

Software Testing and Quality Assurance

Unit-2 Levels of Testing



Unit-2 Levels of Testing

2.0. Unit Testing

2.1. Control Flow Testing

2.2. Data Flow Testing

2.3. Domain Testing

2.4. System Integration Testing

2.5. System Test

2.6. Acceptance Testing



Unit Testing

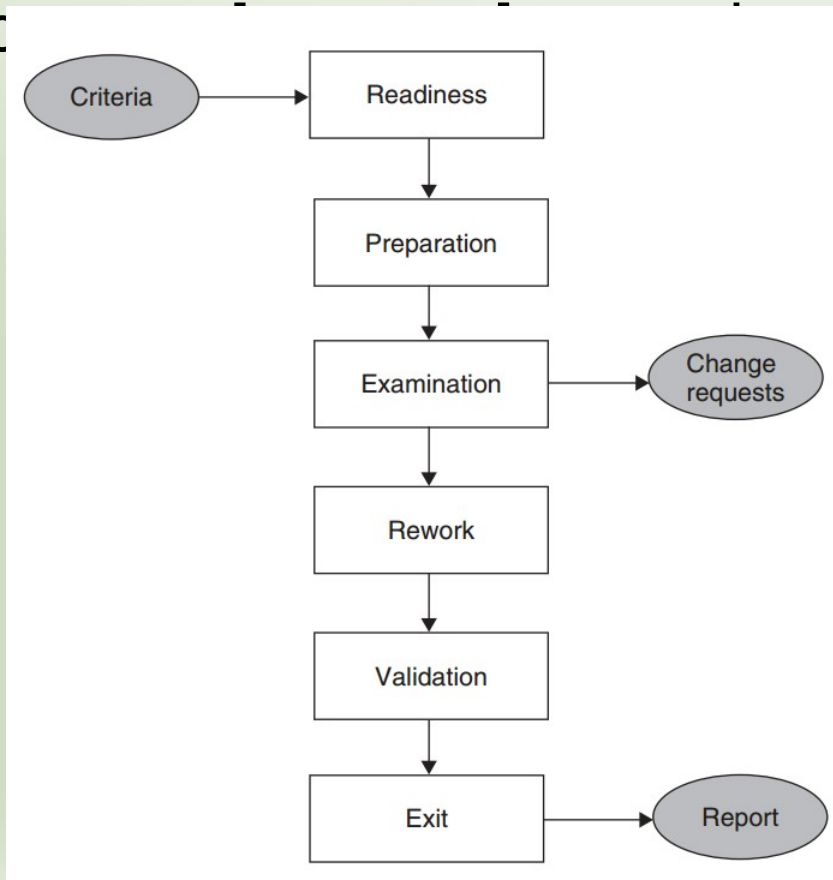
- ❑ Unit Testing : Refers to testing program units in isolation.
- ❑ Units are **functions, procedures, or methods**, even **a class**.
- ❑ unit testing removes **dependencies** on other program units and helps to verify the unit alone produce desirable result.
- ❑ Intuitively, a programmer needs to test a unit via
 - **Execute every line of code.**
 - **Execute every predicate** in the unit to evaluate them to true and false separately.
 - Observe that the unit performs its intended function and ensure that it contains **no known errors**.
- Unit testing is conducted in two complementary phases:
 - **Static unit testing** : is a non execution based via code review
 - **Dynamic unit testing**: is execution based.



Unit Testing

- In Static Unit Testing:

- In static unit testing, code is reviewed by applying techniques commonly known as **Walkthrough**.

