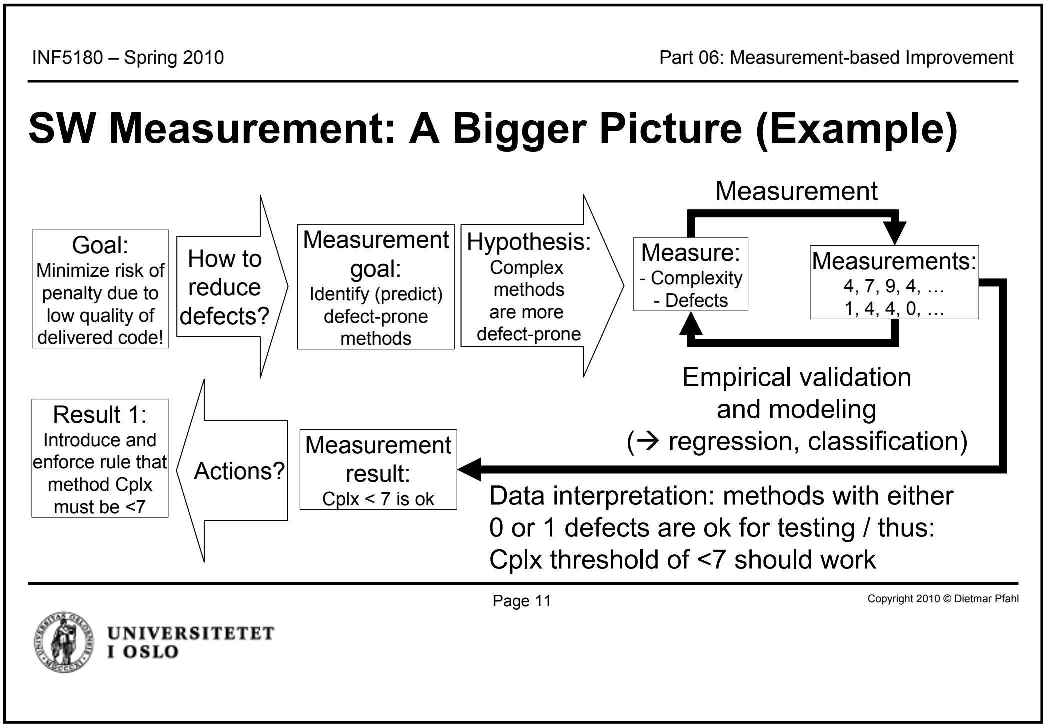
entity	attribute	unit	scale	value
Human	Height	cm	ratio	178

- **Measuring non-physical properties:**

entity	attribute	unit	scale	value
Human	Intelligence/IQ	index	ordinal	135
Program	Modifiability	?	?	?

- **Software properties are non-physical**
  - **size, complexity, functionality, reliability, maturity, portability, flexibility, maintainability, correctness, testability, coupling, coherence, interoperability, …**

Software Measurement: How do it?



SW Measurement: A Bigger Picture (Example)

Goal: Minimize risk of penalty due to low quality of delivered code!

How to reduce defects?

Measurement goal: Identify (predict) defect-prone methods

Hypothesis: Complex methods are more defect-prone

Measurement

Measure:
- Complexity
- Defects

Measurements:
4, 7, 9, 4, …
1, 4, 4, 0, …

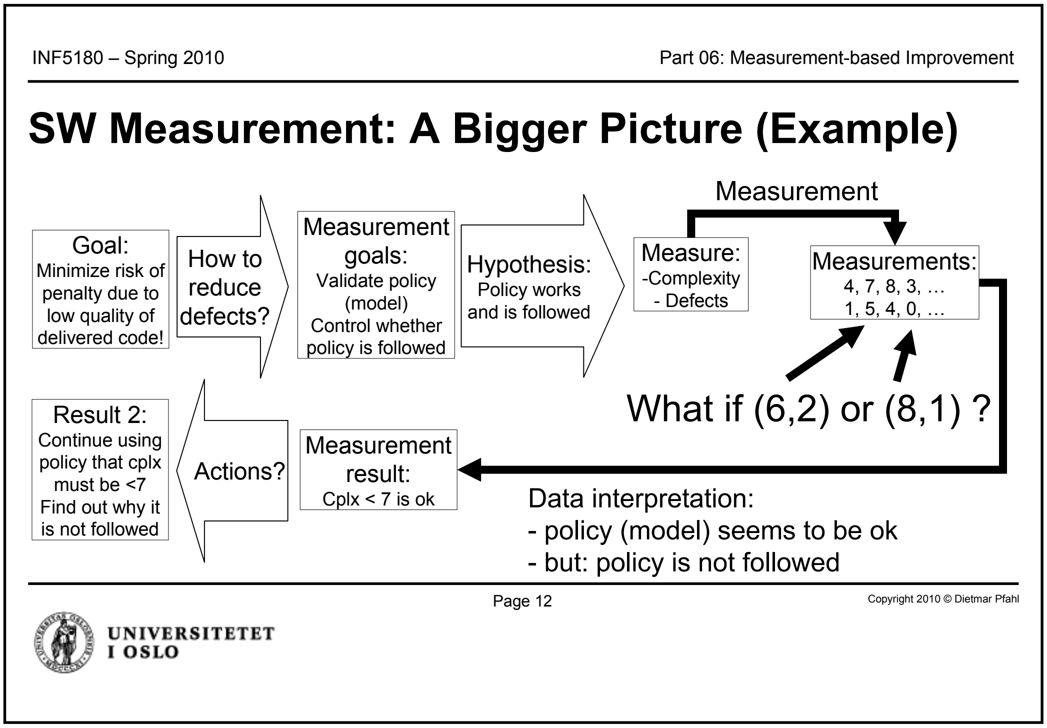
Empirical validation and modeling (→ regression, classification)

Data interpretation: methods with either 0 or 1 defects are ok for testing / thus: Cplx threshold of <7 should work

Measurement result: Cplx < 7 is ok

Actions?

Result 1: Introduce and enforce rule that method Cplx must be <7



SW Measurement: A Bigger Picture (Example)

Goal: Minimize risk of penalty due to low quality of delivered code!

How to reduce defects?

Measurement goals: Validate policy (model) Control whether policy is followed

Hypothesis: Policy works and is followed

Measurement

Measure:
-Complexity
- Defects

Measurements:
4, 7, 8, 3, …
1, 5, 4, 0, …

What if (6,2) or (8,1) ?

Data interpretation:
- policy (model) seems to be ok
- but: policy is not followed

Measurement result: Cplx < 7 is ok

Actions?

Result 2: Continue using policy that cplx must be <7 Find out why it is not followed

SW Measurement: How to plan and run it?

- **These steps are required to implement a measurement program:**
  - Identify the business goals
  - Derive the measurement goals
  - Document the software development process(es)
  - Define measures (metrics) required to reach goals
  - Define data collection procedures
  - Assemble a measurement tool(set)
  - Create a measurement database
  - Collect data
  - Define feedback mechanism
  - Package measurement results
  - Continuously control/improve the measurement program

Software Measurement: Who benefits?

SW Measurement: Who benefits?

- **Managers**
  - What does each process cost?
  - How productive is development?
  - How good is the product (code, design)?
  - Will the user be satisfied with the product?
  - How can we improve?
- **Engineers**
  - Are the requirements testable?
  - Have we found all (severe) defects?
  - Have we met our product or process goals?
  - What can we predict about our software product in the future?

SW Measurement: What does it (not)?

- **SW Measurement is supposed to help us understand the technical process that is used to develop software**
  - The process is measured to control/improve its capability/performance
  - The product is measured to control/improve its quality

But …

- **SW Measurement does not (yet?) provide a commonly agreed set of appropriate metrics for all kinds of software projects/products/processes**
- **SW Measurement should be used very carefully when it comes to evaluate/compare people!**

Measurement and Measure

Measurement:

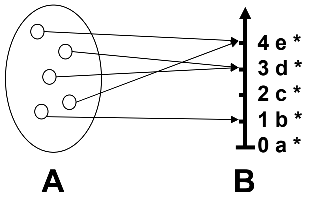
- Measurement is the process through which values are assigned to attributes of entities of the real world.

Measure:

- A measure is the result of the measurement process, so it is the assignment of a value to an entity with the goal of characterizing a specified attribute.

Source: Sandro Morasca, “Software Measurement”, in “Handbook of Software Engineering and Knowledge Engineering - Volume 1: Fundamentals” (refereed book), pp. 239 - 276, Knowledge Systems Institute, Skokie, IL, USA, 2001, ISBN: 981-02- 4973-X.

