

1. State-of-the-Art in Software Quality Assessment

Welf Löwe

ARiSA™ - Applied Research in System Analysis

<http://www.arisa.se>

Växjö, 19 August 2005

Goals of this talk

- ⇒ Motivate Software Quality Assessment
- ⇒ Introduce State-of-the-Art and Standards in Software Quality Assessment
- ⇒ Introduce some notions to speak the same language
- ⇒ Basis for project discussions

Agenda

- ⇒ What is Software Quality?
- ⇒ Why is it worth spending time/money on it?
- ⇒ Are there standards?
- ⇒ How do we measure quality?

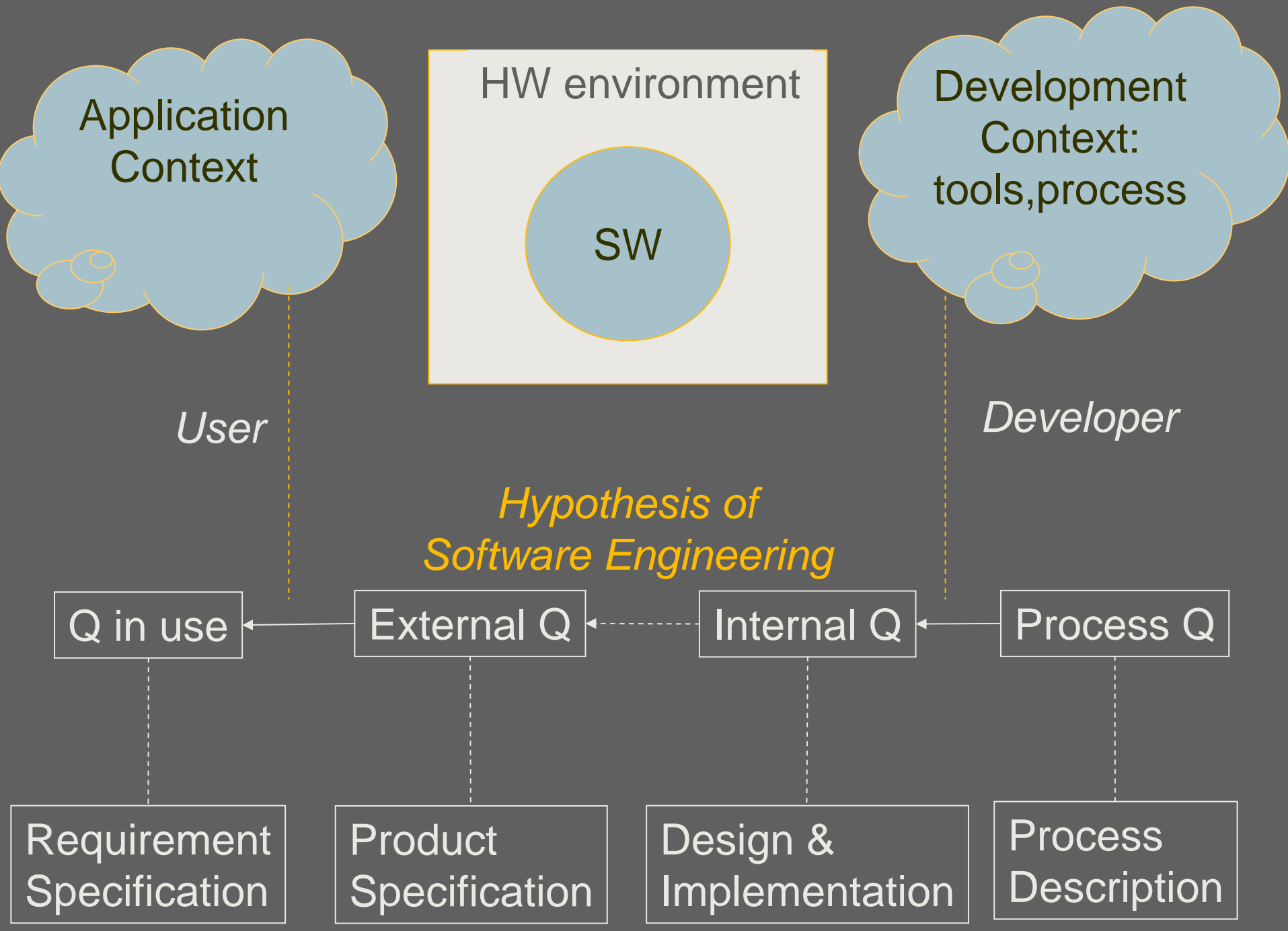
Agenda

- ⇒ What is Software Quality?
- ⇒ Why is it worth spending time/money on it?
- ⇒ Are there standards?
- ⇒ How do we measure quality

Software Quality – Definition

Software Quality is the entirety of properties and attributes of a product or a process relating to their fitness to fulfill certain requirements.

(DIN 55350/11)



Software Quality Levels

- ⇒ **Quality in use**: How well does the software, in a Hardware environment, support /enable the user to accomplish a task.
- ⇒ **External Quality**: How well does the software, in a Hardware environment, fit the product specifications (functionality, reliability, usability, efficiency)
- ⇒ **Internal Quality**: How easy is the software to understand, maintain, etc. (properties not visible to user, maintainability)
- ⇒ **Process Quality**: How well do methods, tools, ... support the construction of the software

Hypothesis of Software Engineering

Process and internal quality affects external quality and quality in use.

Our focus is the assessment of **internal quality** we can derive some limited ideas about **external quality** too.

Agenda

- ⇒ What is Software Quality?
- ⇒ Why is it worth spending time/money on it?
- ⇒ Are there standards?
- ⇒ How do we measure quality?

Costs

⇒ Assessing and improving software quality costs

⇒ Lack quality costs too

- **Money**: due to high maintenance expenses, currently 50-80% of total costs of ownership of software
- **Lives**: bad quality is a danger in safety critical applications

Ariane 5 crash

June 4, 1996

- ⇒ Maiden flight of the European Ariane 5 launcher
 - crashed about 40 seconds after takeoff
 - lost was about \$US half a billion
- ⇒ Explosion was the result of a software error
 - Uncaught exception due to floating-point error:
 - conversion from a 64-bit integer to a 16-bit signed integer applied to a larger than expected number .
 - Module was re-used without proper testing from Ariane 4
 - Error was not supposed to happen with Ariane 4
 - No exception handler

Mars Climate Orbiter

September 23, 1999

⇒ Mars Climate Orbiter

- disappeared as it began to orbit Mars.
- Cost about \$US 125 million

⇒ Failure due to error in a transfer of information between a team in Colorado and a team in California

- One team used English units (e.g., inches, feet and pounds)
- Other team used metric units for a key spacecraft operation.

Internet viruses and worms

- ⇒ Exploit well known weak points in software
 - Blaster worm (\$US 525 millions)
 - Sobig.F (\$US 500 millions – 1billions)
- ⇒ Software developers do not devote enough effort to applying lessons learned from these cases
- ⇒ Same types of vulnerabilities continue to be seen in newer versions of products that were in earlier versions.

Other Historical Examples

- ➔ 1980, the North American Aerospace Defense Command (NORAD) reported (falsely) that the U.S. was under missile attack.
- ➔ 1988 shooting down of Airbus 320 by the USS Vincennes - cryptic and misleading output displayed by tracking software
- ➔ 1991 patriot missile failure - inaccurate calculation of time due to computer arithmetic errors
- ➔ First operational launch attempt of the space shuttle, whose real-time operating software consists of about 500,000 lines of code, failed - synchronization problem among its flight-control computers.
- ➔ 9 hour breakdown of AT&T's long-distance telephone network - caused by an untested code patch
- ➔ ...

