Chapter-3

A GOAL-BASED FRAMEWORK FOR SOFTWARE MEASUREMENT

Prepared by: Mintesinot A.

Software metrics is a term that embraces many activities , all of which involve some degree of software measurement:

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

Introduction

- The framework for software measurement is based on two principles activities
  - ➢Classifying the entities to be examined
  - ➢Using measurement goals to identify relevant metrics
- Goal-based framework supports evaluations of sofware products , processes, and sofware development organizations.

Classifying Software Measures

- The first obligation of any software measurement activity is identifying the entities and attributes that we want to measure
- Software entities can be classified as follows
    - ➢**Processes:** Collection of software-related activities.
    - ➢**Products:** Artifacts, deliverables or documents that result from a process activity.
    - ➢**Resources:** Entities required by a process activity.
- Within each class of entity, we distinguish between internal and external attributes of a product, process, or resource:

Classifying Software Measures (cont.)

- Internal attribute: are those that can be measured by examining the product, process, or resource on its own, separate from its behavior. (e.g. Program size, complexity, dependencies).
- External attributes: are those that can be measured only with respect to how the product, process or resource relates to the environment. (Ease of Navigation, Number of failures)
- Managers often want to be able to measure and predict external attributes.
  - For example, the cost-effectiveness of an activity or the productivity of the staff can be very useful in ensuring that the quality stays high while the price stays low.

Classifying Software Measures (cont.)