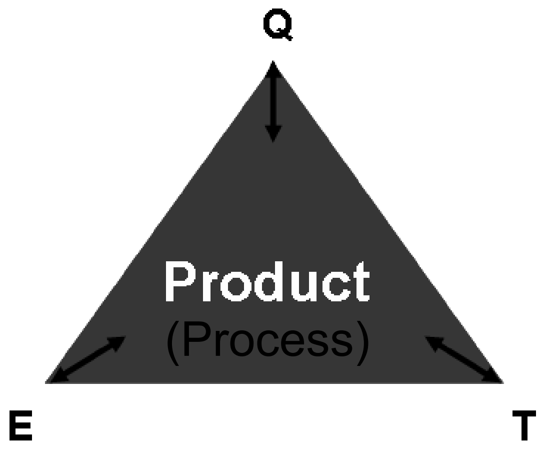
Measure ~~(Metric)~~



- **Measure:**
  - **Let A be a set of empirical (physical) objects**
  - **Let B be a set of formal objects, such as numbers (or symbols)**
  - **A *measure m* is defined to be a mapping from A to B, i.e., m: A → B**

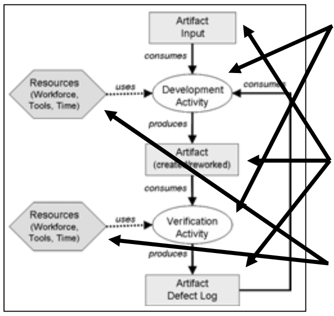
Note: this is neither (exactly) the definition of the mathematical measure (μ: σ(A) → [0, ∞), with σ(A) is the σ-algebra of A) nor of the mathematical metric (d: X × X → R with d(x, y) ≥ 0, d(x, y) = 0 if and only if x = y, d(x, y) = d(y, x), and d(x, z) ≤ d(x, y) + d(y, z)).

What to Measure?



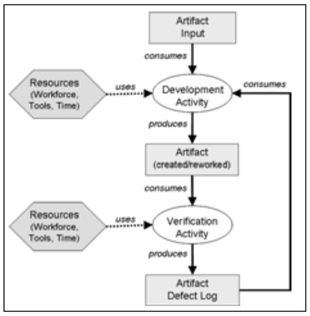
Entity

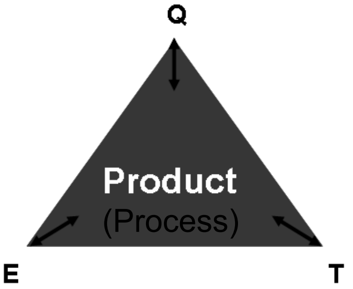
- An entity in software measurement can represent any of the following:
  - Processes/Activities: any activity related to software development and/or maintenance (e.g., requirements analysis, design, testing) – these can be at different levels of granularity
  - Products: any artifact produced or changed during software development and/or maintenance (e.g., source code, software design documents)
  - Resources: people, hardware or software needed to perform the processes



Attribute

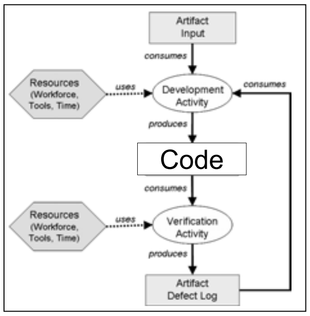
- **An attribute in software measurement could be …**





Attribute (cont'd)

- **An attribute is a feature or property of an entity**
  - e.g., blood pressure of a person, cost of a journey, duration of the software specification process



There are two general types of attributes:

- **Internal attributes can be measured based on the entity itself (→ static)**
  - e.g., entity: code, internal attribute: size, modularity, coupling
- **External attributes can be measured only with respect to how the entity relates to its environment (behavior, usage → dynamic)**
  - e.g., entity: code, external attribute: reliability, maintainability

Example Software Process Attributes

- Process Efficiency:
  - How fast, how much effort, how much quantity/quality per time or effort unit?
- Process Effectiveness:
  - Do we get the quantity/quality we want?
- Process Maturity:
  - CMMI level (cf. Part 09)
- People/Organisation-related:
  - Skills, knowledge, learning, motivation
- Method/Technique/Tool-related:
  - Effectiveness, Efficiency, Learnability, Cost

Cost (Effort) Measurement

- Effort consumption in the project
  - Includes overtime, excludes line activities like department meetings etc
  - How to distinguish productive time from unproductive time?
  - How to distinguish defect correction, change management and “pure development”?
  - Allocation of effort over phases / increments?
- Necessary training costs
  - Close competence gap to be able to do the project
- Tool costs
  - Pure purchase and possible license costs
  - (Tool) Training costs
  - Learning curve costs?
- NB: To be able to investigate cost improvement, cost/effort data must be related to amount of produced output/value (→ **productivity**)

Time Measurement

- Time-to-market is often considered as very important
  - How do you define "time-to-market"?
  - How do you monitor this parameter?
- Time must be precisely defined!
  - Number of work hours or days, number of calendar days, weeks, months … ???
  - Requires that the projects/increments have clearly defined start and end times

Example Software Product Attributes

- Size
  - Length, Complexity, Functionality
- Modularity
- Cohesion
- Coupling
- Quality
- Cost

- Quality (→ ISO 9126)
  - Functionality
  - Reliability
  - Usability
  - Efficiency
  - Maintainability
  - Portability

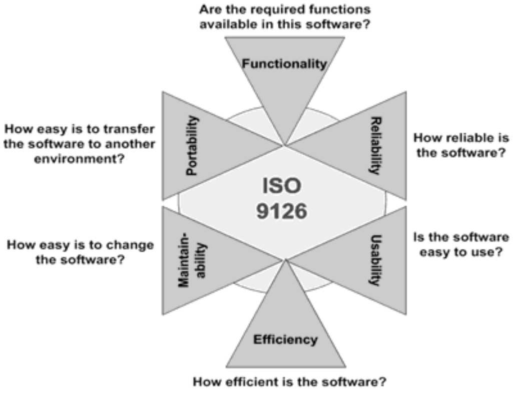
Definition: Software Quality Characteristic

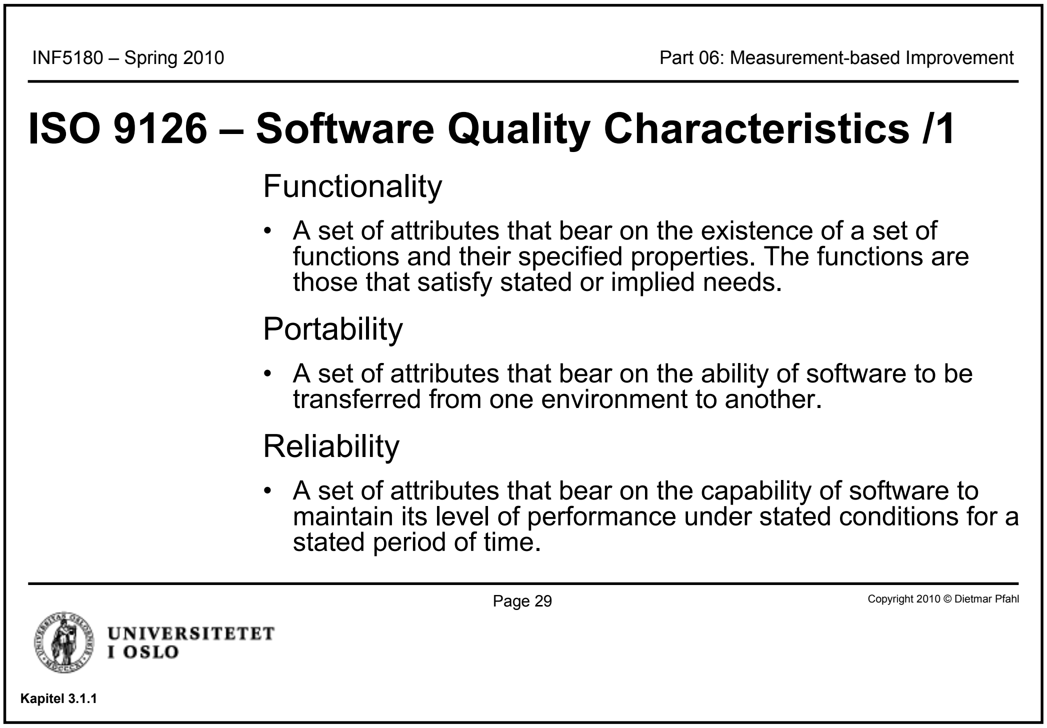
ISO 9126:

“A set of attributes of a software product by which its quality is described and evaluated. A software quality characteristic may be refined into multiple levels of sub-characteristics.”

ISO 9126 – Quality Model (Parts 1-3)

- Software Quality can be measured by evaluating the following characteristics:
  - Functionality
  - Reliability
  - Usability
  - Efficiency
  - Maintainability
  - Portability


Are the required functions available in this software?
Functionality
How easy is to transfer the software to another environment?
Portability
Reliability
How reliable is the software?
ISO 9126
How easy is to change the software?
Maintain-ability
Usability
Is the software easy to use?
Efficiency
How efficient is the software?



