Course Name:-Software Metrics

Chapter one: Overview of software metrics

compiled by Samuel. A

Outline

- **What is measurement?**
- **Measurement in Everyday Life**
- **Measurement in Software Engineering**
- **Neglect of Measurement in Software Engineering**
- **Objectives for Software Measurement**
- **Why Measure Software?**
- **What is Software *Metric***
- **Motivation for Metrics**
- **Scope of Software Metrics**
- **Key Metrics to Monitor Your React Native App's Performance**

What is 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.
- measurement captures the information about the attributes of entities
- Ex

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 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

Measurement in Software Engineering cont,,

- 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

Why Measure Software?

- Determine the quality of the current product or process
- Predict qualities of a product/process
- Improve quality of a product/process

Objectives for Software Measurement

- 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?**
- **How productive is the staff?**
- **How good is the code being developed?**
- **Will the user be satisfied with the product?**
- **How can we improve?**

2. Developers:

- **Are the requirements testable?**
- **Have we found all the faults?**
- **Have we met our product or process goals?**
- **What will happen in the future?**

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

Biggest Information Technology Failures

- **A large drug wholesaler failed large implementation of ERP**

In the early 90s, FoxMeyer, a healthcare service company. The IT system, a multi-million dollar project, was the first of its kind launched in the pharmaceutical industry. The implementation cost for was budgeted at $65 million

This failure was the result of poor planning and implementation.

Biggest Information Technology Failures

- **A large drug wholesaler failed large implementation of ERP**
  - In the early 90s, FoxMeyer, a healthcare service company. The IT system, a multi-million dollar project, was the first of its kind launched in the pharmaceutical industry. The implementation cost for was budgeted at $65 million. This failure was the result of poor planning and implementation.
    - No restructuring of the business process
    - Insufficient testing
    - Overly ambitious project scope
    - Dominance of IT specialists' own interest
    - Poor management support
    - Lack of end-user cooperation

Biggest Information Technology Failures

- **A $2 billion air traffic control system failed due to insufficient computer memory**
  - The $2.4 billion system, made by Lockheed Martin Corp
  - On April 30, 2014, hundreds of LAX flights were delayed or canceled because all computers in the airport crashed due to a bug in the En Route Automation Modernization (ERAM) system.

Biggest Information Technology Failures

Biggest Information Technology Failures

- **The Ariane 5 launch became one of the biggest information technology failure**
  - the rocket exploded, burst into a million pieces, and crashed into the open field.
  - The failure lead to an extra cost of $370 million and turned a large, potentially innovative project into burning little pieces of dust.

Biggest Information Technology Failures

Motivation for Metrics

- Estimate the cost & schedule of future projects
- Evaluate the productivity impacts of new tools and techniques
- Establish productivity trends over time
- Improve software quality
- Forecast future staffing needs
- Anticipate and reduce future maintenance needs

What is Software *Metric*

- A **software metric** is a standard of measure of a degree to which a software system or process possesses some property.
- Even if a metric is not a measurement (metrics are functions, while measurements are the numbers obtained by the application of metrics),

Cont,,

- *Metric* - quantitative measure of degree to which a system, component or process possesses a given attribute. “A handle or guess about a given attribute.”
  - E.g., Number of errors found per person hours expended

- *Measure* - quantitative indication of extent, amount, dimension, capacity, or size of some attribute of a product or process.
  - E.g., Number of errors

