SOFTWARE MEASUREMENT AND METRICS

Tuesday, May 23, 2017

By Saman Gunawardena

ICE BREAKER

- Imagine you are developing a software with a team of 5 of your friends. You need to develop the software in 5 months.
  - How would you ensure that it will be developed in 5 months
  - How would you ensure that it will have the appropriate quality
  - How would you measure how much you all have learnt experienced at the beginning of the project to end of the project

Sources

- *Software Engineering*, I. Sommerville, Addison-Wesley, Fifth Edn., 1996.
- *Software Metrics: A Rigorous & Practical Approach*, Norman E. Fenton, Shari I. Pfleeger, Thompson Computer Press, 1996.

Roadmap

- Measurement Basics
- **What are metrics? Why do we need them?**
- Metrics for cost estimation
- Metrics for software quality evaluation
- Object-Oriented metrics in practice

Measurements Basics

- Measurement is part of many systems that govern our lives
- Measurement also plays a key role in Software and Software Development

Measurements Basics …

- Measurement
  - Comparison & calculations according to a well defined set of rules
  - The process by which numbers or symbols are assigned to *attributes* of *entities* in the real world in a way as to describe them according to clearly defined rules

Measurements Basics …

- Entity
  - An object (i.e. person or room) or an event (i.e. journey or the testing phase of software) in the real world
- Attribute
  - Feature or property of an entity (i.e. area or color of a room; elapsed time of testing phase)

Measurements Basics …

- Attributes and entities often interchanged in every day speech
  - "It's hot today"
- OK for every day speech, but not acceptable for scientific measure
  - Ambiguous to say "we measure a room"

Measurements Basics . . .

- Attributes often defined by using numbers or symbols:
  - Height: inches, feet, centimeters
  - Price: dollars, pounds sterling, Euros
  - Clothing: small, medium, large
- These numbers & symbols are abstractions used to reflect our perception of the real world

Measurements Basics …

- Two Kinds of Quantification:

1. Measurement
   - Direct quantification
2. Calculation
   - Indirect quantification

Direct and Indirect Measures

Direct Measures

- *Measured* directly in terms of the observed attribute (usually by counting)
  - Length of source-code, Duration of process, Number of defects discovered

Indirect Measures

- *Calculated* from other direct and indirect measures
  - Module Defect Density = Number of defects discovered / Length of source

Software Engineering Measurements . . .

- Direct Measurements in SE:
  - *Length* of source code
  - *Duration* of a phase
  - *Number* of defects found during phase
  - *Time* a programmer spends on a project

Software Engineering Measurements . . .

- Indirect Measurements in SE:
  - *Programmer productivity*
  - *Module Defect Density*
  - *Defect detection efficiency*
  - *Requirements stability*
  - *Test Effectiveness ratio*
  - *System Spoilage*

Objective & Subjective Measurements

- Objective measurement uses hard data that can be obtained by counting, stacking, weighing, timing, etc. Examples include number of defects, hours worked, or completed deliverables. An objective measurement should result in identical values for a given measure, when measured by two or more qualified observers.

Objective & Subjective Measurements

- Subjective data is normally observed or perceived. It is a person's perception of a product or activity, and includes personal attitudes, feelings and opinions, such as how easy a system is to use, or the skill level needed to execute the system. With subjective measurement, even qualified observers may determine different values for a given measure, since their subjective judgment is involved in arriving at the measured value. The reliability of subjective measurement can be improved through the use of guidelines, which define the characteristics that make the measurement result one value or another

Objective & Subjective Measurements

- Objective measurement is more reliable than subjective measurement, but as a general rule, subjective measurement is considered more important. The more difficult something is to measure, the more valuable it is. For example, it is more important to know how effective a person is in performing a job (subjective measurement), than knowing they got to work on time (objective measurement).

Objective & Subjective Measurements

- Following are a few other examples of objective and subjective measurements
  - The size of a software program measured in LOC is an objective product measure. Any informed person, working from the same definition of LOC, should obtain the same measure value for a given program.
  - The classification of software as user-friendly is a subjective product measure. For a scale of 1-5, customers of the software would likely rate the product differently. The reliability of the measure could be improved by providing customers with a guideline that describes how having or not having a particular attribute affects the scale.
  - Development time is an objective process measure.
  - Level of programmer experience is a subjective process measure.

Measurements Basics …

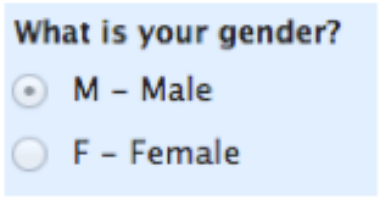
- Types of Measurements Scales:
  1. Nominal
  2. Ordinal
  3. Interval
  4. Ratio

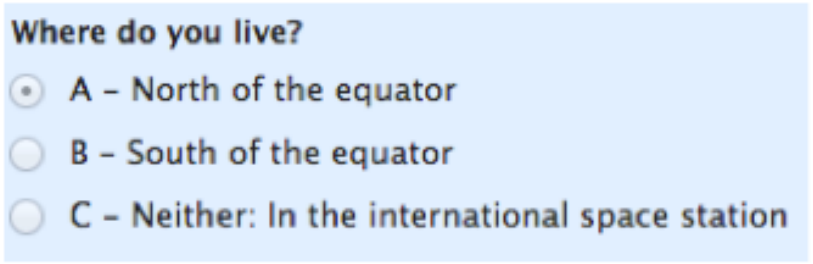
Measurements Basics …

- Nominal
  - Most primitive form of measurement
  - Places elements in classification scheme
  - No notion of ordering of the classes
  - Example: Where was the bug introduced – requirements, design, or code?

