Chapter one

What measurement?

It is the process by which numbers or symbols are assigned to attributes of entities in the real world, in such a way as to describe them according to clearly defined rules.

So, measurement captures the information about the attributes of entities.

- An entity is an **object** such as a person or an **event** such as a journey, software design, software testing in the real world.
- An attribute is a feature or property of an entity such as the height of a person, cost of a journey, etc.
    - Attributes are mostly defined by numbers or symbols. For example, the price can be specified in number of dollars, clothing size can be specified in terms of small, medium, large.
- In the real world, even though we are thinking of measuring the things, actually we are measuring the attributes of those things.
- The accuracy of a measurement depends on the measuring instrument as well as on the definition of the measurement.
- After obtaining the measurements, we have to analyze them and we have to derive conclusions about the entities.

Examples of Entities and Attributes

Entity	Attribute
Software Design	Defects discovered in design reviews
Software Design Specification	Number of pages
Software Code	Number of lines of code, number of operations
Software Development Team	Team size, average team experience
Person	Height, intelligence etc.

Measurement in Everyday Life

Measurement is not only used by professional technologists, but also used by all of us in everyday life. That is, measurement governs many aspects of everyday life:

- Economic indicators determine prices, pay raises
- Medical system measurements enable diagnosis of specific illnesses
- Measurements in atmospheric systems are the basis of weather prediction
- Similarly, height and size measurements will ensure whether the cloth will fit properly or not.

- Thus, measurement will help us compare an item with another.

Measurement is a direct quantification whereas calculation is an indirect one where we combine different measurements using some formulae.

Measurement in Software Engineering

Software measurement is an essential component of good software engineering.

- Many of the best software developers measure characteristics of their software to get some sense of whether:
  - the requirements are consistent and complete,
  - the design is of high quality, and
  - the code is ready to be released.
- Effective project managers measure attributes of **processes** and **products** to be able to tell when software will be **ready for delivery** and whether a **budget will be exceeded**.
- Organizations use **process evaluation measurements** to select software suppliers.
- Informed customers measure the aspects of the final product to determine:
  - if it meets the requirements and
  - is of sufficient quality.
- Finally, maintainers must be able to assess the **current product** to see what should be upgraded and improved.

Software Engineering activities involves managing, costing, planning, modeling, analyzing, specifying, designing, implementing, testing, and maintaining software products. Hence, measurement plays a significant role in software engineering, A rigorous approach will be necessary for measuring the attributes of a software product.

Neglect of Measurement in Software Engineering

In many instances, measurement is considered a luxury in software development. This lead for the following failures in many projects:

- We fail to set measurable targets for our software products
- We fail to understand and quantify the component cost of software projects
- We do not quantify or predict the quality of the products we produce
- Too much reliance
  - We allow anecdotal evidence to convince us to try yet another revolutionary new development technology, without doing a carefully controlled study to determine if the technology is efficient and effective. Promotional materials for software development tools and techniques typically include the following types of claims:
    - ✓ “Our new technique guarantees 100% reliability.”
    - ✓ “Our tool improves productivity by 200%!!”
    - ✓ “Build your code with half the staff in a quarter of the time.”
    - ✓ “Cuts test time by 2/3.”

These claims are generally not supported by scientific studies.

Thus, for controlling software products, measuring the attributes is necessary. Every measurement action must be motivated by a particular goal or need that is clearly defined and easily understandable. The measurement objectives must be specific, tried to what managers, developers and users need to know. Measurement is required to assess the status of the project, product, processes, and resources.

In software engineering, measurement is essential for the following three basic activities −

- To understand what is happening during development and maintenance
- To control what is happening in the project
- To improve processes and goals

Software metrics is a standard of measure that contains many activities which involve some degree of measurement. It can be classified into three categories: product metrics, process metrics, and project metrics.

- **Product metrics** describe the characteristics of the product such as size, complexity, design features, performance, and quality level.
- **Process metrics** can be used to improve software development and maintenance. Examples include the effectiveness of defect removal during development, the pattern of testing defect arrival, and the response time of the fix process.
- **Project metrics** describe the project characteristics and execution. Examples include the number of software developers, the staffing pattern over the life cycle of the software, cost, schedule, and productivity.

Some metrics belong to multiple categories. For example, the in-process quality metrics of a project are both process metrics and project metrics.

Objectives for Software Measurement

Even when a project is not in trouble, measurement is not only useful but also necessary. Measurement is needed at least for assessing the status of your projects, products, processes, and resources.

It is not enough to assert that we must measure to gain control. But, every measurement action must be motivated by a particular goal or need that is clearly defined and easily understandable. The measurement objectives must be specific, tied to what the managers, developers, and users need to know.
Here the kinds of information needed to understand and control a software development project, from both manager and developer perspectives.

