

Software Measurements and Metrics

Massimo Cossentino
(cossentino@pa.icar.cnr.it)

Outline

- Software metric fundamentals
- Quality models (McCall, Bohem, FURPS+)
- Measuring Software
- Software Metrics
 - Product Metrics
 - Process Metrics
 - Architecture-based Metrics
- Limits of Software Metrics

Objectives

- To explain the concept of a software metric;
- To explain how measurement may be used in assessing software quality, software development plans and the limitations of software measurement;
- To introduce some metrics and the rationale for choosing the right one

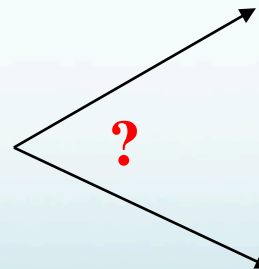
What is a software metric

- Any type of measurement which relates to a software system, process or related documentation
 - Lines of code in a program, number of person-days required to develop a component, ...
- Allows the software and the software process to be quantified.
- May be used to predict product attributes or to control the software process.
- Product metrics can be used for general predictions or to identify anomalous components.

Why measure?

- Gilb's principle of fuzzy targets:
 - Projects without clear goals will not achieve goals clearly
- Tom DeMarco
 - You can neither predict nor control what you cannot measure

Project goal:
I want to build a small car



Chevrolet Spark

Length= 359 cm
995 cm³
50 HP
152 KM/h Vmax
10K€



Lotus Elise

Length= 379 cm
1796 cm³
192 HP
241 KM/h Vmax
50K€

Metrics Domains

- Process
 - duration or effort of tasks, no. of changes in requirements
 - Resources
 - no. of staff working on a task; staff overturn
 - staff experience/skills
 - Product
 - requirements document
 - architecture document
 - design document
 - implementation (code, libraries)
- } size (n. of pages or lines of code)
complexity
functionality

Direct and Indirect Metrics

- Direct metric
 - Only one attribute or entity is involved
 - Lines of code, Number of Methods, Number of Requirements
- Indirect metric
 - Measuring an attribute by combining more than one attribute
 - Defect Density = # of defects per Lines of Code
 - Requirement Stability = # of initial requirements / # of total requirements

Internal and External Attributes

- Internal attributes
 - Can be measured in terms of only the entity itself
 - Size, reuse, time, effort, age, price, ...
- External attributes
 - Can only be measured with respect to the entity's environment
 - Comprehensibility, maintainability, reliability, usability,...
- External attributes often concern **product quality**

Quality Models

Outline

- Software metric fundamentals
- Quality models (McCall, Bohem, FURPS+)
- Measuring Software
- Software Metrics
 - Product Metrics
 - Process Metrics
 - Architecture-based Metrics
- Limits of Software Metrics



Quality Models

- Two classical and well known software quality models:
 - Boehm (1977)
 - McCall (1978)
- A more recent (and diffused) model is FURPS+ by R. Grady and D. Caswell (1987)

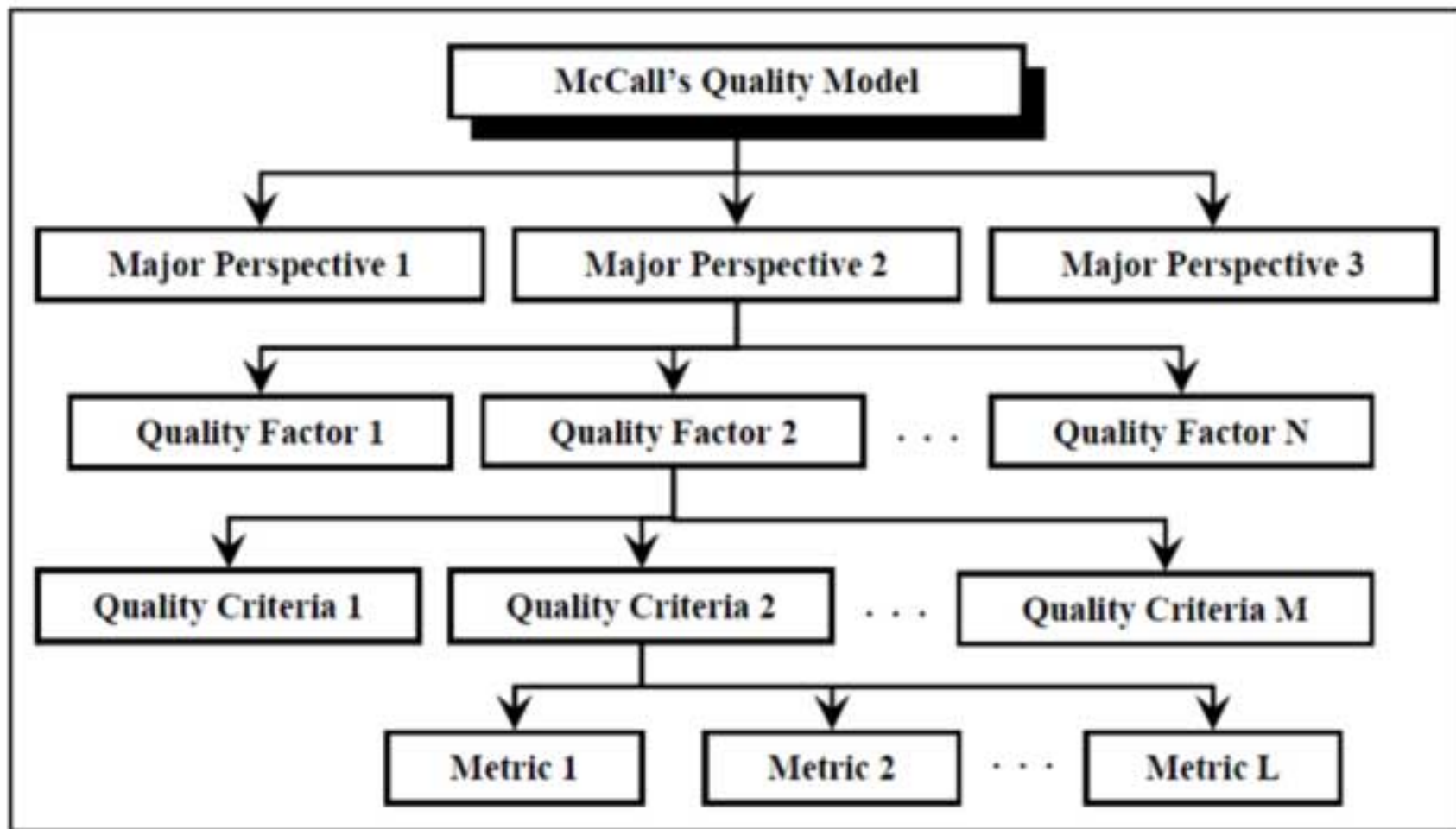
Quality Models /2

- Common objectives of these models are:
 - The benefits and costs of software are represented in their totality with no overlap between the attributes.
 - The presence, or absence, of these attributes can be measured objectively.
 - The degree to which each of these attributes is present reflects the overall quality of the software product.
 - These attribute facilitate continuous improvement, allowing cause and effect analysis which maps to these attributes, or measure of the attribute.