Measurements Basics …

Nominal Measurement Example





Measurements Basics ...

- Ordinal
  - Augments nominal scale with information about ordering of the classes or categories
  - Classes ordered with respect to the attribute
  - Numbers represent ranking, so arithmetic operations not allowed

Measurements Basics …

Ordinal Measurement Example

How do you feel today?

- 1 – Very Unhappy
- 2 – Unhappy
- 3 – OK
- 4 – Happy
- 5 – Very Happy

How satisfied are you with our service?

- 1 – Very Unsatisfied
- 2 – Somewhat Unsatisfied
- 3 – Neutral
- 4 – Somewhat Satisfied
- 5 – Very Satisfied

Measurements Basics ...

- Interval
  - Carries more information about entities, making it more powerful than nominal and ordinal
  - Captures information about size of the intervals that separates classes
  - Addition and subtraction allowed, but not multiplication and division

Measurements Basics …

Interval Measurement Example

Measurements Basics . . .

- Ratio
  - Most useful scale of measurements
  - Most common scale in physical sciences
  - Contains zero element → lack of attribute
  - Preserves ordering, size, and ratios
  - All arithmetic can be applied

Measurements Basics …

Ratio Measurement Example

Measurements Basics …

Provides:	Nominal	Ordinal	Interval	Ratio
The "order" of values is known		✔	✔	✔
"Counts," aka "Frequency of Distribution"	✔	✔	✔	✔
Mode	✔	✔	✔	✔
Median		✔	✔	✔
Mean			✔	✔
Can quantify the difference between each value			✔	✔
Can add or subtract values			✔	✔
Can multiple and divide values				✔
Has "true zero"				✔

Why Metrics?

When you can measure what you are speaking about and express it in numbers, you know something about it; but when you cannot measure, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science.

— Lord Kelvin

Why Measure Software?

Estimate cost and effort	measure correlation between specifications and final product
Improve productivity	measure value and cost of software
Improve software quality	measure usability, efficiency, maintainability ...
Improve reliability	measure mean time to failure, etc.
Evaluate methods and tools	measure productivity, quality, reliability ...

"You cannot control what you cannot measure" — De Marco, 1982
"What is not measurable, make measurable" — Galileo

What are Software Metrics?

Software metrics

- Any type of measurement which relates to a software system, process or related documentation
  - Lines of code in a program
  - the Fog index (calculates readability of a piece of documentation)

    $0.4 \times (\#\text{ words} / \#\text{ sentences}) + (\%\text{ words} \geq 3\text{ syllables})$
  - number of person-days required to implement a use-case

(Measures vs Metrics)

Mathematically, a metric is a function m measuring the *distance between two objects* such that:

1. $\forall$ x, m(x,x) = 0
2. $\forall$ x, y, m(x,y) = m(y,x)
3. $\forall$ x, y, z, m(x,z) $\leq$ m(x,y) + m(y,z)

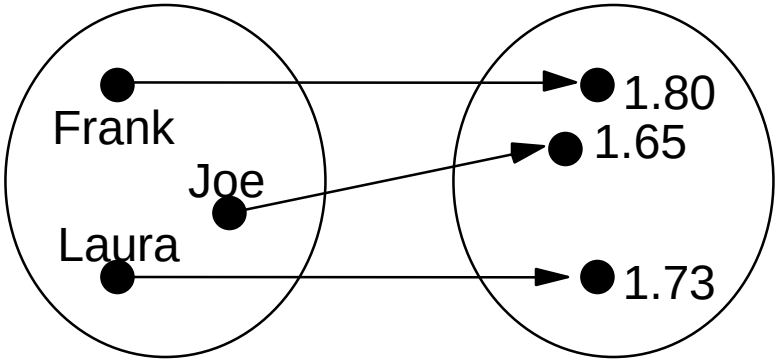
So, technically "software metrics" is an abuse of terminology, and we should instead talk about "software measures".

Measurement Mapping

Measure & Measurement

A measure is a function mapping an *attribute* of a real world entity (= the domain) onto a *symbol* in a set with known mathematical relations (= the range).

A measurement is the symbol assigned to the real world attribute by the measure.



Example: measure mapping height attribute of person on a number representing height in meters.

Purpose: *Manipulate symbol(s) in the range to draw conclusions about attribute(s) in the domain*

Preciseness

To be precise, the definition of the measure must specify:

- ***domain:*** do we measure people's height or width?
- ***range:*** do we measure height in centimetres or inches?
- ***mapping rules:*** do we allow shoes to be worn?

Possible Problems

Example: Compare productivity in lines of code per time unit.

