- **Strive for consistency** - Consistent sequences of actions should be required in similar situations. Identical terminology should be used in prompts, menus, and help screens. Consistent commands should be employed throughout.
- **Enable frequent users to use short-cuts** - The user's desire to reduce the number of interactions increases with the frequency of use. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.
- **Offer informative feedback** - For every operator action, there should be some system feedback. For frequent and minor actions, the response must be modest, while for infrequent and major actions, the response must be more substantial.
- **Design dialog to yield closure** - Sequences of actions should be organized into groups with a beginning, middle, and end. The informative feedback at the completion of a group of actions gives the operators the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans and options from their minds, and this indicates that the way ahead is clear to prepare for the next group of actions.
- **Offer simple error handling** - As much as possible, design the system so the user will not make a serious error. If an error is made, the system should be able to detect it and offer simple, comprehensible mechanisms for handling the error.
- **Permit easy reversal of actions** - This feature relieves anxiety, since the user knows that errors can be undone. Easy reversal of actions encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data entry, or a complete group of actions.
- **Support internal locus of control** - Experienced operators strongly desire the sense that they are in charge of the system and that the system responds to their actions. Design the system to make users the initiators of actions rather than the responders.
- **Reduce short-term memory load** - The limitation of human information processing in short-term memory requires the displays to be kept simple, multiple page displays be consolidated, window-motion frequency be

reduced, and sufficient training time be allotted for codes, mnemonics, and sequences of actions.

Chapter Nine
Software Design Complexity

The term complexity stands for state of events or things, which have multiple interconnected links and highly complicated structures. In software programming, as the design of software is realized, the number of elements and their interconnections gradually emerge to be huge, which becomes too difficult to understand at once.

Software design complexity is difficult to assess without using complexity metrics and measures. Let us see three important software complexity measures.

Halstead's Complexity Measures

In 1977, Mr. Maurice 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 a 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 goto 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.

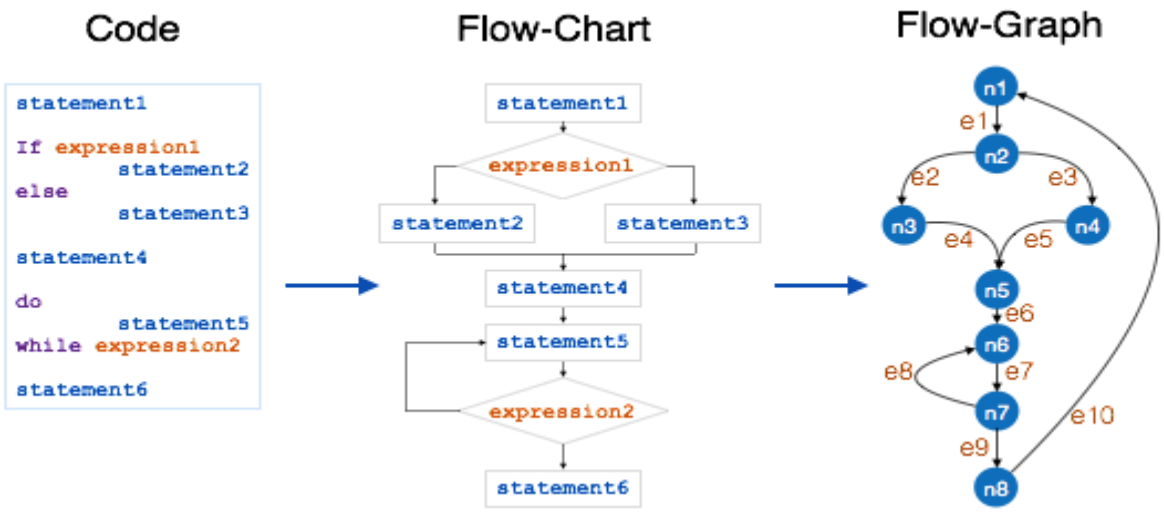
To calculate Cyclomatic complexity of a program module, we use the formula -