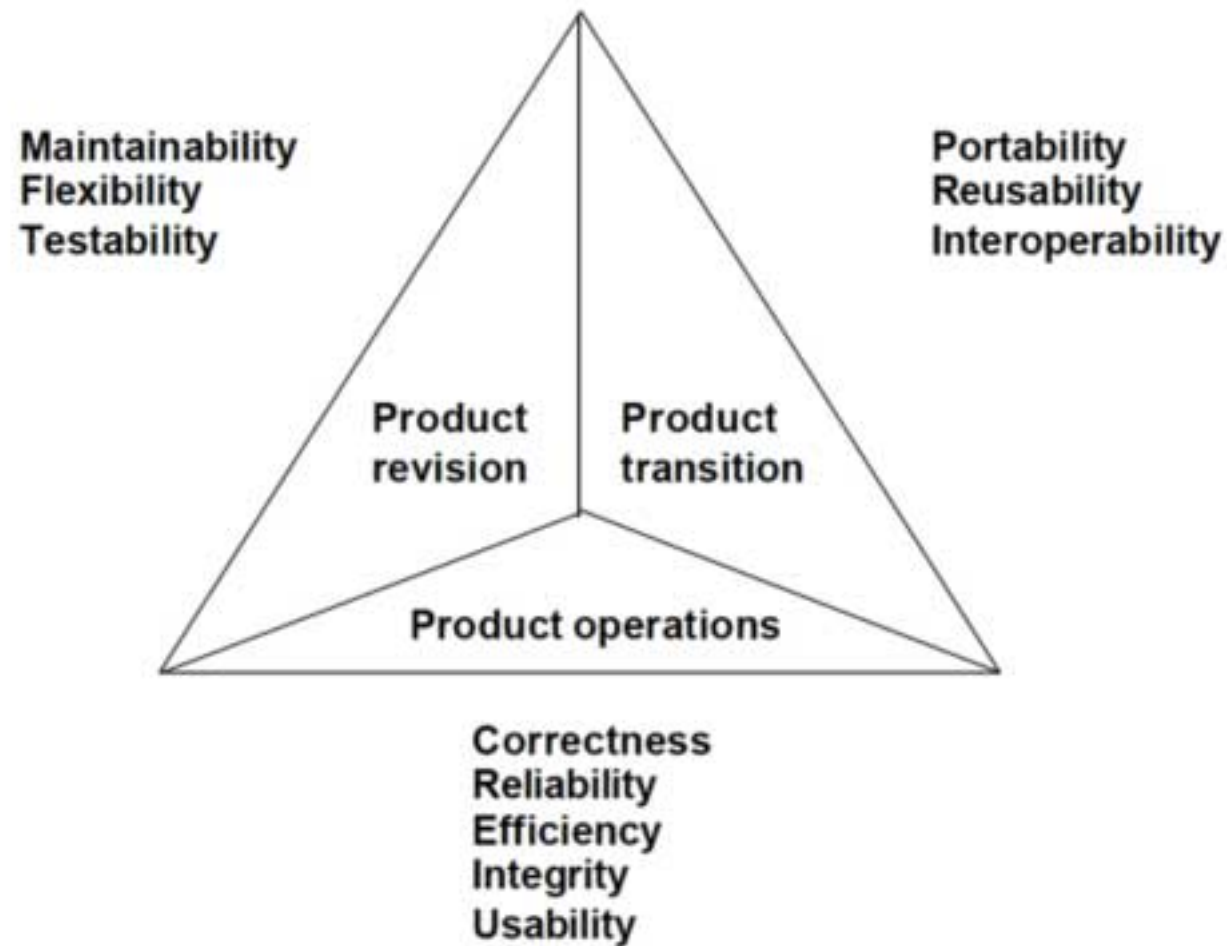
McCall's Quality Model

- Jim McCall produced this model for the US Air Force to bridge the gap between users and developers.
- McCall identified three main perspectives:
 - **Product revision** (ability to change).
 - **Product transition** (adaptability to new environments).
 - **Product operations** (basic operational characteristics).

McCall's Quality Model



McCall's Quality Factors



Quality Factors in McCall perspectives (1/2)

- **Product revision** (quality factors that influence the ability to change the software product):
 - Maintainability, the ability to find and fix a defect.
 - Flexibility, the ability to make changes required as dictated by the business.
 - Testability, the ability to Validate the software requirements.
- **Product transition** (quality factors that influence the ability to adapt the software to new environments):
 - Portability, the ability to transfer the software from one environment to another.
 - Reusability, the ease of using existing software components in a different context.
 - Interoperability, the extent, or ease, to which software components work together.

Quality Factors in McCall perspectives (2/2)

- **Product operations** (quality factors that influence the extent to which the software fulfils its specification):
 - Correctness, the functionality matches the specification.
 - Reliability, the extent to which the system fails.
 - Efficiency, system resource (including cpu, disk, memory, network) usage.
 - Integrity, protection from unauthorized access.
 - Usability, ease of use.

Boehm's Quality Model (1/3)

- A hierarchical model of software quality characteristics
 - Characteristics qualitatively define software quality as a set of attributes and metrics (measurements).
- At the highest level of his model, Boehm defined three **primary uses**:
 - **As-is utility**, the extent to which the as-is software can be used (i.e. ease of use, reliability and efficiency).
 - **Maintainability**, ease of identifying what needs to be changed, the ease of modification and retesting.
 - **Portability**, ease of changing software to accommodate a new environment.

Boehm's Quality Model (2/3)

- The three primary uses had **quality factors** associated with them, representing the next level of Boehm's hierarchical model.
- Boehm identified seven quality factors, namely:
 - **Portability**, the extent to which the software will work under different computer configurations (i.e. operating systems, databases etc.).
 - **Reliability**, the extent to which the software performs as required, i.e. the absence of defects.
 - **Efficiency**, optimum use of system resources during correct execution.
 - **Usability**, ease of use.
 - **Testability**, ease of validation, that the software meets the requirements.
 - **Understandability**, the extent to which the software is easily comprehended with regard to purpose and structure.
 - **Flexibility**, the ease of changing the software to meet revised requirements.

Boehm's Quality Model (3/3)

- These quality factors are further broken down into **Primitive constructs** that can be measured, for example:
 - Testability is broken down into:
 - accessibility,
 - communicativeness,
 - structure,
 - self descriptiveness.

The intention is to be able to measure the lowest level of the model.

