Parul® University

Software Engineering (303105254)

Parul® University

UNIT-5

Coding and Unit Testing

Parul® University

Programming Principles and Guidelines

1. Select data structures that will meet the need of the design.
2. Keep conditional logic as simple as possible.
3. Understand the software architecture and make interfaces that are according to it.
4. Select meaningful variable names and follow other local coding standards.
5. Write code that's self-documenting.
6. Create a visual layout.
7. Constrain your algorithm by structured programming practice.

Programming Practices

- **Control construct**

  The single entry and exit constructs need to be used.
- **Use of goto**

  The goto statements make the program unstructured. So avoid use of goto statements as possible.
- **Information hiding**
- **Nesting**

  Structure inside another structure is called as nesting. If there is too deep nesting then it becomes hard to understand the code as well as complex.
- **User defined data types**

  User can define data type to enhance the readability of the code.

Image source : Google

Programming Practices (Contd.)

- **Modular size**
  The size of program may be large or small. There is no rule for size of the program. So as possible generate different module but not of large size.
- **Side effects**
  Sometimes if some part of code may change then it may generate some kind of problems called as side effects.
- **Robustness**
  If any kind of exception is generated, the program should generate some kind of output. Then it is called as robustness. In this situation the programs do not crash.
- **Switch case with defaults**
  Inside the switch case statement if any value which is unpredictable is given as argument then there should be default case to execute it

Parul® University

Coding Standards

Good software development organizations normally require their programmers to adhere to some well-defined and standard style of coding called coding standards.

Parul® University

Coding Standards (Contd.)

- Most software development organizations formulate their own coding standards that suit them most, and need their engineers to follow these standards strictly.
- The purpose of requiring all engineers of an organization to adhere to a standard style of coding is the following:

A coding standard gives a uniform appearance to the codes written by different engineers.

It enhances code understanding.

It encourages good programming practices.

Coding Standards (Contd.)

- **A coding standard lists several rules to be followed such as, the way variables are to be named, the way the code is to be laid out, error return conventions, etc.**
- **The following are some representative coding standards:**

Rules for limiting the use of global
These rules list what types of data can be declared global and what cannot.

Naming conventions for global & local variables & constant identifiers
A possible naming convention can be that global variable names always start with a capital letter, local variable names are made of small letters, and constant names are always capital letters.

Parul® University

Coding Standards (Contd.)

Contents of the headers preceding codes for different modules

- **The information contained in the headers of different modules should be standard for an organization.**
- **The exact format in which the header information is organized in the header can also be specified.**

The following are some standard header data
Module Name Creation Date Author’s Name Modification history Synopsis of the module Global variables accessed/modifiedby the module Different functions supported, along with their input/output parameters

