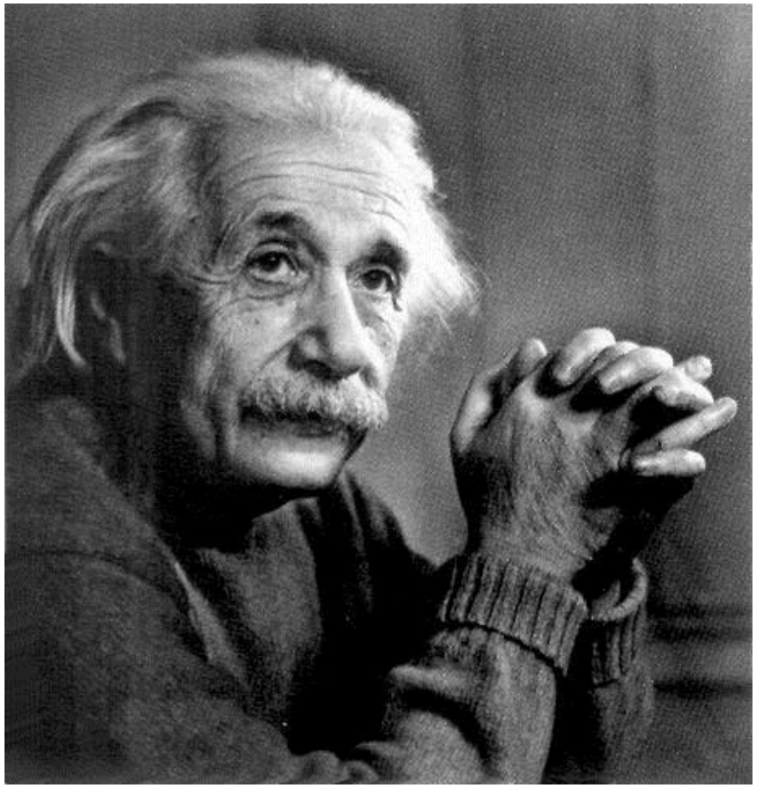
Measurement and Metrics

SWEN 256 – Software Process & Project Management

Measurements & Metrics



"Not everything that can be counted counts, and not everything that counts can be counted."

- Albert Einstein

Why Measurements and Metrics?

- Software measurement is concerned with deriving a **quantitative** (numeric) value for an attribute of a software product or process (largely qualitative)
- This allows for **objective** comparisons between techniques and processes
- Although some companies have introduced measurement programs, the systematic use of measurement is still uncommon
- There are few standards in this area

Definitions

- **Measure** – provides a quantitative indication of the size of some product or process attribute
- **Measurement** – the act of obtaining a measure
- **Metric** – a quantitative measure of the degree to which a system, component, or process possesses a given attribute

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
- Allow the software and the software process to be quantified
- Measures of the software **process or product**
- May be used to predict product attributes or to control the software process

Categories

- Product
  - Assess the quality of the **design** and construction of the software product being built.
- Process & Project
  - Quantitative measures that enable software engineers to gain insight into the **efficiency** of the software process and the projects conducted using the process framework

Process Metrics

- **Private** process metrics (e.g., defect rates by individual or module) are only known to by the individual or team concerned.
- **Public** process metrics enable organizations to make strategic changes to improve the software process.
- Metrics should **not** be used to **evaluate** the performance of individuals. **←Why?**
- *Statistical* software process **improvement** helps and organization to discover where they are strong and where they are weak

Product Metrics

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

Project Metrics

- A software team can use software project metrics to **adapt project workflow** and technical activities
- Project metrics are **used** to avoid development schedule delays, to mitigate potential risks, and to assess product quality on an on-going basis
- Every project should measure its inputs (resources), outputs (deliverables), and results (effectiveness of deliverables)

Those who cannot learn from history are doomed to repeat it

George Santayana

Metrics Assumptions

- A software property **can be** measured
- The **relationship exists** between what we can measure and what we want to know
- This relationship has been formalized and validated
- It may be difficult to relate what can be measured to desirable quality attributes

Integrating Metrics with Process

- Many software developers do not collect measures.
- Without measurement it is impossible to determine whether a process is **improving or not**
- **Baseline metrics** data should be collected from a large, representative sampling of past software projects
- Getting this **historic** project data is very difficult, if the previous developers did not collect data in an on-going manner