FURPS+ MODEL

- The FURPS quality model has been developed by Grady and Caswel in Hewlett-Packard for classifying software quality attributes (both functional and non-functional).
- HP used this model for evaluating customer satisfaction
- FURPS is an acronym built by considering the initials of the following categories of software attributes:
 - **Functionality, Usability, Reliability, Performance, Supportability**
- FURPS+ is now widely used in the software industry and adopted in the Unified Process for non functional requirements.
- The + was later added to the model after various campaigns at HP to extend the acronym to emphasize various attributes.

Software requirements in FURPS

- The FURPS model addresses the following requirements within the categories used for the acronym:
 - **Functionality** - features, capabilities, security.
 - **Usability** - human factors, help, documentation.
 - **Reliability** - frequency of failure, recoverability, predictability.
 - **Performance** - response times, throughput, accuracy, availability, resource usage.
 - **Supportability** - adaptability, maintainability, internationalization, configurability.

The “+” in FURPS+

- The “+” in FURPS+ indicates other requirements not specifically included in previous categories, such as:
 - **Implementation**—resource limitations, languages and tools, hardware, ...
 - **Interface**—constraints imposed by interfacing with external systems.
 - **Operations**—system management in its operational setting.
 - **Packaging**
 - **Legal**—licensing and so forth.

FURPS MODEL

- **Functionality:** Capability (Size & Generality of Feature Set), Reusability (Compatibility, Interoperability, Portability), Security (Safety & Exploitability)
- **Usability:** Human Factors, Aesthetics, Consistency, Documentation, Responsiveness
- **Reliability:** Availability (Failure Frequency), Robustness/Durability/Resilience, Failure Extent & Time-Length (Recoverability/Survivability), Predictability (Stability), Accuracy (Frequency/Severity of Error)
- **Performance:** Speed, Efficiency, Resource Consumption (power, ram, cache, etc.), Throughput, Capacity, Scalability
- **Supportability:** Serviceability, Maintainability, Sustainability, Repair Speed, Testability, Flexibility (Modifiability, Configurability, Adaptability, Extensibility, Modularity), Installability, Localizability

Using the FURPS+ Model

- The FURPS+ Model may have different uses during the development process, for instance:
 - To elicit non functional requirements
 - To evaluate software qualities (e.g. completeness)
 - To evaluate customer satisfaction

Eliciting non-functional requirements with FURPS

- There are very few methods for eliciting non functional requirements. FURPS may be used for that by adopting check lists of questions


Category	Example questions
Usability	<ul style="list-style-type: none">What is the level of expertise of the user?What user interface standards are familiar to the user?What documentation should be provided to the user?
Reliability <i>(including robustness, safety, and security)</i>	<ul style="list-style-type: none">How reliable, available, and robust should the system be?Is restarting the system acceptable in the event of a failure?How much data can the system lose?How should the system handle exceptions?Are there safety requirements of the system?Are there security requirements of the system?
Performance	<ul style="list-style-type: none">How responsive should the system be?Are any user tasks time critical?How many concurrent users should it support?How large is a typical data store for comparable systems?What is the worst latency that is acceptable to users?
Supportability <i>(including maintainability and portability)</i>	<ul style="list-style-type: none">What are the foreseen extensions to the system?Who maintains the system?Are there plans to port the system to different software or hardware environments?

Evaluating Customer Satisfaction

- Customer satisfaction was evaluated in HP by considering FURPS attributes
- Many approaches may be used for that, most of them adopt a percentage evaluation of satisfaction:
 1. Percent of completely satisfied customers
 2. Percent of satisfied customers (satisfied and completely satisfied)
 3. Percent of dissatisfied customers (dissatisfied and completely dissatisfied)
 4. Percent of nonsatisfied (neutral, dissatisfied, and completely dissatisfied)

Measuring Software

Outline

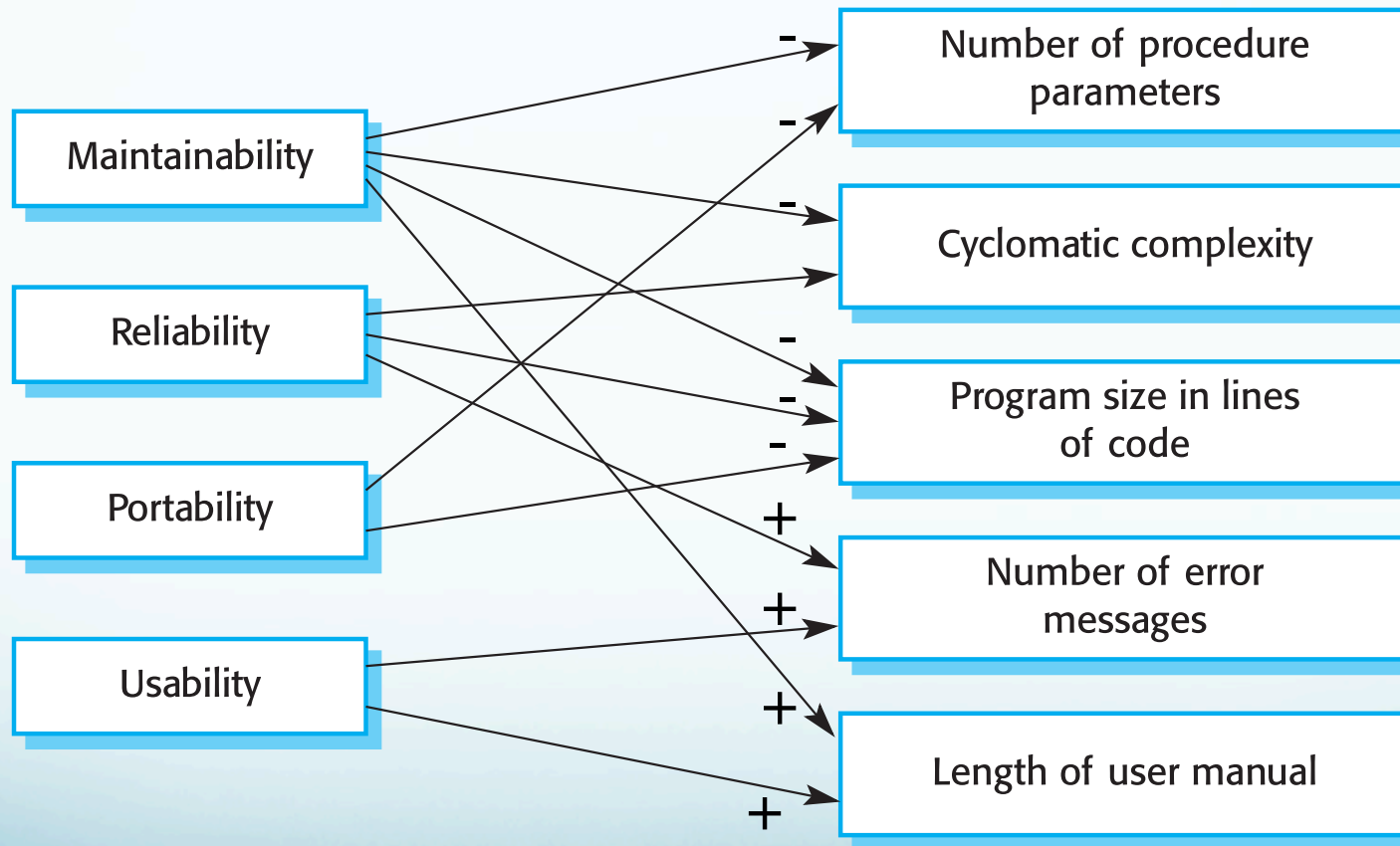
- Software metric fundamentals
- Quality models (McCall, Bohem, FURPS+)
- Measuring Software 
- Software Metrics
 - Product Metrics
 - Process Metrics
 - Architecture-based Metrics
- Limits of Software Metrics