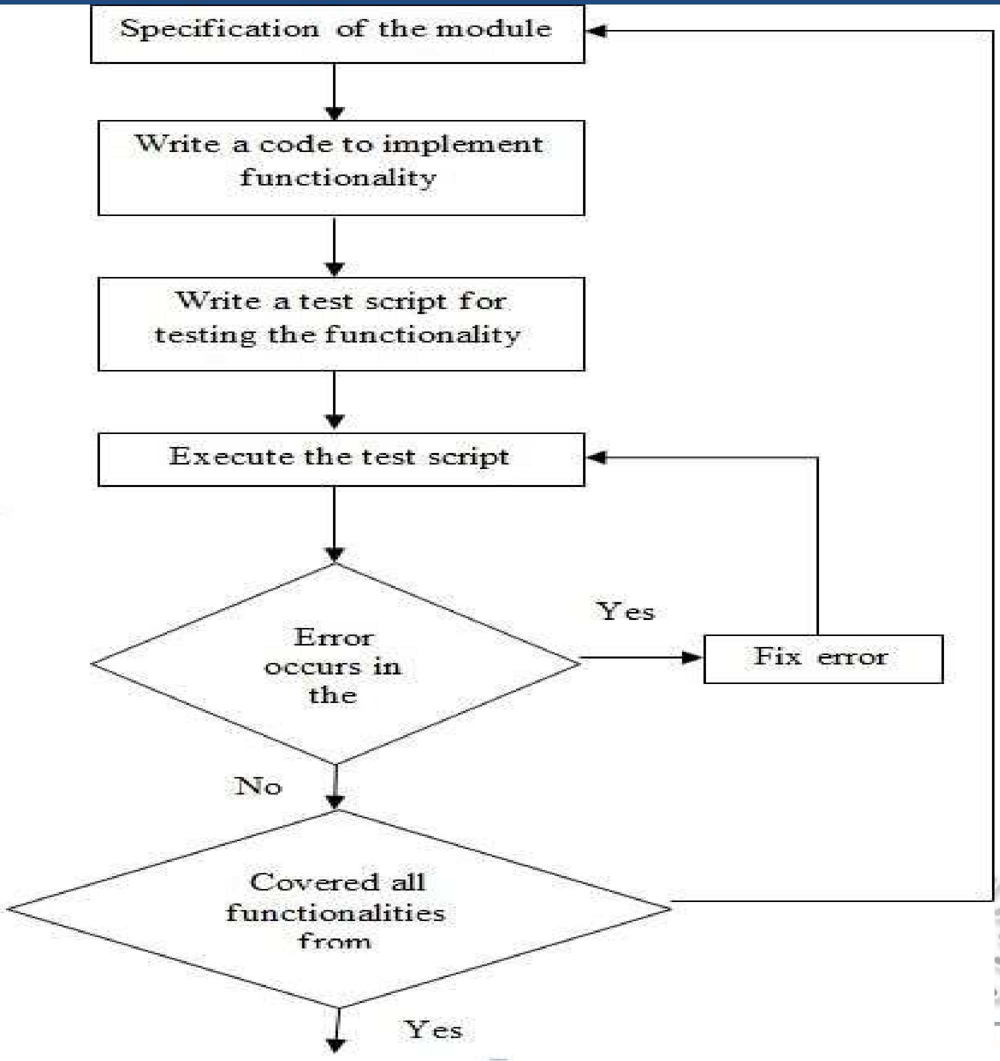
Coding Standards (Contd.)

Error return conventions and exception handling mechanisms

- **The way error conditions are reported by different functions in a program are handled should be standard within an organization.**
- **For example, different functions while encountering an error condition should either return a 0 or 1 consistently.**

Incremental Development of Code

- Then test that code based on some cases then execute the test script.
- It should be checked that any kind of errors are generated then fix the errors.
- If any kind of errors are not generated then covered all the functionalities mentioned in the specification, the process is terminated.
- Each and every functionality is written and immediately tested is one of its advantages


Specification of the module
Write a code to implement functionality
Write a test script for testing the functionality
Execute the test script
Error occurs in the
Yes
Fix error
No
Covered all functionalities from
Yes

Coding Guidelines

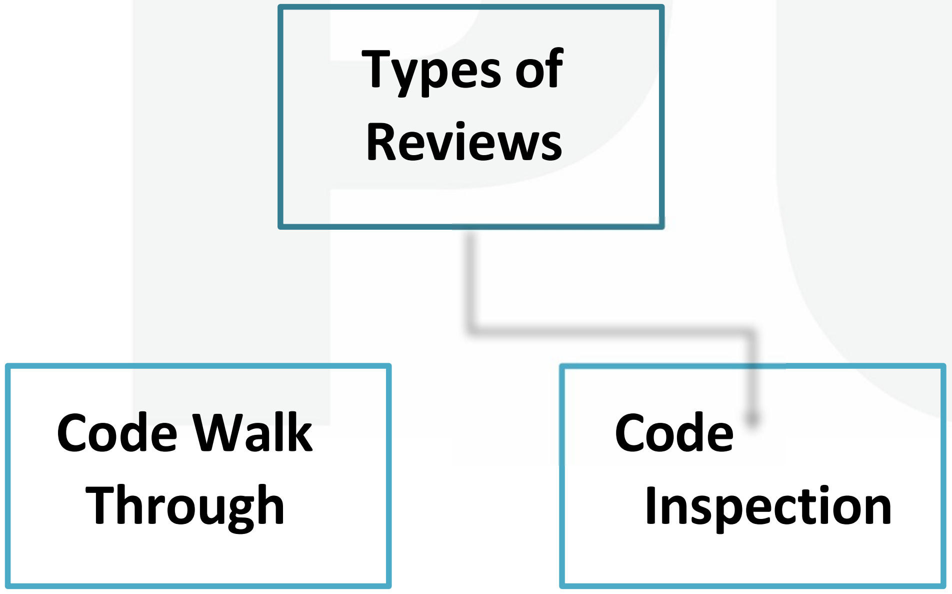
- The following are some representative coding guidelines
- Do not use a coding style that is too clever or too difficult to understand
- The code should be well-documented
- The length of any function should not exceed 10 source lines
- Do not use goto statements