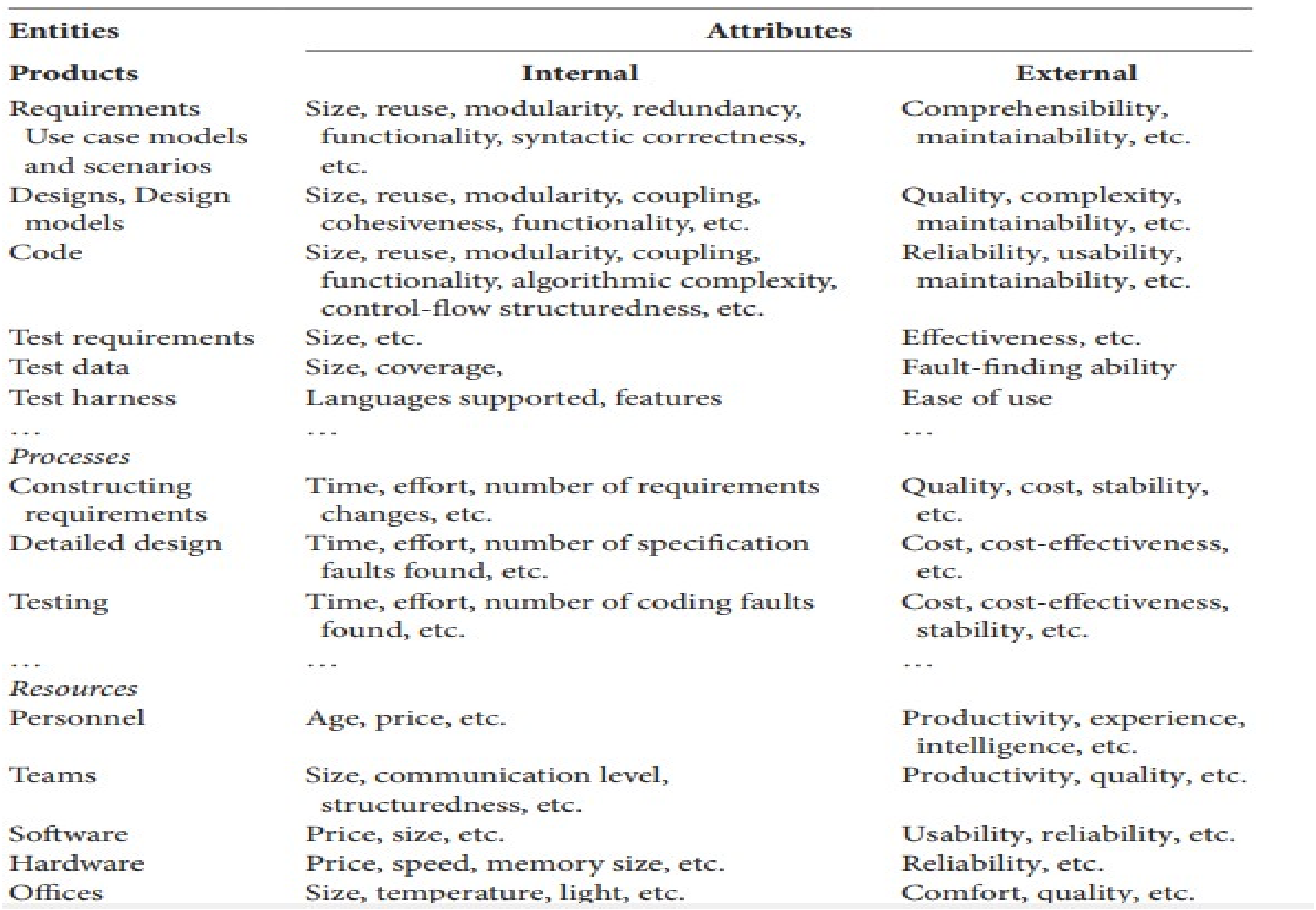
Internal

- Size, Effort, Cost
- Code Complexity
- Functionality
- Modularity
- Redundancy
- Syntactic Correctness
- Reuse

External

- Usability
- Integrity
- Efficiency
- Testability
- Reusability
- Portability
- Interoperability

Components of Software Measurement



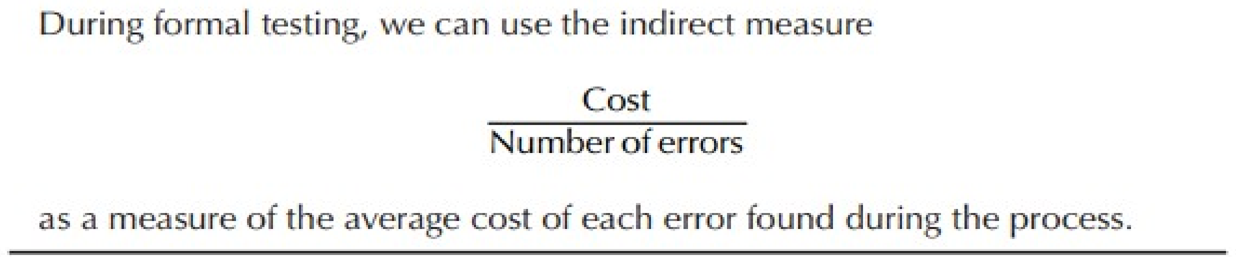
Entities	**Attributes**	
Products	**Internal**	**External**
Requirements Use case models and scenarios	Size, reuse, modularity, redundancy, functionality, syntactic correctness, etc.	Comprehensibility, maintainability, etc.
Designs, Design models	Size, reuse, modularity, coupling, cohesiveness, functionality, etc.	Quality, complexity, maintainability, etc.
Code	Size, reuse, modularity, coupling, functionality, algorithmic complexity, control-flow structuredness, etc.	Reliability, usability, maintainability, etc.
Test requirements	Size, etc.	Effectiveness, etc.
Test data	Size, coverage,	Fault-finding ability
Test harness	Languages supported, features	Ease of use
...	...	...
Processes		
Constructing requirements	Time, effort, number of requirements changes, etc.	Quality, cost, stability, etc.
Detailed design	Time, effort, number of specification faults found, etc.	Cost, cost-effectiveness, etc.
Testing	Time, effort, number of coding faults found, etc.	Cost, cost-effectiveness, stability, etc.
...	...	...
Resources		
Personnel	Age, price, etc.	Productivity, experience, intelligence, etc.
Teams	Size, communication level, structuredness, etc.	Productivity, quality, etc.
Software	Price, size, etc.	Usability, reliability, etc.
Hardware	Price, speed, memory size, etc.	Reliability, etc.
Offices	Size, temperature, light, etc.	Comfort, quality, etc.

Processes

- We often have questions about our software-development activities and processes that measurement can help us to answer.
- We want to know *how long it takes for a process to complete*, *how much it will cost*, whether it is effective or efficient, and how it compares with other processes that we could have chosen.
- Following are some of the internal attributes that can be measured directly for a process
  - The duration of the process or one of its activities.
  - The effort associated with the process or one of its activities.
  - The number of incidents of a specified type arising during the process or one of its activities.