Cont,,

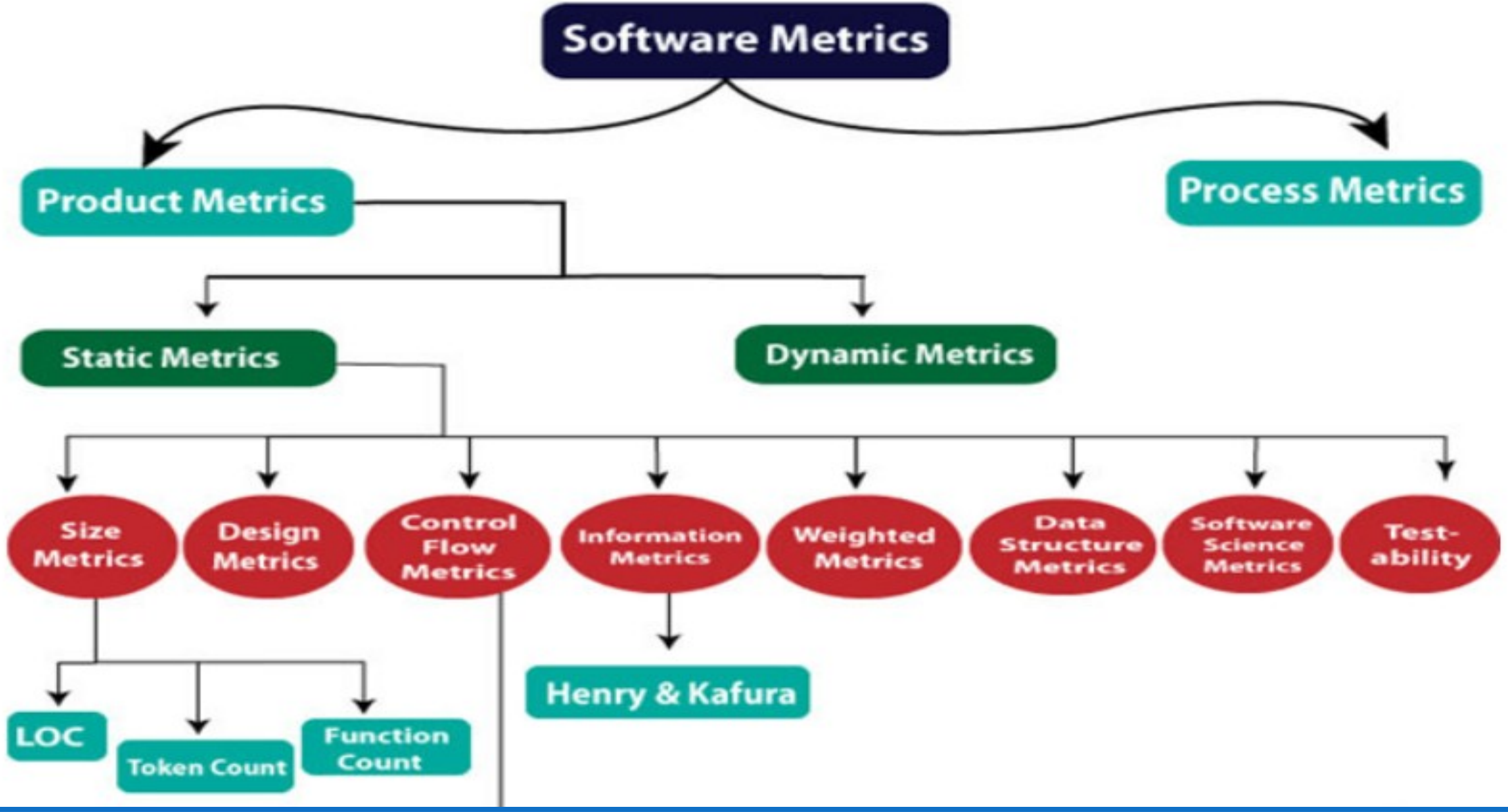
- Software metrics is a standard of measure that contains many activities which involve some degree of measurement.
- It can be classified into three categories:
  - product metrics,
  - process metrics, and
  - project metrics.
- But Some metrics belong to multiple categories. For example, the in-process quality metrics of a project are both process metrics and project metrics.