ISO 9126 – Software Quality Characteristics /1

Functionality

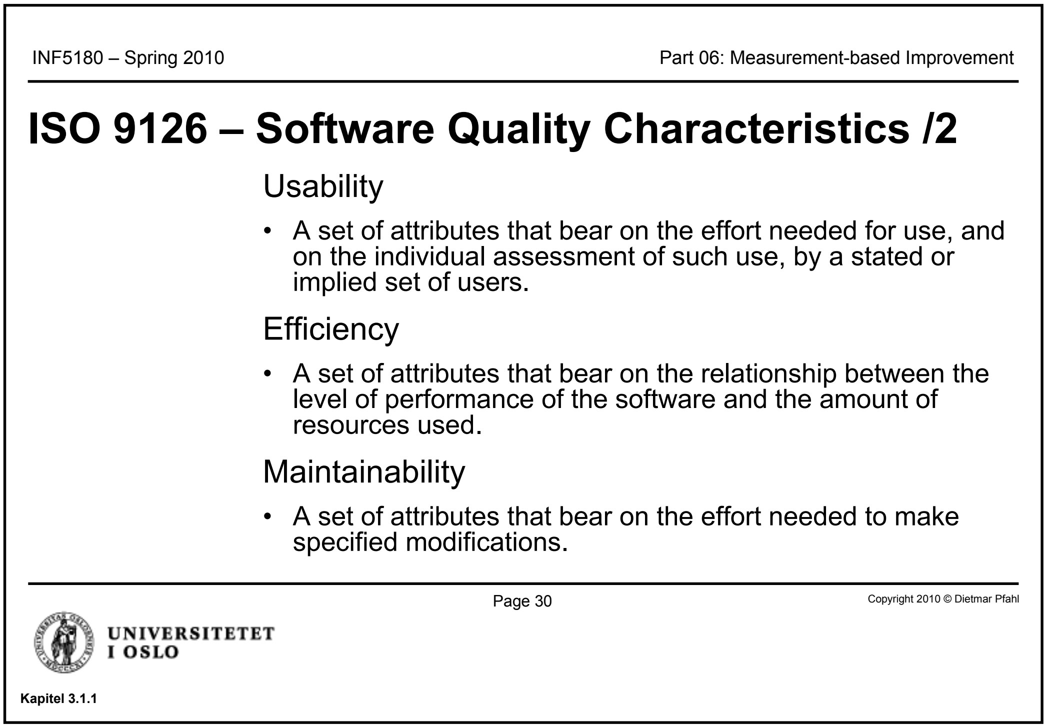
- A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.

Portability

- A set of attributes that bear on the ability of software to be transferred from one environment to another.

Reliability

- A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.



ISO 9126 – Software Quality Characteristics /2

Usability

- A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.

Efficiency

- A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used.

Maintainability

- A set of attributes that bear on the effort needed to make specified modifications.

Quality Model: ISO 9126

1 : n relation between Characteristics and Attributes (Sub-Characteristics)

Characteristics	Attributes		
Functionality	Suitability	Interoperability	Accuracy
	Security	Compliance	
Reliability	Maturity	Recoverability	Fault Tolerance
	Compliance		
Usability	Understandability	Learnability	Operability
	Attractiveness	Compliance	
Efficiency	Time Behaviour	Resource Behaviour	Compliance
Maintainability	Analyzability	Stability	Changeability
	Testability	Compliance	
Portability	Adaptability	Installability	Co-existence
	Replaceability	Compliance	

ISO 9126 – Future Developments

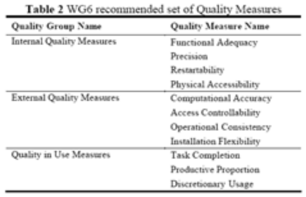
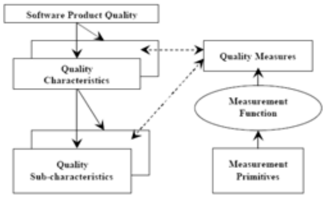


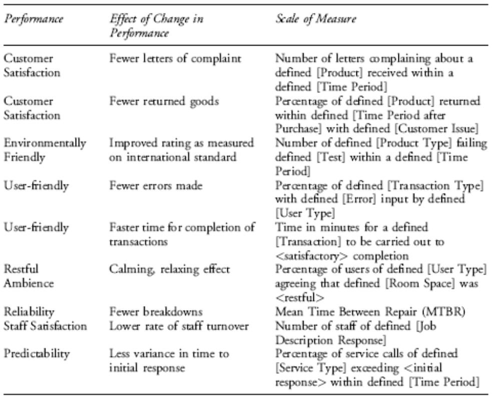
Table 2 WG6 recommended set of Quality Measures

Quality Group Name	Quality Measure Name
Internal Quality Measures	Functional Adequacy Precision Restartability Physical Accessibility
External Quality Measures	Computational Accuracy Access Controllability Operational Consistency Installation Flexibility
Quality in Use Measures	Task Completion Productive Proportion Discretionary Usage



- A new series of standards is currently under development.
- Name: Software Product Quality Requirements and Evaluation (SQuaRE - ISO 25000).
- This series of standards will replace the current ISO 9126 (and ISO 14598) series of standards.
  - Note: the new standard will replace the word ”metric” by “measure”

Alternative Quality Model: Performance Measures by Tom Gilb*



Performance	*Effect of Change in Performance*	*Scale of Measure*
Customer Satisfaction	Fewer letters of complaint	Number of letters complaining about a defined [Product] received within a defined [Time Period]
Customer Satisfaction	Fewer returned goods	Percentage of defined [Product] returned within defined [Time Period after Purchase] with defined [Customer Issue]
Environmentally Friendly	Improved rating as measured on international standard	Number of defined [Product Type] failing defined [Test] within a defined [Time Period]
User-friendly	Fewer errors made	Percentage of defined [Transaction Type] with defined [Error] input by defined [User Type]
User-friendly	Faster time for completion of transactions	Time in minutes for a defined [Transaction] to be carried out to <satisfactory> completion
Restful Ambience	Calming, relaxing effect	Percentage of users of defined [User Type] agreeing that defined [Room Space] was <restful>
Reliability	Fewer breakdowns	Mean Time Between Repair (MTBR)
Staff Satisfaction	Lower rate of staff turnover	Number of staff of defined [Job Description Response]
Predictability	Less variance in time to initial response	Percentage of service calls of defined [Service Type] exceeding <initial response> within defined [Time Period]

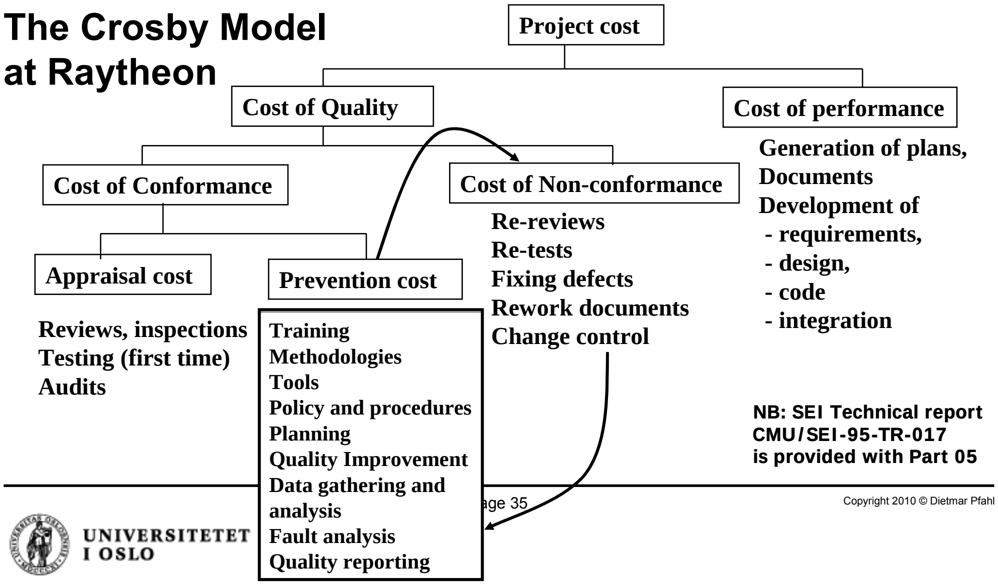
*see www.gilb.com
Taken from "A Handbook for Systems Engineering, Requirements Engineering and Software Engineering Using Planguage"

Crosby's Cost of Quality

- Crosby defines quality as "conformance to requirements"
- Quality costs have 3 components:
  - (Internal & External) Failure cost: what it costs to find and correct a failure plus what it costs to be operational again.
  - Appraisal (or Inspection) cost: what it costs to evaluate the product in order to determine its quality.
  - Prevention cost: what it costs to identify the causes of failure (e.g., through root-cause analysis) and to prevent similar failure to happen in the future.

[Crosby] Philip B. Crosby, *Quality is Free, The Art of Making quality Certain.* New York: Mentor, New American Library, 1979.