SW Metrics

Assumptions and Considerations

- A software property can be measured.
- A relationship exists between what we can measure and what we want to know.
- We can only measure internal attributes but we are often more interested in external software attributes.
- It may be difficult to relate what can be measured to desirable external quality attributes.

Internal and external attributes of software



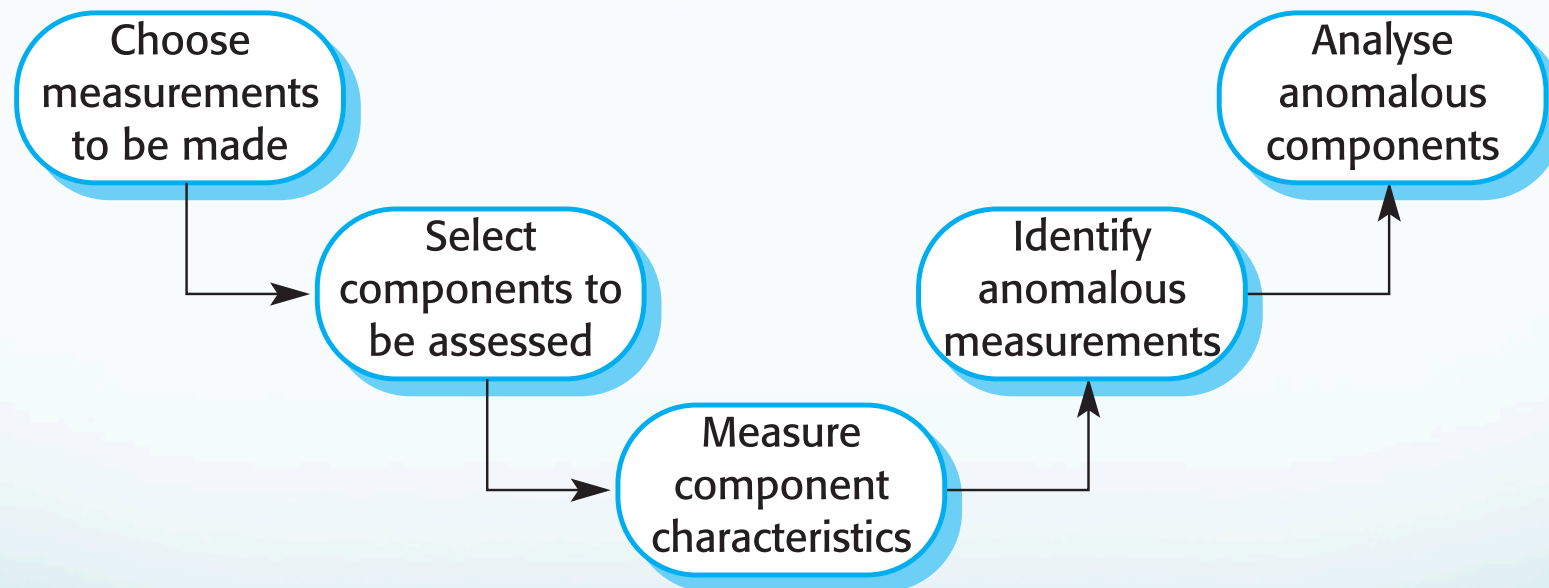
External

Internal

The measurement process

- A software measurement process is often part of a quality control process.
- Data collected during this process should be maintained as an organisational resource.
 - Quick obsolescence may be a problem
- Once a measurement database has been established, **comparisons across projects** become possible.

Product measurement process



Data collection

- Data should be collected immediately (not in retrospect) and, if possible, automatically.
- Three types of automatic data collection
 - Static product analysis;
 - Dynamic product analysis;
 - Process data collation.

Data accuracy

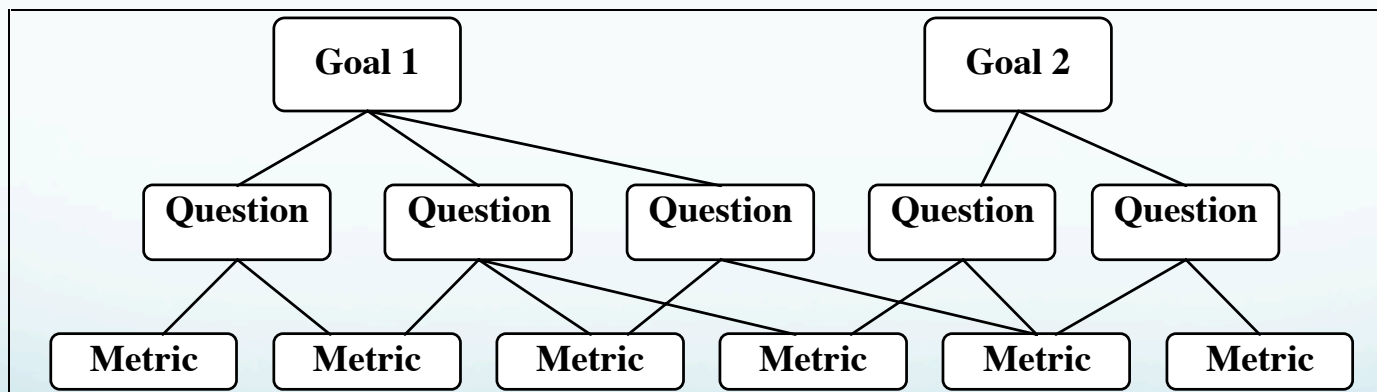
- **Don't collect unnecessary data**
 - **The questions to be answered should be decided in advance and the required data identified.**
- Tell people why the data is being collected.
 - It should not be part of personnel evaluation.
- Don't rely on memory
 - Collect data when it is generated not after a project has finished.