The Types of Faults

Algorithmic	Logic is wrong Code reviews
Syntax	Wrong syntax; typos Compiler
Computation/ Precision	Not enough accuracy
Documentation	Misleading documentation
Stress/Overload	Maximum load violated
Capacity/Boundary	Boundary cases are usually special cases
Timing/Coordination	Synchronization issues Very hard to replicate
Throughput/Performance	System performs below expectations
Recovery	System restarted from abnormal state
Hardware & related software	Compatibility issues
Standards	Makes for difficult maintenance

Code Review

- CodeReview is carried out after the module is successfully compiled and all the syntax errors have been eliminated.
- Code Reviews are extremely cost-effective strategies for reduction in coding errors and to produce high quality code.



Parul® University

Code Walk Through

- Code walk through is an informal code analysis technique.
- The main objectives of the walk through are to discover the algorithmic and logical errors in the code.
- A few members of the development team are given the code few days before the walk through meeting to read and understand code.
- Each member selects some test cases and simulates execution of the code by hand
- The members note down their findings to discuss these in a walk through meeting where the coder of the module is present.

Code Inspection

- The aim of Code Inspection is to discover some common types of errors caused due to improper programming.
- In other words, during Code Inspection the code is examined for the presence of certain kinds of errors.
    - For instance, consider the classical error of writing a procedure that modifies a parameter while the calling routine calls that procedure with a constant actual parameter.
    - It is more likely that such an error will be discovered by looking for these kinds of mistakes in the code.
    - In addition, commitment to coding standards is also checked.

Few Classical Programming Errors

- Use of uninitialized variables
- Jumps into loops
- Non terminating loops
- Array indices out of bounds
- Improper storage allocation and de-allocation
- Mismatches between actual and formal parameter in procedure calls
- Use of incorrect logical operators or incorrect precedence among operators
- Improper modification of loop variables

Software Documentation

- When various kinds of software products are developed, various kinds of documents are also developed as part of any software engineering process e.g.
  - Users' manual,
  - Software requirements specification (SRS) documents,
  - Design documents,
  - Test documents,
  - Installation manual, etc

Software Testing

Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user.

Don’t view testing as a “safety net” that will catch all errors that occurred because of weak software engineering practice.

Who tests the Software?

Developer

Understands the system but, will test "gently"
and, is driven by "delivery"