Processes(cont.)

- E.g:



During formal testing, we can use the indirect measure

$$\frac{\text{Cost}}{\text{Number of errors}}$$

as a measure of the average cost of each error found during the process.

- E.g: AT&T developers wanted to know the effectiveness of their software inspections. In particular, managers needed to evacuate the cost of inspections against benefits received. To do this, they measured the average amount of effort expended per thousand lines of code reviewed. This information combined with measures of the number of faults discovered during the inspections, allowed managers to perform a cost-benefit analysis.

Processes(cont.)

- Cost is not the only process attribute that we can examine.
- Controllability, observability, and stability are also important in managing a large project.
- These attributes are clearly external ones.
- For example, stability of the design process can depend on the particular period of time, as well as on which designers are involved.

Products

- Products are Any artifact or document produced during the software life cycle can be measured and assessed.
- For example developers often build prototypes for examination only, so that they can understand requirements or evaluate possible designs; these prototypes may be measured in some way.
- External product attributes depend on both product behavior and environment, each attribute measure should take these characteristics into account.
  - The understandability of a document depends on the experience and credentials of the person reading it;
  - The maintainability of a system may depend on the skills of the maintainers and the tools available to them.

Products(cont.)

- Internal product attributes are sometimes easy to measure.
- We can determine the size of a product by measuring the number of pages it fills or the number of words it contains.
- The products are concrete, we have a better understanding of attributes like size, effort, and cost.
- Other internal product attributes are more difficult to measure, because opinions differ as to what they mean and how to measured them. For example the complexity of codes.

Importance Of Internal Attributes

- Many software engineering methods proposed and developed in the last 40 years provide rules, tools, and any heuristics for providing software products.
- These methods give structure to the products and the common wisdom is that this structure makes software products easier to understand, analyze, test, and modify.
- The structure involves two aspects of development:
  - The development process, as certain products need to be produced at certain stages, and
  - The products themselves, as the products must conform to certain structural principles.

Resources

- The resources that we are likely to measure include any input for software production.
- Resources include personnel, materials, tools and methods.
- Resources are measured to determine their magnitude, cost and quality.
- Cost is often measured across all types of resources, so that managers can see how the cost of inputs affects the cost of the outputs.
- Resource measure combines a process measure (input) with a product measure (output).

Resources (cont.)

- We measure resources to determine
  - Their magnitude (how many staff are working on this project?)
  - Their cost (how much are we paying for testing tools?)
  - Their quality (how experienced are our designers?).
  - These measures help us to understand and control the process.
  - For example, if we are producing poor-quality software, resource measurements may show us that the software quality is the result of using too few people or people with the wrong skills.

Resources (cont.)

- Productivity is usually measured as some form of the following equation:

$$\frac{\text{Amount of output}}{\text{Effort input}}$$

- Resource measure combines a process measure (input) with a product measure (output).
- For software development, the measure of ***output*** is usually computed as ***the amount of code or functionality*** produced as the final product, while the ***input*** measure is ***the number of person-months*** used to specify, design, code, and test the software.