Agenda

- ⇒ What is Software Quality?
- ⇒ Why is it worth spending time/money on it?
- ⇒ Are there standards?
- ⇒ How do we measure quality?

Applicable Standards

- ⇒ ISO 9000-9004 general Quality Management
- ⇒ ISO 9126 Software Quality having 4 parts
 - ISO 9126-1: Software Quality Model
 - ISO 9126-2: External Software Quality
 - ISO 9126-3: Internal Software Quality
 - ISO 9126-4: Quality in Use
- ⇒ ISO 14598(1-6): Measurement Process

ISO 9000 Family

⇒ The ISO 9000 family is primarily concerned with "quality management". This means what the organization does to fulfill:

- the customer's **quality requirements**, and
- applicable **regulatory requirements**, while aiming to
- enhance **customer satisfaction**, and
- achieve continual **improvement** of its performance in pursuit of these objectives.

ISO 9000 Family

- ⇒ International Organization for Standardization ISO 9000 quality standard family
 - Implemented by some 634 000 organizations
 - in 152 countries.
- ⇒ Generic, i.e. applicable
 - to any organization, large or small, whatever its product
 - including whether its "product" is actually a service,
 - in any sector of activity, and
 - whether it is a business enterprise, a public administration, or a government department.

Process Maturity Models

- ⇒ Evaluation of the maturity of an organization development process
- ⇒ Provides means of measuring maturity, of complete process and/or of separate process areas
- ⇒ Examples
 - ISO 9001:2000, Quality management systems - Requirements of the ISO 9000 Family
 - Capability Maturity Model (CMM)

CMM Maturity Levels

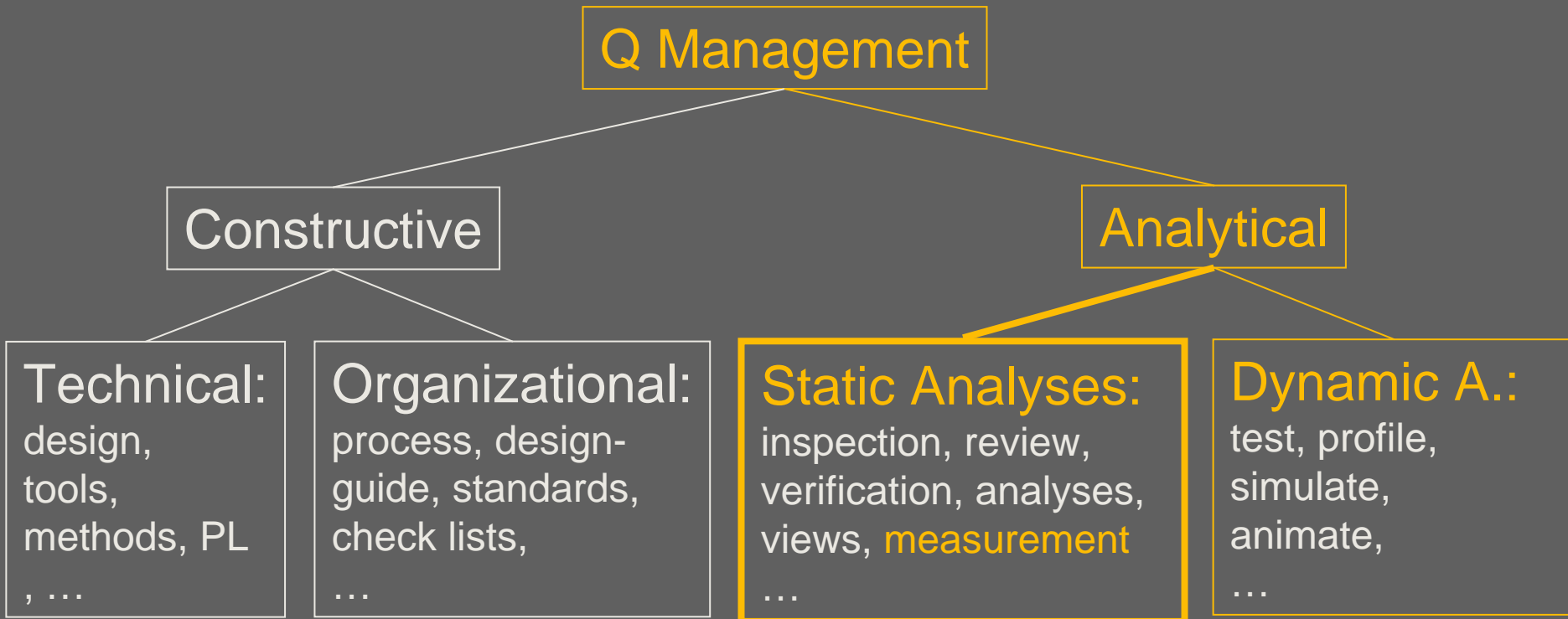
1. **Initial** – chaotic unpredictable (cost, schedule, quality)
2. **Repeatable** – intuitive; cost/quality highly variable, some control of schedule, informal/ad hoc procedures.
3. **Defined** – qualitative; reliable costs and schedules, improving but unpredictable quality performance.
4. **Managed** – quantitative reasonable statistical control over product quality.
5. **Optimizing** – quantitative basis for continuous improvement.