GQM-Approach

- Goal – Question – Metric (Basili)
- Approach to select metrics
 - Avoids “let’s collect a lot of data and decide afterwards what we do with the values”
- GQM briefly is composed of:
 1. Express goals of data collection
 2. Derive from each goal the questions that must be answered to determine if goals are achieved
 3. Analyze questions and define metrics
 4. Design and test data collection forms
 5. Collect and validate data
 6. Analyze data

The GQM approach

- Conceptual Level → Goal
- Operational Level → Question
- Quantitative Level → Metric

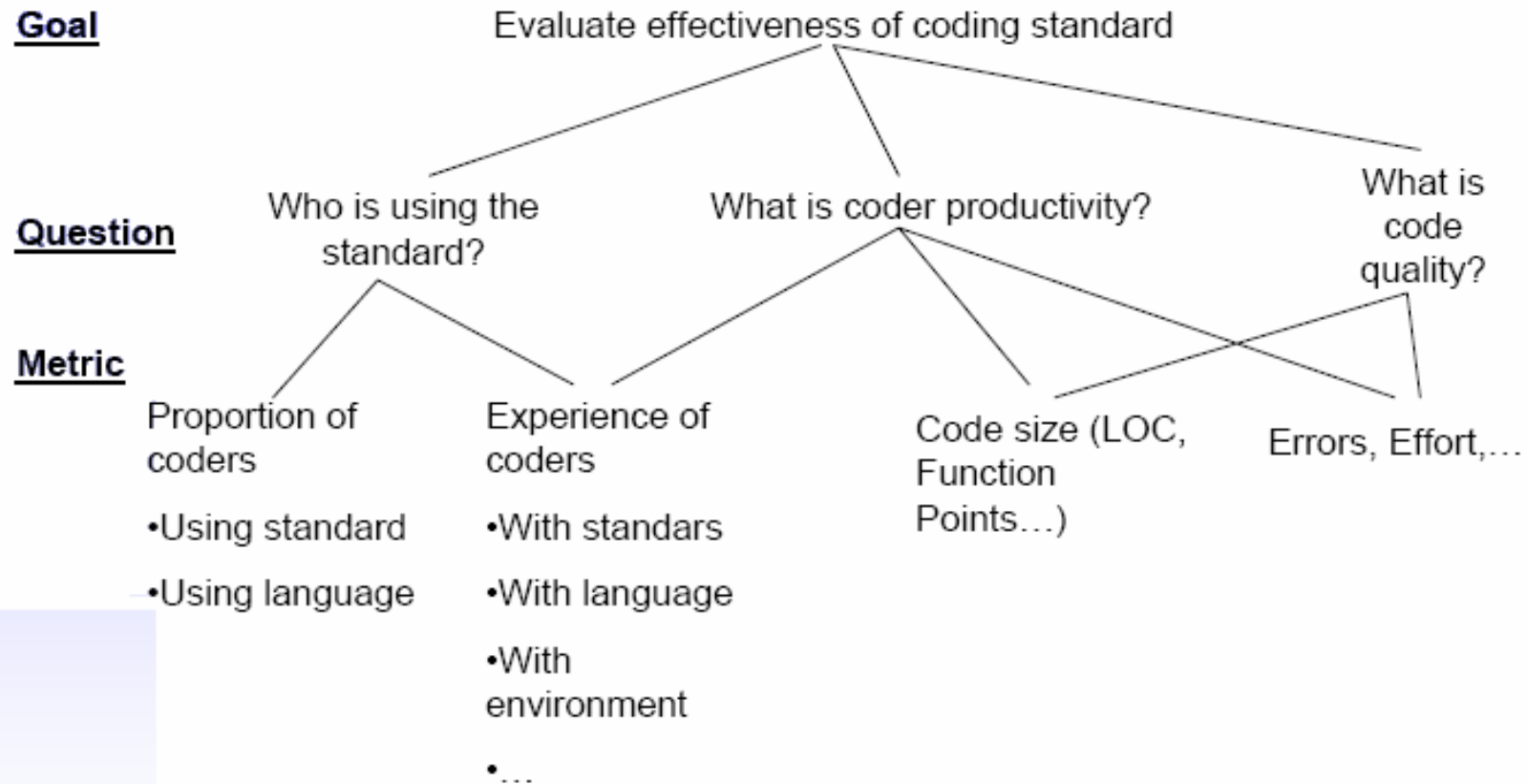


GQM MEASUREMENT GOALS DEFINITION

- **Measurement goals** should be formally defined and well structured on the basis of pursued **improvement goals**
- Next table can be useful in defining measurement goals

What?	the object under measurement
Why?	understanding, controlling, or improving the object
What aspect?	the quality focus of the object that measurement focuses on
Who?	the people that measures the object
Context	the environment in which measurement takes place

GQM: Example



GQM Example 2

Goal	Purpose Issue Object (process) Viewpoint	Improve the timeliness of change request processing from the project manager's viewpoint
Question		What is the current change request processing speed?
Metrics		Average cycle time Standard deviation % cases outside of the upper limit
Question		Is the performance of the process improving?
Metrics		$\frac{\text{Current average cycle time}}{\text{Baseline average cycle time}} * 100$ Subjective rating of manager's satisfaction