Do we use the same units to compare?	What is a “line of code”? What is the “time unit”?
Is the context the same?	Were programmers familiar with the language?
Is “code size” really what we want to produce?	What about code quality?
How do we want to interpret results?	Average productivity of a programmer? Programmer X is twice as productive as Y?
What do we want to do with the results?	Do you reward “productive” programmers? Do you compare productivity of software processes?

GQM

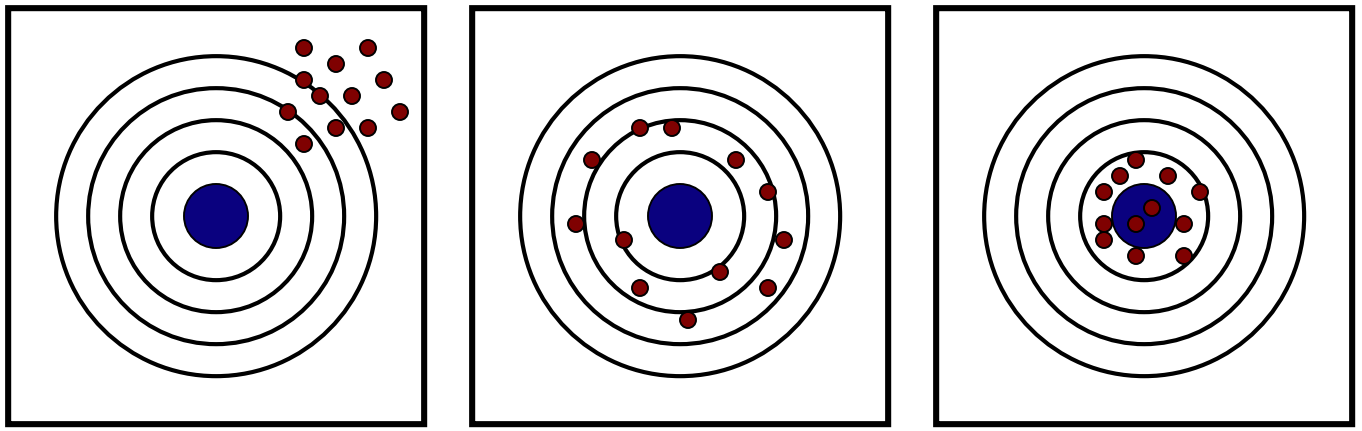
Goal — Question — Metrics approach

[Basili et al. 1984]

- Define *Goal*
  - e.g., "How effective is the coding standard XYZ?"
- Break down into *Questions*
  - "Who is using XYZ?"
  - "What is productivity/quality with/without XYZ?"
- Pick suitable *Metrics*
  - Proportion of developers using XYZ
  - Their experience with XYZ ...
  - Resulting code size, complexity, robustness ...

Validity and reliability

- A good metric is both valid *(measures what it is intended to measure)* and reliable *(yields consistent results)*



Reliable but not valid

Valid but not reliable

Valid and reliable

See: Stephen H. Kan, *Metrics and Models in Software Quality Engineering*, Addison Wesley, 2002. Ch. 3.4

Some Desirable Properties of Metrics

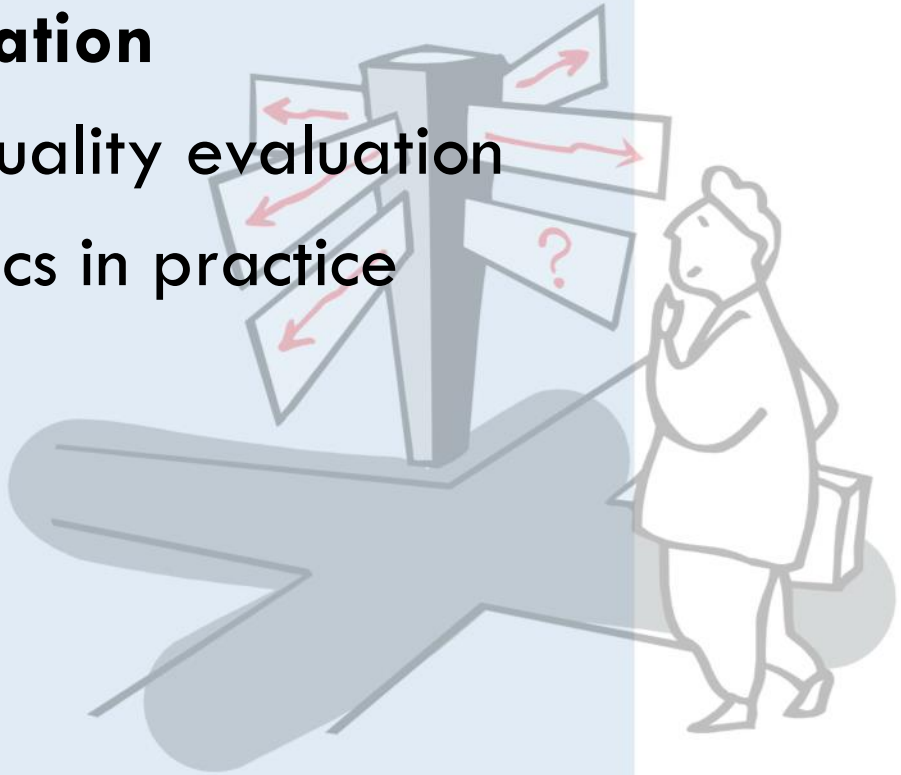
- Valid and reliable (consistent)
- Objective, precise
- Intuitive
- Robust (failure-tolerant)
- Automatable and economical (practical)
- …