Testing without plan is of no point

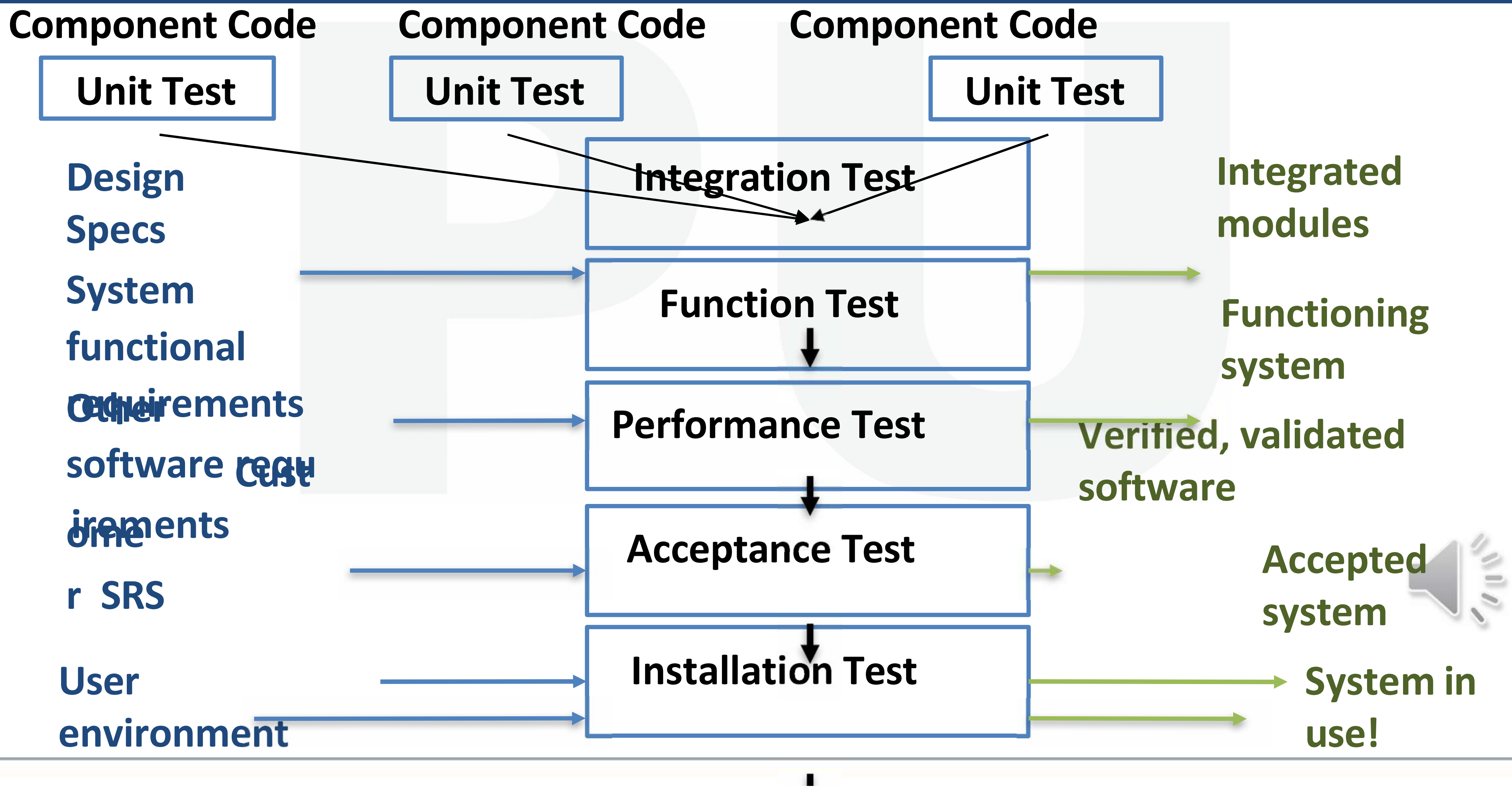
It wastes time and effort

Tester

Must learn about the system,
but, will attempt to break it and,
is driven by quality

Testing need a strategy Dev team needs to work with Test team,
"Egoless Programming"

When to test Software?



Verification vs Validation

Verification

Are we building the product right?

The objective of Verification is to make sure that the product being develop is as per the requirements and design specifications.

Validation

Are we building the right product?

The objective of Validation is to make sure that the product actually meet up the user's requirements, and check whether the specifications were correct in the first place.

Verification vs Validation (Contd.)

Verification	Validation
❑ Process of evaluating products of a development phase to find out whether they meet the specified requirements.	❑ Process of evaluating software at the end of the development to determine whether software meets the customer expectations and requirements.
❑ Activities involved: Reviews, Meetings and Inspections	❑ Activities involved: Testing like black box testing, white box testing, gray box testing
❑ Carried out by QA team	❑ Carried out by testing team
❑ Execution of code is not comes under Verification	❑ Execution of code is comes under Validation
❑ Explains whether the outputs are according to inputs or not	❑ Describes whether the software is accepted by the user or not

Software Testing Strategy

Unit Testing

- It concentrate on each unit of the software as implemented in source code.
- It focuses on each component individual, ensuring that it functions properly as a unit.