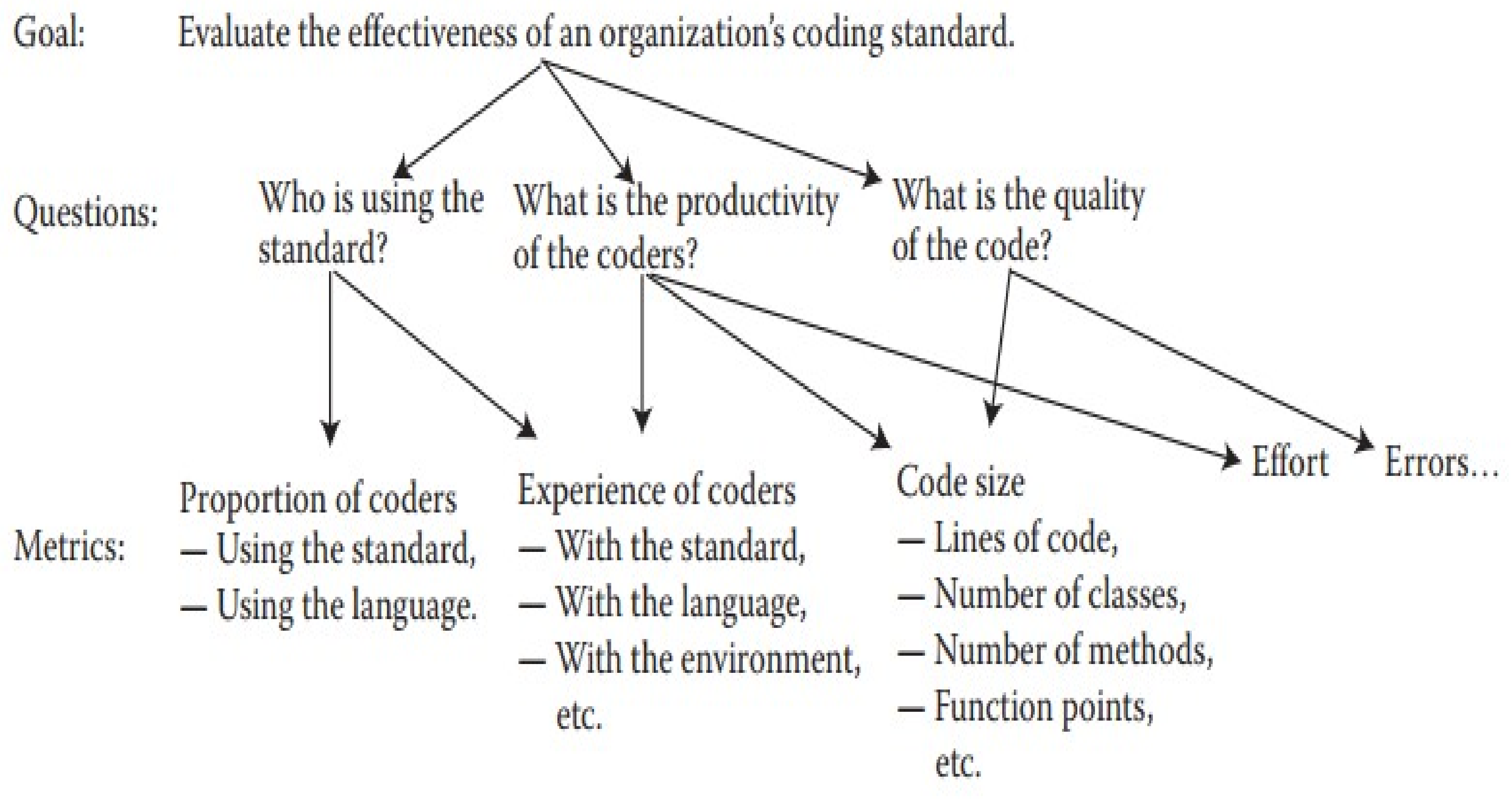
Determining What To Measure

- A particular measurement is useful only if it helps you to understand the underlying process or one of its resultant products.
- The improvement in the process or products can be performed only when the project has clearly defined goals for processes and products.
- A clear understanding of goals can be used to generate suggested metrics for a given project in the context of a process maturity framework.