Caveat: Attempts to define formally desirable properties have been heavily disputed …

See: 2.Brian Henderson-Sellers, *Object-Oriented Metrics: Measures of Complexity*, Prentice-Hall, 1996, Ch. 2.6

Roadmap

- What are metrics? Why do we need them?
- **Metrics for cost estimation**
- Metrics for software quality evaluation
- Object-Oriented metrics in practice



Cost estimation objectives

Cost estimation and planning/scheduling are closely related activities

Goals

- To establish a budget for a software project
- To provide a means of controlling project costs
- To monitor progress against the budget
  - comparing planned with estimated costs
- To establish a cost database for future estimation

Estimation techniques

Expert judgement	cheap, but risky!
Estimation by analogy	limited applicability
Parkinson's Law	unlimited risk!
Pricing to win	i.e., you do what you can with the money
Top-down estimation	may miss low-level problems
Bottom-up estimation	may underestimate integration costs
Algorithmic cost modelling	requires correlation data

Each method has strengths and weaknesses!

Estimation should be based on several methods

Algorithmic cost modelling

- Cost is estimated as a *mathematical function of product, project and process attributes* whose values are estimated by project managers
- The function is derived from a study of *historical costing data*
- Most commonly used product attribute for cost estimation is *LOC* (code size)
- Most models are basically similar but with different attribute values

Measurement-based estimation

A. Measure

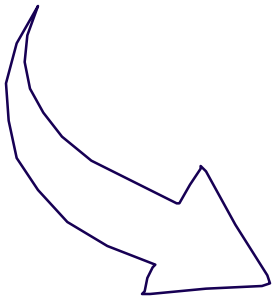
Develop a *system model* and measure its size

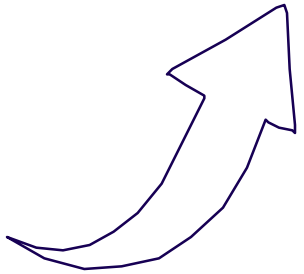
B. Estimate

Determine the effort with respect to an *empirical database* of measurements from *similar projects*

C. Interpret

Adapt the effort with respect to a specific *Development Project Plan*





Lines of code

Lines of Code as a measure of system size?

- Easy to measure; but *not well-defined* for modern languages
  - *What's a line of code?*

- A *poor indicator of productivity*
  - Ignores software reuse, code duplication, benefits of redesign
  - The lower level the language, the more productive the programmer!
  - The more verbose the programmer, the higher the productivity!

Function points

Function Points (Albrecht, 1979)

- Based on a combination of program characteristics:
  - external inputs and outputs
  - user interactions
  - external interfaces
  - files used by the system
- A weight is associated with each of these
- The function point count is computed by multiplying each raw count by the weight and summing all values
- Function point count modified by complexity of the project

Function points

Good points, bad points

- Can be measured already after design
- FPs can be used to estimate LOC depending on the average number of LOC per FP for a given language
- LOC can vary wildly in relation to FP
- FPs are very subjective — depend on the estimator. They cannot be counted automatically

Programmer productivity

A measure of the rate at which individual engineers involved in software development produce software and associated documentation

Productivity metrics

- Size-related measures based on some output from the software process. This may be lines of delivered source code, object code instructions, etc.
- Function-related measures based on an estimate of the functionality of the delivered software. Function-points are the best known of this type of measure

…

Programmer productivity …

Productivity estimates

- Real-time embedded systems, 40-160 LOC/P-month
- Systems programs , 150-400 LOC/P-month
- Commercial applications, 200-800 LOC/P-month

Quality and productivity

- *All metrics based on volume/unit time are flawed because they do not take quality into account*
  - Productivity may generally be increased at the cost of quality
  - It is not clear how productivity/quality metrics are related

The COCOMO model

- Developed at TRW, a US defence contractor
- Based on a *cost database* of more than 60 different projects
- Exists in three stages
  1. ***Basic*** — Gives a "ball-park" estimate based on product attributes
  2. ***Intermediate*** — Modifies basic estimate using project and process attributes
  3. ***Advanced*** — Estimates project phases and parts separately

Basic COCOMO Formula

- $\text{Effort} = C \times PM^S \times M$
  - Effort is measured in person-months
  - C is a *complexity factor*
  - PM is a product metric (size or functionality, usually KLOC)
  - exponent S is close to 1, but increasing for large projects
  - M is a multiplier based on process, product and development attributes (~ 1)