Integration Testing

- It focus is on design and construction of
- software architecture
- Integration testing is the process of testing the interface between two software units or modules

Software Testing Strategy (Contd.)

Validation Testing	
	• Software is validated against requirements established as a part of requirement modeling • It give assurance that software meets all informational, functional, behavioral and performance requirements.
System Testing	
	• The software and other software elements are tested as a whole • Software once validated, must be combined with other system elements e.g. hardware, people, database etc…

Unit Testing

- Unit is the smallest part of a software system that is testable.
- It may include code files, classes and methods which can be tested individually for correctness.
- Unit Testing validates small building block of a complex system before testing an integrated large module or whole system
- The unit test focuses on the internal processing logic and data structures within the boundaries of a component.

Unit Testing (Contd.)

- The module is tested to ensure that information properly flows into and out of the program unit
- Local data structures are examined to ensure that data stored temporarily maintains its integrity during execution
- All independent paths through the control structures are exercised to ensure that all statements in module have been executed at least once
- Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restricted processing
- All error handling paths are tested.

Unit Testing (Contd.)

- Component-testing (Unit Testing) may be done in isolation from rest of the system
- In such case the missing software is replaced by Stubs and Drivers and simulate the interface between the software components in a simple manner

Unit Testing (Contd.)

- Let's take an example to understand it in a better way.
- Suppose there is an application consisting of three modules say, module A, module B & module C.
- Developer has design in such a way that module B depends on module A & module C depends on module B
- The developer has developed the module B and now wanted to test it.
- But the module A and module C has not been developed yet.
- In that case to test the module B completely we can replace

Unit Testing (Contd.)

- Driver and/or Stub software must be developed for each unit test
- A driver is nothing more than a "main program"
  - It accepts test case data
  - Passes such data to the component and
  - Prints relevant results.
- Driver
  - Used in Bottom up approach
  - Lowest modules are tested first.
  - Stimulates the higher level of components
  - Dummy program for Higher level component

Unit Testing (Contd.)

- Stubs serve to replace modules that are subordinate (called by) the component to be tested.
- A stub or "dummy subprogram"
  - Uses the subordinate module's interface
  - May do minimal data manipulation
  - Prints verification of entry and
  - Returns control to the module undergoing testing
- Stubs
  - Used in Top down approach
  - Top most module is tested first
  - Stimulates the lower level of components
  - Dummy program of lower level components

Metrics

- Metrics are quantitative measure that the software engineer to gain the efficiency of the process

Types of Measures

Size Measure

Complexity Measure

Size Measure

Size oriented measure is derived by considering the size of software that has been produced.
Any organization builds a simple record of size measure for the software projects. It is built based on past experiences.
Set of size measure is given below:

- Size = Kilo Line of Code
- Effort = Person month
- Productivity = KLOC/Person-month
- Cost = $/KLOC
- Quality = Number of faults / KLOC
- Documentation = Pages/KLOC

Size measure is based on line of code computation.

Complexity Measure

- If the complexity is measured in terms of line of code then it may vary from system to system.
- Complexity can be done by various methods such as cyclomatic complexity, Halstead measure and Knot count measure.

- **Cyclomatic complexity**
  - Independent path is any path through use of the program that introduces at least one new set of processing statements or a new condition.
  - Cyclomatic complexity **is software metric that provides a quantitative measure of the logical complexity of a program**.
  - It defines number of independent paths in the basis of set of program and provides us with an **upper bound for the number of tests that must be conducted to ensure all statements have been executed at least once.**

Cyclomatic Complexity

It can be computed 3 ways:

- The number of regions corresponds to cyclomatic complexity.
- Cyclomatic complexity V (G) can be defined as

  $$V(G) = E - N + 2$$

  Where E is number of flow graph edges and N is the number of flow graph nodes.
- Cyclomatic complexity V (G) can also be defined as

  $$V(G) = P + 1$$

  Where P is the number of predicate nodes contained in the graph.