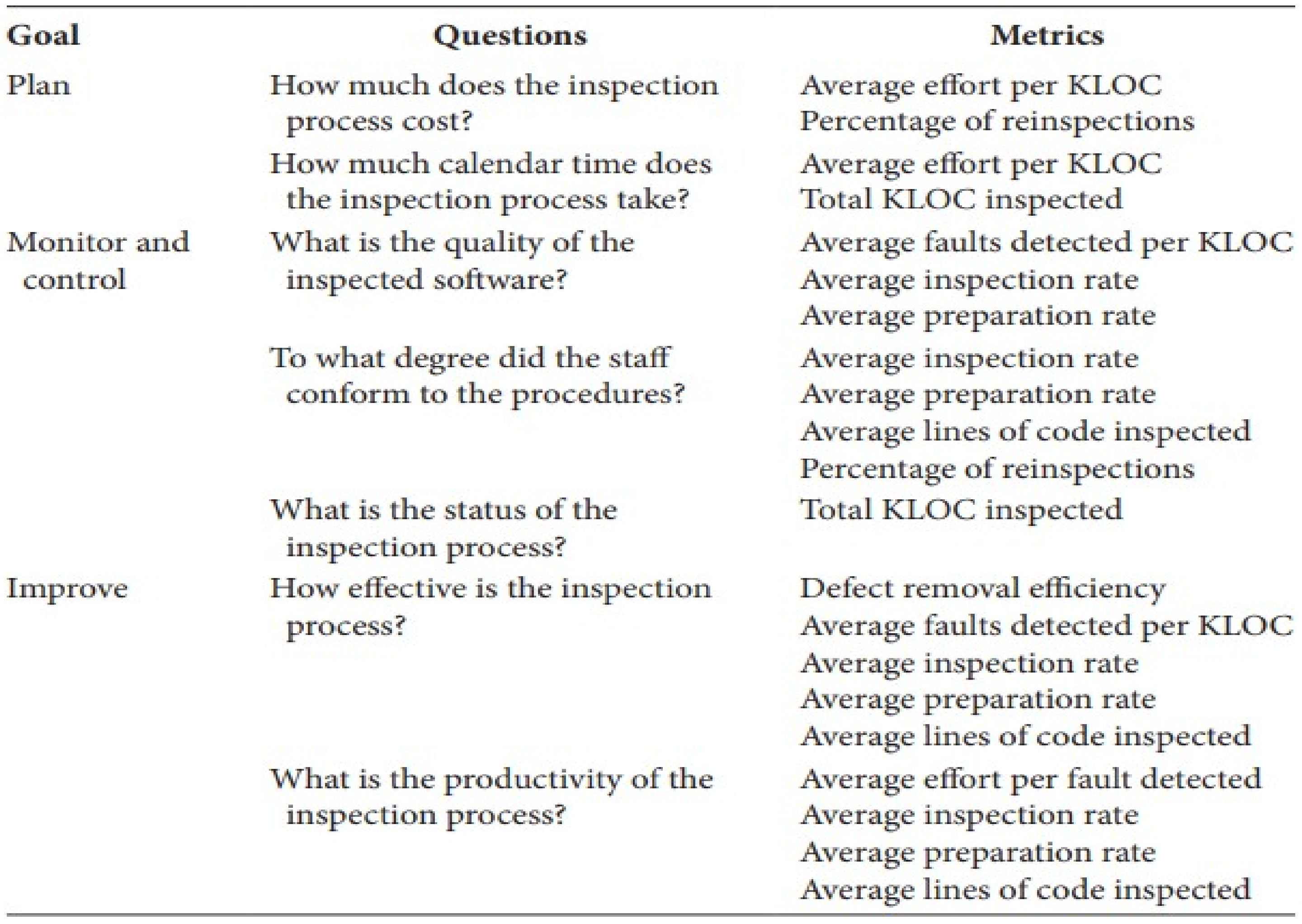
Goal-Question-Metric

- The GQM approach to process and metrics has proven to be a particular effective approach to selecting and implementing the metrics.
- To use GQM, You express the overall goals of your organization $\longrightarrow$ Ask relevant questions $\longrightarrow$ Measure.
- So, GQM approach provides a framework involving the following three steps
  - ➢ Listing the major goals of the development or maintenance project
  - ➢ Deriving the questions from each goal that must be answered to determine if the goals are being met
  - ➢ Decide what must be measured in order to be able to answer the questions adequately

Deriving metrics from goals and questions



Examples Of AT&T goals, questions and metrics



Goal	Questions	Metrics
Plan	How much does the inspection process cost?	Average effort per KLOC
		Percentage of reinspections
	How much calendar time does the inspection process take?	Average effort per KLOC
		Total KLOC inspected
Monitor and control	What is the quality of the inspected software?	Average faults detected per KLOC
		Average inspection rate
		Average preparation rate
	To what degree did the staff conform to the procedures?	Average inspection rate
		Average preparation rate
		Average lines of code inspected
		Percentage of reinspections
	What is the status of the inspection process?	Total KLOC inspected
Improve	How effective is the inspection process?	Defect removal efficiency
		Average faults detected per KLOC
		Average inspection rate
		Average preparation rate
		Average lines of code inspected
	What is the productivity of the inspection process?	Average effort per fault detected
		Average inspection rate
		Average preparation rate
		Average lines of code inspected

Templates for goal definition:

- Typical goals are expressed in terms of productivity, quality, risk, customer satisfaction, etc. Goals and questions are to be constructed in terms of their audience.
- To help generate the goals, questions, and metrics, Basili & Rombach provided a series of templates.
  - ➢**Purpose**: To (characterize, evaluate, predict, motivate, etc.) the (process, product, model, metric, etc.) in order to understand, assess, manage, engineer, learn, improve, etc. **Example**: To characterize the product in order to learn it, maintenance in order to improve it

Templates for goal definition(cont.)