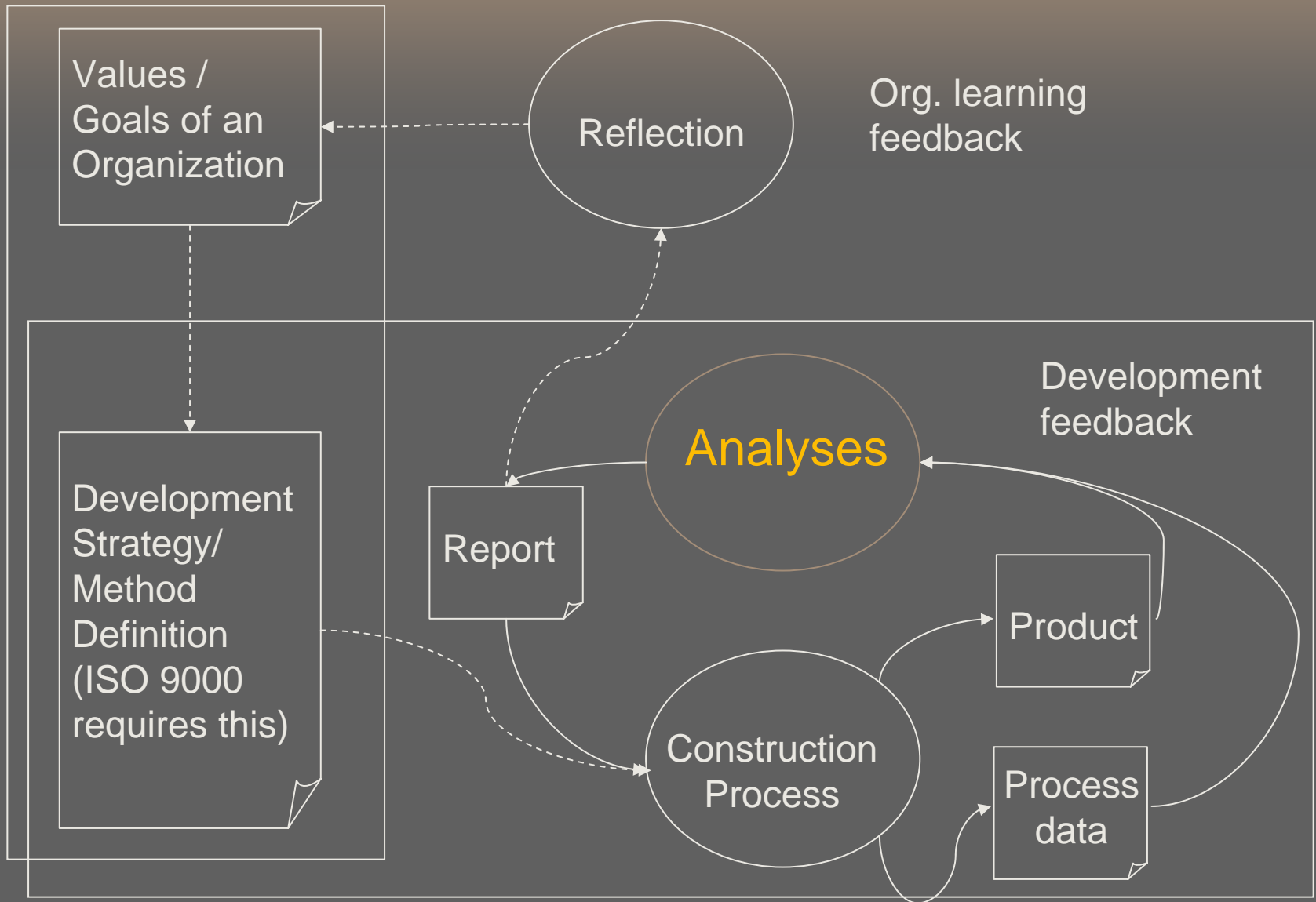
Quality Management Problem

How to get **quantitative** reasonable statistical control over **internal** (external) product quality or even ...

How to get **quantitative** basis for continuous **internal** (external) product quality **improvement**?

Aspects in Quality Management





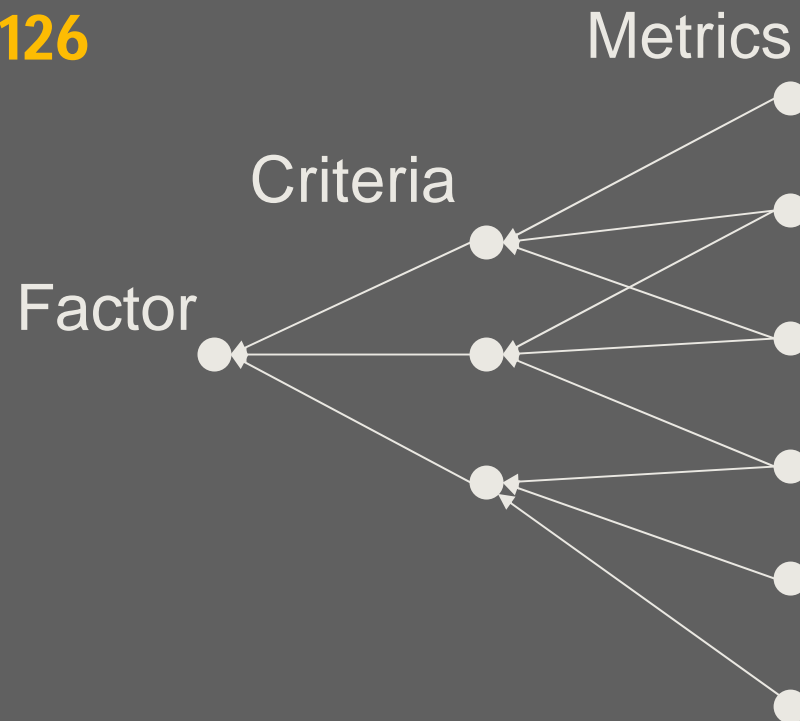
Quality Models - QM

- ⇒ “Operationalization” (bringing things to work) and “Quantification” of quality
- ⇒ Questions to discuss:
 - Subject of Quality (specific product, domain, type)
 - Quality goal(s),
 - Criteria (supporting the goal) and factors (directly measurable, supporting criteria)
 - Measurement/decision procedure for factors
- ⇒ Generic (Meta-) Models
 - Factor-Criteria-Metrics (FCM) Model, 1977
 - Goal-Question-Metrics (GQM) Model, 1984

Factor-Criteria-Metrics – FCM

McCall, Richards '77

Basis of ISO 9126



FCM based suggested QM

- ⇒ ISO 9126-1 suggests a FCM based QM having 6 factors and 31 criteria, applicable to **every kind of software**
 - functionality
 - reliability
 - usability
 - efficiency
 - maintainability
 - portability
- ⇒ ISO 9126-2 considers external metrics
- ⇒ ISO 9126-3 considers internal metrics

Standard Criteria and Factors

ISO 9126

Addressing external qualities:

- ⇒ **Functionality**: Suitability, Accuracy, Interoperability, Security, Compliance
- ⇒ **Reliability**: Maturity, Fault-tolerance, Recoverability, Compliance
- ⇒ **Usability**: Understandability, Learnability, Operability, Attractiveness, Compliance
- ⇒ **Efficiency**: Time behavior, Resource utilization, Compliance

Standard Criteria and Factors

ISO 9126

Addressing (mainly) internal qualities:

- ➡ **Re-usability**: Understandability, Learnability, Programmability in reuse context, Attractiveness, Compliance
- ➡ **Maintainability**: Analyzability, Changeability, Stability, Testability, Compliance
- ➡ **Portability**: Adaptability, (Installability), Co-existence, Conformance, Replaceable, Compliance

Generic Quality Model

- ⇒ Derived from ISO 9126 standard
- ⇒ Maps standard metrics to ISO 9126 standard criteria and factors, cf. FCM
- ⇒ Gives an overview and first impression on internal software quality,
- ⇒ Allows to compare different projects, software products

The One and Only Quality Model ?

- ⇒ Quality Models are, in general, tailored to goals of
 - Companies
 - Projects
 - Customers
 - Branch standards etc.
- ⇒ Quality Models change over time:
 - adapt to reality (Customer's request, programmers' behavior)
 - adapt to emerging standards (norms, principle rules)
 - Merger of the two