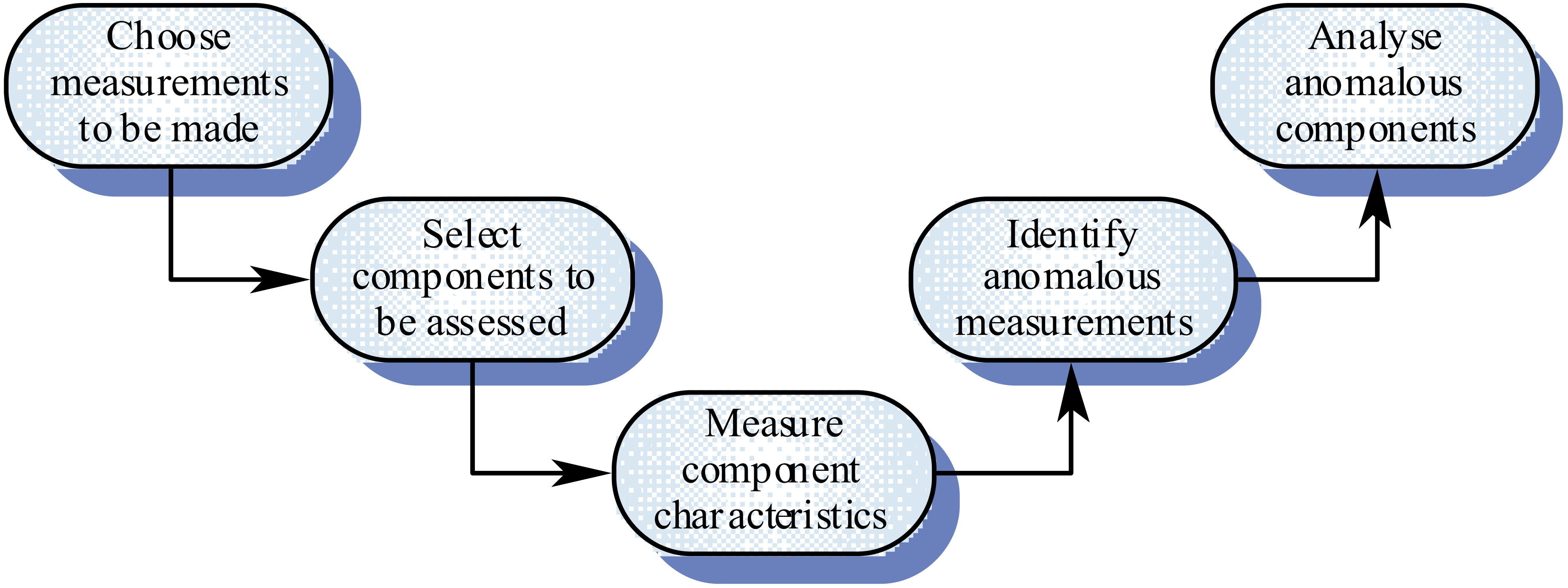
Software Measurement

- **Direct** measures of a software engineering process include **co$t** and **effort**
- Direct measures of the product include lines of code (LOC), execution speed, memory size, defects reported over some time period
- **Indirect** product measures examine the quality of the software product itself (e.g., functionality, complexity, efficiency, reliability, maintainability)

The Measurement Process

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

Product Measurement Process


Choose measurements to be made
Select components to be assessed
Measure component characteristics
Identify anomalous measurements
Analyse anomalous components