- **Perspective** − Examine the (cost, effectiveness, correctness, defects, changes, product measures, etc.) from the viewpoint of the developer, manager, customer, etc. **Example**: Examine the defects from the viewpoint of the customer.
- **Environment** − The environment consists of the following: process factors, people factors, problem factors, methods, tools, constraints, etc. **Example**: The customers of this software are those who have no knowledge about the tools.

Measurement For Process Improvement

- Normally measurement is useful for:
    - Understanding the process and products
    - Establishing a baseline
    - Accessing and predicting the outcome
- The Capability Maturity Model Integration (CMMI) for Development provides an ordinal ranking of development organizations from ***initial*** (the least predictable and controllable, and least understood) to ***optimizing*** (the most predictable and controllable).

Measurement For Process Improvement(cont.)

1. **Level 1 - Initial:** Processes are ad hoc and "success depends on the competence and heroics of the people in the organization."
2. **Level 2 - Managed:** Processes are planned; "the projects employ skilled people … have adequate resources … involve relevant stakeholders; are monitored, controlled, and reviewed … ."
3. **Level 3 - Defined:** "Processes are well characterized and understood, and are described in standards procedures, tools, and methods."

Measurement For Process Improvement(cont.)

4. **Level 4 - Quantitatively managed:** “organization and projects establish quantitative objectives for quality and process performance and use them as criteria in managing projects.”
5. **Level - Optimizing:** “organization continually improves its processes based on a quantitative understanding of its business objectives and performance needs”

CMMI-Development version 1.3 Level 2 Managed

- For example, to reach CMMI-Development version 1.3 Level 2 Managed, a process must satisfy 15 goals in 7 process areas:

1. *Configuration management goals*:
    - Establish baselines
    - Track and control changes
    - Establish integrity.
2. *Measurement and analysis goals*:
    - Align measurement and analysis activities
    - Provide measurement results.

CMMI-Development version 1.3 Level 2 Managed(cont.)

3. *Project monitoring and control goals*:
   - Monitor project against plan
   - Manage corrective actions to closure.
4. *Project planning goals*:
   - Establish estimates
   - Develop a project plan
   - Obtain commitment to the plan.
5. *Process and quality assurance goals*:
   - Objectively evaluate processes and work products
   - Provide objective insight.

CMMI-Development version 1.3 Level 2 Managed(cont.)

6. *Requirements management goals*:
   - Manage requirements.
7. *Supplier agreement management goals*:
   - Establish supplier agreements
   - Satisfy supplier agreements

CMMI-Development version 1.3 Level 2 Managed(cont.)

- For example, answers to the following questions determine whether configuration management process area goals are met: Does the development process

- Identify configuration items?
- Establish a configuration management system?
- Create or release baselines?
- Track change requests?
- Control changes to configuration items?
- Establish configuration management records?
- Perform configuration audits?

Combining GQM with Process Maturity

- The goal and question analysis will be the same, but the metric will vary with maturity.
- The more mature the process, the richer will be the measurements.
- The GQM paradigm, in concert with the process maturity, has been used as the basis for several tools that assist managers in designing measurement programs.
- GQM helps to understand the need for measuring the attribute, and process maturity.
- Together they provide a context for measurement.

Combining GQM with Process Maturity(cont.)

- Suppose you are using the Goal-Question-Metric paradigm to decide what your project should measure.
- You may have identified at least one of the following high-level goals:
  - Improving productivity
  - Improving quality
  - Reducing risk
- For example, the goal of improving productivity can be interpreted as several subgoals affecting resources:
  - Assuring adequate staff skills
  - Assuring adequate managerial skills
  - Assuring adequate host software engineering technology

Applying The Framework

- **Cost and Effort Estimation:**
  - Cost and effort estimation focuses on predicting the attributes of cost or effort for the development process.
- **Productivity Measures and Models**
  - The most commonly used model for productivity measurement expresses productivity as the ratio “process output influenced by the personnel” divided by “personnel effort or cost during the process.”
- **Quality Models and Measures**
  - usually involve product measurement, as their ultimate goal is predicting product quality
- **Reliability Models**
  - The accepted view of reliability is described as the likelihood of successful operation, so reliability is a relevant attribute only for executable code.