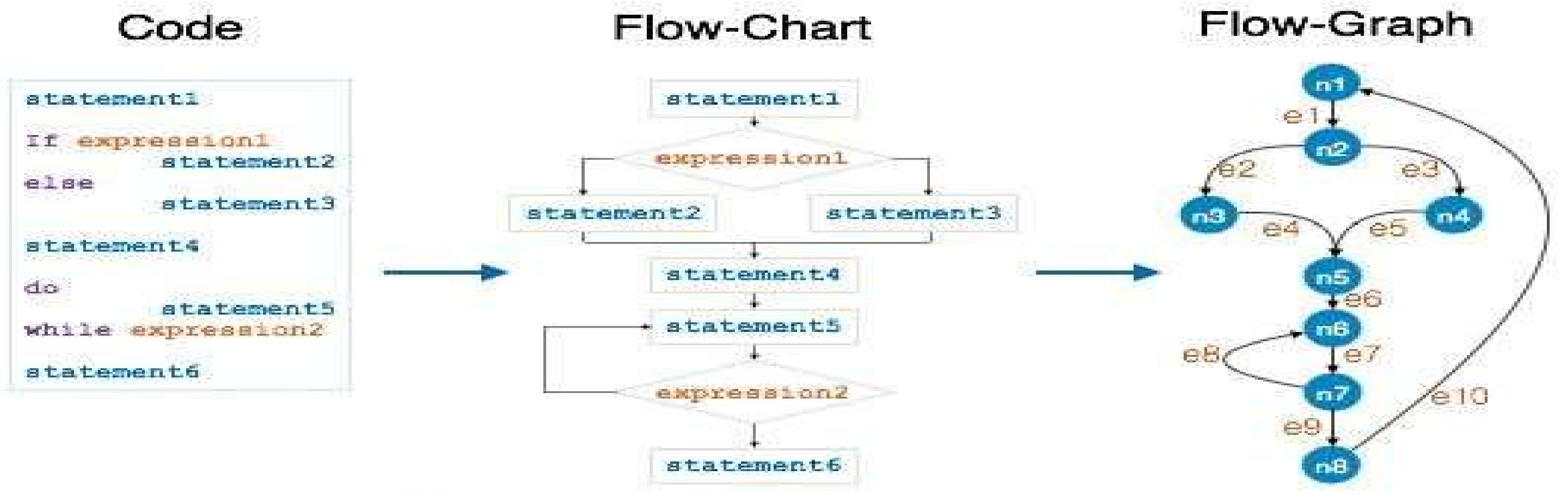
Parul® University

DIGITAL LEARNING CONTENT

Cyclomatic Complexity

```
V(G) = e – n + 2

Where
e is total number of edges
n is total number of nodes
```



```
statement1
If expression1
          statement2
else
          statement3
statement4
do
          statement5
while expression2
statement6
```

The Cyclomatic complexity of the above module is

```
e = 10
n = 8
Cyclomatic Complexity = 10 - 8 + 2
                      = 4
```

Halstead Measure

- **Halstead's theory of software science is one of "the best known and most thoroughly studied composite measures of (software) complexity".**
- **Software science assigns quantitative laws to the development of computer software, using a set of primitive measures that may be derived after code is generated or estimated once design is complete.**

Parul® University

Halstead Measure (Contd.)

- These follow:
- n1= the number of distinct operators that appear in a program. n2= the number of distinct operands that appear in a program. N1= the total number of operator occurrences. N2= the total number of operand occurrences.

	Symbol	Formula
Program Length	N	N = N1 + N2
Program Vocabulary	n	n = n1 + n2
Volume	V	V = N * (log2n)
Difficulty	D	D = (n1/2) * (n2/2)
Effort	E	E = D * V

Halstead Measure (Contd.)

- **Program Length**

  The length of a program is total usage of operators and operands in the program.

  Length = N1 + N2

- **Program vocabulary**

  The program vocabulary is the number of unique operators and operands used in the program.