1. Managers:

• **What does each process cost?** We can measure the time and effort involved in the various processes that comprise software production. For example, we can identify the cost to elicit requirements, the cost to specify the system, the cost to design the system, and the cost to code and test the system. In this way, we gain understanding not only of the total project cost but also of the contribution of each activity to the whole.
• **How productive is the staff?** We can measure the time it takes for staff to specify the system, design it, code it, and test it. Then, using Measurement measures of the size of specifications, design, code, and test plans, for example, we can determine how productive the staff is at each activity. This information is useful when changes are proposed; the manager can use the productivity figures to estimate the cost and duration of the change.
• **How good is the code being developed?** By carefully recording faults, failures, and changes as they occur, we can measure software quality, enabling us to compare different products, predict the effects of change, assess the effects of new practices, and set targets for process and product improvement.
• **Will the user be satisfied with the product?** We can measure functionality by determining if all of the requirements requested have actually been implemented properly. And we can measure usability, reliability, response time, and other characteristics to suggest whether our customers will be happy with both functionality and performance.
• **How can we improve?** We can measure the time it takes to perform each major development activity and calculate its effect on quality and productivity. Then we can weigh the costs and benefits of each practice to determine if the benefit is worth the cost. Alternatively, we can try several variations of a practice and measure the results to decide which is best; for example, we can compare two design methods to see which one yields the higher-quality code.

2. Developers:

• **Are the requirements testable?** We can analyze each requirement to determine if its satisfaction is expressed in a measurable, objective way. For example, suppose a requirement states that a web-based system must be “fast”; the requirement can be replaced by one that states that the mean response time to a set of specific of inputs must be less than 2 second for specified browsers and number of concurrent users.
• **Have we found all the faults?** We can measure the number of faults in the specification, design, code, and test plans and trace them back to their root causes. Using models of expected detection rates, we can use this information to decide whether inspections and testing have been effective and whether a product can be released for the next phase of development.
• **Have we met our product or process goals?** We can measure characteristics of the products and processes that tell us whether we have met standards, satisfied a requirement, or met a process goal. For example, certification may require that fewer than 20 failures have been reported per beta-test site over a given period of time. Or a standard may mandate that all modules must pass code inspections. The testing process may require that unit testing must achieve 90% statement coverage.
• **What will happen in the future?** We can measure attributes of existing products and current processes to make predictions about future ones. For example, measures of size of specifications can be used to predict the size of the target system, predictions about future maintenance

problems can be made from measures of structural properties of the design documents, and predictions about the reliability of software in operational use can be made by measuring reliability during testing.

Scope of Software Metrics

Software metrics contains many activities which include the following −

- Cost and effort estimation
- Productivity measures and model
- Data collection
- Quantity models and measures
- Reliability models
- Performance and evaluation models
- Structural and complexity metrics
- Capability – maturity assessment
- Management by metrics
- Evaluation of methods and tools

Software measurement is a diverse collection of these activities that range from models predicting software project costs at a specific stage to measures of program structure.

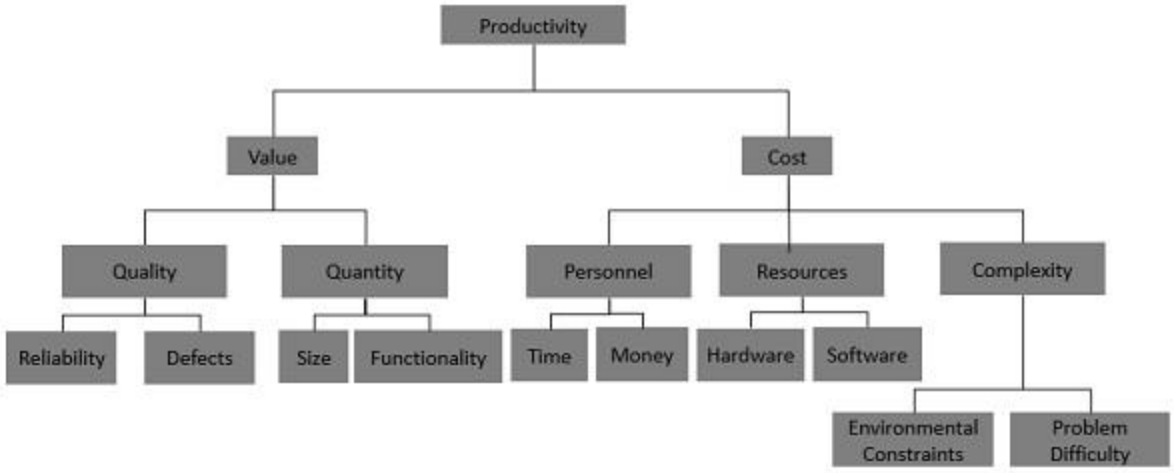
Cost and Effort Estimation

Effort is expressed as a function of one or more variables such as the size of the program, the capability of the developers and the level of reuse. Cost and effort estimation models have been proposed to predict the project cost during early phases in the software life cycle. The different models proposed are −

- Boehm’s COCOMO model
- Putnam’s slim model
- Albrecht’s function point model

Productivity Model and Measures

Productivity can be considered as a function of the value and the cost. Each can be decomposed into different measurable size, functionality, time, money, etc. Different possible components of a productivity model can be expressed in the following diagram.