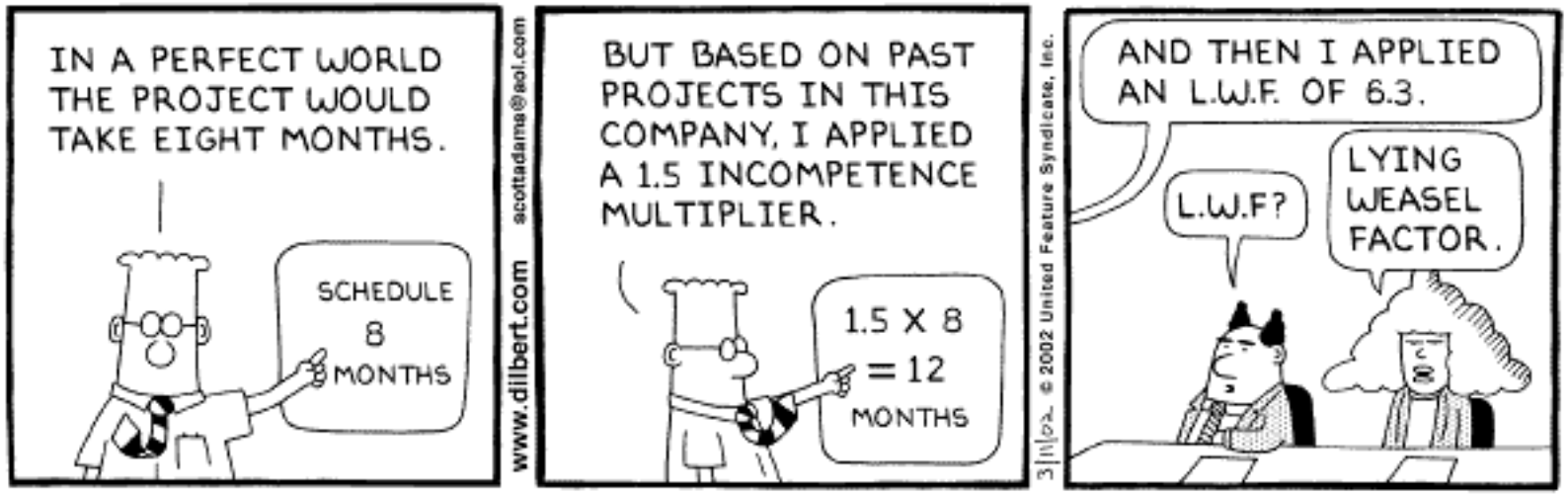
COCOMO Project classes

Organic mode: *small teams*, familiar environment, *well-understood applications*, no difficult non-functional requirements (EASY)	$Effort = 2.4\ (KDSI)^{1.05} \times M$
Semi-detached mode: Project team may have *experience mixture*, system may have more *significant non-functional constraints*, organization may have less familiarity with application (HARDER)	$Effort = 3\ (KDSI)^{1.12} \times M$
Embedded: *Hardware/software systems*, *tight constraints*, unusual for team to have deep application experience (HARD)	$Effort = 3.6\ (KDSI)^{1.2} \times M$

KDSI = Kilo Delivered Source Instructions

COCOMO assumptions and problems

- Implicit productivity estimate
  - Organic mode = 16 LOC/day
  - Embedded mode = 4 LOC/day
- Time required is a function of total effort *not team size*
- Not clear how to adapt model to *personnel availability*



http://en.wikipedia.org/wiki/Fair_use

COCOMO assumptions and problems

...

- *Staff required* can't be computed by dividing the development time by the required schedule
- The number of people working on a project varies depending on the *phase of the project*
- The more people who work on the project, the *more total effort* is usually required (!)
- Very *rapid build-up* of people often correlates with *schedule slippage*