$$V(G) = e - n + 2$$

Where

e is total number of edges

n is total number of nodes



The Cyclomatic complexity of the above module is

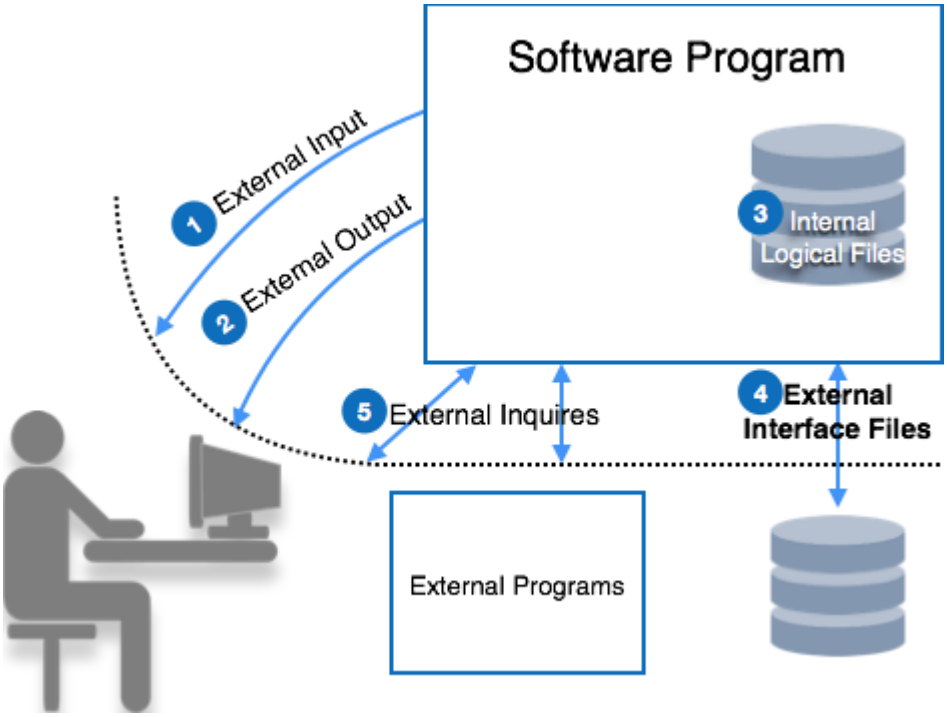
```
e = 10

n = 8

Cyclomatic Complexity = 10 - 8 + 2

                      = 4
```

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

Function Point

It is widely used to measure the size of software. Function Point concentrates on functionality provided by the system. Features and functionality of the system are used to measure the software complexity.

Function point counts on five parameters, named as External Input, External Output, Logical Internal Files, External Interface Files, and External Inquiry. To consider the complexity of software each parameter is further categorized as simple, average or complex.



Let us see parameters of function point:

External Input

Every unique input to the system, from outside, is considered as external input. Uniqueness of input is measured, as no two inputs should have same formats. These inputs can either be data or control parameters.

- **Simple** - if input count is low and affects less internal files
- **Complex** - if input count is high and affects more internal files
- **Average** - in-between simple and complex.

External Output

All output types provided by the system are counted in this category. Output is considered unique if their output format and/or processing are unique.

- **Simple** - if output count is low
- **Complex** - if output count is high
- **Average** - in between simple and complex.

Logical Internal Files

Every software system maintains internal files in order to maintain its functional information and to function properly. These files hold logical data of the system. This logical data may contain both functional data and control data.

- **Simple** - if number of record types are low
- **Complex** - if number of record types are high
- **Average** - in between simple and complex.

External Interface Files

Software system may need to share its files with some external software or it may need to pass the file for processing or as parameter to some function. All these files are counted as external interface files.

- **Simple** - if number of record types in shared file are low
- **Complex** - if number of record types in shared file are high
- **Average** - in between simple and complex.

External Inquiry

An inquiry is a combination of input and output, where user sends some data to inquire about as input and the system responds to the user with the output of inquiry processed. The complexity of a query is more than External Input and External Output. Query is said to be unique if its input and output are unique in terms of format and data.

- **Simple** - if query needs low processing and yields small amount of output data
- **Complex** - if query needs high process and yields large amount of output data
- **Average** - in between simple and complex.

Each of these parameters in the system is given weightage according to their class and complexity. The table below mentions the weightage given to each parameter:

Parameter	Simple	Average	Complex
Inputs	3	4	6
Outputs	4	5	7
Enquiry	3	4	6
Files	7	10	15
Interfaces	5	7	10

The table above yields raw Function Points. These function points are adjusted according to the environment complexity. System is described using fourteen different characteristics:

- Data communications
- Distributed processing
- Performance objectives
- Operation configuration load

- Transaction rate
- Online data entry,
- End user efficiency
- Online update
- Complex processing logic
- Re-usability
- Installation ease
- Operational ease
- Multiple sites
- Desire to facilitate changes

These characteristics factors are then rated from 0 to 5, as mentioned below:

- No influence
- Incidental
- Moderate
- Average
- Significant
- Essential

All ratings are then summed up as N. The value of N ranges from 0 to 70 (14 types of characteristics x 5 types of ratings). It is used to calculate Complexity Adjustment Factors (CAF), using the following formulae:

```
CAF = 0.65 + 0.01N
```

Then,

```
Delivered Function Points (FP)= CAF x Raw FP
```

This FP can then be used in various metrics, such as:

Cost = $ / FP

Quality = Errors / FP