Measuring Internal Product Attributes

Code Size: This is an internal product attribute, by measuring the code size of any software, measure the software size. There are many techniques used for measuring the code size of the software.

1. **Counting Lines of Code to Measure Code Size**

   Total size:

   LOC=NCLOC+CLOC

   In all languages, using a compiler.

   Total code size measure using this way:

   - Count of the number of physical lines (including blank lines)
   - Count of all lines except blank lines and comments
   - Count of all statements except comments (statements taking more than one line count as only one line)
   - Count of all lines except blank lines, comments, declarations, and headings
   - Count of all statements except blank lines, comments, declarations, and headings
   - Count of only the ESs, not including exception conditions

2. **Halstead's Approach**

   This is another approach for code size measuring.

   $\mu 1$ = Number of unique operators

   $\mu 2$ = Number of unique operands

   N1 = Total occurrences of operators

   N2 = Total occurrences of operands

   $V = N \times \log 2\mu$

   where $\mu = (\mu 1 + \mu 2)$ and $N=(N1+N2)$

   We use a program to calculate it easily

3. **Number of bytes of computer storage:** We can build a program for calculating the total byte of computer storage for measuring code size.
4. **Number of characters (CHAR):**

   $CHAR = \alpha\ LOC$

   we can use KLOC (thousands of LOCs) or KDSI (thousands of delivered source instructions) to measure program size.

Design size:

- **Packages:** Number of sub packages, number of classes, interfaces (Java), or abstract classes (C++)
- **Design patterns:**

- ❖ Number of different design patterns used in a design
- ❖ Number of design pattern realizations for each pattern type
- ❖ Number of classes, interfaces, or abstract classes that play roles in each pattern realization

- ➢ **Classes, interfaces, or abstract classes:** Number of public methods or operations, number of attributes
- ➢ **Methods or operations:** Number of parameters, number of overloaded versions of a method or operation

Requirement's analysis and specification size:

- ❖ **Use case diagrams:** The number of use cases, actors, and relationships of various types indicate the requirements analysis and specification size, we can measure it manually easily.
- ❖ **Use case:** Number of scenarios, size of scenarios in terms of steps, or activity diagram model elements indicate the requirements analysis and specification size, we can measure it manually easily.
- ❖ **UML class diagram:** The number of classes, Actors and Relationships of various types indicate the requirements analysis and specification size, we can measure it manual easily.

STRUCTURAL MEASURES

Structural attributes are measured with the help of complexity, length, coupling, and cohesion.

We can think of structure from at least two perspectives:

1. Control flow structure: measure using graph

2. Data flow structure

Using control Flow graph

Depth of expression

- ✓ ▪ Primes: if F is a prime ≠ P1, then α((F) = 1.
- ✓ ▪ Sequence: α (F1; …; Fn) = max (α(F1); …; α(Fn))
- ✓ ▪ Nesting: α (F (F1, …, Fn)) = 1 + max(α(F1), …, α(Fn))

Cyclomatic Complexity Measure: the number of linearly independent paths.

v(F) = e − n + 2 or, v(F) = 1 + d ev(F) = v(F) – m

Here, e = edges. n = nodes. d = predicate nodes. Ev(F) = essential complexity. M = number of sub-flowgraphs.

Measure procedure: Programmatic / Manually. Requirement: Control Flowgraph.

Test Effectiveness Ratio (TER): the extent to which the test cases satisfy a particular testing strategy. TER = (Number of requirements executed at least once) / (Total number of test requirements for criterion)

Design level attribute:

We find out the level of design of software system for knowing the design output like as design is bad or good. We follow more approaches for measuring the design level such as Models of Modularity and Information Flow, Global Modularity, Morphology, Tree Impurity, Internal Reuse, Information Flow, etc.

Models of Modularity and Information Flow: we can find out all of the modules within the system and show the information flow from one module to another module. We can see the dependency modules when the information will be shared among the modules. It will be easy to see the manual from the software system.

Global Modularity: "Global modularity" is difficult to define because there are many different views of what modularity means. For example, consider average module length as an intuitive measure of global modularity. metrics was developed to calculate the average size of program modules as a measure of structuredness. We can measure global modularity using the program for automated calculation.

Morphology:

The "shape" of the overall system structure when expressed pictorially

- Size: Measured as number of nodes, number of edges, or a combination of these. Measure procedure: Programmatic. Requirement: Graph / program module.
- Depth: Measured as the length of the longest path from the root node to a leaf node. Measure procedure: Programmatic. o Requirement: Graph / program module.
- Width: Measured as the maximum number of nodes at any one level. Measure procedure: Manually. Requirement: Graph / program module.
- Ratio: Ratio = edges / node.

Measure procedure: Programmatic. Requirement: Graph / program module.

Tree Impurity: The more a system deviates from being a pure tree structure towards being a graph structure, the worse the design is … it is one of the few system design metrics* to have been validated on a real project. We can easily measure it using the following function

$$G(m)=2(e-n+2)/(n-1)(n-2)$$

When we find out the graph manually then we use that information to estimate the above function automated.

Internal Reuse: We call internal reuse the extent to which modules within a product are used multiple times within the same product. We use the following method to estimate the Internal Reuse

$$R(r)=(e-n+1)$$

The above method will be automated when the graph's edge and nodes find out the manually checking.

Coupling in Object-Oriented Systems:

connections between elements from one module to others. Instability metric,

$$I = Ce / Ca + Ce.$$

Here, Ce = Efferent coupling. [Fan-in], Ca = Afferent coupling. [Fan-out].

Cohesion in Object-Oriented Systems:

connection between elements in an individual module.

$$TCC(C) = NDC(C)/NP(C)$$

$$LCC(C) = (NDC(C) + NIC(C))/NP(C)$$

$$RC(P) = (R(P) + 1)/N(P)$$

Here, TCC = Tight Class Cohesion LCC = Loose Class Cohesion NDC = Number of Direct Cohesion NIC = Number of Indirect Connection NP = Number of Possible Connections RC = Relational Cohesion R(P) = number of relations between classes and interfaces N(P) = number of classes and interfaces in the package.

In OOP language measure manually.