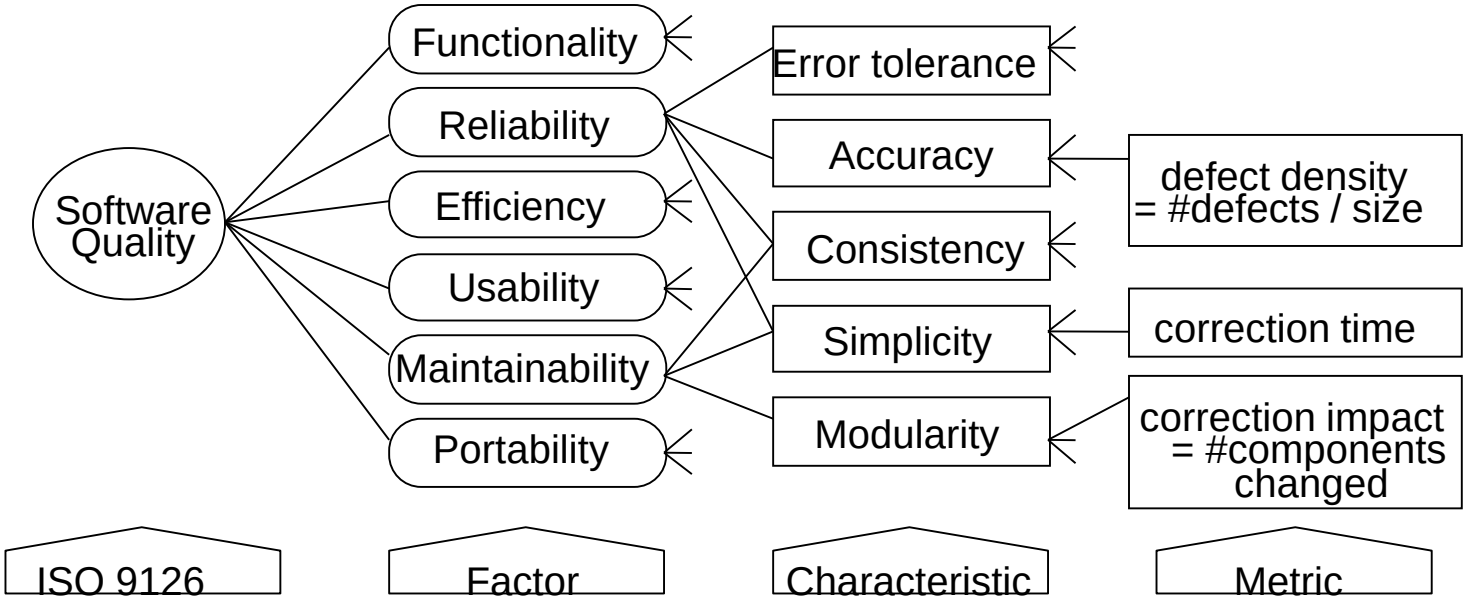
Roadmap

- What are metrics? Why do we need them?
- Metrics for cost estimation
- **Metrics for software quality evaluation**
- Object-Oriented metrics in practice

Quantitative Quality Model

Quality according to ISO 9126 standard

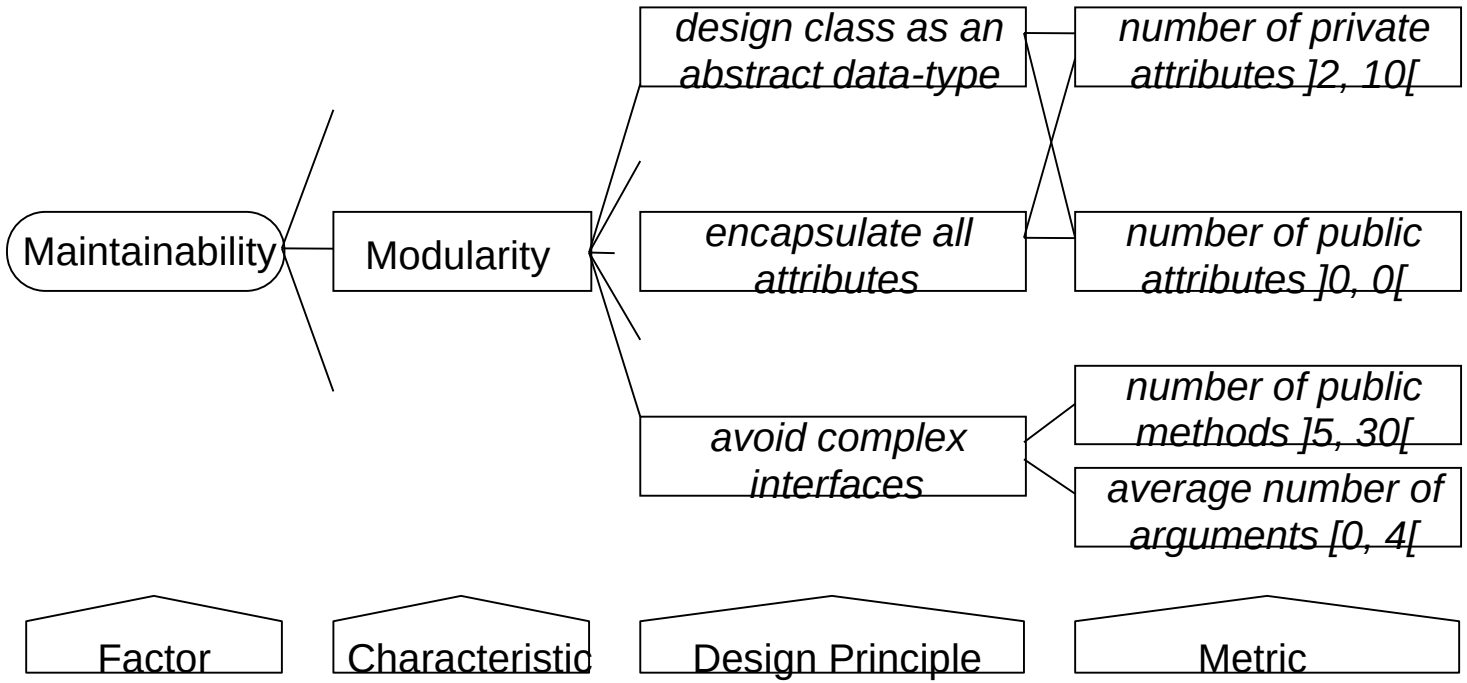
- Divide-and conquer approach via "hierarchical quality model"
- Leaves are simple metrics, measuring basic attributes



“Define your own” Quality Model

Define the quality model with the development team

- Team chooses the characteristics, design principles, metrics ... and the thresholds