Applying The Framework (cont.)

- **Structural and Complexity Metrics:**
  - By measuring specific internal attributes like control-flow structure, information flow, and number of paths of various types, one may attempt to quantify those aspects of the code that make the code difficult to understand.
- **Management by Metrics**
  - Managers often use metrics to set targets for their development projects. These targets are sometimes derived from best practice, as found in a sample of existing projects
- **Evaluation of Methods and Tools**
  - The proposed tool or method is tried first on a small project, and the results are evaluated to determine whether further investment and broader implementation are in order.

Software Measurement Validation

- Even when you know which entity and attribute you want to assess, there are many measures from which to choose.
- Sometimes, managers are confused by measurement which is not surprising.
- One of the roots of this confusion is the lack of software measurement validation.

Software Measurement Validation(cont.)

- The validation approach depends on distinguishing *measurement* from *prediction*
  - **Measures or measurement systems**
    - assess an existing entity by numerically characterizing one or more of its attribute.
  - **Prediction systems**
    - to predict some attribute of a future entity, involving a mathematical model with associated prediction procedures.

Software Measurement Validation (cont.)

- **Validating software Measures:** is the process of ensuring that the measure is a proper numerical characterization of the claimed attribute by showing that the representation condition is satisfied.
- The formal requirement for a validating measure involves demonstrating that it characterizes the stated attribute in the sense of measurement theory.

Software Measurement Validation (cont.)

- **Validating a prediction system** in a given environment is the process of establishing the accuracy of the prediction system by empirical means; that is, by comparing model performance with known data in the given environment.
- It involves ***experimentation*** and ***hypothesis testing***.
- To validate the prediction system formally you must first decide how stochastic it is, and then compare performance of the prediction system with known data points.

Software Measurement Validation (cont.)

- Two types of Prediction systems
  - *Deterministic prediction systems*- getting the same output for a given input
  - *stochastic prediction systems*- the output for a given input will vary probabilistically with respect to a given model.
- Boehm has stated that under certain circumstances the COCOMO effort prediction system will be accurate to within 20%;
  - An *acceptance range* for a prediction system is a statement of the maximum difference between prediction and actual value.

Performing Software Measurement Validation

- Software engineering community has always been aware of the need for validation.
- Thus, a measure must be viewed in the context in which it will be used.
- Validation must take into account the measurement's purpose; measure *X* may be valid for some uses but not for others.

Choosing Appropriate Prediction Systems

- We divide prediction systems into the following classes:

i. ***Class 1***. Using internal attribute measures of early life-cycle products to predict measures of internal attributes of later life-cycle products.
   - For example, measures of size, modularity, and reuse of a specification are used to predict size and structuredness of the final code.

ii. ***Class 2***. Using early life-cycle process attribute measures and resource attribute measures to predict measures of attributes of later life-cycle processes and resources.
   - For example, the number of faults found during formal design review is used to predict cost of implementation.

Choosing Appropriate Prediction Systems(cont.)

iii. ***Class 3.*** Using internal product attribute measures to predict process attributes.

- For example, measures of structuredness are used to predict time to perform some maintenance task, or number of faults found during unit testing

iv. ***Class 4***. Using process measures to predict later process measures.

- For example, measures of failures during one operational period are used to predict likely failure occurrences in a subsequent operational period. In examples like this, where an external product attribute (reliability) is effectively defined in terms of process attributes (operational failures)

Choosing Appropriate Prediction Systems(cont.)

v. ***Class 5***. *Using* internal *structural* attributes to predict external and process attributes.

- Examples of these prediction systems include using module coupling or other structural measures to predict the faultproneness of a component, where fault-proneness is based on failures during testing or operation that are traced to module faults. These prediction systems tend to work only on the specific systems where the prediction system parameters are determined.

"Thank You"