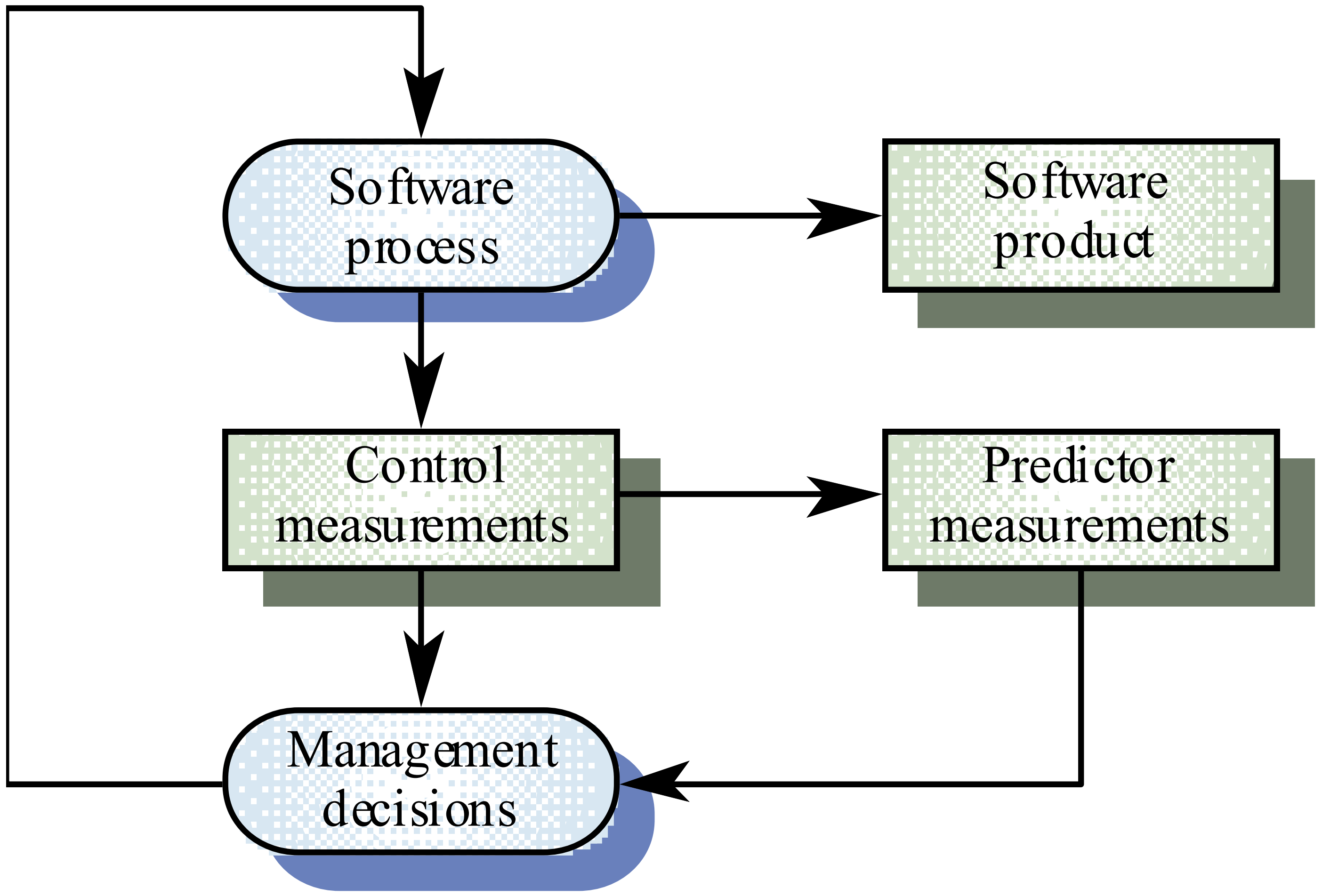
Data Collection

- A metrics program should be based on a set of product and process data
- 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

What Are Metrics For?



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

Metrics Types and Examples

Size-Oriented Metrics

- Derived by normalizing (dividing) any direct measure (e.g., defects or human effort) associated with the product or project by LOC
- Size-oriented metrics are widely used but their validity and applicability is a matter of some debate

Function-Oriented Metrics

- Function points are computed from direct measures of the information domain of a **business** software application and assessment of its **complexity**
- Once computed **function points** are used like LOC to normalize measures for software productivity, quality, and other attributes
- The relationship of LOC and function points depends on the language used to implement the software

Web Engineering Metrics Examples

- Number of static Web pages (Nsp)
- Number of dynamic Web pages (Ndp)
- Customization index: C = Nsp / (Ndp + Nsp)
- Number of internal page links
- Number of persistent data objects
- Number of external systems interfaced
- Number of static content objects
- Number of dynamic content objects
- Number of executable functions

Software Metrics Examples

- Fan in/Fan-out – Fan-in is a measure of the number of functions that call some other function (say X). Fan-out is the number of functions which are called by function X. 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.
- Length of code – This is a measure of the size of a program. Generally, the larger the size of the code of a program's components, the more complex and error-prone that component is likely to be.
- Cyclomatic complexity – This is a measure of the control complexity of a program. This control complexity may be related to program understandability. The computation of cyclomatic complexity is covered in Chapter 20.
- Length of identifiers – This is a measure of the average length of distinct identifiers in a program. 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 depth of nesting of if-statements in a program. Deeply nested if statements are hard to understand and are potentially error-prone.
- Fog index – This is a measure of the average length of words and sentences in documents. The higher the value for the Fog index, the more difficult the document may be to understand.