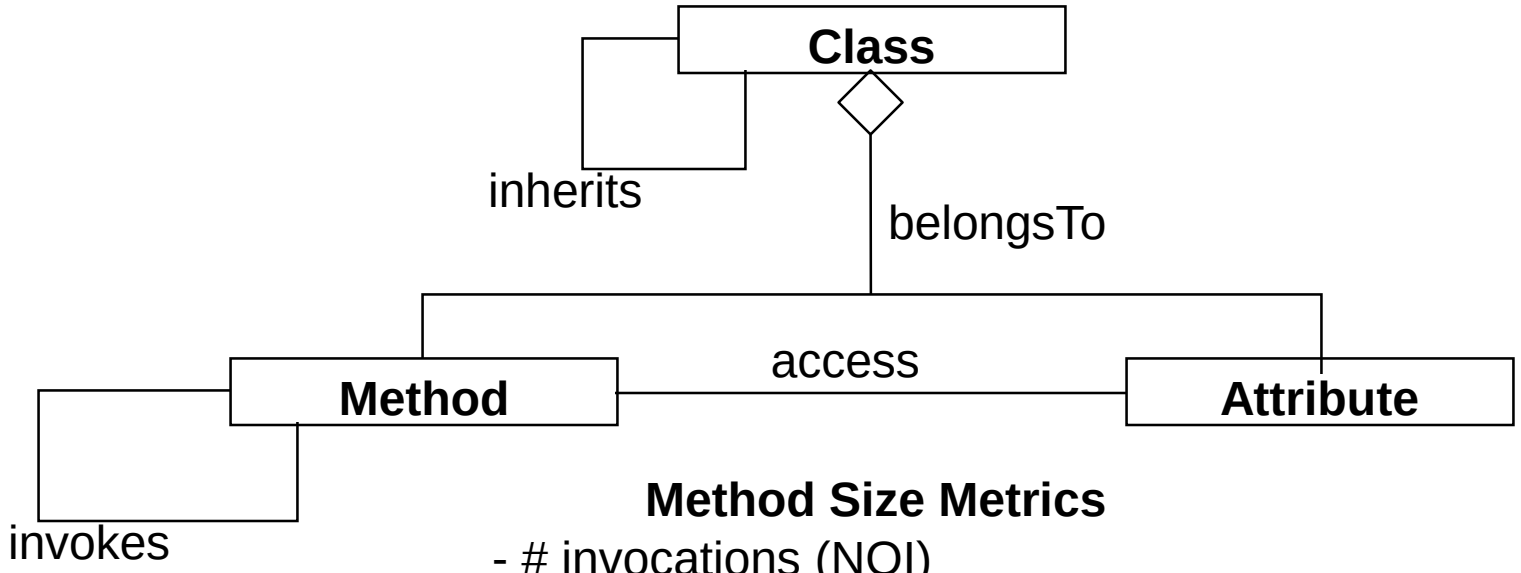
Sample Size (and Inheritance) Metrics

Inheritance Metrics

- hierarchy nesting level (HNL)
- # immediate children (NOC)
- # inherited methods, unmodified (NMI)
- #overridden methods (NMO)

Class Size Metrics

- # methods (NOM)
- # attributes, instance/class (NIA, NCA)
- # S of method size (WMC)



Method Size Metrics

- # invocations (NOI)
- # statements (NOS)
- # lines of code (LOC)
- # arguments (NOA)

Sample Coupling & Cohesion Metrics

The following definitions stem from [Chid91a], later republished as [Chid94a]

Coupling Between Objects (CBO)

CBO = number of other classes to which given class is coupled

Interpret as "number of other classes a class requires to compile"

Lack of Cohesion in Methods (LCOM)

LCOM = number of disjoint sets (= not accessing same attribute) of local methods

Coupling & Cohesion Metrics

Beware!

Researchers disagree whether coupling/cohesion methods are valid

- Classes that are observed to be cohesive may have a high LCOM value
  - due to accessor methods
- Classes that are not much coupled may have high CBO value
  - no distinction between data, method or inheritance coupling

Sample Quality Metrics (I)

Productivity (Process Metric)

- functionality / time
- functionality in LOC or FP; time in hours, weeks, months
  - be careful to compare: the same unit does not always represent the same
- Does not take into account the quality of the functionality!

Sample Quality Metrics (II)

Reliability (Product Metric)

- mean time to failure =
  mean of probability density function PDF
  - for software one must take into account the fact that repairs will influence the rest of the function ⇒ quite complicated formulas
- average time between failures = # failures / time
  - time in execution time or calendar time
  - necessary to calibrate the probability density function
- mean time between failure = MTTF + mean time to repair
  - to know when your system will be available, take into account repair

Sample Quality Metrics (III)

Correctness (Product Metric)

- "a system is correct or not, so one cannot measure correctness"
- defect density = # known defects / product size
  - product size in LOC or FP
  - # known defects is a time based count!
- *do not compare across projects* unless your data collection is sound!

Sample Quality Metrics (IV)

Maintainability (Product Metric)

- #time to repair certain categories of changes
- "mean time to repair" vs. "average time to repair"
  - similar to "mean time to failure" and "average time between failures"
- beware of the units
  - "categories of changes" is subjective
  - time =?
    problem recognition time + administrative delay time + problem analysis time + change time + testing & reviewing time

Can you answer the following questions?

- During which phases in a software project would you use metrics?
- Is the Fog index a "good" metric?
- How would you measure your own software productivity?
- Why are coupling/cohesion metrics important? Why then are they so rarely used?

