UNIT 3rd

SOFTWARE METRICS

Software metric is a measure of software characteristics which are quantifiable or countable. Software metrics are important for many reasons, including measuring software performance planning work items, measuring productivity, and many other uses.

The goal of tracking and analysing software metrics is to determine the quality of the current product or process, improve that quality and predict the quality once the software development project is complete.

Increase return on investment (ROI)

Identify areas of improvement

Manage workloads

Reduce overtime

Reduce costs

These goals can be achieved by providing information and clarity throughout the organization about complex software development projects. Metrics are an important component of quality assurance, management, debugging, performance, and estimating costs, and they're valuable for both developers and development team leaders:

Managers can use software metrics to identify, prioritize, track and communicate any issues to foster better team productivity. This enables effective management and allows assessment and prioritization of problems within software development projects. The sooner managers can detect software problems, the easier and less-expensive the troubleshooting process.

Software development teams can use software metrics to communicate the status of software development projects, pinpoint and address issues, and monitor, improve on, and better manage their workflow.

Halstead's Software Metrics

According to Halstead's "A computer program is an implementation of an algorithm considered to be a collection of tokens which can be classified as either operators or operand."

Token Count

In these metrics, a computer program is considered to be a collection of tokens, which may be classified as either operators or operands. All software science metrics can be defined in terms of these basic symbols. These symbols are called as a token.

The basic measures are

n1 = count of unique operators.
n2 = count of unique operands.
N1 = count of total occurrences of operators.
N2 = count of total occurrence of operands.

In terms of the total tokens used, the size of the program can be expressed as N = N1 + N2.

Halstead metrics are:

Program Volume (V)

The unit of measurement of volume is the standard unit for size "bits." It is the actual size of a program if a uniform binary encoding for the vocabulary is used.

V=N*log2n

Program Level (L)

The value of L ranges between zero and one, with L=1 representing a program written at the highest possible level (i.e., with minimum size).

L=V*/V

Program Difficulty

The difficulty level or error-proneness (D) of the program is proportional to the number of the unique operator in the program.

D= (n1/2) * (N2/n2)

Programming Effort (E)

The unit of measurement of E is elementary mental discriminations.

E=V/L=D*V

Estimated Program Length

According to Halstead, The first Hypothesis of software science is that the length of a well-structured program is a function only of the number of unique operators and operands.

N=N1+N2

And estimated program length is denoted by N^

N^ = n1log2n1 + n2log2n2

The following alternate expressions have been published to estimate program length:

NJ = log2 (n1!) + log2 (n2!)

NB = n1 * log2n2 + n2 * log2n1

NC = n1 * sqrt(n1) + n2 * sqrt(n2)

NS = (n * log2n) / 2

Potential Minimum Volume

The potential minimum volume V* is defined as the volume of the most short program in which a problem can be coded.

V* = (2 + n2*) * log2 (2 + n2*)

Here, n2* is the count of unique input and output parameters

Size of Vocabulary (n)

The size of the vocabulary of a program, which consists of the number of unique tokens used to build a program, is defined as:

n=n1+n2

Where

n=vocabulary of a program
n1=number of unique operators
n2=number of unique operands

Halstead's Complexity Measures

Howard Halstead introduced metrics to measure software complexity. Halstead's metrics depends upon the actual implementation of program and its measures, which are computed directly from the operators and operands from source code, in static manner. It allows to evaluate testing time, vocabulary, size, difficulty, errors, and efforts for C/C++/Java source code.

According to Halstead, "A computer program is an implementation of an algorithm considered to be a collection of tokens which can be classified as either operators or operands". Halstead metrics think a program as sequence of operators and their associated operands.

He defines various indicators to check complexity of module.

Parameter	Meaning
n1	Number of unique operators
n2	Number of unique operands
N1	Number of total occurrence of operators

N2	Number of total occurrence of operands

When we select source file to view its complexity details in Metric Viewer, the following result is seen in Metric Report:

Metric	Meaning	Mathematical Representation
n	Vocabulary	n1 + n2
N	Size	N1 + N2
V	Volume	Length * Log2 Vocabulary
D	Difficulty	(n1/2) * (N1/n2)
E	Efforts	Difficulty * Volume
B	Errors	Volume / 3000
T	Testing time	Time = Efforts / S, where S=18 seconds.

Cyclomatic Complexity Measures

Every program encompasses statements to execute in order to perform some task and other decision-making statements that decide what statements need to be executed. These decision-making constructs change the flow of the program.

If we compare two programs of same size, the one with more decision-making statements will be more complex as the control of program jumps frequently.

McCabe, in 1976, proposed Cyclomatic Complexity Measure to quantify complexity of given software. It is graph driven model that is based on decision-making constructs of program such as if-else, do-while, repeat-until, switch-case and go to statements.

Process to make flow control graph:

- Break program in smaller blocks, delimited by decision-making constructs.
- Create nodes representing each of these nodes.
- Connect nodes as follows:
- If control can branch from block i to block j
- Draw an arc
- From exit node to entry node
- Draw an arc.

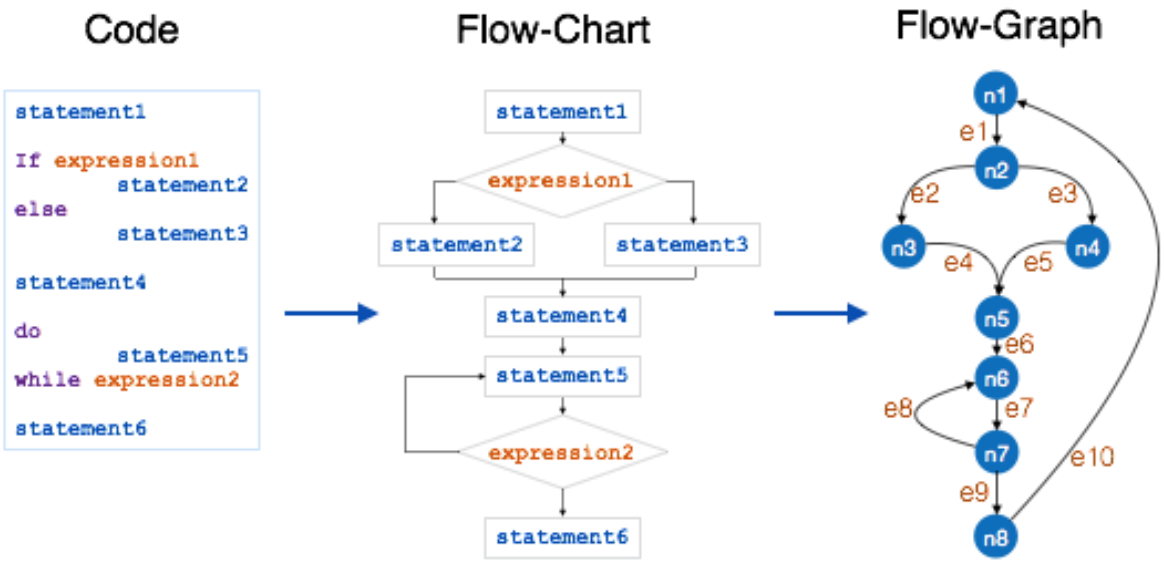
To calculate Cyclomatic complexity of a program module, we use the formula -

V (G) = e – n + 2

Where

e is total number of edges

n is total number of nodes



The Cyclomatic complexity of the above module is

e = 10

n = 8

Cyclomatic Complexity = 10 - 8 + 2

= 4

According to P. Jorgensen, Cyclomatic Complexity of a module should not exceed 10.