Cont,

- **Product metrics** describe the characteristics of the product such as size, complexity , design features, performance, and quality level.
- **Process metrics** are the measures of various characteristics of the software development process. can be used to improve software development and maintenance.
  - For example, the efficiency of fault detection. They are used to measure the characteristics of methods, techniques, and tools that are used for developing software.

Cont,

- **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
  - Project metrics are the metrics used by the project manager to check the project's progress. Data from the past projects are used to collect various metrics, like time and cost; these estimates are used as a base of new software.
    - these metrics are used to decrease the development costs, time efforts and risks.


Software Metrics
Product Metrics
Process Metrics
Static Metrics
Dynamic Metrics
Size Metrics
Design Metrics
Control Flow Metrics
Information Metrics
Weighted Metrics
Data Structure Metrics
Software Science Metrics
Test-ability
LOC
Token Count
Function Count
Henry & Kafura

Scope of Software Metrics

- Software metrics contains many activities which include the following:-
- **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

Cont,,

- **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.

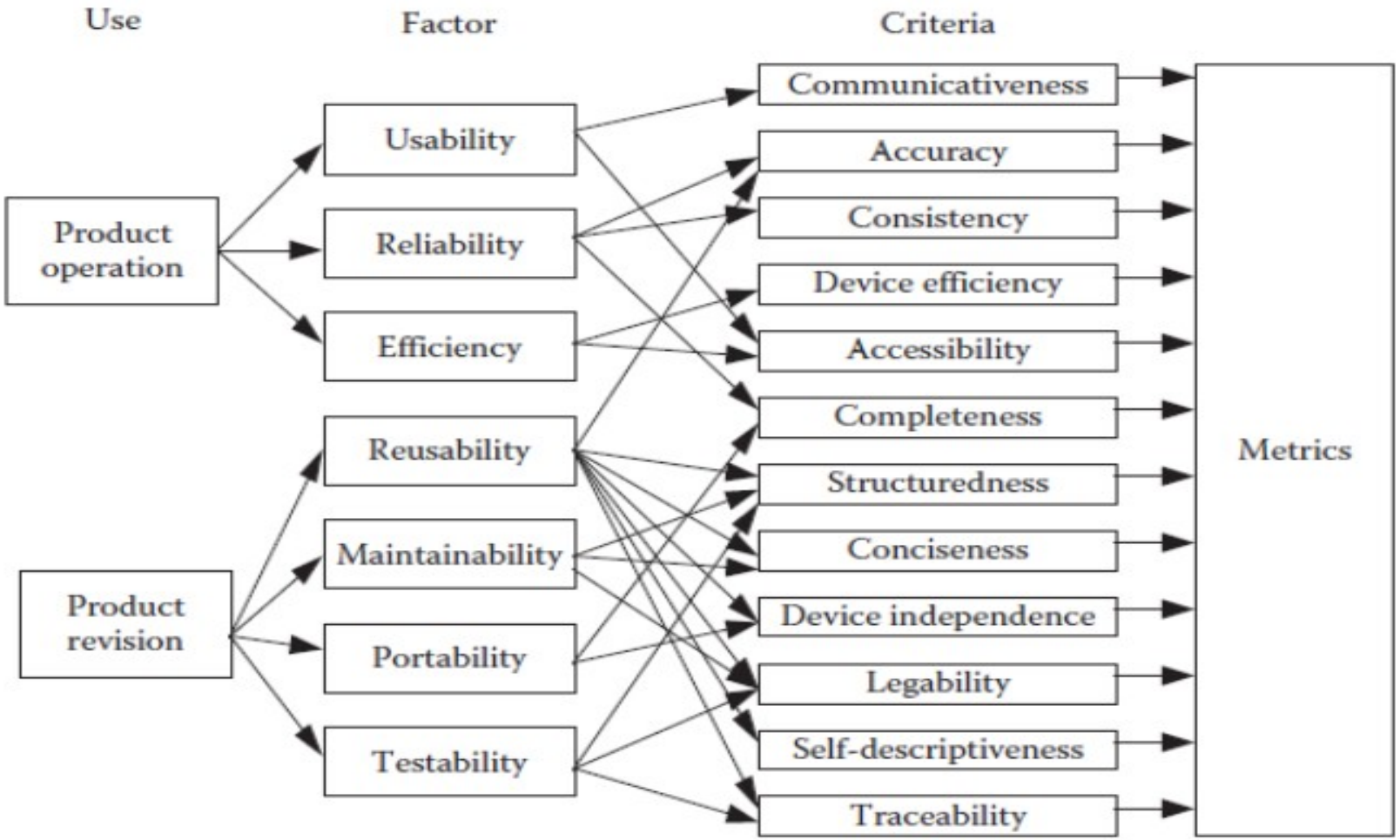
Cont,,

- **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.

Cont,,

- **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.
- 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

- The basic problem in reliability theory is to predict when a system will eventually fail.
- A **software reliability model** indicates the form of a random process that defines the behavior of **software** failures to time.
- **Software reliability models** have appeared as people try to understand the features of how and why **software** fails, and attempt to quantify **software reliability**.

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.

Cont,

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

Cont,,

- **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.
  - measurement is an important part of software project management. Customers and developers alike rely on measurement-based charts and graphs to help them decide if a project is on track.

- **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.

Question,,

- What product, process and project metrics do big tech companies (Google, Facebook, Netflix etc.) use?

Chapter two: Measurement basics

Contents

- Metrology
- Direct and Indirect Measurement
- Measurement quality
- Measurement scales and scale types
- Measuring Software Quality using Quality Metrics
- **Google’s HEART Framework for Measuring UX**
- Validating the Measurement Systems

Metrology

- Metrology is “the science of measurement, embracing both experimental and theoretical determinations at any level of uncertainty in any field of science and technology,” as defined by the International Bureau of Weights and Measures (BIPM, 2004).