GQM QUESTIONS AND HYPOTHESES

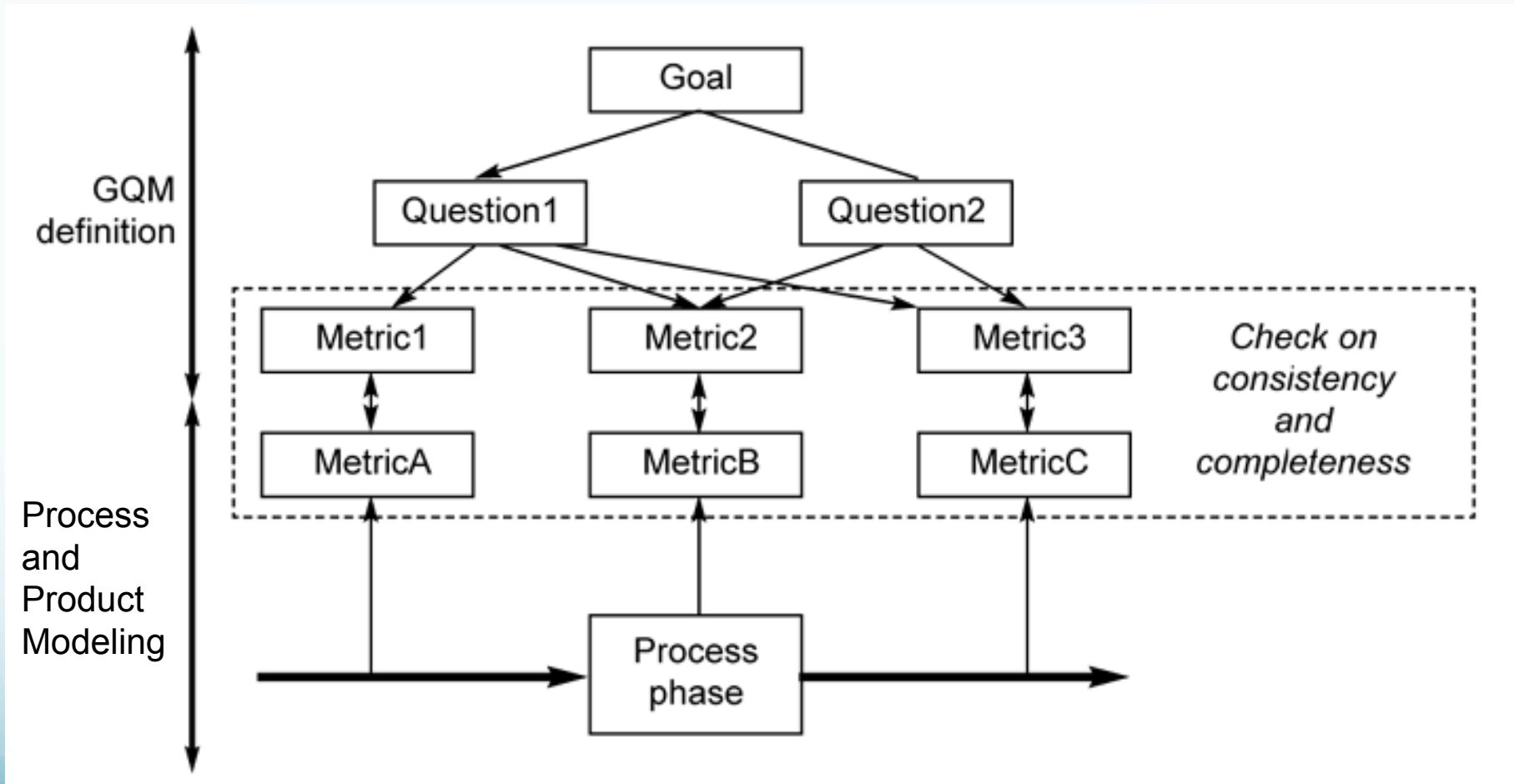
- The main idea in this phase is gaining operational definitions, i.e. **a question is a goal refined to the operational level.**
- Hypotheses are expected answers, and they are going to be examined during the measurement.

GQM Metrics Definition

- The next step is finding a way to provide all quantitative information necessary for answering the questions.
- **Questions are refined into** quantitative processes and/or **product measurements**.
- To make sure that no mistakes were made so far, completeness and consistency check should be performed with respect to the predefined models.

->

GQM Metrics Definition



Software Metrics

Product Metrics

Outline

- Software metric fundamentals
- Quality models (McCall, Bohem, FURPS+)
- Measuring Software
- Software Metrics
 - Product Metrics
 - Process Metrics
 - Architecture-based Metrics
- Limits of Software Metrics



Product metrics

- Classes of product metric
 - **Dynamic metrics** which are collected by measurements made of a program in execution;
 - help assess efficiency and reliability;
 - **Static metrics** which are collected by measurements made of the system representations;
 - help assess complexity, understandability and maintainability.

Dynamic and static metrics

- Dynamic metrics are closely related to software quality attributes
 - It is relatively easy to measure the response time of a system (performance attribute) or the number of failures (reliability attribute).
- Static metrics have an indirect relationship with quality attributes
 - You need to try and derive a relationship between these metrics and properties such as complexity, understandability and maintainability.

Some Software product metrics

Software Metric	Description
Fan in/Fan-out	<p>Fan-in is a measure of the <i>number of functions or methods that call some other function or method</i> (say X).</p> <p>Fan-out is <i>the number of functions that are called by function X</i>.</p> <p>A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extensive knock-on effects. A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the control logic needed to coordinate the called components.</p>
Length of code	<p>This is a measure of the <i>size of a program</i>. Generally, the larger the size of the code of a component, the more complex and error-prone that component is likely to be. Length of code has been shown to be one of the most reliable metrics for predicting error-proneness in components.</p>
Cyclomatic complexity	<p>This is a measure of the <i>control complexity</i> of a program. This control complexity may be related to program understandability.</p>

Some Software product metrics/2

Software Metric	Description
Length of identifiers	This is a measure of <i>the average length of distinct identifiers in a program</i> . The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.
Depth of conditional nesting	This is a measure of the <i>depth of nesting of if-statements</i> in a program. Deeply nested if statements are hard to understand and are potentially error-prone.
Fog index	This is a measure of the <i>average length of words and sentences in documents</i> . The higher the value for the Fog index, the more difficult the document is to understand.

Measurement analysis

- It is not always obvious what data means
 - Analysing collected data is very difficult.
- Professional statisticians should be consulted if available.
- Data analysis must take local circumstances into account.

Measurement surprises

- Reducing the number of faults in a program leads to an increased number of help desk calls
 - The program is now thought of as more reliable and so has a wider more diverse market. The percentage of users who call the help desk may have decreased but the total may increase;
 - A more reliable system is used in a different way from a system where users work around the faults. This leads to more help desk calls.

Some Product Metrics

Size and Complexity Metrics (1/2)

Lines of Code (LOC)

- How to deal with...
 - Empty lines?
 - Comment?
 - Multiple statements in one line?
- Counting method must be stated explicitly
- Variation of LOC for equal program in different languages
- Productivity = LOC / hour
 - Wrong incentive: verbose programming style

Size and Complexity Metrics (2/2)

- Cyclomatic Complexity.
 - It is measured by calculating the McCabe's cyclomatic number of a module
 - Measures complexity of a module

G is a control flow graph

e edges and n nodes

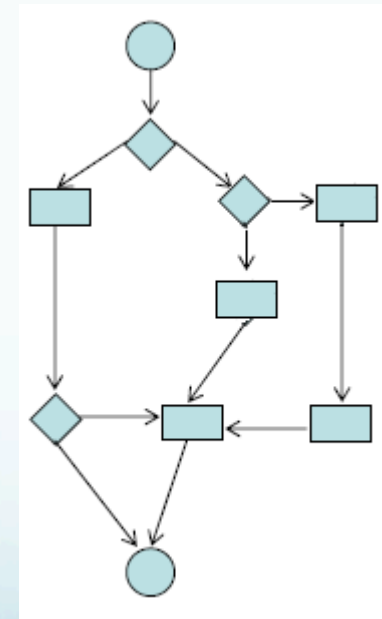
$$V(G) = e - n + 2$$

(number of linearly independent paths in G)

- (example on the right):
 $V(G) = 12 - 10 + 2 = 4$
More simply, d is number of decision nodes

$$V(G) = d + 1$$

- Heuristic: should be $V(G) < 10$
- It may be useful to consider the Total, Average (per method) and Maximum Cyclomatic Complexity



Metric (Lorenz,1993)*	Rules of Thumb and Comments
1. Average Method Size (LOC)	Should be less than 24 LOC for C++
2. Average Number of Methods per Class	Should be less than 20. Bigger averages indicate too much responsibility in too few classes
3. Average Number of Instance Variables per Class	Should be less than 6. More instance variables indicate that one class is doing more than it should.
4. Class Hierarchy Nesting Level (Depth of Inheritance Tree, DIT)	Should be less than 6, starting from the framework classes or the root class.