The Crosby Model at Raytheon



Project cost

Cost of Quality

Cost of performance

Generation of plans,
Documents
Development of
- requirements,
- design,
- code
- integration

Cost of Conformance

Cost of Non-conformance

Re-reviews
Re-tests
Fixing defects
Rework documents
Change control

Appraisal cost

Reviews, inspections
Testing (first time)
Audits

Prevention cost

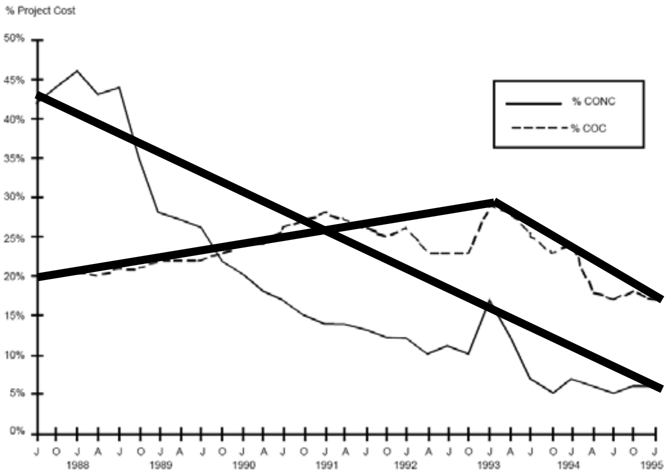
Training
Methodologies
Tools
Policy and procedures
Planning
Quality Improvement
Data gathering and analysis
Fault analysis
Quality reporting

NB: SEI Technical report CMU/SEI-95-TR-017 is provided with Part 05

"Conformance"-Evolution over 6 Years

CONC = Cost of Non-Conformance

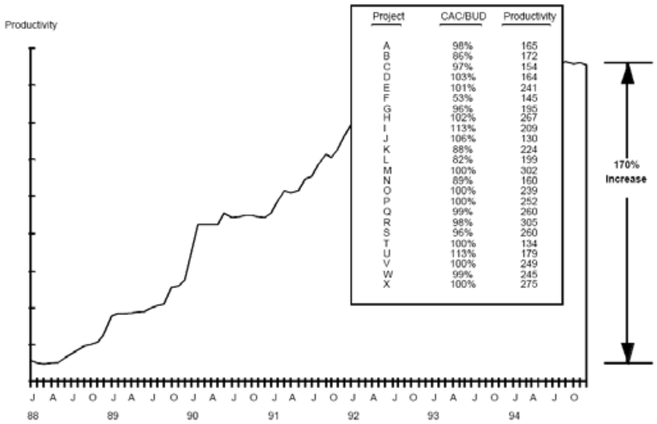
COC = Cost of Conformance



Increase in Productivity over 6 Years

Productivity index = 100 x (productivity – base_productivity) / base_productivity

NB: productivity of each point is the weigthed average of all staff members per project



Project	CAC/BUD	Productivity
A	98%	165
B	86%	172
C	97%	154
D	103%	164
E	101%	241
F	53%	145
G	96%	195
H	102%	267
I	113%	209
J	106%	130
K	88%	224
L	82%	199
M	100%	302
N	89%	160
O	100%	239
P	100%	252
Q	99%	260
R	98%	305
S	96%	260
T	100%	134
U	113%	179
V	100%	249
W	99%	245
X	100%	275

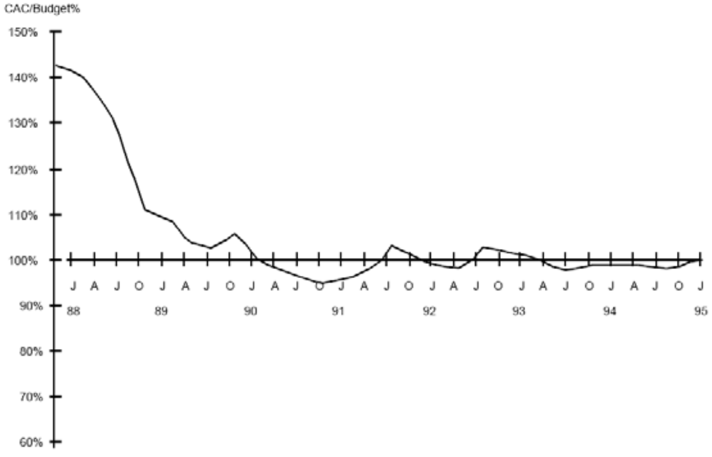
UNIVERSITETET I OSLO

Productivity = equivalent delivered source instructions (EDSI) / person-month of development effort

Prediction Accuracy in Projects (7 Years)

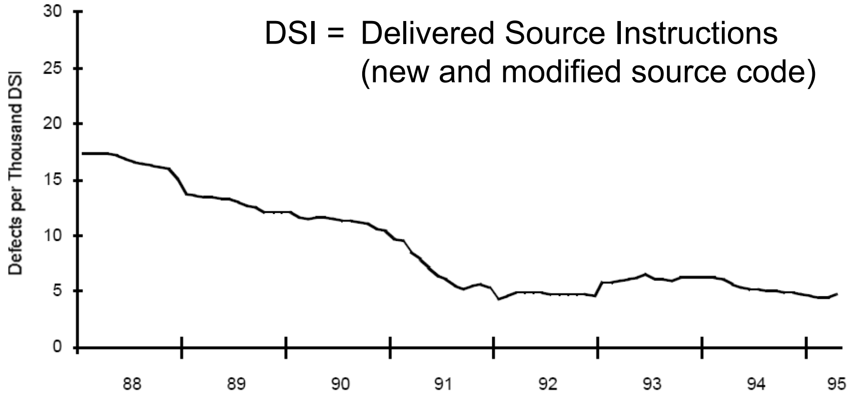
CAC = (actual) cost at completion

BUD = budgeted cost (planned, predicted)



UNIVERSITETET I OSLO

Defect Density (over 7 Years)



DSI = Delivered Source Instructions
(new and modified source code)

Exercise

Situation/Problem:

- The system development organization ”Your IT-partner Inc.” has until now described all system development processes in a paper-based handbook.
- Recently, the handbook has been transformed into a web-based version providing “links” between related documents. In other words, while the paper-handbook was sequential the web-version has a network structure .
- The IT-manager was very satisfied with the paper-based handbooks and requests that an empirical comparison be done before they are actually replaced by the web-based version.

Task:

Sketch a plan for a measurement program in the organization.

The measurement program will have as objective to decide which of the two versions is most effective for the organization.

Software Measurement Details

<cf. papers by Sandro Morasca and Lionel Briand in the reading materials>

Measure m, Scale: Definition

- A **measure m** is a mapping m: σ(A) → B which yields for every empirical object a ∈A a formal object (measurement value) m(a) ∈ B. This mapping must not be arbitrary, hence leading to the following definition of a scale.
- Let ***A*** = (A,R$_1$, …, R$_n$, o$_1$,…, o$_m$) be an **empirical relational system** and ***B*** = (B, S$_1$,…, S$_n$, •$_1$,…, •$_m$) a **formal relational system** and **m** a measure.

  The Triple **(*A*, *B*, m)** is a **scale** if and only if for all i, j and for all a, b, a$_1$, …, a$_k$ ∈ A the following holds:

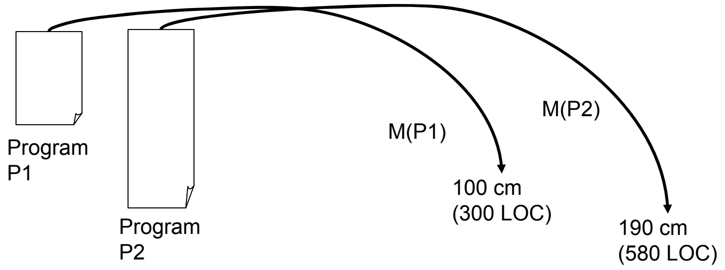
  $R_i (a_1, \dots, a_k) \Leftrightarrow S_i (m(a_1), \dots, m(a_k))$

  and $m(a\ o_j\ b) = m(a) \bullet_j m(b)$

- Example: If B is the set of real numbers, the triple **(*A*, *B*, m)** is a ratio scale.

Representational Measurement Theory: Idea

- Empirical relation preserved under measurement M as numerical relation



P1 shorter than P2 M(P1) < M(P2)

Empirical vs. Formal Relational System

- Definition ERS:

  A = (A, R_1, …, R_n, o_1, …, o_m)

  A is a non-empty set of empirical objects that are to be measured

  - Example entity: program → attribute to be measured: length

  R_i are k_i-ary empirical relations on A with i = 1, …, n.

  - Example: empirical relations “equally long“, “longer”, “shorter”, etc.

  o_j are binary operations on the empirical objects in A with j=1,…,m.

  - Example: concatenation of programs

- Definition FRS:

  B = (B, S_1, …, S_n, $*_1$, …, $*_m$)

  B is a non-empty set of formal objects

  - Examples: symbols, numbers or vectors

  S_i are k_i-ary relations on B with i = 1, …, n

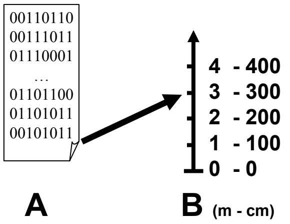
  - Examples: the relations “greater than” or "equal to or greater than"

  $*_j$ are binary operations on the formal objects in B with j=1,…,m

  - Examples: addition or multiplication

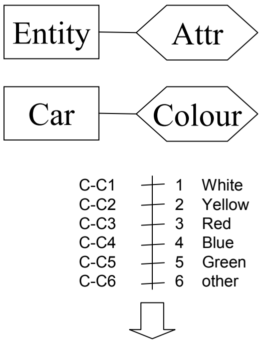
Measurement Unit

Entity: Program
Attribute: Length



- **A Unit of Measurement is a standardised quantity of a physical (or non-physical) property**
- **Questions:**
  - **What other units of program length can you think of?**
  - **What is the unit of temperature (or a project milestone)?**
  - **What is the unit of problem (or program) complexity, or of experience, intelligence?**
  - **What is the unit of color (or defect type)?**
  - **What is the unit of a count?**

Scale Types: Nominal Scale

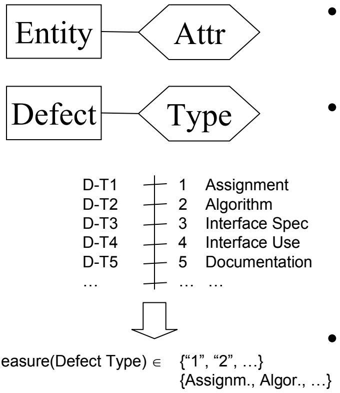


Measure (Car Colour) ∈ {"1", "2", "3", "4", "5", "6"}
{White, …, other}

Nominal Scales:

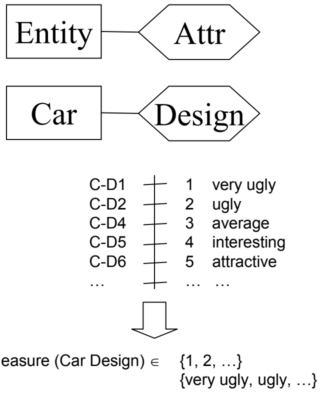
- **Define classes or categories, and then place each entity in a particular class or category, based on the value of the attribute.**
- **Properties:**
  - **The system of empirical relations consists only of different classes**
  - **There is no notion of ordering among the classes.**
  - **Any distinct numbering or symbolic representation of the classes is an acceptable measure, but there is no notion of magnitude associated with the numbers or symbols.**
- **NB: Nominal-scale measurement places elements in a classification scheme. The classes are not ordered; even if the classes are numbered from 1 to n for identification, there is no implied ordering of the classes.**

Example: Nominal Scale



- **Classification of objects based on their colour, id, type, …**
- **Classification of defects in a software:**
  - **Wrong/Missing Value Assignment**
  - **Wrong/Missing Algorithm**
  - **Wrong/Missing Interface Spec**
  - **Wrong/Missing Interface Use**
  - **Wrong/Missing Documentation, …**
- **One-to-one mapping between M and M'**

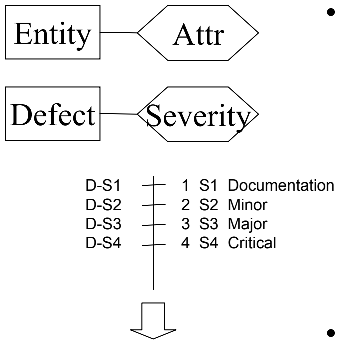
Scale Types: Ordinal Scale



Ordinal Scales

- **The ordinal scale augments the nominal scale by ordering the classes or categories.**
- **Properties:**
  - **The system of empirical relations consists of classes that are ordered with respect to the attribute.**
  - **Any mapping that preserves the ordering (that is, any monotonic function) is acceptable.**
  - **The numbers represent ranking only, so addition, subtraction, and other arithmetic operations have no meaning.**

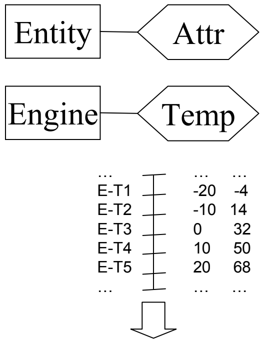
Example: Ordinal Scale



Measure (Defect Severity) ∈ {S1, …, S4}
{1, …, 4}

- **Classification of defects according to severity (→ effects / correction effort):**
  - **Wrong/Missing documentation**
  - **Minor (incorrect program behaviour; one module affected; easy to correct)**
  - **Major (incorrect program behaviour; several modules affected)**
  - **Critical (uncontrolled program behaviour; program execution interrupted)**
- **If M(x) > M(y) then M'(x) > M'(y)**

Scale Types: Interval Scale

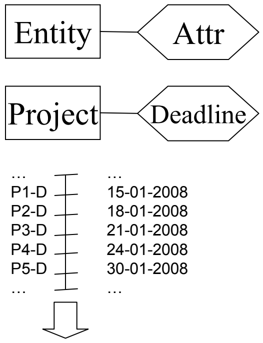


Measure (Engine Temperature) ∈ [min, max]

Interval Scales

- **Interval scale carries more information than ordinal and nominal scale. It captures information about the size of the intervals that separate the classes, so that we can in some sense understand the magnitude of the distance from one class to another.**
- **Properties:**
  - **An interval scale preserves order, as with an ordinal scale.**
  - **An interval scale preserves differences but not ratios.**
    - **That is, we know the difference between any two of the ordered classes in the range of the mapping, but computing the ratio of two classes in the range does not make sense.**
  - **Addition and subtraction are acceptable on the interval scale, but not multiplication and division.**

Example: Interval Scale



…	…
P1-D	15-01-2008
P2-D	18-01-2008
P3-D	21-01-2008
P4-D	24-01-2008
P5-D	30-01-2008
…	…

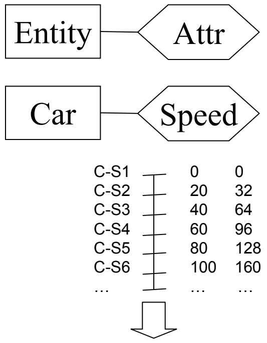
Measure (Project Deadline) ∈ "Calendar"

- **Temperature in Celsius and Fahrenheit**
- **Project deadlines**
  - **Project 1: Jan 15, 2008**
  - **Project 2: Jan 18, 2008**
  - **Project 3: Jan 21, 2008**
  - **Project 4: Jan 24, 2008**
  - **Project 5: Jan 30, 2008**

  Which project finished last?

  Which project took the longest (time)?
- **M' = aM + b, a > 0 (e.g., M' = 9/5M + 32)**

Scale Types: Ratio Scale



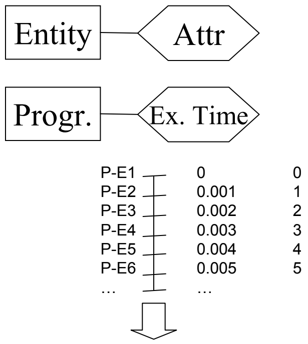
C-S1	0	0
C-S2	20	32
C-S3	40	64
C-S4	60	96
C-S5	80	128
C-S6	100	160
…	…	…

Measure (Car Speed) ∈ [0, 1000]

Ratio Scales

- **Sometimes we would like to be able to say that one liquid is twice as hot as another, or that one project took twice as long as another. This needs the ratio scale, which is the most useful scale of measurement, and quite common in the physical sciences.**
- **Properties:**
  - **It is a measurement mapping that preserves ordering, the size of intervals between entities, and ratios between entities.**
  - **There is a zero element, representing total lack of the attribute. ["natural zero"]**
  - **The measurement mapping must start at zero and increase (or decrease) at equal intervals, known as units.**
  - **All arithmetic operations can be meaningfully applied to the classes in the range of the mapping.**

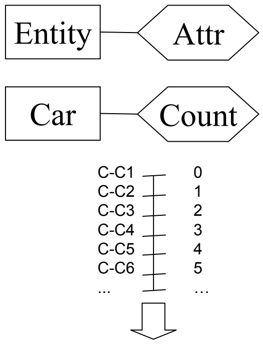
Example: Ratio Scale



Measure (Progr. Exec. Time) ∈ [0, ∞)

- **Measuring execution time of a software program:**
  - **Seconds**
  - **Minutes**
  - **Hours**
  - **…**
- **M’ = aM, a > 0**

Scale Types: Absolute Scale

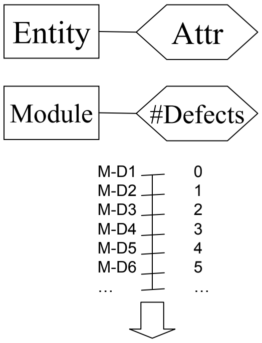


Measure (Car Count) ∈ IN_0

Absolute Scales

- **The absolute scale is the most restrictive of all. For any two measures, M and M', there is only one admissible transformation: the identity transformation.**
- **Properties:**
  - **The measurement for an absolute scale is made simply by counting the number of elements in the entity set.**
  - **The attribute always takes the form “number of occurrences of x in the entity set.”**
  - **There is only one possible measurement mapping.**
  - **All arithmetic manipulation of the resulting count is meaningful.**

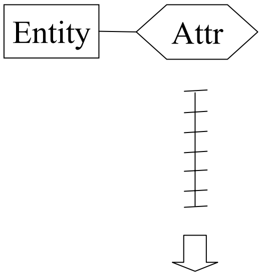
Example: Absolute Scale



Measure (Module Defect Count) ∈ IN_0

- **The count of defects detected in a module is absolute (but quality in terms of number of defects is not).**
- **The count of people working on a project is absolute (but staffing in terms of number of people is not).**
- **M’ ≡ M = {0, 1, 2, …}**

Measurement Scale Types (Summary)



Measure (Attribute) is well-defined, if scale and unit are clearly specified; specification of the unit makes the measure unambiguous!

- Nominal scale: classification of objects, where the fact that objects are different is preserved
- Ordinal scale: objects are ranked/ordered according to some criteria, but no information about the distance between the values is given
- Interval scale: differences between values are meaningful
- Ratio scale: there is a meaningful “zero” value, and ratios between values are meaningful
- Absolute scale: no transformation (other than identity) is meaningful (→ no unit needed)

NB: Scale types can be defined in terms of admissible transformations

Measurement Scale Types [Mor01] /1

Scale Type	Characterization	Example (generic)	Example (SE)
Nominal	Divides the set of objects into categories, with no particular ordering among them	Labeling, classification	Name of programming language, name of defect type
Ordinal	Divides the set of entities into categories that are ordered	Preference, ranking, difficulty	Ranking of failures (as measure of failure severity)
Interval	Comparing the differences between values is meaningful	Calendar time, temperature (Fahrenheit, Reaumur, Celsius)	Beginning and end date of activities (as measures of time distance)
Ratio	There is a meaningful “zero” value, and ratios between values are meaningful	Length, weight, time intervals, absolute temperature (Kelvin)	Lines of code (as measure of attribute “Program length/size”)
Absolute	There are no meaningful transformations of values other than identity	Object count	Count (as measure of attribute “Number of lines of code”)

Measurement Scale Types [Mor01] /2

Scale Type	Admissible Transformation	Indicators of Central Tendency
Nominal	Bijection (one-to-one mapping)	Mode
Ordinal	Monotonically increasing transformation	Mode + Median
Interval	Positive linear transformation M’= a M + b (a>0)	Mode + Median + Arithmetic Mean
Ratio	Proportionality M’= a M (a>0)	Mode + Median + Arithmetic Mean + Geometric Mean
Absolute	Identity M’ ≡ M	Mode + Median + Arithmetic Mean + Geometric Mean

The classification of scales has an important impact on their practical use, in particular on the statistical techniques and indices that can be used.

