Chapter 10

Maintenance Measures

Overview

- Characteristics of Measurement Procedure (Empirical, Objective and Encodable)
- Software Measurement (Process, Product, Resource)
- Software Measure and Software Metric
- Objectives of Software Measurement (Evaluation, Control, Assessment, Improvement, Prediction)
- Some Example Measures
    - Size
    - Complexity
    - Quality
    - Understandability
    - Maintainability
    - Cost Estimation
- Guidelines for Selecting Maintenance Measures (clearly defined objectives, personal involvement, ease of use)

Characteristics of Measurement Procedure

A measurement procedure must demonstrate a number of characteristics like:

Empirical: The result of measurement should describe empirically established facts. Finkelstein [96] captured the importance of this when he said that the precise, concise, objective and empirical nature of measurement 'gives its primacy in science'.

Objective: During measurement, observations should be carried out with integrity, objectively, reliably, efficiently and without bias or ambiguity

Encodable: An attribute can be encoded or characterised using different symbols such as numbers and graphic representations. In this chapter, encoding will be restricted to numbers.

Software Measurement

Def: 'Process of objectively and empirically quantifying an attribute of a software system and the process connected with its development, use, maintenance and evolution'.

There are three software maintenance-related entities whose attributes can be subjected to measurement: process, product and resource

- A process is any software-related activity such as change analysis, specification, design, coding and testing.
- A resource is input to a process, for example personnel, hardware and software.
- A product is any intermediate or final output resulting from a software process such as system documentation (for example, specification and design), program listings, test data, source code and object code.

Software Measurement

A diagrammatic representation of the relationship between these three entities is presented in Figure

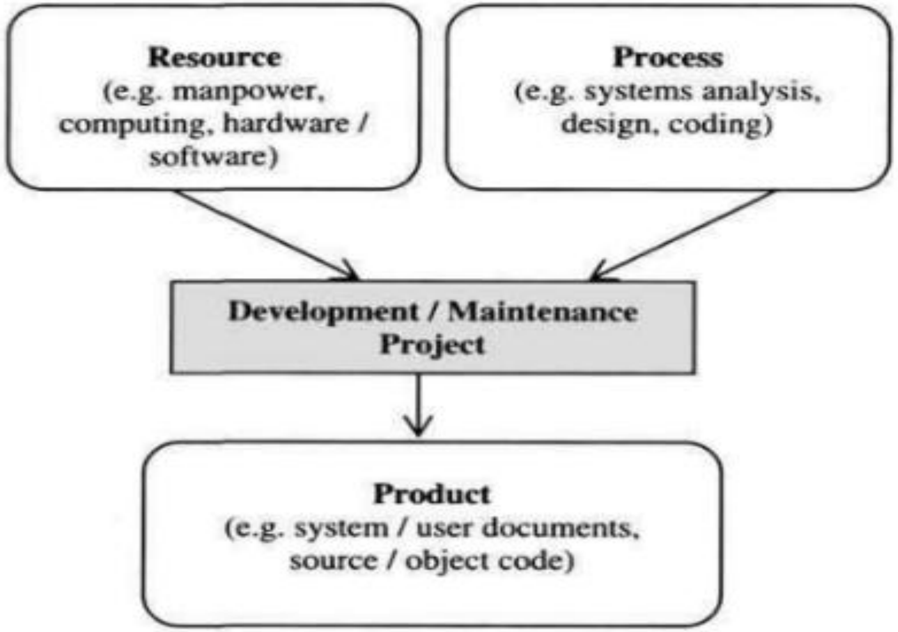

Resource
(e.g. manpower, computing, hardware / software)
Process
(e.g. systems analysis, design, coding)
Development / Maintenance Project
Product
(e.g. system / user documents, source / object code)

Figure 12.1 Relation between a resource, process and product

Software Measure and Software Metric

The result of a software measurement process is known as a software measure. The measure characterises the property of interest for a class of entities.

A measure is the assignment of these numbers (as in the above example) or any other selected symbol during measurement.

Two factors can be used to assess the maintainability of a program, P: understandability and modifiability.
Understandability is the time, Tl, required per module to determine the changes required.
Modifiability" is the time, T2, required per module to effect these changes.