* From: Lorenz, M., Object-Oriented Software Development: A Practical Guide, Englewood Cliffs, N.J.: PTR Prentice Hall, 1993.

Metric (Lorenz,1993)	Rules of Thumb and Comments
Average Number of Comment Lines (per Method)	Should be greater than 1.
Number of Problem Reports per Class	Should be low (no specifics provided).
Number of Times Class Is Reused	If a class is not being reused in different applications (especially an abstract class), it might need to be redesigned.
Number of Classes and Methods Thrown Away	Should occur at a steady rate throughout most of the development process.

Object-Oriented Metrics

Metric (Sommerville, 2006*)	Rules of Thumb and Comments
Method fan-in/fan-out	This is directly related to fan-in and fan-out as described above and means essentially the same thing. However, it may be appropriate to <u>make a distinction between calls from other methods within the object and calls from external methods.</u>
Weighted methods per class (WMC)	This is the number of methods that are included in a class weighted by the complexity of each method. The larger the value for this metric, the more complex the object class. Complex objects are more likely to be more difficult to understand.
Number of overriding operations	This is the number of operations in a super-class that are over-ridden in a sub-class. A high value for this metric indicates that the super-class used may not be an appropriate parent for the sub-class.

* From: I. Sommerville. Software Engineering. 7th Edition. Addison Wesley. 2004.

Object-Oriented Metrics

(from other authors)

Metric	Rules of Thumb and Comments
Unweighted Class Size (UWCS)	<p>This is calculated from the number of methods plus the number of attributes of a class.</p> <p>Smaller class sizes usually indicate a better designed system reflecting better distributed responsibilities.</p>
Lack of cohesion of methods (LCOM)	<p>It measures the correlation between the methods and the local instance variables of a class.</p> <p>It is calculated as the ratio of methods in a class that do not access a specific data field, averaged over all data fields in the class.</p> <p>High cohesion indicates good class subdivision. Lack of cohesion or low cohesion increases complexity. Classes with low cohesion could probably be subdivided into two or more subclasses with increased cohesion.</p>
Number of Children (NOC)	<p>It relates to the class as a node of the inheritance tree. NOC is the number of immediate successors of the class</p>

Object-Oriented Metrics

(from other authors)

Metric	Rules of Thumb and Comments
Coupling Between Object classes (CBO)	It is the number of other classes to which the class is coupled
Response For Class (RFC)	It measures the complexity of the class in terms of method calls. It is calculated by adding the number of methods in the class (not including inherited methods) plus the number of distinct method calls made by the methods in the class (each method call is counted only once even if it is called from different methods).

Process Metrics

Outline

- Software metric fundamentals
- Quality models (McCall, Bohem, FURPS+)
- Measuring Software
- Software Metrics
 - Product Metrics
 - Process Metrics
 - Architecture-based Metrics
- Limits of Software Metrics



The software process

- A structured set of activities required to develop a software system
 - Specification;
 - Design;
 - Validation;
 - Evolution.
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

Process measurement

- Wherever possible, quantitative process data should be collected
 - Where organisations do not have clearly defined process standards this is very difficult as you don't know what to measure.
 - A process may have to be defined before any measurement is possible.
- Process measurements should be used to assess process improvements
 - Measurements should NOT drive the improvements → Organizational objectives should drive the improvement.

Classes of process measurement

- Time taken for process activities to be completed
 - E.g. Calendar time or effort to complete an activity or process.
- Resources required for processes or activities
 - E.g. Total effort in person-days.
- Number of occurrences of a particular event
 - E.g. Number of defects discovered.

Outline

- Software metric fundamentals
- Quality models (McCall, Bohem, FURPS+)
- Measuring Software
- Software Metrics
 - Product Metrics
 - Process Metrics
 - Architecture-based Metrics
- Limits of Software Metrics



Architecture Metrics

Architecture Metrics

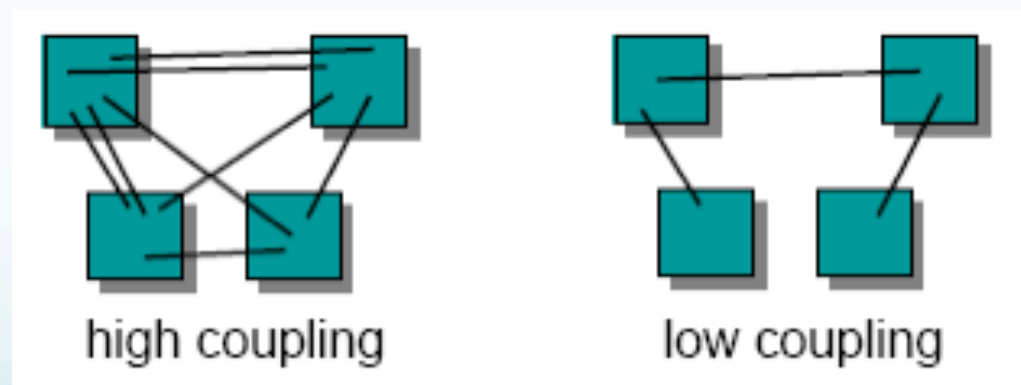
- Software Architecture Metrics deal with the organization of the software components and their relationships
- Architecture metrics may have a dramatic impact on non functional requirements of software

Modularity

- Degree of modularity is an indicator for quality.
- It has positive impact on:
 - subdivision of work
 - (design, implementation, test, maintain)
 - reuse

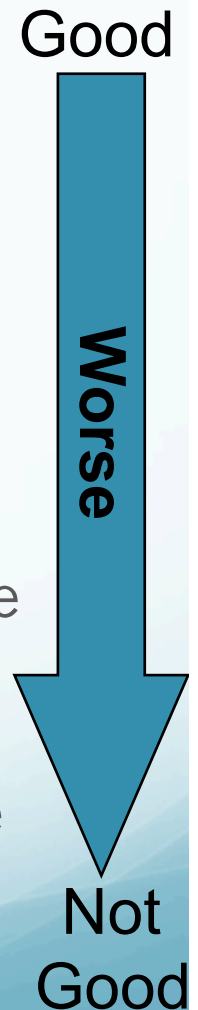
Dependency: Coupling

- Coupling is the degree of interdependence between modules
- Heuristic: *minimize coupling between modules*