Goal-Question-Metrics - GQM

Basili, Weiss 84

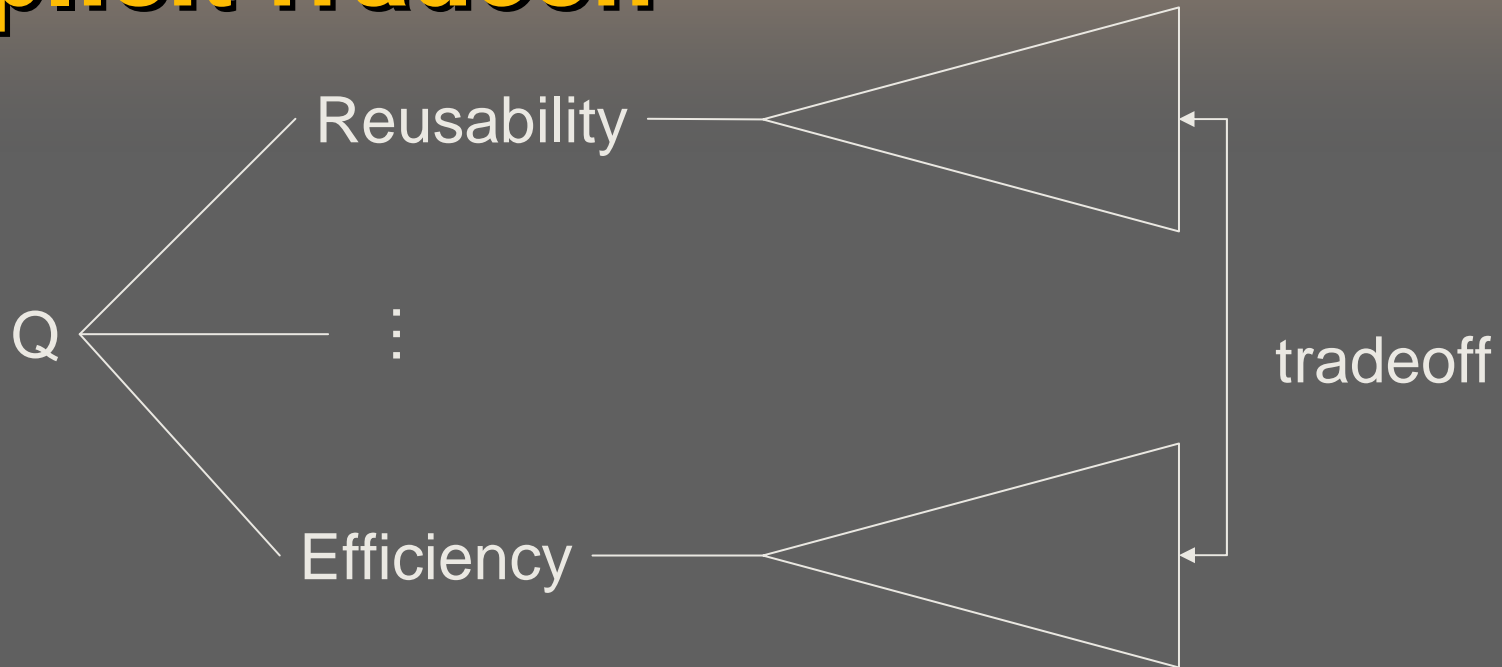
⇒ Goal

- Subject: type of document, level of detail (e.g. source code)
- Aspect (e.g. understandability)
- Objective (e.g. add functionality)
- Role (e.g. project manager)

⇒ Question(s) to clarify the current situation and required changes in the organization to reach the goal

⇒ Metrics/measurement answering each question

Explicit Tradeoff



- ⇒ Contradictions within one QM possible,
- ⇒ Almost guaranteed when integrating different QM (e.g. reusability, efficiency, security)

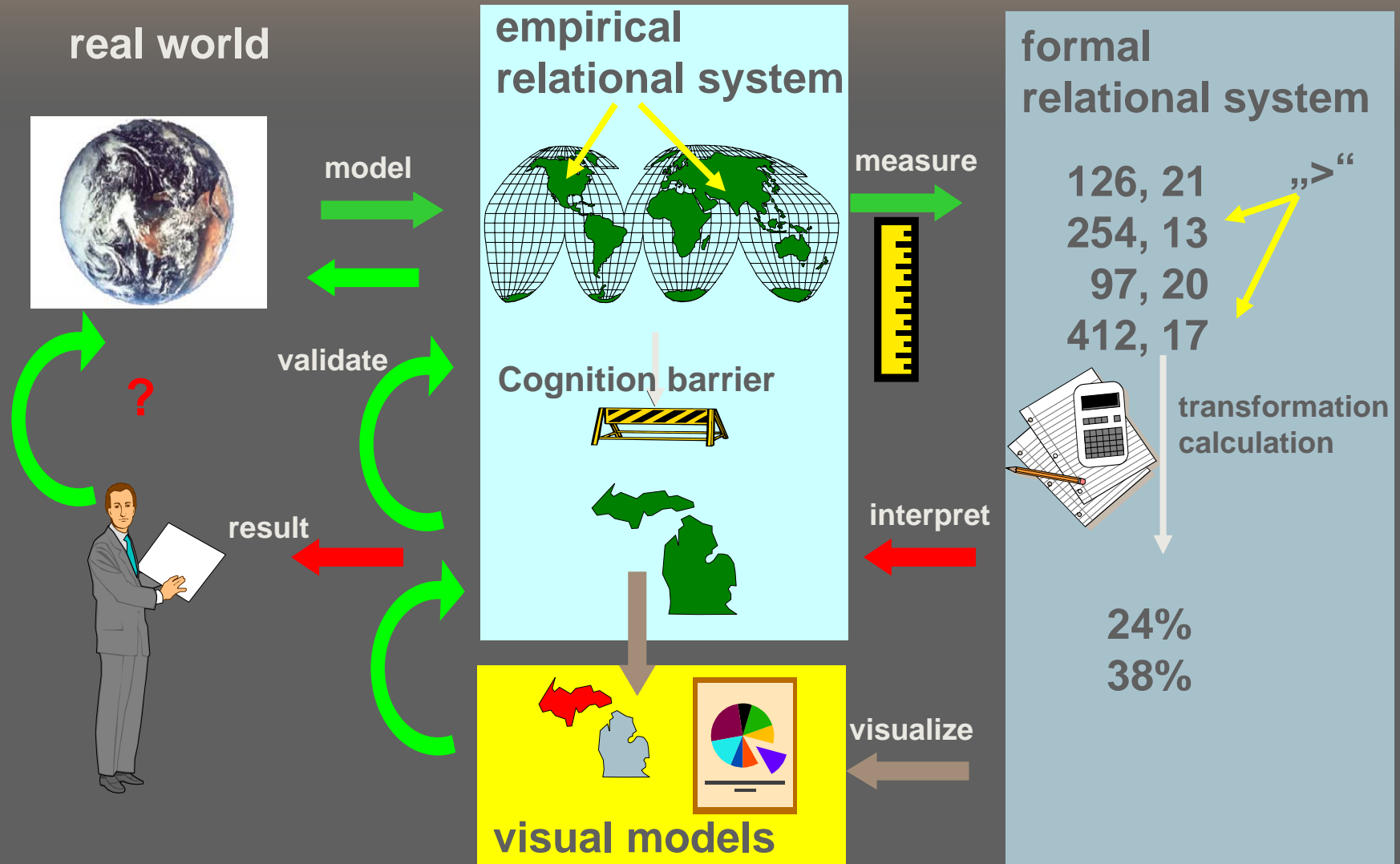
Agenda

- ⇒ What is Software Quality?
- ⇒ Why is it worth spending time/money on it?
- ⇒ Are there standards?
- ⇒ How do we measure quality?

Software Measurement

- ⇒ Software measurement: well-defined mapping of software entities to metrics
- ⇒ Software entities:
 - Code (source or binary)
 - Documentation
 - Design, architecture descriptions
 - Version control system information
 - Bug database entries
- ⇒ Used to judge, describe or predict certain properties of a software system

Measurement



Measurement in a Nutshell

1. Question to the real world
2. Build a model (that comprise suitable properties to answer the question) – empirical relational system
3. Measure relevant properties in objects – map to a formal relational system
4. Calculations/conclusion on the formal model
5. Interpretation in the empirical model and the real world
6. Visualization as a side effect
7. Validate consistency (empirical), Process must yield reasonable results

Relational Systems

- ⇒ Both empirical and formal relational systems are build to
 - Abstract from objects in the real world
 - Relate objects in the real world
- ⇒ Relations (partial orders, equivalence, i.e. classification)
- ⇒ $R = R_1, R_2, \dots R_n$ number of relations of our empirical domain, D the objects of that domain: Empirical Relational System $E(D, R)$
- ⇒ Further abstraction of both objects and relations to a Formal Relational System $F'(D', R')$

Relational Systems Properties

- ⇒ **Completeness**: all relevant objects and relations contained, to evaluate our factors, cf. **FCM**, and answer our questions, cf. **GQM**
- ⇒ **Non-ambiguity**: clear definition of the objects and relations
- ⇒ **Comprehensibility**: understandability, i.e. is it clear what the objects and relations mean?
- ⇒ **Consistency**: derive objects and relations in a formal / algorithmic / repeatable way