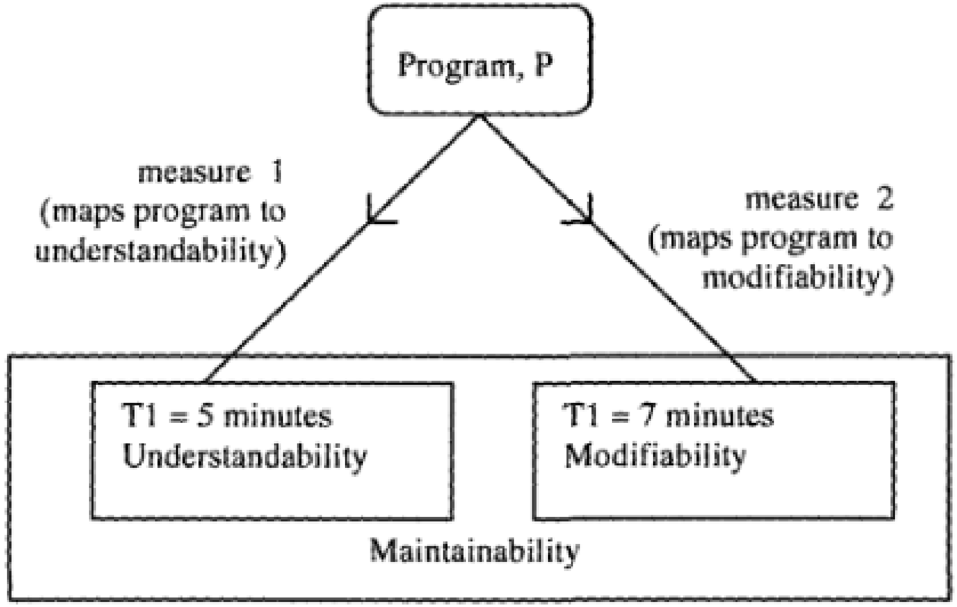


Figure 12.2 Relation between an entity, measure and attribute

Software Measure and Software Metric

Another term that has been widely used in the literature in connection with software measurement is software metric. It is sometimes used as a synonym for software measure.

The type of measure obtained during software measurement within software maintenance departments depends to some extent on the objective.

Objectives of Software Measurement

EPICA

Software measurement can be undertaken for several reasons, notably for evaluation, control, assessment, improvement and prediction.

1. **Evaluation**: There is a need for maintainers to evaluate different methods, program libraries and tools before arriving at a decision as to which is best suited to a given task. For instance, during reuse, the maintainer may be faced with the problem of evaluating the suitability of a candidate component obtained from the reuse library.
2. **Control**: There is a need to control the process of software change to ensure that change requests are dealt with promptly and within budget. As DeMarco says, "you cannot control what you cannot measure"
3. **Assessment**: In order to control a process or product, it is important to be able to assess or to characterise it first.
   For instance, a manager may need to assess a system to determine whether or not it is economically feasible to continue maintaining it.
   Also, in order to determine whether or not the maintenance process being used is achieving or will achieve the desired effect, an assessment of the process must be undertaken.

Objectives of Software Measurement

4. **Improvement**: There is a need to improve various characteristics of the software system or process such as quality and productivity. It is difficult to assess and monitor such improvements without an objective means of measuring the characteristics. With measures of quality and productivity, management is able to set targets and monitor progress in achieving those targets. If the targets are not met, then corrective actions can be taken.
5. **Prediction**: There is a need to make predictions about various aspects of the software product, process and cost.
   For instance, measures obtained from program code can be used to predict the time required to implement a given change.
   These measures can assist a manager in the allocation of time, personnel, hardware and software resources to a maintenance project. To a programmer, the measures can serve as indicators of the difficulty associated with understanding a program and can also signify the speed and accuracy with which change can be affected.

Some Example Measures

Most commonly used source of measures in the software system is the source code. So, code-based measures such as size, complexity, quality, understandability and maintainability are discussed here.

1. **Size**: One of the commonest ways of measuring the size of a program is by counting the number of lines of code. Moller and Paulish define lines of code (LOC) as "the count of program lines of code excluding comment or blank lines".
   The advantage of this measure is that it is easy to determine and also correlates strongly with other measures such as effort and error density.