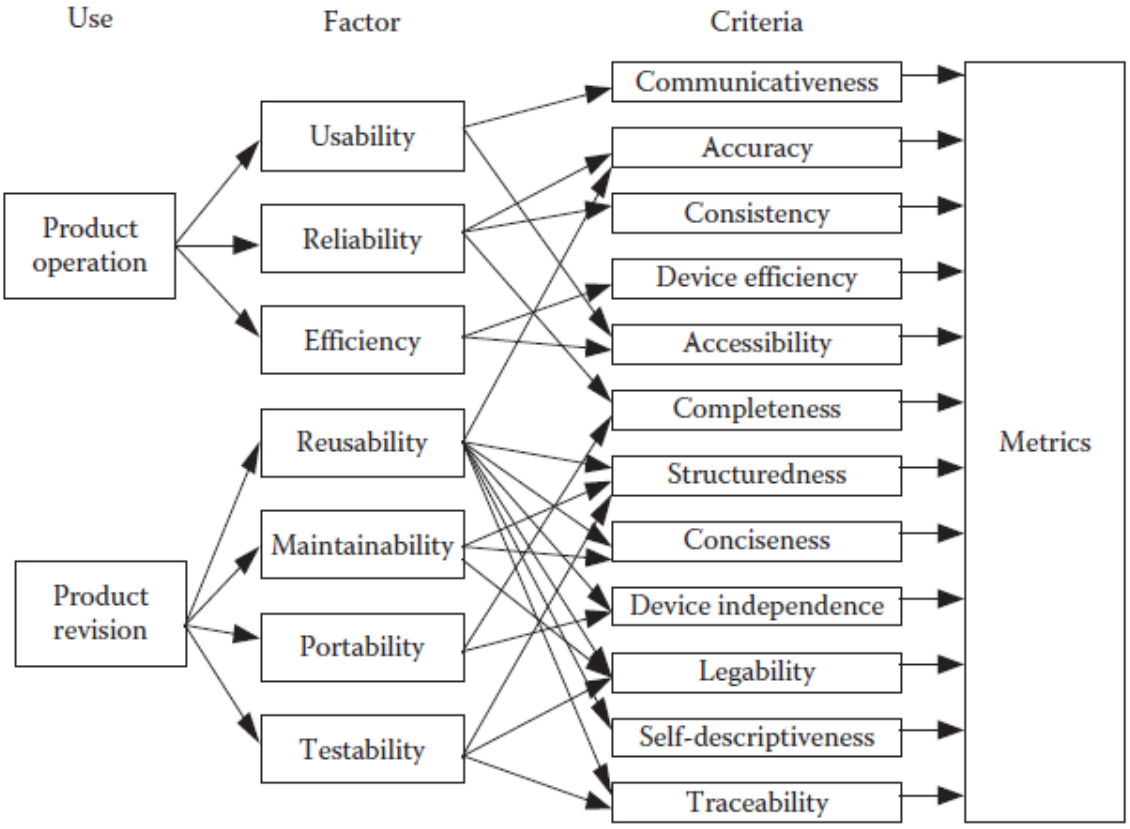
Data Collection

The quality of any measurement program is clearly dependent on careful data collection. Data collected can be distilled into simple charts and graphs so that the managers can understand the progress and problem of the development. Data collection is also essential for scientific investigation of relationships and trends.

Quality Models and Measures

Quality models have been developed for the measurement of quality of the product without which productivity is meaningless. These quality models can be combined with productivity model for measuring the correct productivity. These models are usually constructed in a tree-like fashion. The upper branches hold important high-level quality factors such as reliability and usability.

The notion of divide and conquer approach has been implemented as a standard approach to measuring software quality.



Software quality model

Reliability Models

Most quality models include reliability as a component factor, however, the need to predict and measure reliability has led to a separate specialization in reliability modeling and prediction. The basic problem in reliability theory is to predict when a system will eventually fail.

Performance Evaluation and Models

It includes externally observable system performance characteristics such as response times and completion rates, and the internal working of the system such as the efficiency of algorithms. It is another aspect of quality.

Structural and Complexity Metrics

Here we measure the structural attributes of representations of the software, which are available in advance of execution. Then we try to establish empirically predictive theories to support quality assurance, quality control, and quality prediction.

Capability Maturity Assessment

This model can assess many different attributes of development including the use of tools, standard practices and more. It is based on the key practices that every good contractor should be using.

Management by Metrics

For managing the software project, measurement has a vital role. For checking whether the project is on track, users and developers can rely on the measurement-based chart and graph. The standard set of measurements and reporting methods are especially important when the software is embedded in a product where the customers are not usually well-versed in software terminology.

Evaluation of Methods and Tools

This depends on the experimental design, proper identification of factors likely to affect the outcome and appropriate measurement of factor attributes.