Metrology..

- Metrology can be divided into three subfields: scientific metrology, applied metrology, and legal metrology.
- Legal metrology is the end of the line, concerning regulatory requirements of well established measurements and measuring instruments for the protection of consumers and fair trade.
- In applied metrology, the measurement science is developed toward manufacturing and other processes, ensuring the suitability of measurement instruments, their calibration, and quality control.
- Scientific metrology is the basis of all subfields, and concerns the development of new measurement methods, the realization of measurement standards, and the transfer of these standards to users.

What are the Software attributes and How we measure it?

- Unfortunately, we have no comparably deep understanding of software attributes.
- Nor do we have the associated sophisticated measurement tools.
- Questions that are relatively easy to answer for non-software entities are difficult for software. For example, consider the following questions:
- How much must we know about an attribute to consider measuring (e.g., program complexity)?

- Many systems consist of programs in a variety of languages. For example, the GNU/Linux distribution includes code written in at least 19 different languages (Wheeler 2002).

- In order to deal with code written in such a variety of languages, David Wheeler's code analysis tool uses a simple scheme for counting LOC:
- "a physical source line of code is a line ending in a newline or end-of-file marker, and which contains at least one non whitespace non-comment character.

Measuring lines of code

- Are non-executable lines counted?
  - Declarations
  - Compiler Directives
  - Comments
  - Blank lines

Examples of specific measures used in software engineering

	Entity	Attribute	Measure
1	Completed project	Duration	Months from start to finish
2	Completed project	Duration	Days from start to finish
3	Program code	Length	Number of lines of code (LOC)
4	Program code	Length	Number of executable statements
5	Integration testing process	Duration	Hours from start to finish
6	Integration testing process	Rate at which faults are found	Number of faults found per KLOC (thousand LOC)
7	Tester	Efficiency	Number of faults found per KLOC (thousand LOC)
8	Program code	Quality	Number of faults found per KLOC (thousand LOC)
9	Program code	Reliability	Mean time to failure (MTTF) in CPU hours
10	Program code	Reliability	Rate of occurrence of failures (ROCOF) in CPU hours

Direct and Indirect Measurement

- Direct measure – relates an attribute to a number or symbol without reference to no other object or attribute.
- *Direct measurement* of an attribute of an entity involves no other attribute or entity.
- For example, *length* of a physical object can be measured without reference to any other object or attribute.