   There are no standards for LOC measurement and it is dependent on the programming language in question. Also it is too simplistic and does not reflect cost or productivity. Despite these criticisms, this measure is still widely used.

Some Example Measures

2. **Complexity**:: According to Zuse, it is the difficulty of maintaining, changing and understanding programs. One of the major problems that software maintainers face is dealing with the increasing complexity of the source code that they have to modify.

   The more complex a program is, the more likely it is for the maintainer to make an error when implementing a change

   Over one hundred complexity measures have been proposed in the literature. For example, McCabe's cyclomatic complexity, Halstead's difficulty measure , Basili-Hutchen's measure, and Prather's measure.

Some Example Measures

2. **Complexity::** Two of the most popular code complexity measures: McCabe's cyclomatic complexity and Halstead's difficulty measure
    a. **McCabe's Cyclomatic Complexity:**
    McCabe views a program as a directed graph in which lines of program statements are represented by nodes and the flow of control between the statements is represented by the edges. McCabe's cyclomatic complexity (also known as the cyclomatic number) is the number of 'linearly independent' paths through the program (or flow graph) and this value is computed using the formula:.

$$v(F) = e - n + 2$$

where n = total number of nodes; e = total number of edges or arcs; and $v(F)$ is the cyclomatic number.

Some Example Measures

2. **Complexity::**
    a. McCabe's Cyclomatic Complexity:
    Advantages:
        i. It helps to identify highly complex programs that may need to be modified in order to reduce complexity.
        ii. The cyclomatic number can be used as an estimate of the amount of time required to understand and modify a program.
        iii. The flow graph generated can be used to identify the possible test paths during testing.

Some Example Measures

2. **Complexity::**
    a. McCabe's Cyclomatic Complexity:

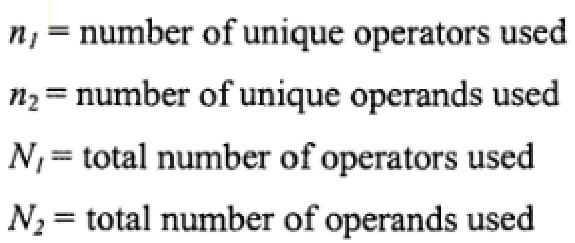
        Disadvantages:

        i. It takes no account of the complexity of the conditions in a program, for example multiple use of Boolean expressions, and over-use of flags.
        ii. In its original form, it failed to take account of the degree of nesting in a program. As such, two programs may have been considered to be equally complex based on cyclomatic number whereas in actual fact, one had a greater level of nesting than the other.

Some Example Measures

2. **Complexity::**
   b. Halstead's Measures:
   Halstead proposed a number of equations to calculate program attributes such as program length, volume and level, potential volume, language level clarity, implementation time and error rates.

   The measures for these attributes can be computed from four basic counts:



n_1 = number of unique operators used

n_2 = number of unique operands used

N_1 = total number of operators used

N_2 = total number of operands used

An operand is a variable or con

An operator is an entity that ca
change the value of an operanc
the order in which it is changed. Operators include arithmetic operators (for example, *, /, + and -), keywords (for example, PROCEDURE, WHILE, REPEAT and DO), logical operators (for example, greater than, equal to and less than), and delimiters.

Some Example Measures

2. **Complexity::**
   b. Halstead's Measures:
   The following formulae can be used to calculate the program length and program effort:
   - Observed program length, $N = N_1 + N_2$;
   - Calculated program length, $= n_1\log_2 n_1 + n_2\log_2 n_2$
   - Program effort, $E = \dfrac{n_1 * N_2 * (N_1 + N_2) * \log(n_1 + n_2)}{2 * n_2}$

The four basic counts can also be used to compute other attributes such as **programming time**, **number of bugs** in a program and **understandability**.

Some Example Measures

2. **Complexity::**
    b. Halstead's Measures:
    Advantages:
    i) Easy to calculate and do not require an in-depth analysis of programming features and control flow.

    ii) Can be applied to any language

    iii) Empirical evidence shows that these measures can be used as good predictors of programming effort and number of bugs

Some Example Measures

2. **Complexity::**
    b. Halstead's Measures:
    Disadvantages:
    i) Experiments which were used to test the measures were badly designed [179] and statistically flawed

    ii) Counting rules involved in the design of the measures were not fully defined

    iii) Failure to consider declarations and input/output statements as a unique operator for each unique label

    iv) measures are code-based; it is assumed that 'software = programs. Measures fail to capture the contribution that this documentation makes to the programming effort or program understanding.