Example: Indicator of central tendency of a distribution of values (“Location”).

Mode = most frequent value of distribution
Median = the value such that not more than 50% of the values of the distribution are less than the median and not more than 50% of the values of the distribution are greater than the median

Measurement Scale – Summary

- There are 5 different types of measurement scales
- The type of the measurement scale determines
  - how measurement data can be treated statistically
    - indicators of central tendency
    - types of statistical distributions
    - types and power of statistical analyses (test, correlation, etc.)
  - whether statements involving measurement data are meaningful

Meaningfulness of Measurement-Based Statements

Definition:

A statement involving measurements is meaningful, if its truth value remains unchanged under any admissible transformation

Are the following statements meaningful?

Scale?	Meaningful?	Statement:
1. ratio	yes	1. “Peter is twice as tall as Hermann”
2. interval*	no*	2. “Peter’s temperature is 10% higher than Hermann’s“
3. ordinal	yes	3. “Defect X is more severe than defect Y”
4. ordinal	no	4. “Defect X is twice as severe as defect Y”
5. ratio	yes	5. “The cost for correcting defect X is twice as high as the cost for correcting defect Y”
6. interval	no	6. The average temperature of city A (30 ºC) is twice as high as the average temperature of city B (15 ºC)
7. interval	no	7. “Project Milestone 3 (end of coding) took ten times longer than Project Milestone 0 (project start)”
8. interval	yes	8. “Coding took as long as requirements analysis”

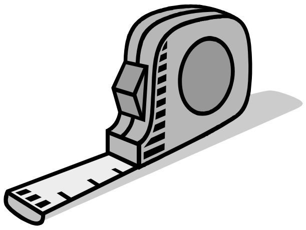
Meaningfulness of Measurement-Based Statements

Procedure to check for meaningfulness:

1. Apply the admissible transformation to measures in a statement S and obtain a transformed statement S’.

2. If S’ can be shown to be equivalent to S, then the statement S is meaningful for the scale associated with the admissible transformation.

Meaningfulness – Example 1



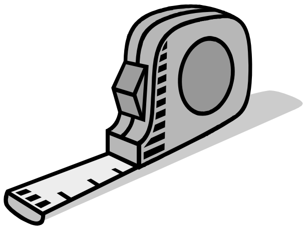
Ratio Scale

- Is statement (1) on the right meaningful, if X is measured on a ratio scale?
- Apply any admissible transformation M'=aM (a>0) for ratio scales:
- By arithmetic manipulation, (2) can always be made equivalent to (1). Therefore, the first statement is meaningful for a ratio scale.

(1) $$\frac{x_1 + x_2}{2} = m$$

(2) $$\frac{a \cdot x_1 + a \cdot x_2}{2} = a \cdot m$$

Meaningfulness – Example 2



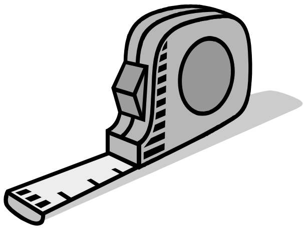
Interval Scale

- Is statement (1) on the right meaningful, if X is measured on an interval scale?
- Apply any admissible transformation M'=aM+b (a>0) for interval scales:
- By arithmetic manipulation, (2) can always be made equivalent to (1). Therefore, the first statement is meaningful for an interval scale.

(1) $$\frac{x_1 + x_2}{2} = m$$

(2) $$\frac{a \cdot x_1 + b + a \cdot x_2 + b}{2} = a \cdot m + b$$

Meaningfulness – Example 3



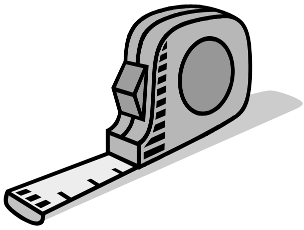
Ordinal Scale

- Is statement (1) on the right meaningful, if X is measured on an ordinal scale?
- Apply an admissible transformation for ordinal scales, e.g., x'=x³:
- For any pair of measurements x_1 and x_2, there exists always one admissible transformation such that statement (2) is false when (1) is true. Therefore, statement (1) is not meaningful for an ordinal scale.

(1) $$\frac{x_1 + x_2}{2} = m$$

(2) $$\frac{x_1^3 + x_2^3}{2} = m^3 = \left(\frac{x_1 + x_2}{2}\right)^3$$

Meaningfulness – Geometric Mean



Scale Type ?

- The geometric mean of a data set [a_1, a_2, ..., a_n] is given by

$$\left(\prod_{i=1}^{n} a_i\right)^{1/n} = \sqrt[n]{a_1 \cdot a_2 \cdot \ldots \cdot a_n}$$

- On which scale type is the geometric mean meaningful?

Objective vs. Subjective Measurement

- Objective Measurement
  - Usually the measurement process can be automated
  - (Almost) no random measurement error, i.e., the process is perfectly reliable
- Subjective Measurement
  - Human involvement in the measurement process
  - If we repeat the measurement of the same object(s) several times, we might not get exactly the same measured value every time, i.e., the measurement process is not perfectly reliable

Objective vs. Subjective Measurement (cont'd)

Examples:

- Subjective Measurement
  - Classification of defects into severity classes
  - Function Points (when counted manually)
  - Software Process Assessments
- Objective Measurement
  - Lines of Code
  - Cyclomatic Complexity
  - Memory Size
  - Test Coverage

To which category belong …
- Effort ?
- Time ?
- Defect Count ?

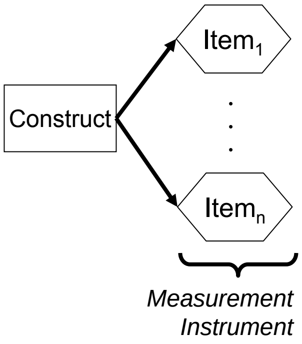
Why Use Subjective Measures?

- **It is not always possible to develop objective measures**
  - **e.g., when trying to measure abstract concepts like "skill", "competence", "functionality", "process capability", or "organizational maturity"**

Remarks on Subjective Measures

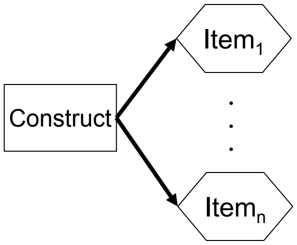
- Well developed subjective measures have proven to be useful
  - e.g., to select suppliers, to identify skill gaps, to assign priorities (e.g., for requirements)
- It is possible to have objective and subjective measures for the same attribute
  - e.g., measures of code size: LOC and Function Points
- Rule of Thumb:
  - If an objective measure is available, then it is preferable

Basic Concepts in Subjective Measurement



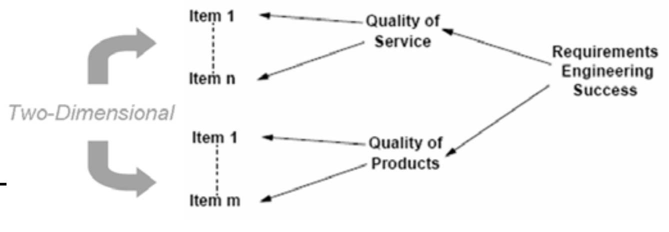
- **Construct:** A conceptual object that cannot be directly observed and therefore cannot be directly measured (i.e., we estimate the quantity we are interested in rather than directly measure it); for example:
  - User Satisfaction
  - Competence of a Software Engineer
  - Efficiency of a Process
  - Maturity of an Organization
- **Item:** A subjective measurement scale that is used to measure a construct
  - A question on a questionnaire is an item

The Dimensionality of Constructs

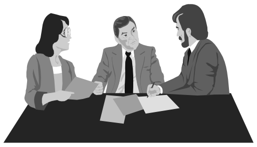


One-Dimensional

- Constructs can be one-dimensional or multi-dimensional
- If a construct is multidimensional, then each dimension covers a different and distinct aspect of the construct
  - e.g., the different dimensions of customer satisfaction

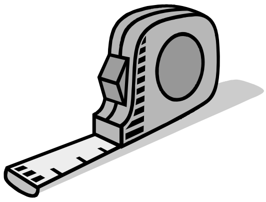


Procedures for Subjective Measurement



- Subjective Measures usually entail a well-defined Measurement Procedure that precisely describes:
  - How to collect the data (usually via questionnaires on paper or online)
  - How to conduct interviews
  - How to review documents (software artifacts)
  - In which order to assess the dimensions/items of the instrument, etc.
- Examples: ISO9000 Audit, CMM/CMMI Assessment, Function Points

Commonly Used Subjective Measurement Scales



- Likert-Type Scale
  - Evaluation-Type
  - Frequency-Type
  - Agreement-Type
- Semantic Differential Scale

Likert Type Scales

- Evaluation-type

  Example:
  - Familiarity with and comprehension of the software development environment:
    - □ Little
    - □ Unsatisfactory
    - □ Satisfactory
    - □ Excellent

- Frequency-type

  Example:
  - Customers provided information to the project team about the requirements:
    - □ Never
    - □ Rarely
    - □ Occasionally
    - □ Most of the time

- Agreement-type

  Example:
  - The tasks supported by the software at the customer site were changing frequently:
    - □ Strongly Agree
    - □ Agree
    - □ Disagree
    - □ Strongly Disagree

Semantic Differential Scale

- Items which include semantic opposites
- Example:
  - Processing of requests for changes to existing systems: the manner, method, and required time with which the MIS staff responds to user requests for changes in existing computer-based information systems or services.