Empirical vs. Formal Relational Systems

- ⇒ Empirical Relational System: relations depend on the persons involved, i.e. observers
- ⇒ Formal Relational System: relations independent of the persons involved (uses measurement)

Operations on Relational Systems

- ⇒ $E(D, R, O)$ with $O = O_1, O_2, \dots, O_m$ closed (binary) operations on the domain D
- ⇒ $F(D', R', O')$ with $O' = O'_1, O'_2, \dots, O'_k$ closed (binary) operations on the domain D'
- ⇒ Must correspond to meaningful operations in the real world, i.e. have an interpretation (homomorphism relation)
- ⇒ Might be interpreted as relations as well

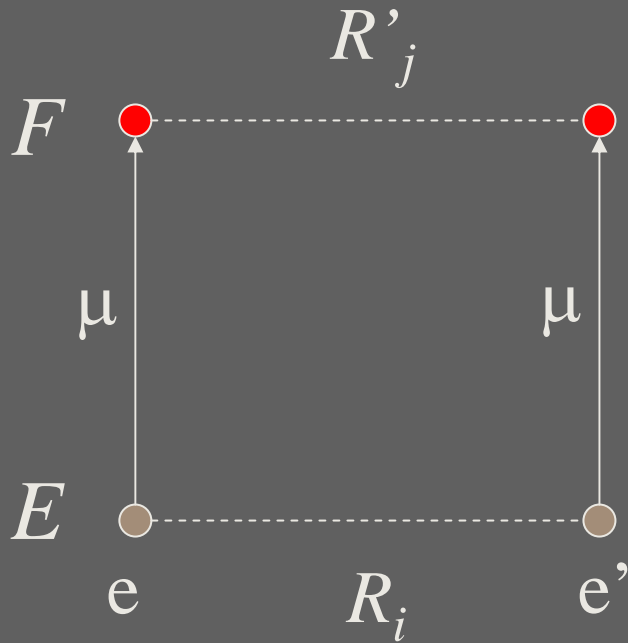
Example

- ⇒ Size of a person (comparison in the real world leads to a “taller than”), modeled by lines of certain length in our empirical relational system (comparison of lines leads to a “longer than” relation homomorphic to “taller than” in the real world)
- ⇒ An operation piling up two people in the real world corresponds to the homomorphic operation of adding length in our empirical relational system
- ⇒ Abstraction to a formal system by measuring the length of lines and assigning length values (integer objects) to them
- ⇒ Our operation is homomorphic to integer addition

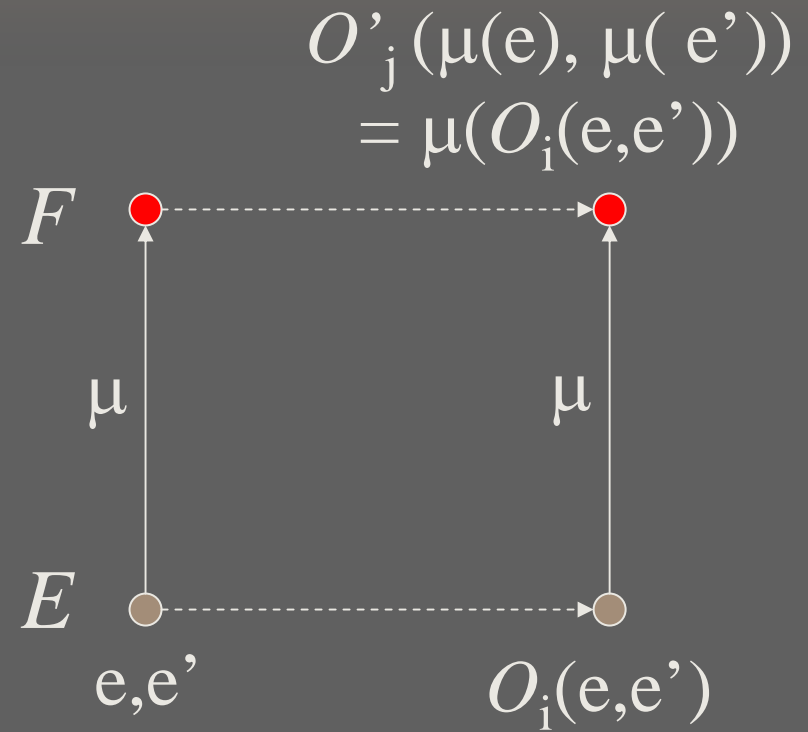
Measures

- ⇒ Measure definition $\mu: E \rightarrow F$
- ⇒ (E, F, μ) is a **scale** iff μ is a homomorphism wrt. the relations and operation in E and F , i.e. it holds
 - Representation condition, and
 - Interpretation condition
- ⇒ In practice (and standards) **measures** and **metrics** are used as synonyms

Homomorphisms



Representation condition



Interpretation condition

Scale Types

- ⇒ Calculations/conclusions are functions g transforming elements of the formal domain
 $g: \mu(D) \rightarrow D'$, often $\mu(D) \rightarrow \subset \mu(D)$
- ⇒ Not all possible calculations/conclusions g are meaningful in
 - Real world
 - Empirical model
 - Formal model
- ⇒ Scale types – **hierarchy** – define restrictions, i.e. meaningful usage of (calculations/conclusions over) measurement values
- ⇒ Calculations/conclusions have to be allowed on the underlying scale type

Scale Types

| Scale Types | Intended Conclusions | Admissible transformations | Admissible statistics |
|-------------|--|---------------------------------|----------------------------|
| Nominal | Differentiate objects (classifications) | Bijective mapping | Type (mode), frequency |
| Ordinal | Order objects | Strongly monotone functions | Percentage |
| Interval | Ratio between intervals (e.g. twice as large is pointless) | $g(x)=ax+b$ ($a \neq 0, b>0$) | Arithmetic mean, deviation |
| Rational | Ratio between values (e.g. twice as large) | $g(x)=ax$ ($a \neq 0$) | Geometric mean |
| Absolute | Values | Identity | no |

Are there standard metrics?

- ⇒ Standard metrics from ISO 9126 standard criteria
 - About 150 metrics related to factors and criteria, all require human input (i.e. are subjective, non repeatable)
 - Some “pure internal” metrics, not related
- ⇒ Well known metrics (reported in literature, on web pages, tool documentations etc.)
 - We reviewed 120 different complex metrics
 - Plus 263 simple counting metrics and statistics
 - About 50 are validated in a broad sense / applied in practice

Metrics Classes and Sub-Classes

Classified wrt. to internal properties assessed

⇒ Architecture of the system:

- Cohesion, Coupling, Inheritance

⇒ Design and coding conventions

- Documentation, Design convention, Coding convention

⇒ Complexity

- Size, Interface size, Structural complexity

Architecture - Coupling / Cohesion

⇒ Coupling:

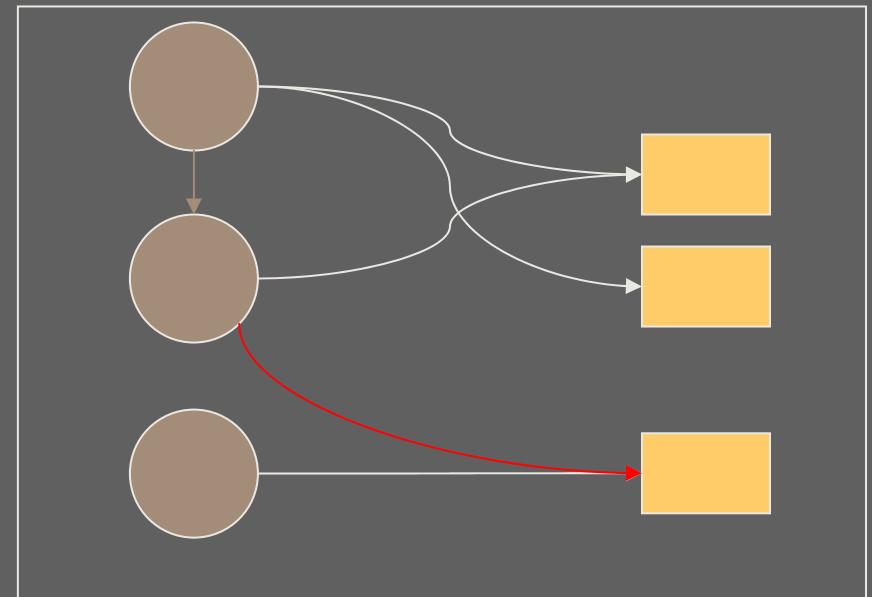
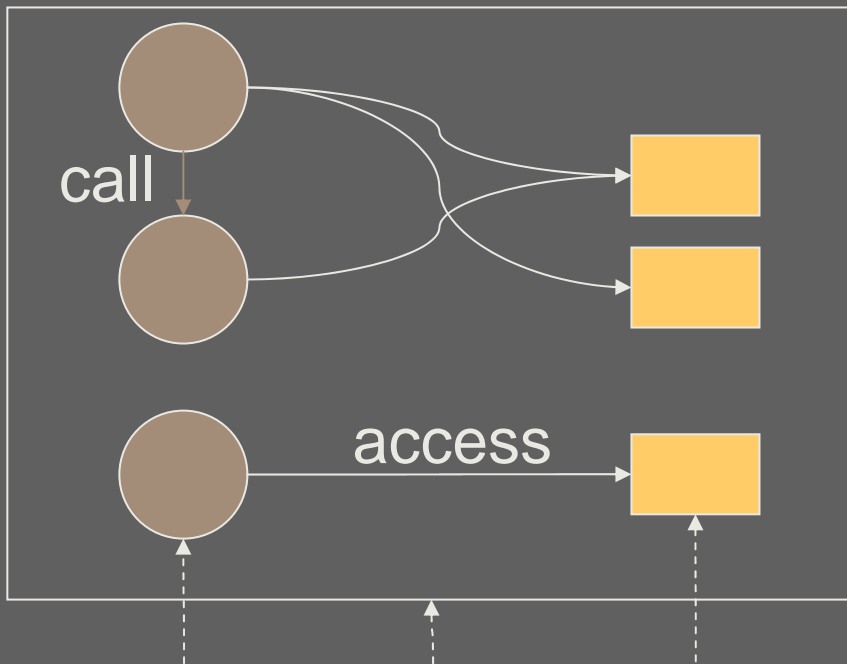
- Connectivity between software entities
- E.g. a class is coupled to another if it refers to its attribute / invokes its methods, computation based on class usage graph

⇒ Cohesion:

- Connectivity between software entities contained in the same container software entity
- E.g. connectivity of classes in a package or methods and attributes in a class

Example Coupling / Cohesion

Left module is less cohesive than the right one since it has a smaller degree of coupling



function module attributes

GQM Example: Goal and Question

⇒ **Goal:** enforce object-oriented design conventions known from software engineering theory (e.g. for companies changing from C to C++)

⇒ **Questions:**

- Right abstraction of classes and packages? No god packages and classes, one class – one concept, packages are components.
- Inheritance should be used but not at all cost?

GQM Example: Metrics

- ⇒ God classes:
 - High complexity
 - High coupling
- ⇒ One class – one concept:
 - Classes implementing more than one concept have low cohesion
 - Classes that do not implement a concept, i.e. concept is distributed over several classes, have high coupling
- ⇒ One package one component
 - Packages that are no components have high coupling and low cohesion, too
- ⇒ Inheritance usage: measured directly

What do we measure?

- ⇒ Software product metrics:
 - structure and behavior of software
- ⇒ Software process metrics:
 - Trends as derived from software product metrics over development time
 - Direct process metrics (e.g. time between two check-ins)
- ⇒ Organization:
 - Derived by process metrics
 - Direct (e.g. team size and structure, ...)

Next ...

- ⇒ Quantitative reasonable statistical control over internal product quality possible:
ARiSA – First Contact Analysis
- ⇒ Quantitative basis for continuous internal product quality improvement requires additionally
ARiSA – Quality Monitor

Back to our Problem

- ⇒ Quantitative reasonable statistical control over internal product quality possible:
 - Define quality goals / derive them from company goals
 - Define a quality model (according to FCM, GQM)
 - Perform Measurements on relevant software entities
 - Interpretation of results,
 - Feedback for construction process and organization
- ⇒ Quantitative basis for continuous internal product quality improvement requires additionally:
 - Measurements over development stages
 - Interpretation of trends

Description schema

⇒ Description:

- E.g. lines of code (LOC) counts the lines of source code of a certain software entity

⇒ Scope:

- Software entity the metric is applied to (i.e. derived from)
- E.g. LOC is applied to System, Class and Methods

⇒ Property:

- Usage of the metric, property
- E.g. LOC measures the complexity

⇒ Cross reference:

- Other related metrics
- E.g. LOC is related Number of AST nodes

Number of Methods - NOM

- ⇒ **Description**: NOM = number of methods of a
- ⇒ **Scope**: Class
- ⇒ **Property**: Complexity
- ⇒ **Cross reference**: _

Weighted Method Count - WMC

- ⇒ **Description**: WMC = sum of complexity of methods of a class according some method complexity measure
- ⇒ **Scope**: Class
- ⇒ **Property**: Complexity
- ⇒ **Cross reference**: Method complexity measures like LOC, LND, MCC, ...

Depth Of Inheritance Tree - DOT

- ⇒ **Description:** DOT = depth of inheritance tree of a
- ⇒ **Scope:** System
- ⇒ **Property:** Complexity
- ⇒ **Cross reference:** DIT

Depth In Inheritance Tree - DIT

- ⇒ **Description**: DIT = depth of a certain class in an inheritance tree of a system
- ⇒ **Scope**: Class
- ⇒ **Property**: Complexity
- ⇒ **Cross reference**: DOT

Number of Children - NOC

- ⇒ **Description**: NOC = # of children of a class in the inheritance tree
- ⇒ **Scope**: Class
- ⇒ **Property**: Complexity, Architecture
- ⇒ **Cross reference**: NOD, DIT