Some Example Measures

3. **Quality:**
   Quality is defined as 'fitness for purpose'. It means it does what the user expects it to do.

   A quality maintenance process is one which enables the maintainer to implement the desired change. Different measures can be used to characterise product and process quality

   a. **Product Quality:**
      One way of measuring the quality of a software system is by keeping track of the number of change requests received from the users after the system becomes operational. This measure is computed by
      (number of unique change requests made by customers for the first year of field use of a given release) / (the number of thousand lines of code)

      The measure excludes feature enhancement change requests which are not contained in the software requirements specification.
      The other measure of product quality is the number of faults that are detected.

Some Example Measures

3. **Quality:**
   b. **Process Quality:**
   This describes the degree to which the maintenance process being used is assisting personnel in satisfying change requests. Two measures of process quality are schedule and productivity

   Schedule: The difference between the planned and actual work time to achieve the milestone of first customer delivery, divided by the planned work time"
   This measure is expressed as a percentage.
   A negative number signifies a slip and a positive number signifies early delivery.

   Productivity: It is computed by dividing the number of lines of code that have been added or modified by the effort in staff days required to make the addition or modification.
   Effort is the total time from analysing the change requests to a successful implementation of the change.

Some Example Measures

4. **Understandability**:
   Program understandability is the ease with which the program can be understood, that is, the ability to determine what a program does and how it works by reading its source code and accompanying documentation.

   It also includes other external factors such as the available documentation, the maintenance process and maintenance personnel.

   Understandability usually has an **inverse relation to complexity;** as the complexity of a program increases, the understandability tends to decrease. From this perspective, understandability can be **computed indirectly from McCabe's cyclomatic complexity and Halstead's program** effort measure. Understandability can also be estimated from **subjective ratings of the quality of documentation**, **consistency of programming style** and the **conciseness of the program text**.

Some Example Measures

5. **Maintainability:**
   Software maintainability is "the ease with which the software can be understood, corrected, adapted, and/or enhanced"

   Maintainability is an external attribute since its computation requires knowledge from the software product as well as external factors such as the maintenance process and the maintenance personnel.

   An example of a maintainability measure that depends on an external factor is the Mean Time To Repair (MTTR): the mean time required to effect a change

   Maintainability can also be perceived as an internal attribute if it is derived solely from the software system. Several internal attributes of program source code can impact on maintainability, for example modularity. Thus, measures for these other internal attributes would need to be obtained in order to compute maintainability.

Some Example Measures

6. **Cost Estimation:**

   The cost of a maintenance project is the resources - personnel, machines, time and money - expended on effecting change.

   One way of estimating the cost of a maintenance task is from historical data collected for a similar task.

   A second way of estimating cost is through mathematical models. One of these was Boehm's COCOMO model adapted for maintenance.

   According to Boehm, the cost of maintenance is affected by attributes of factors called cost drivers. Examples of cost drivers are database size, program complexity, use of modern programming practices and applications experience of the maintenance personnel.

   A third measure of cost is time in person-months required to modify a program.

Guidelines for Selecting Maintenance Measures

The main purpose of maintenance activities is to ensure that a software system can be easily modified, adapted and enhanced to accommodate changes.

Proposed guidelines include well-defined objectives, fitness for purpose, ease of use, low implementation cost and sensitivity.

- **Clearly defined objectives**: Prior to deciding on the use of a measurement for maintenance-related purposes, it is essential to define clearly and unambiguously what objectives need to be achieved. These objectives will determine the measures to be used and the data to be collected.
- **Personnel involvement**: The purpose of measurement in an organisation needs to be made clear to those involved in the programme. However objectives should be clearly defined before, else personnel may feel that the measures will be used for punitive purposes and this can impede the programme.

Guidelines for Selecting Maintenance Measures

- **Ease of use:**
  The measures that are finally selected to be used need to be easy to use, take not too much time to administer, be unobtrusive, and possibly subject to automation. As indicated earlier in this chapter, one of the reasons why source code-based measures are very popular is because they can be easily automated and collected in an unobtrusive way.