Slow □ □ □ □ □ □ □ Fast

Timely □ □ □ □ □ □ □ Untimely

Assigning Numbers to Scale Responses

- Likert-Type Scales:

❑ Strongly Agree	→ 1
❑ Agree	→ 2
❑ Disagree	→ 3
❑ Strongly Disagree	→ 4

- Ordinal Scale
- But: Often the distances between the four response categories are approximately (conceptually) equidistant and thus are treated like approximate interval scales.

- Semantic Differential Scale:

Slow	□ □ □ □ □ □ □	Fast
	1 2 3 4 5 6 7	

- Ordinal scale, but again, often treated as interval scales

Software Measures: Validity & Reliability

Why is Validity an Issue?

Many

Important

Questions

How to measure

- **“modularity”?**
- **“cohesion”?**
- **“coupling”?**

→ Many suggestions have been made by many people!

→ Do these suggestions work?

Theoretical Validation

Problem 1:

- **How do we know whether a proposed measure adequately reflects my intuition / understanding about the attribute it purports to measure?**

Answer:

- **We have to make our intuition / understanding about the characteristics (properties) of the measured attribute explicit – then we can check whether the measure “reproduces” our assumptions**

Problem 2:

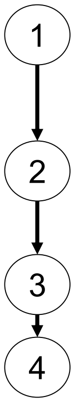
- **Do we all have the same intuition / understanding about the characteristics / properties of an attribute?**

Answers:

- **If we all make our assumptions explicit, we can check**
- **If we encounter differences, we can try to identify a set of necessary “core characteristics / properties” of the attribute under consideration.**

→ “Measurement Concepts”

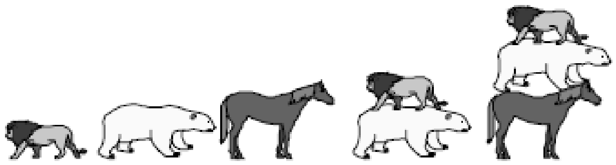
Theoretical Validation: Method



- Define an Empirical Relational System (ERS) with
  - A : set of objects to be measured
  - R_i : empirical relations between elements of A
  - o_j : binary operations on the empirical objects in A
- Define a Formal Relational System (FRS) with
  - B : set of formal objects
  - S_i : formal relations between the elements of B
  - $*_j$: binary operations on the formal objects in B
- Define measure(s) that map empirical objects (from A) into formal objects (in B)
- Show that the measure(s) preserve the Representation Condition

Empirical Relational System: Example

- Suppose we want to study the “height” (attribute) of “animals” (entities).
- The height of animals gives rise to empirical relations like “high”, “higher than”, “much higher than”



- A = { Lion, Bear, Horse, ... }
- R_1:= “HIGHER THAN”

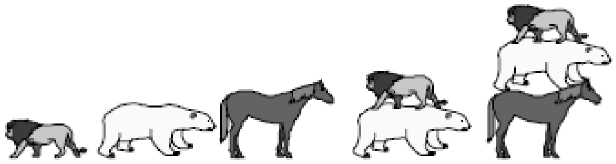
  $R_1(\text{Entity}_1, \text{Entity}_2)$ = Entity_1 IS HIGHER THAN Entity_2
- o_1:= “STANDING ON THE BACK OF” = ∇

  $R_1(\text{Entity}_1 \nabla \text{Entity}_2, \text{Entity}_3)$

Empirical Relational System: Example

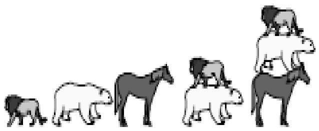
NB:

No numbers are involved → An Empirical Relation System embodies our understanding of the attribute.



- The Horse IS HIGHER THAN the Bear
- The Bear IS HIGHER THAN the Lion
- The Horse IS HIGHER THAN the Lion (R_1 is transitive)
- Lion ∇ Bear IS HIGHER THAN the Horse

Example: ERS, FRS, Measure (with Scale)



- m: ({Bear, Lion, Horse}, “Is Higher Than”, ∇)

  → ({1, 2, 2.5} ,>, +)
- Each entity of A is mapped into a number of B:

  m(Lion) = 1, m(Bear) = 2, m(Horse) = 2.5
- Each relation R_i is mapped into a relation S_i:

  “Is Higher Than” : >
- Each operation o_i is mapped into a numerical operation $\bullet_i$:

  ∇ : +

Measure m, Scale: Definition

- A **measure m** is a mapping m: σ(A) → B which yields for every empirical object a $\in$A a formal object (measurement value) m(a) $\in$ B. This mapping must not be arbitrary, hence leading to the following definition of a scale.
- Let ***A*** = ($A, R_1, \ldots, R_n, o_1, \ldots, o_m$) be an **empirical relational system** and ***B*** = ($B, S_1, \ldots, S_n, \bullet_1, \ldots, \bullet_m$) a **formal relational system** and **m** a measure.

  The Triple **(*A*, *B*, m)** is a **scale** if and only if for all i, j and for all a, b, $a_1, \ldots, a_k \in A$ the following holds:

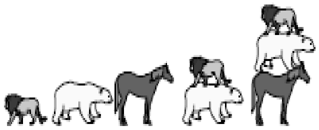
  $$R_i(a_1, \ldots, a_k) \Leftrightarrow S_i(m(a_1), \ldots, m(a_k))$$

  and $m(a\ o_j\ b) = m(a) \bullet_j m(b)$
- Example: If B is the set of real numbers, the triple **(*A*, *B*, m)** is a ratio scale.

Representation Condition

Representation Condition

Recall:
$R_i(a_1, \ldots, a_k) \Leftrightarrow S_i(m(a_1), \ldots, m(a_k))$
and
$m(a\ o_j\ b) = m(a) \bullet_j m(b)$



- **Definition:** All empirical relations must be preserved in the formal relational system.
- **Examples:**

  Horse “IS HIGHER THAN” Bear ⇔ m(Horse) > m(Bear)

  Bear “IS HIGHER THAN” Lion ⇔ m(Bear) > m(Lion)

  Horse “IS HIGHER THAN” Lion ⇔ m(Horse) > m(Lion)

  Lion ∇ Bear “IS HIGHER THAN” Horse

  ⇔ m(Lion ∇ Bear) > m(Horse)

  ⇔ m(Lion) + m(Bear) > m(Horse)

Theoretical Validation

Problem 1:

- **How do we know whether a proposed measure adequately reflects my intuition / understanding about the attribute it purports to measure?**

Answer:

- **We have to make our intuition / understanding about the characteristics (properties) of the measured attribute explicit – then we can check whether the measure “reproduces” our assumptions**

Problem 2:

- **Do we all have the same intuition / understanding about the characteristics / properties of an attribute?**

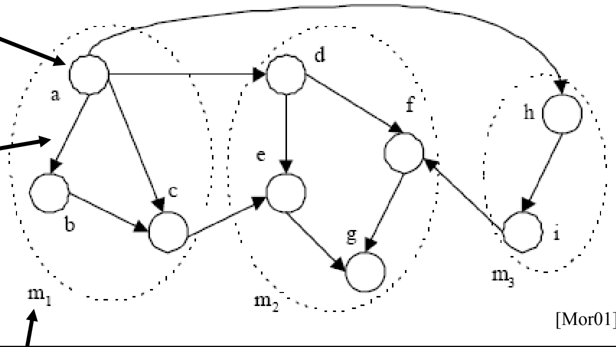
Answers:

- **If we all make our assumptions explicit, we can check**
- **If we encounter differences, we can try to identify a set of necessary “core characteristics / properties” of the attribute under consideration.**

→ “Measurement Concepts”

Example: System Complexity [BMB96]

Example System S:



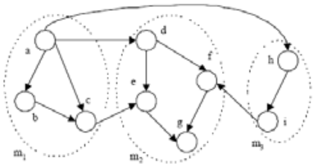
Element of S (node)

Relationship between Elements of S (edge)

Module

[Mor01]

Example: System Complexity [BMB96]



Formal Characterization of Software System:

- A system S is represented as a pair <E, R>
- E represents the set of elements of S
- R is a binary relationship on E ($R \subseteq E \times E$) representing the set of relationships between elements of S
- A module m of S is defined as: $m=<E_m, R_m>$ iff:
  - $E_m \subseteq E$
  - $R_m \subseteq E_m \times E_m$
  - $R_m \subseteq R$