Number of Descendants - NOD

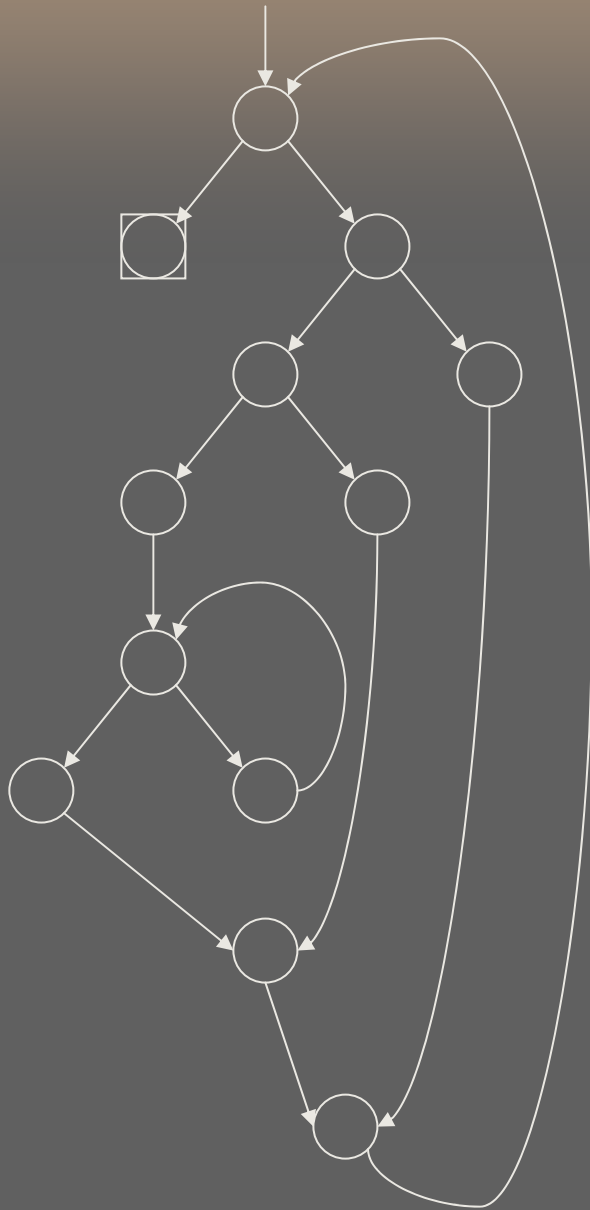
- ⇒ **Description**: NOD = transitive # of children of a class in the inheritance tree
- ⇒ **Scope**: Class
- ⇒ **Property**: Complexity, Architecture
- ⇒ **Cross reference**: NOC, DIT

Loop Nesting Depth - LND

- ⇒ **Description:** LND = nesting depth of loops in the method
 - For reducible languages (each loop has a dedicated entry point called loop head): computed as # of loop statements contained in another
 - For irreducible languages: computed as # of reductions until the interval graph of a method stabilizes or # of T1, T2 analyses
- ⇒ **Scope:** Method
- ⇒ **Property:** Complexity (control flow complexity)
- ⇒ **Cross reference:** Could be used in WMC for method's complexity

McCabe Cyclomatic Complexity - CC

- ⇒ **Description**: MCC = number of linearly independent path of a basic block graph / control flow graph (i.e. code graphs)
 - Let $B_m = (V, E)$ a basic block graph of a method m with
 - $MCC(m) \approx |E| - |V| + 2$
 - Preferred implementation, if B_m computed anyway
- ⇒ **Scope**: System, Method
- ⇒ **Property**: Complexity (control flow complexity)
- ⇒ **Cross reference**: Could be used in WMC for method's complexity



```

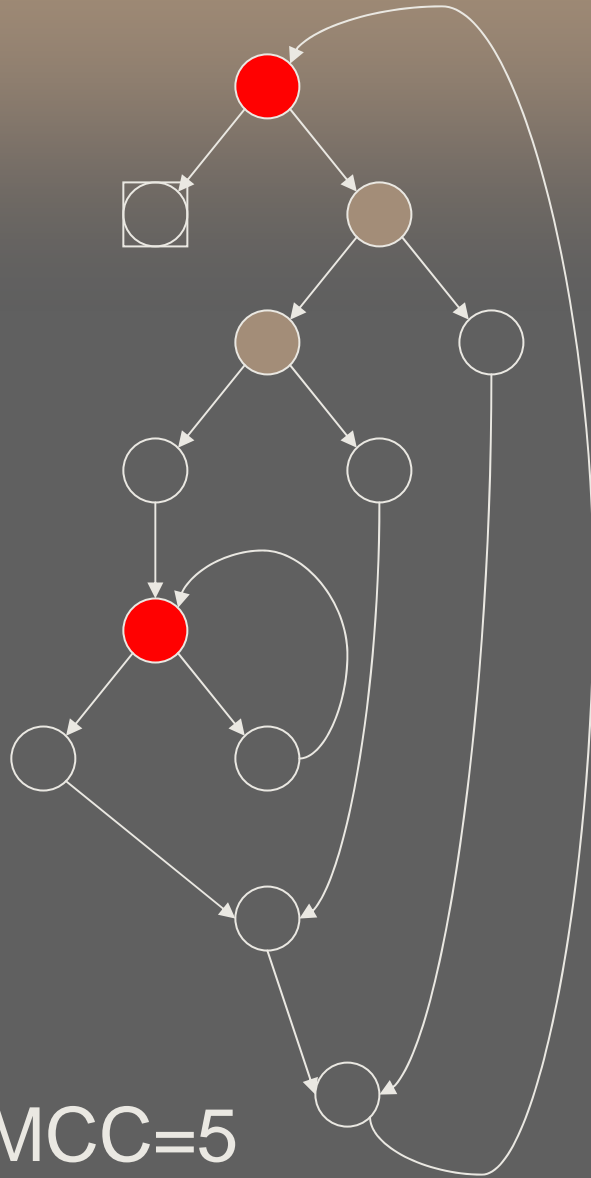
while (x<0) {
  if (x<0) then {
    if (x<0) then {
      while (x<0){
        x:=y; }
      x:=y; }
    else{
      x:=y; } }
  else{
    x:=y; } }

```

$$\Rightarrow |E| = 14$$

$$\Rightarrow |V| = 11$$

$$\Rightarrow MCC(m) = 14 - 11 + 2 = 5$$



MCC=5



MCC=14-11+2=5

Data Abstract Coupling - DAC

- ⇒ **Description**: DAC = number of complex attributes, i.e. attributes referring to other (complex, user defined) classes
- ⇒ **Scope**: Class
- ⇒ **Property**: Design (Coupling with other classes), Complexity, Architecture
- ⇒ **Cross reference**: TCC, RFC, NIV (Number of Instance Variables)

Response set For a Class - RFC

⇒ **Description:** $\text{RFC} = |\text{RS}|$ with

$$\text{RS} = M \cup \bigcup_{m \in M} R_m$$

M set of methods defined and R_m set of public methods potentially called by $m \in M$

⇒ **Scope:** Class

⇒ **Property:** Design (Coupling with other classes), Complexity, Architecture

⇒ **Cross reference:** Cf. DAC

Tight Class Cohesion - TCC

⇒ Description:

- Relative number of directly connected methods, i.e. methods sharing at least one attribute
- $I(m)$ = set of attributes m accesses
- $NDC = |\{(m_i, m_j) \mid i < j, \text{ with } (I_i \cap I_j) \neq \emptyset\}|$
i.e. # of method pairs using a common attribute and
- NPC = # of method pairs that could **possibly** use a common attribute: $|M| * (|M|-1) / 2$
- $TCC = NDC / NPC$

⇒ Scope: Class

⇒ Property: Design (Cohesion)

⇒ Cross reference: LCOM, TPC, DAC

Architecture - Coupling / Cohesion

⇒ Coupling:

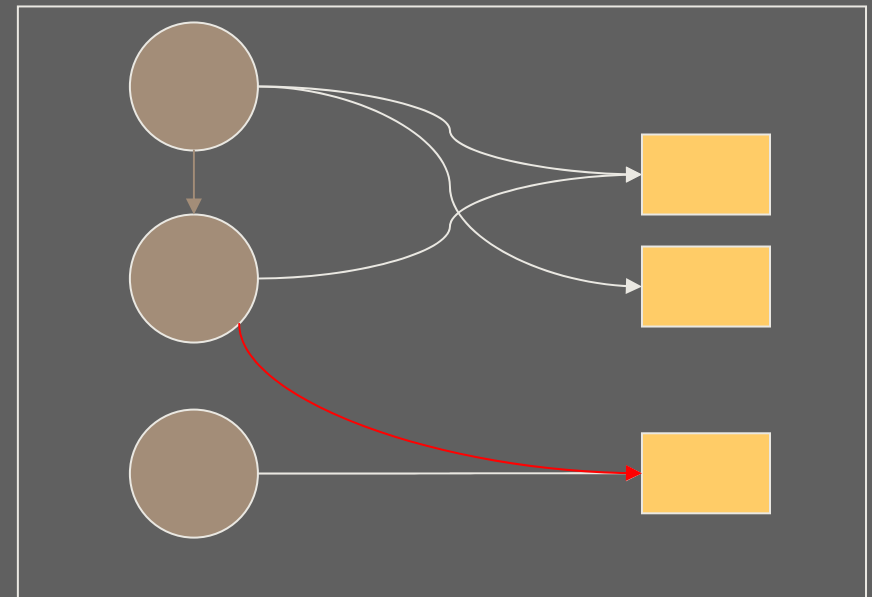
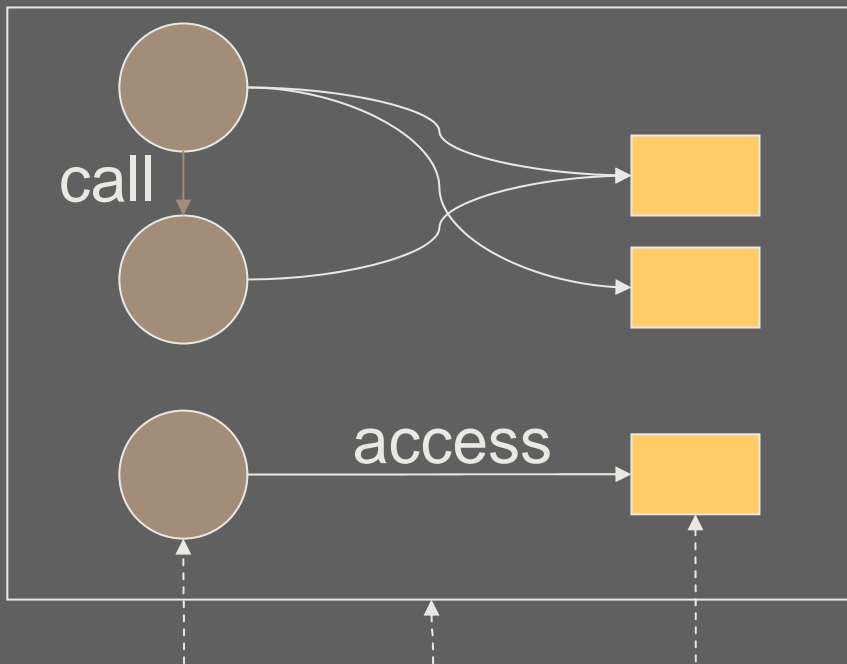
- Connectivity between software entities
- E.g. a class is coupled to another if it refers to its attribute / invokes its methods, computation based on class usage graph

⇒ Cohesion:

- Connectivity between software entities contained in the same container software entity
- E.g. connectivity of classes in a package or methods and attributes in a class

Example Coupling / Cohesion

Left module is less cohesive than the right one since it has a smaller degree of coupling



function module attributes

Lack of Cohesion On Methods - LCOM

⇒ Description:

- Modified TCC

- $LCOM(c) = \max (NNC - NDC, 0)$

- $I(m)$ = set of attributes m accesses

- $NNC = | \{ (m_i, m_j) \mid i < j, \text{ with } (I_i \cap I_j) = \emptyset \} |$

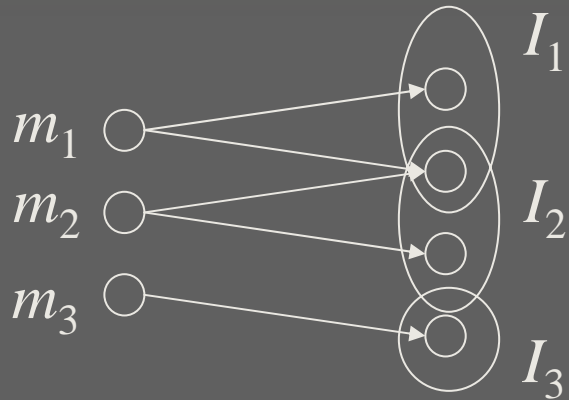
- $NDC = | \{ (m_i, m_j) \mid i < j, \text{ with } (I_i \cap I_j) \neq \emptyset \} |$

⇒ Scope: Class

⇒ Property: Cohesion

⇒ Cross reference: TCC, TPC

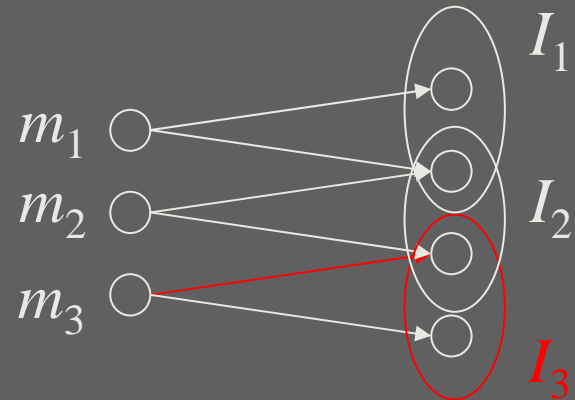
LCOM Example



$$NDC = \{(m_1, m_2)\}$$

$$NNC = \{(m_1, m_3) (m_2, m_3)\}$$

$$LOCM = \max(2-1, 0) = 1$$



$$NDC = \{(m_1, m_2) (m_2, m_3)\}$$

$$NNC = \{(m_1, m_3)\}$$

$$LOCM = \max(1-2, 0) = 0$$

Package DAC

- ⇒ **Description**: PDAC = number of complex attributes over all classes in a package referring to (complex, user defined) classes outside a package
- ⇒ **Scope**: Package
- ⇒ **Property**: Architecture (Coupling with other packages)
- ⇒ **Cross reference**: DAC, TPC

Tight Package Cohesion - TPC

⇒ Description:

- NDC = # of class pairs of a package using another and
- NPC = # of class pair of a package that could possibly use another $|C| * (|C|-1) / 2$
- $TPC = NDC / NPC$

⇒ Scope: Package

⇒ Property: Architecture (Cohesion of classes)

⇒ Cross reference: TCC, PDAC

What do we measure?

- ⇒ Software product metrics:
 - structure and behavior of SW
- ⇒ Software process metrics:
 - Derived by SW product metrics over development time
 - Direct process metrics (e.g. time between two check-ins)
- ⇒ Organization:
 - Derived by process metrics
 - Direct (e.g. team size, structure, ...)

Who are the stakeholders

- ⇒ Customer (buys the software)
- ⇒ End user (uses the software)
- ⇒ Developer (writes the software)
- ⇒ Software producers (sells the software)
- ⇒ Maintainers (maintains the software)

Classification of Measurements

- ⇒ **Internal** attributes – related to the subject itself – vs. **External** attributes – related to the subject environment
- ⇒ **Direct** measures – directly observable – vs. **Indirect/derived** measures – calculated from direct and other indirect
- ⇒ **Objective** measurement – independent of human judgment – vs. **Subjective** measurement – depends on human judgment
- ⇒ **Point-in-time** measurement vs. **Trend** measurement – relates measures of different status in the development

Monetary Impact of poor Quality

Standish group - 1995

- ⇒ 31.1% of projects canceled before completed cost \$81 billion
- ⇒ 52.7% of projects exceed their budget - costing 189% of original estimates cost \$59 billion
- ⇒ Large companies
 - 9% of software projects completed on-time and on-budget
 - On average, delivered systems have approximately only 42% of originally-proposed features and functions
- ⇒ Smaller companies
 - 16.2% of software projects completed on-time and on-budget
 - 78.4% of smaller companies projects get deployed with at least 74.2% of their original features and functions.