  Vocabulary n = n1 + n2

- **Program Volume**

  The program volume can be defined as the maximum number of bits to encode the program.

  V=Nlog2n

  Halstead shows that length N can be estimated

  N= n1log2n1+ n2log2n2

Parul® University

DIGITAL LEARNING CONTENT

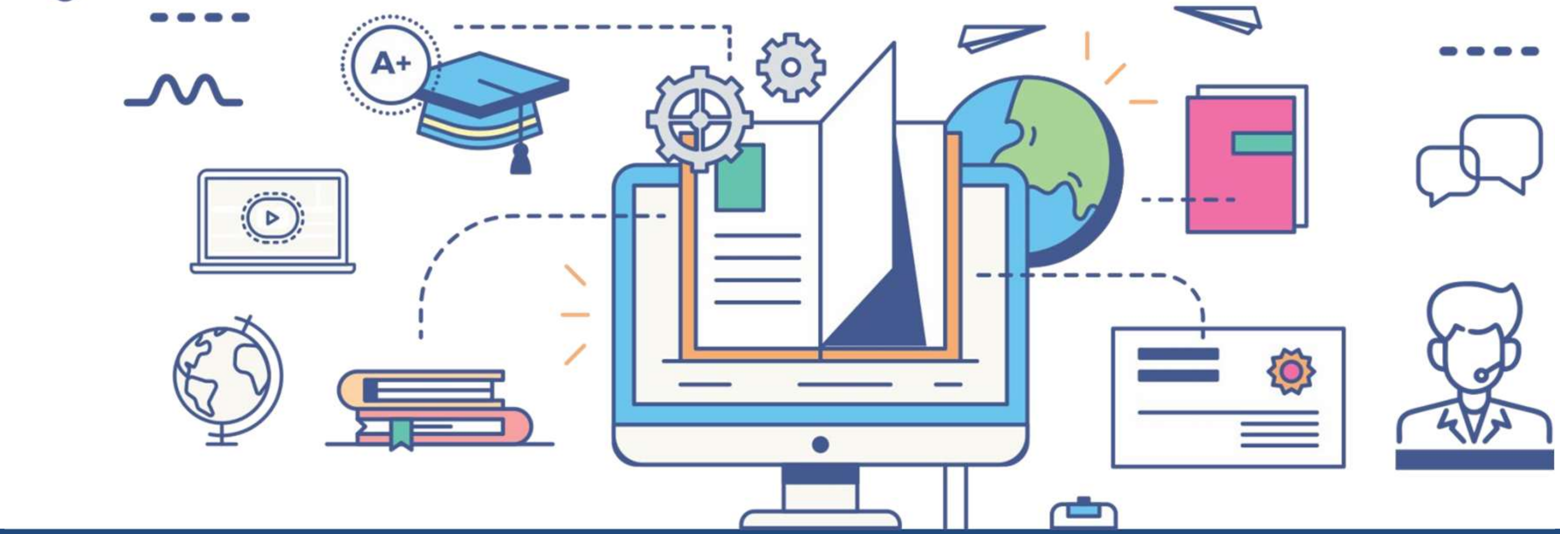
Knot Count

- Knot is a crossing of control flows. These crossings occur due to non-structural
- jumps in the program.
- Typically the goto statements cause this kind of non-structural jump. This metric is designed for FORTRAN language.
- If the knot is more intertwined then that means the program is more complex.
- The code with large knots is generally extremely difficult to read and understand.

Comparison of Different Metrics

Size measure	Cyclomatic Complexity	Halstead's measure	Knot count
▪ This is simple method of obtaining the metrics. It is based on lines of code.	▪ This measure is based on the control flow of the programming constructs such as if then else, do-while, repeat-until and so on.	▪ The measurable quantities of the program are operators and operands.	▪ It is basically designed for the FORTRAN programs.
▪ Modules of the same size can have different complexities.	▪ For larger number of decisions larger is the complexity.	▪ It is based on length and volume of the program.	▪ More number of knots indicates more complex in the program.

DIGITAL LEARNING CONTENT



Parul® University

www.paruluniversity.ac.in