Direct and Indirect Measurement

- Indirect measure -used when an attribute must be measured by combining several of its aspects (e.g., density).
- measures of the *density* of a physical object can be derived in terms of *mass* and *volume*; we then use a model to show us that the relationship between the three is *Density = Mass/Volume.*

Requires a model of how measures are related to each other

- Direct
  - *Size* of source code (measured by LOC)
  - *Schedule* of the testing process (measured by elapsed time in hours)
  - *Number of defects discovered* (measured by counting defects)
  - *Time* a programmer spends on a project (measured by months worked)
- Indirect/ derived measures that are commonly used in software engineering.
  - Programmer productivity (LOC/work months of effort)
  - Module defect density (number of defects/module size)
  - Defect detection efficiency (# defects detected/total defects)
  - Requirements stability (initial # requirements/total # requirements)
  - Test effectiveness ratio (number of items covered/total number of items)
  - System spoilage (effort spent fixing faults/total project effort)

- The most common of all, and the most controversial, is the measure for programmer productivity, as it emphasizes size of output without taking into consideration the code’s functionality or complexity.

- The defect detection efficiency measure is computed with respect to a specific testing or review phase; the total number of defects refers to the total number discovered during the entire product life cycle.
- Japanese software developers routinely compute the system spoilage measure; it indicates how much effort is wasted in fixing faults, rather than in building new code.

Measurement scale types

Reading Assignment

Nominal

Ordinal

Interval

Ratio

Absolute

1.Nominal

- define classes or categories, and place each category in a particular class or category
- nominal scale measurement places elements in a classification scheme. The classes are not ordered; even if the classes are numbered from 1 to *n* for identification, there is no implied ordering of the c

$$M_1(x) = \begin{cases} 1 \text{ if } x \text{ is specification fault} \\ 2 \text{ if } x \text{ is design fault} \\ 3 \text{ if } x \text{ is code fault} \end{cases}$$

$$M_2(x) = \begin{cases} 101 \text{ if } x \text{ is specification fault} \\ 2.73 \text{ if } x \text{ is design fault} \\ 69 \text{ if } x \text{ is code fault} \end{cases}$$

2.Ordinal scale

- Augments nominal scale with ordering information.
- We can often augment the nominal scale with information about an ordering of the classes or categories creating an *ordinal scale*.
- The ordering leads to analysis not possible with nominal measures.
- The ordinal scale has the following characteristics:

2.Ordinal scale

- The empirical relation system consists of classes that are ordered with respect to the attribute.
- Any mapping that preserves the ordering is acceptable.
- The numbers represent ranking only, so addition, subtraction, and other arithmetic operations have no meaning

2.Ordinal scale

- Example – software “complexity” – two valid measures

Value	Meaning
1	Trivial
2	Simple
3	Moderate
4	Complex
5	Incomprehensible

Value	Meaning
2	Trivial
4	Simple
6	Moderate
9	Complex
12	Incomprehensible

3.Interval scale

- Captures information about size of intervals that separate classes.
- Example – program length can be measured by lines of code, number of characters, etc. Number of characters may be obtained by multiplying the number of lines by the average number of characters per line