Measurement – Part 2 (AGENDA)

- **Empirical Investigation**
- **Software Engineering Measurement**
  - Measurements Used in SE
  - Software Metrics
- **Classifying Software Measures**
  - Processes
  - Product
  - Resource
- **Measuring Software Size**
- **Measuring Software Quality**

Investigation

- Empirical Investigation
  - Empirical data vs. advice and counsel?
  - Better to rely on scientific research for objective decisions, vs. advice and counsel

Empirical Investigations

- Several types of Assessments
  - Surveys
    - Research in the large
  - Case Studies
    - Research in the typical
  - Formal Experiments
    - Research in the small

Software Engineering Measurements

- Computer Science provides theoretical foundations for building software
- SE focuses on implementing software in controlled and scientific manner

Software Engineering Measurements . . .

- Engineering disciplines use methods based on models and theories
- Measurements are the underpinning of scientific process

Software Engineering Measurements . . .

- Imagine electrical, mechanical, or civil engineering without measurements?
- Measurements still viewed as luxury in Software Engineering
  - Done infrequently, inconsistently, and incompletely

Software Engineering Measurements . . .

- Measurements are essential in engineering disciplines
- Measurements essential for SE even when project not in trouble
  - Helps to control the project, not just run it

Software Engineering Measurements . . .

"You cannot control what you cannot measure"

- Tom Demarco, 1982

- Measurements key for
  - Understanding
  - Controlling
  - Improving

Software Engineering Measurements . . .

- Scope of Software Metrics:
  - Cost and Effort Estimation
  - Productivity Models and Measures
  - Data Collection
  - Quality Models and Measures
  - Reliability Models
  - Performance Evaluation and Models

Software Engineering Measurements ...

- Scope of Software Metrics ...
  - Structural and Complexity Models
  - Management Metrics
  - Evaluation of Methods and Tools
  - Capability Maturity Assessments

Classifying Software Measures

- Software Entity and Attribute Classes to Measure:
  - *Processes*: collection of software related activities
  - *Products*: artifacts, deliverables, or documents from process activity
  - *Resources:* entities required by process activity - personnel, materials

Classifying Software Measures

- Attributes for each class of entity:
  - *Internal:* measured purely in terms of the process, product, or resource itself
  - *External:* measured only with respect to how the process, product or resource relates to its environment

Classifying Software Measures

Products		
Specifications	size, reuse, modularity, functionality, ...	comprehensibility, maintainability, ...
Design	size, reuse, modularity, coupling, cohesiveness, ...	quality, complexity, maintainability, ...
Code	size, reuse, modularity, coupling, functionality, ...	reliability, usability, maintainability, ...
---	---	---
Processes		
Detailed Design	time, effort, number of specification faults found, ...	cost, cost-effectiveness, ...
Testing	time, effort, number of coding faults found, ...	cost, cost-effectiveness, stability, ...
----	---	---
Resources		
Personnel	age, price, ...	productivity, experience, intelligence, ...
Software	price, size, ...	usability, reliability, ...
Hardware	price, speed, memory size, ...	reliability, ...

Measuring Software Size

- Difficulties in estimating software size
  - Simple measures that don't reflect effort, productivity, and cost are rejected
  - Define a fundamental set of attributes for software size; each attribute captures key aspect of software size

Measuring Software Size …

- Fenton & Pfleeger suggestion for measuring software size:
  - *Length:* physical size of product
  - *Functionality:* measures functions
  - *Reuse:* amount of product reused
  - *Complexity (efficiency):* interpretation of complexity: Problem, Algorithmic, Structural, Cognitive

Measuring Software Quality

- McCall and Cavano Quality Framework
  - Developed in 1978
  - Basis for ISO 9126 Standard
  - Same factors defined in 1970s still used today

Measuring Software Quality ...

- ISO 9126 Standard Quality Model
  - Proposed in 1992
  - Provides quality definition
  - Six quality factors:
    - Functionality, Reliability, Efficiency, Usability, Maintainability, Portability

Measuring Software Quality …

- Per Standard, any software quality can be defined in terms of 6 factors
- Quality factors not included:
  - Correctness, Integrity, Flexibility, Testability, Reusability, Interoperability

Achieving Four Uses of Measurement

ective Measureme

Table 8-2 Achieving the Four Uses of Measurement

Use	Questions Answered	Measurement Category	Examples of Measures/Metrics Used
Manage and Control the Process	- How much have we made? - How much is left to make?	Size	- Lines of code (LOC) - Boxes - Procedures - Units of output
	How much progress have we made?	Status	- Earned Value - Amount of scheduled work that is done - % of each activity completed
	How much effort has been expended?	Effort	Labor hours that differentiate requirements, design, implementation and test
	When will the product be completed?	Schedule	Calendar times (months, weeks) of activity completed
Manage and Control the Product	How good is the product?	Quality	- Number of defects found - Mean time to failure - Mean time to repair
	How effectively does the product perform?	Performance	- Technical performance - Measures specified by customers and management
Improve the Process	- How cost-efficient is the process? - What is the current performance?	Time and effort	- Unit costs - Time to complete
Manage the Risks	What are the risks?	Risks	Probability of exceeding constraints or not meeting requirements

Recap

- Measurements Basics
- Empirical Investigation
- Software Engineering Measurements
- Classifying Software Measures
- Measuring Software Size
- Measuring Software Quality