NB: System Complexity is not the same as Psychological or Cognitive Complexity

Example: System Complexity

Properties:

1. Non-Negativity
   - The complexity of a system S is non-negative: Complexity(S) ≥ 0
2. Null Value
   - The complexity of a System S is null if there are no relationships between the elements of the system: $R = Ø \Rightarrow$ Complexity(S) =0.
3. Module Monotonicity
   - The complexity of a system S is not smaller than the sum of the complexities of any two of its modules with no relationships in common:
     ($m_1=<E_{m1}, R_{m1}>$ and $m_2=<E_{m2}, R_{m2}>$ and $m_1 \cup m_2 \subseteq S$ and $R_{m1} \cap R_{m2}= Ø$)
     $\Rightarrow$ Complexity(S) $\geq$ Complexity(m_1) + Complexity(m_2)

Example: System Complexity [BMB96]

Properties (cont'd):

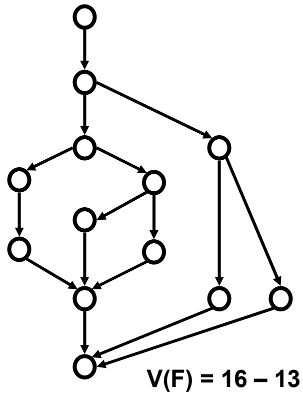
4. Disjoint Module Additivity
   - The complexity of a system S composed of two disjoint modules is equal to the sum of the complexities of the two modules:

     ($S=m_1 \cup m_2$ and $m_1 \cap m_2 = Ø$) $\Rightarrow$ Complexity(S) = Complexity(m_1) + Complexity(m_2)

5. Symmetry
   - The complexity of a system does not depend on the convention chosen to represent the relationships between its elements (e.g., direction of arcs that represent edges):

     ($S^{-1}=<E, R^{-1}>$) $\Rightarrow$ Complexity(S) = Complexity(S^{-1})

Example: System Complexity



V(F) = 16 – 13 + 2 = 5

Proposal of a System Complexity Measure:

- McCabe's Structural Complexity Measure [McC76]:
  - Def.: for a program with (control-)flow graph F, the cyclomatic number is calculated as:

    V(F) = e – n + 2p

    where

    e: #edges of F

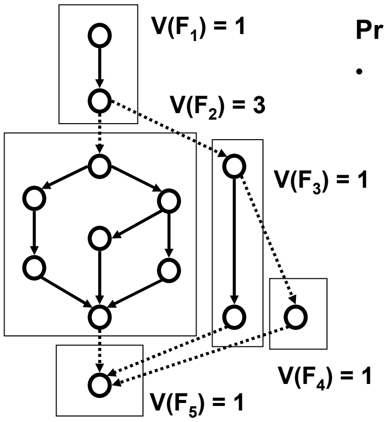
    n: #nodes of F

    p: #programs (modules)

    or, for p=1:

    V(F) = d + 1, where d: #decision nodes of F

Example: System Complexity



Proposal of a System Complexity Measure:

- McCabe's Structural Complexity Measure [McC76]:
  - Def.: for a program with (control-)flow graph F, the cyclomatic number is calculated as:

$V(F) = e – n + 2p$

where

e: #edges of F

n: #nodes of F

p: #programs (modules)

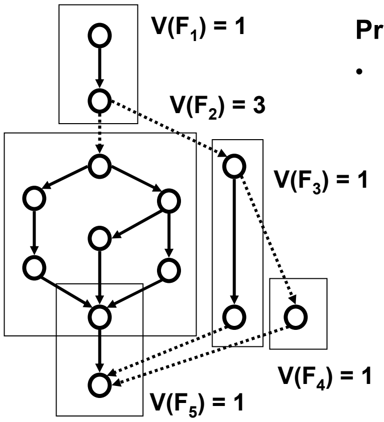
or, for p=1:

$V(F) = d + 1$, where d: #decision nodes of F

$V(F) = 10 – 13 + 2 \times 5 = 7$

with $F_1 \cup F_2 \cup F_3 \cup F_4 \cup F_5 \subseteq F$

Example: System Complexity



Proposal of a System Complexity Measure:

- McCabe's Structural Complexity Measure [McC76]:
  - Def.: for a program with (control-)flow graph F, the cyclomatic number is calculated as:

$V(F) = e – n + 2p$

where

e: #edges of F

n: #nodes of F

p: #programs (modules)

or, for p=1:

$V(F) = d + 1$, where d: #decision nodes of F

$V(F) = 11 – 13 + 2 \times 5 = 6$

with $F_1 \cup F_2 \cup F_3 \cup F_4 \cup F_5 \subseteq F$ and $F_2 \cap F_5 \neq \emptyset$

Example: System Complexity (cont'd)

What does this result tell us about the proposed measure of program complexity?

Answer 1:

- **McCabe's cyclomatic complexity measure does not appropriately capture program complexity**
  - **What about: V(F) := e – n + p (p: #modules)**

Answer 2:

- **We might have to convince ourselves – and the community of researchers and practitioners – that Property 3 (Monotonicity) is not necessary**

Usefulness of Measurement Concepts [Mor01]

- Sets of properties for measurement concepts such as the one described above are useful to:
  - Model intuition about the properties that measures of an attribute should possess
  - Show similarities and differences among measures of different attributes
  - Check whether a given measure is consistent with intuition
    - Note: the check of measurement results can either lead to rejection of a measure or provide supporting evidence for the validity of a measure, but it can never proof validity

Validity of a Measure – 2 Issues

Issue 1

Theoretical Validity

- When I apply a proposed measure, do the measurement results represent my/others intuition/understanding of what “modularity” / “cohesion” / “coupling” mean?

Issue 2

Empirical Validity

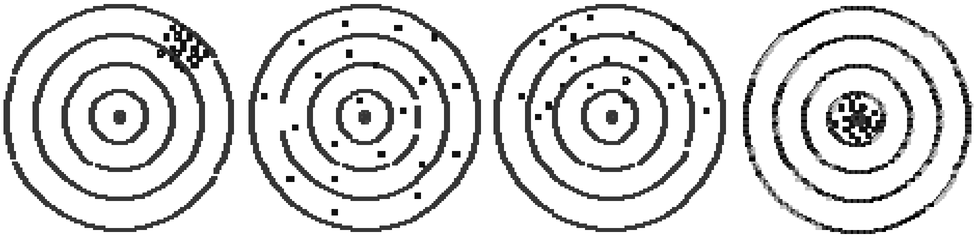
- Is the measure practical, i.e., can it be used to predict values of other interesting attributes (e.g., maintainability), does it help explain other interesting phenomena, can it be collected automatically, is it “cheap”, etc.

Reliability of Measures – Definition

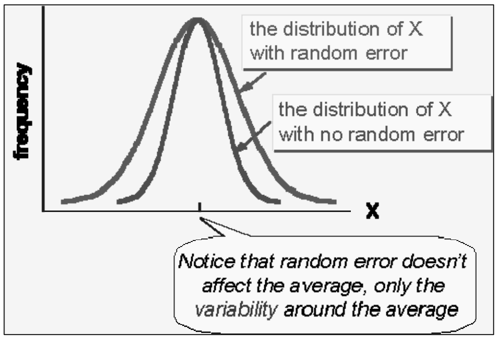
- **Definition:**
  - The extent to which a measurement process will yield exactly the same value if applied repeatedly to the same object

- **Remark:**
  - In software measurement, reliability is mainly an issue related to *Subjective Measures*

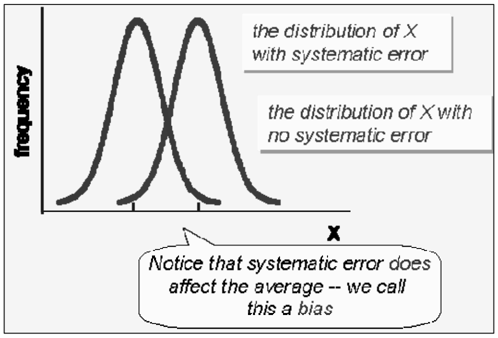
Reliability versus Validity



2 Types of Measurement Error



Random Error (Noise)



Systematic Error (Bias)

Reliability Estimation Techniques – Classes

- It is not possible to assess the reliability of a measure (or measurement instrument) directly, it has to be estimated based on empirical data
  - e.g., by using test data taken from a subset of the actual population
- There are four main classes of Reliability Estimation Techniques:
  1. **Inter-Rater (or Inter-Observer) Reliability (or Agreement):**
     - To assess the degree to which different raters/observers give consistent estimates of the same phenomenon (using the same measure)
  2. **Internal Consistency Reliability:**
     - To asses the consistency of measurement results across items within a (one-dimensional) measurement instrument
  3. **Test-Retest Reliability:**
     - To asses the consistency of a measurement instrument from one time to another
  4. **Parallel Forms (or Alternative Forms) Reliability:**
     - To assess the consistency of the results of two measurement instruments

Reliability Estimation Techniques – Classes

- **Number of administrations** is the number of times that the same object is measured (per observer)
- **Number of instruments** is the number of different but equivalent instruments that would need to be administered

		Number of Instruments	
		One	Two
Number of Administrations (per Observer / Rater)	One	Inter-Rater Internal Consistency	Parallel Forms (immediate)
	Two	Test-Retest	Parallel Forms (delayed)

http://www.socialresearchmethods.net/kb/reltypes.php