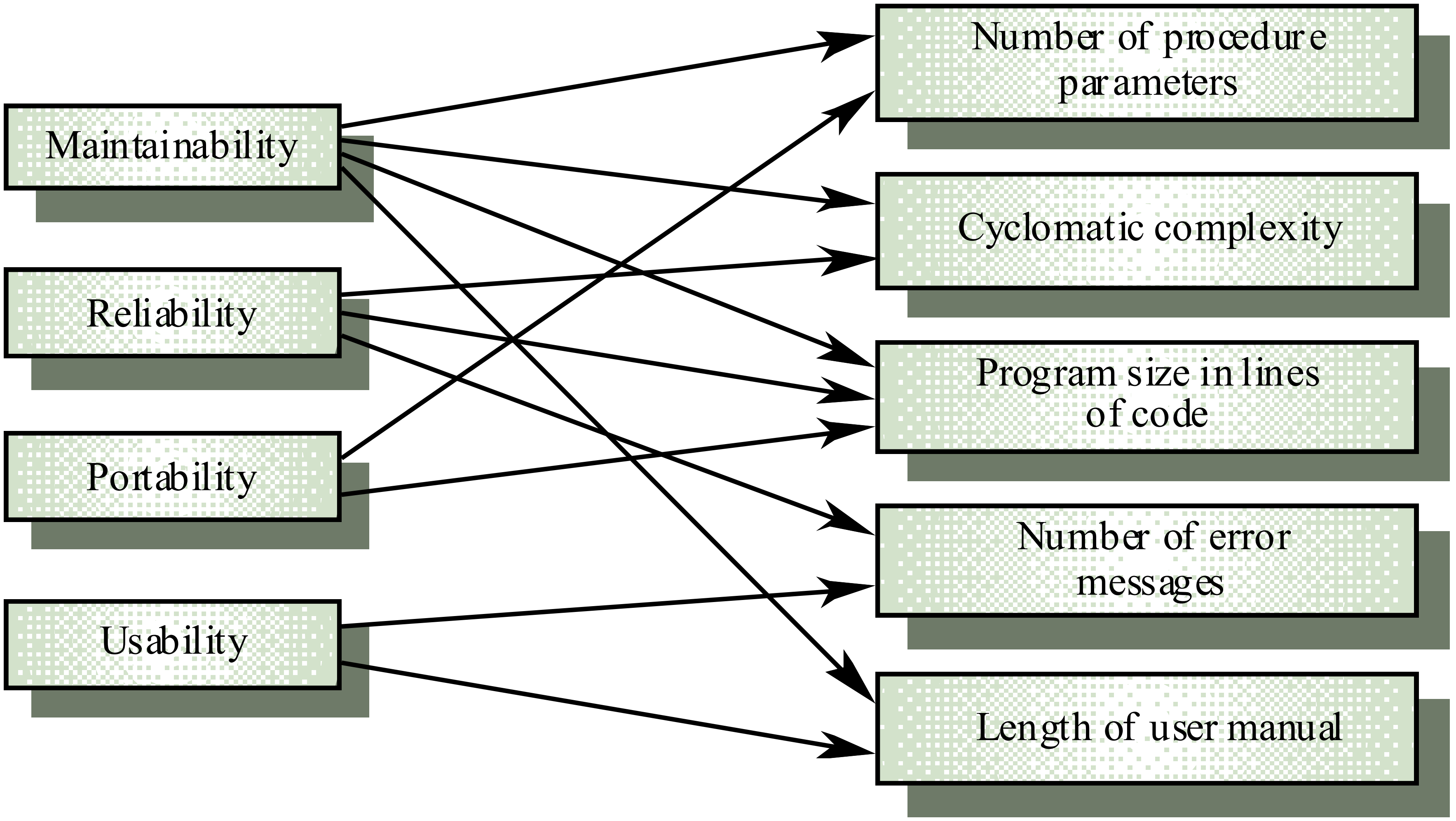
Object-Oriented Metrics Examples

- Depth of inheritance tree – This represents the number of discrete levels in the inheritance tree where sub-classes inherit attributes and operations (methods) from super-classes. The deeper the inheritance tree, the more complex the design as, potentially, many different object classes have to be understood to understand the object classes at the leaves of the tree.
- 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 make a distinction between calls from other methods within the object and calls from external methods.
- Weighted methods per class – This is the number of methods included in a class weighted by the complexity of each method. Therefore, a simple method may have a complexity of 1 and a large and complex method a much higher value. The larger the value for this metric, the more complex the object class. Complex objects are more likely to be more difficult to understand. They may not be logically cohesive so cannot be reused effectively as super-classes in an inheritance tree.
- Number of overriding operations – These are the number of operations in a super-class which 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.

Metrics for Small Organizations

- Most software organizations have fewer than 20 software engineers.
- Best advice is to choose **simple metrics** that provide value to the organization and don't require a lot of effort to collect.
- Even small groups can expect a **significant return** on the investment required to collect metrics, IFF this activity leads to process **improvement.**

Metrics and Quality



Top Ten Metrics (usage)

Number	Metric	Percentage
1	Number of defects found after a release	61%
2	Number of changes or change requests	55%
3	User or customer satisfaction	52%
4	Number of defects found during development	50%
5	Documentation completeness/accuracy	42%
6	Time to identify/correct defects	40%
7	Defect distribution by type/class	37%
8	Error by major function/feature	32%
9	Test coverage of specifications	31%
10	Test coverage of code	31%

Bottom Five Metrics (usage)

Number	Metric	Percentage
1	Module/design complexity	24%
2	Number of source lines delivered	22%
3	Documentation size/complexity	20%
4	Number of reused source lines	16%
5	Number of function points	10%

Questions/Discussion