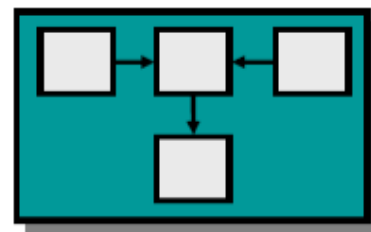
Types of Coupling

- Data coupling
 - Data from one module is used in another
- Data Type coupling
 - two modules use the same datatype
- Control coupling
 - one module may control actions of another module
- Content coupling
 - a module refers to the internals of another module

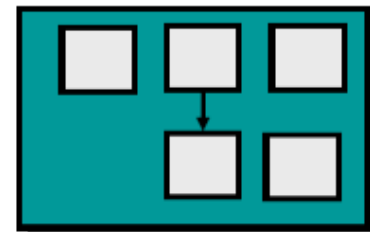


Dependency: Cohesion

- Cohesion is concerned with the interactions within a module
- Heuristic: *Keep together things that belong together.*
- High cohesion within a module reflects good design.



high cohesion



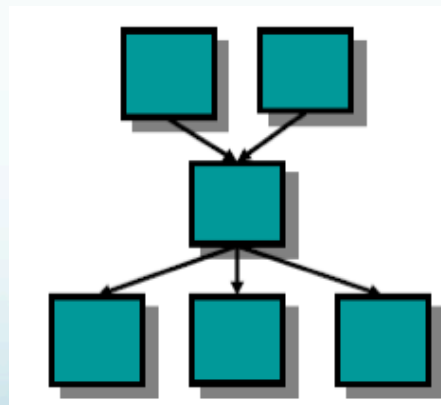
low cohesion

Types of Cohesion

- Functional cohesion:
 - a module performs a single well-defined task
- Communicational/Data cohesion:
 - a module performs multiple functions on the same data
- Temporal cohesion:
 - a module performs a set of functions that must occur in a limited/contiguous time-span.
- Logical cohesion:
 - a module performs a set of similar functions, e.g. output to screen + output to printer + output to file
 - problem: units may change independently
- Heuristic: *describe the purpose of a module in a single sentence using a single verb and a single subject*

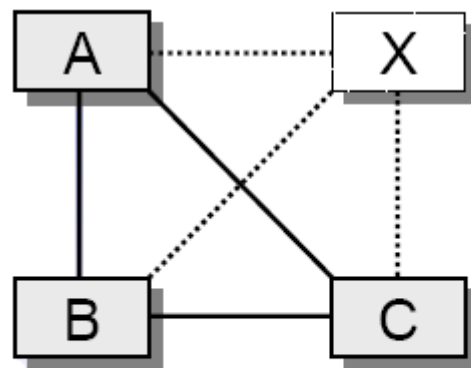
Complexity: Fan-in & Fan-out

- Fan-in = no. of ingoing dependencies
- Fan-out = no. of outgoing dependencies
- Heuristic: *a high fan-in/fan-out indicates a high complexity*

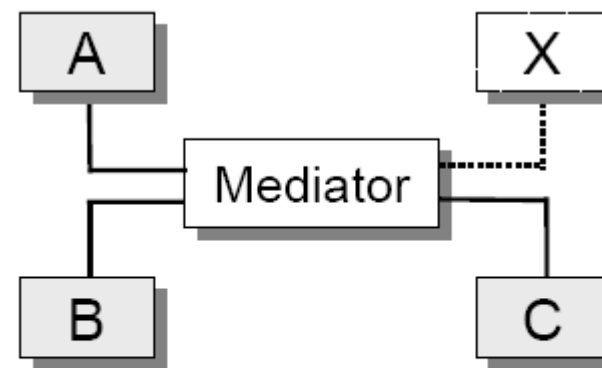


Extensibility

- Metrics:
 - Complexity of topology
 - Number of changes



Difficult to extend
Complexity $O(n^2)$



Easy to extend
Complexity $O(n)$

Outline

- Software metric fundamentals
- Quality models (McCall, Bohem, FURPS+)
- Measuring Software
- Software Metrics
 - Product Metrics
 - Process Metrics
 - Architecture-based Metrics
- Limits of Software Metrics



Limits of software metrics

Limits of software metrics/1

- Most measures are misleadingly precise, yet not very accurate
 - Size doesn't map directly to functionality, complexity, or quality
- Incremental design requires measuring of incomplete functions
- The most meaningful software statistics are time consuming to collect

Limits of software metrics/2

- Many measures only apply after coding has been done
- Performance and resource utilization may only be known after integration and testing
- Often no distinction between work and re-work
- Time lag between problems and their appearance in reports

Limits of software metrics/3

- Difficult to compare relative importance of measures
 - Important measures may be spread across components
- Hard to find reliable historical data to compare with
 - Technology quickly destroys usefulness of historical data
- Changes suggested by one performance indicator may affect others

Capability	Capability	Usability	Performance	Reliability	Installability	Maintainability	Documentation	Availability
Usability								
Performance	●	●						
Reliability	●	○	●					
Installability		○	○	○				
Maintainability	●	○	●	○				
Documentation	●	○				○		
Availability	●	○	○	○	○	○		

● Conflictive
 ○ Support One Another
 Blank = None

Example from CUPRIMDA (capability, usability, performance, reliability, installability, maintainability, documentation, and availability) approach proposed by IBM. Image from: Stephen H. Kan. Metrics and Models in Software Quality Engineering, Second Edition. Addison Wesley. 2002. Pag. 5.

Sources and References

Sources and references

- I. Sommerville. Software Engineering. 7th Edition. Addison Wesley. 2004.
- Stephen H. Kan. Metrics and Models in Software Quality Engineering, Second Edition. Addison Wesley. 2002.
- R.Solingen, E.Berghout: "The Goal/Question/ Metric Method" McGraw-Hill Publishing Company, 1999
- Grady, Robert; Caswell, Deborah (1987). Software Metrics: Establishing a Company-wide Program. Prentice Hall. p. 159. ISBN 0-13-821844-7.
- Grady, Robert (1992). Practical Software Metrics for Project Management and Process Improvement. Prentice Hall. p. 32. ISBN 0-13-821844-7.
- Lorenz, M., Object-Oriented Software Development: A Practical Guide, Englewood Cliffs, N.J.: PTR Prentice Hall, 1993
- GQM Method Application:
<http://ivs.cs.uni-magdeburg.de/sw-eng/us/java/GQM/>

Sources and References /2

- C. Lange. Metrics in Software Architecting. Online at: <http://www.win.tue.nl/~mchaudro/sa2006/>
- C. Larman. Applying UML and Patterns – Second Edition
- B. Bruegge, A. H. Dutoit – Object Oriented Software Engineering. Using UML, Patterns and Java. Third Edition. Prentice Hall. 2010.
- Victor R. Basili, Gianluigi Caldiera, H. Dieter Rombach. The Goal Question Metric Approach. Encyclopedia of Software Engineering. John Wiley & Sons. 1994. Pagg. 528-532.

Thank you for your attention

Any question?

cossentino@pa.icar.cnr.it