3.Interval scale

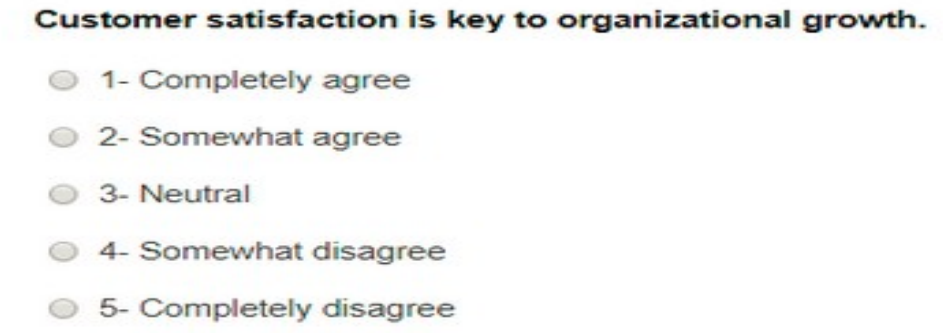
- We have seen how the ordinal scale carries more information about the entities than does the nominal scale, since ordinal scales preserve ordering. The interval scale carries more information still, making it more powerful than nominal or ordinal.
- This scale captures information about the size of the intervals that separate the classes, so that we can in some sense understand the size of the jump from one class to another.

3.Interval scale

- an interval scale can be characterized in the following way
  - An interval scale preserves order, as with an ordinal scale.
  - An interval scale preserves differences but not ratios. That is, we know the difference between any two of the ordered classes in the range of the mapping, but computing the ratio of two classes in the range does not make sense.
  - Addition and subtraction are acceptable on the interval scale, but not multiplication and division

3.Interval scale

- One of the most commonly used interval scale questions is arranged on a five-point Likert Scale question, where each emotion is denoted with a number, and the variables range from extremely dissatisfied to extremely satisfied.



4.Ratio Scale

- Although the interval scale gives us more information and allows more analysis than either nominal or ordinal, we sometimes need to be able to do even more.
- For example, we would like to be able to say that one liquid is twice as hot as another, or that one project took twice as long as another.
- This need for ratios gives rise to the ratio scale, the most useful scale of measurement, and one that is common in the physical sciences.

4.Ratio Scale

A *ratio scale* has the following characteristics:

- It is a measurement mapping that preserves ordering, the size of intervals between entities, and ratios between entities.
- There is a zero element, representing total lack of the attribute.
- The measurement mapping must start at zero and increase at equal intervals, known as units.
- All arithmetic can be meaningfully applied to the classes in the range of the mapping

4.Ratio Scale

Ratio Scale Examples

What is your height in feet and inches?

Less than 5 feet.

5 feet 1 inch – 5 feet 5 inches

5 feet 6 inches- 6 feet

More than 6 feet

5.Absolute scale

- The *absolute scale* has the following properties:
  - The measurement for an absolute scale is made simply by counting the number of elements in the entity set.
  - The attribute always takes the form “number of occurrences of *x* in the entity.”
  - There is only one possible measurement mapping, namely the actual count, and there is only one way to count elements

5.Absolute scale

- lines of code in a module is an absolute scale measure
- the number of failures observed during integration testing can be measured only in one way: by counting the number of failures observed.
- Likewise, the number of people working on a software project can be measured only in one way: by counting the number of people.

5.Absolute scale

- Since there is only one possible measure of an absolute attribute, the set of acceptable transformations for the absolute scale is simply the identity transformation.
- The uniqueness of the measure is an important difference between the ratio scale and absolute scale.

Measuring Software Quality using Quality Metrics

- In Software Engineering, Software Measurement is done based on some Software Metrics where these software metrics are referred to as the measure of various characteristics of a Software.
- In Software engineering Software Quality Assurance (SAQ) assures the quality of the software. Set of activities in SAQ are continuously applied throughout the software process. Software Quality is measured based on some software quality metrics.
- There is a number of metrics available based on which software quality is measured. But among them, there are few most useful metrics which are most essential in software quality measurement.

Measuring Software Quality using Quality Metrics..

1. Code Quality – Code quality metrics measure the quality of code used for the software project development. In code quality both Quantitative metrics like the number of lines, complexity, functions, rate of bugs generation, etc, and Qualitative metrics like readability, code clarity, efficiency, maintainability, etc are measured.

2. Reliability – Reliability metrics express the reliability of software in different conditions. The software is able to provide exact service at the right time or not is checked. Reliability can be checked using Mean Time Between Failure (MTBF) and Mean Time To Repair (MTTR).

Measuring Software Quality using Quality Metrics..

3. Performance – Performance metrics are used to measure the performance of the software. Each software has been developed for some specific purposes. Performance metrics measure the performance of the software by determining whether the software is fulfilling the user requirements or not, by analyzing how much time and resource it is utilizing for providing the service.

4. Usability – Usability metrics check whether the program is user-friendly or not. Each software is used by the end-user. So it is important to measure that the end-user is happy or not by using this software.

Measuring Software Quality using Quality Metrics..

5. Correctness – Correctness is one of the important software quality metrics as this checks whether the system or software is working correctly without any error by satisfying the user. Correctness gives the degree of service each function provides as per developed.

6. Maintainability – Each software product requires maintenance and up-gradation. Maintenance is an expensive and time-consuming process. So if the software product provides easy maintainability then we can say software quality is up to mark. Maintainability metrics include time requires to adapt to new features/functionality, Mean Time to Change (MTTC), performance in changing environments, etc.

Measuring Software Quality using Quality Metrics..

7. Integrity – Software integrity is important in terms of how much it is easy to integrate with other required software's which increases software functionality and what is the control on integration from unauthorized software's which increases the chances of cyberattacks.

8. Security – Security metrics measure how much secure the software is? In the age of cyber terrorism, security is the most essential part of every software. Security assures that there are no unauthorized changes, no fear of cyber attacks, etc when the software product is in use by the end-user.

Google’s HEART Framework for Measuring UX

- What the research team from Google noted was that while small scale frameworks were common place measuring the experience on a large scale via automated means had no framework in place.
- Thus the Heart Framework is specifically targeted at that kind of measurement.

Google’s HEART Framework for Measuring UX

The Heart Metrics

There are five metrics used in the HEART framework:

Happiness

Engagement

Adoption

Retention

Task Success

Google’s HEART Framework for Measuring UX

- Happiness

As you might expect this is a measure of attitude or satisfaction. You’re most likely to record satisfaction on large scale projects through some sort of user survey. An example cited in the report shows that change can impinge on happiness and that an initial drop in happiness following a change does not necessarily have long-term implications.

Google’s HEART Framework for Measuring UX

Engagement

This is a measure of how much a user interacts with a product, of their own volition. As we’ve noted already – it may be a poor metric for enterprise systems because there’s no optional element in usage patterns. If you’ve got to do a job; you’ve got to do the job – whether you like the tool you’re doing the job with or not.

Google’s HEART Framework for Measuring UX

Adoption

Adoption is defined as the number of new users over a certain time frame. It’s a measure of how successful you are at attracting new business. It might be argued, and we can certainly see how, that this is less a measure of user experience and more a measure of customer experience.

Google’s HEART Framework for Measuring UX

Retention

Retention, on the other hand, is a matter of keeping your existing users for x amount of time. That might be an indefinite amount of time for products with long-term utility. However, you’re probably going to want to look at other time scales to work out where drop out from a service is most pronounced so that you can tackle the UX issues that lead to that drop out. A week, a month, a quarter, a year, are all perfectly reasonable intervals as are any other intervals that you can justify as relevant to your business.

Google’s HEART Framework for Measuring UX

Task Success

task success” can be broken down into more subtle components. You might want to examine time spent on any given task (can the process be improved?) or the percentage of successful completions of a specific task once it has begun (e.g. checkout processes or registration processes).

Remote usability testing and benchmarking studies are recommended for measuring these on a large scale.


Thank
you