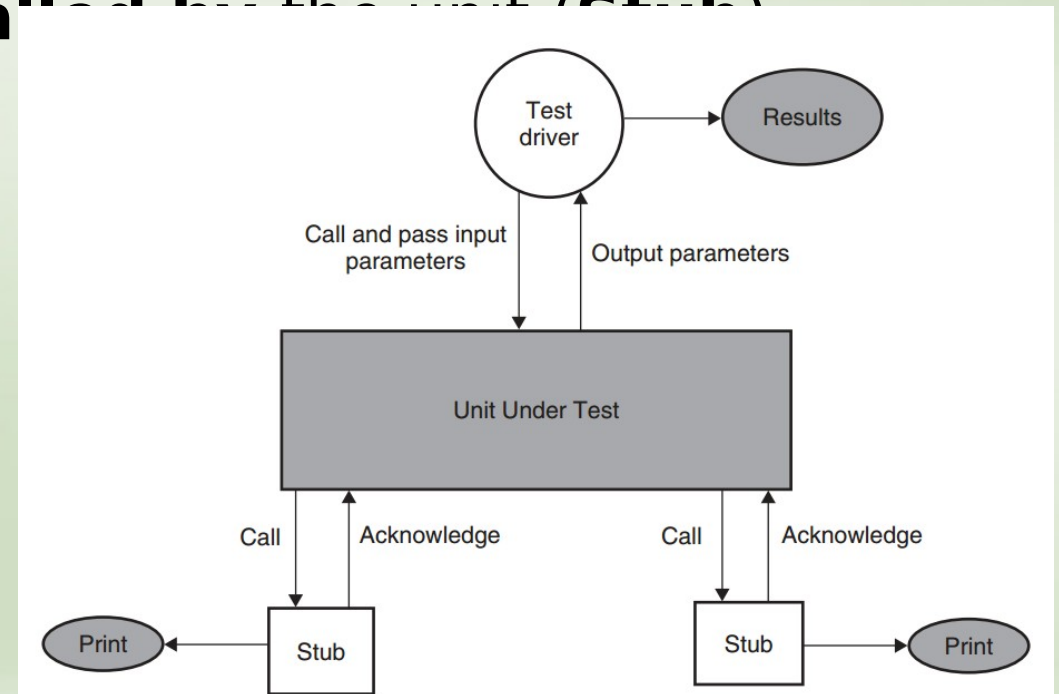


Unit Testing

- In **Dynamic** Unit Testing
 - In this testing, a program unit is actually executed in isolation
 - Dynamic unit testing is created by emulating the context of the unit under test
 - The context of a unit test consists of two parts: (i) **a caller of the unit (test driver)** and (ii) all the units **called by the unit (Stubs)**

Test Driver: is a program that invokes the unit under test with input values.

Stubs: A stub is a “dummy subprogram” that replaces a unit that is called by the unit under test.



Unit Testing

- **Defect Prevention:** include suitable mechanism in the code
 - **Built in tracking, tracing mechanisms**
 - Include **exceptions** such as **division by zero, array index out of bounds, buffer overflow or underflow, File Not found, db connection error** and etc.
- **Debugging:** The process of determining the cause of a program failure and fixing it.

In Unit Testing modules contains the following

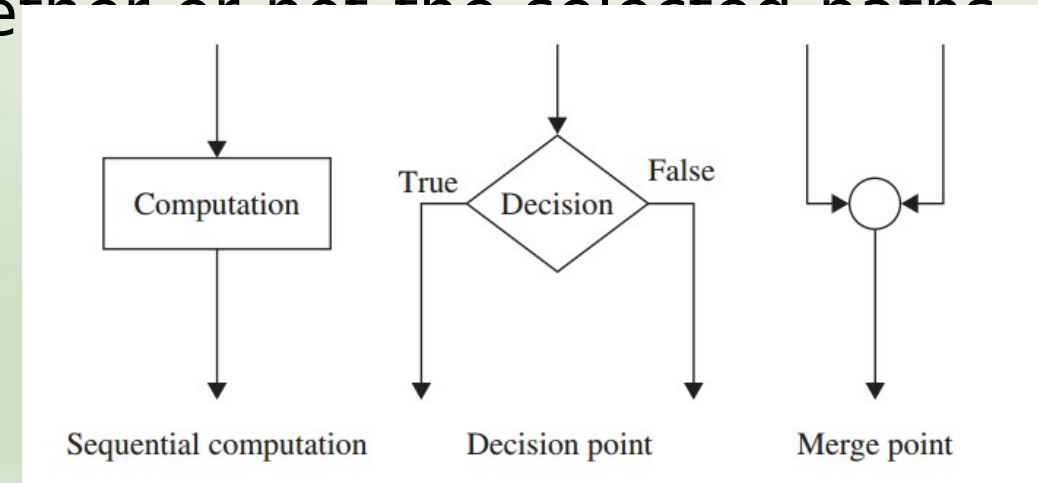
- Sequence of statements



2.1. Control flow testing

Control Flow Testing

- **Control flow** refers to a code that define **sequence, decision, function, class** and or etc of a program.
- Control flow testing is a kind of **structural testing**, which is performed by programmers to test code written by them.
- The main idea in **control flow** testing is to appropriately select a **few paths** in a program unit and observe whether or not the selected paths produce the expected outcome.
- A **control flow graph (CFG)** is a graphical representation of a program unit with.
 - **Sequence, function, loop, decision, Flow direction (arrow)** and others as call



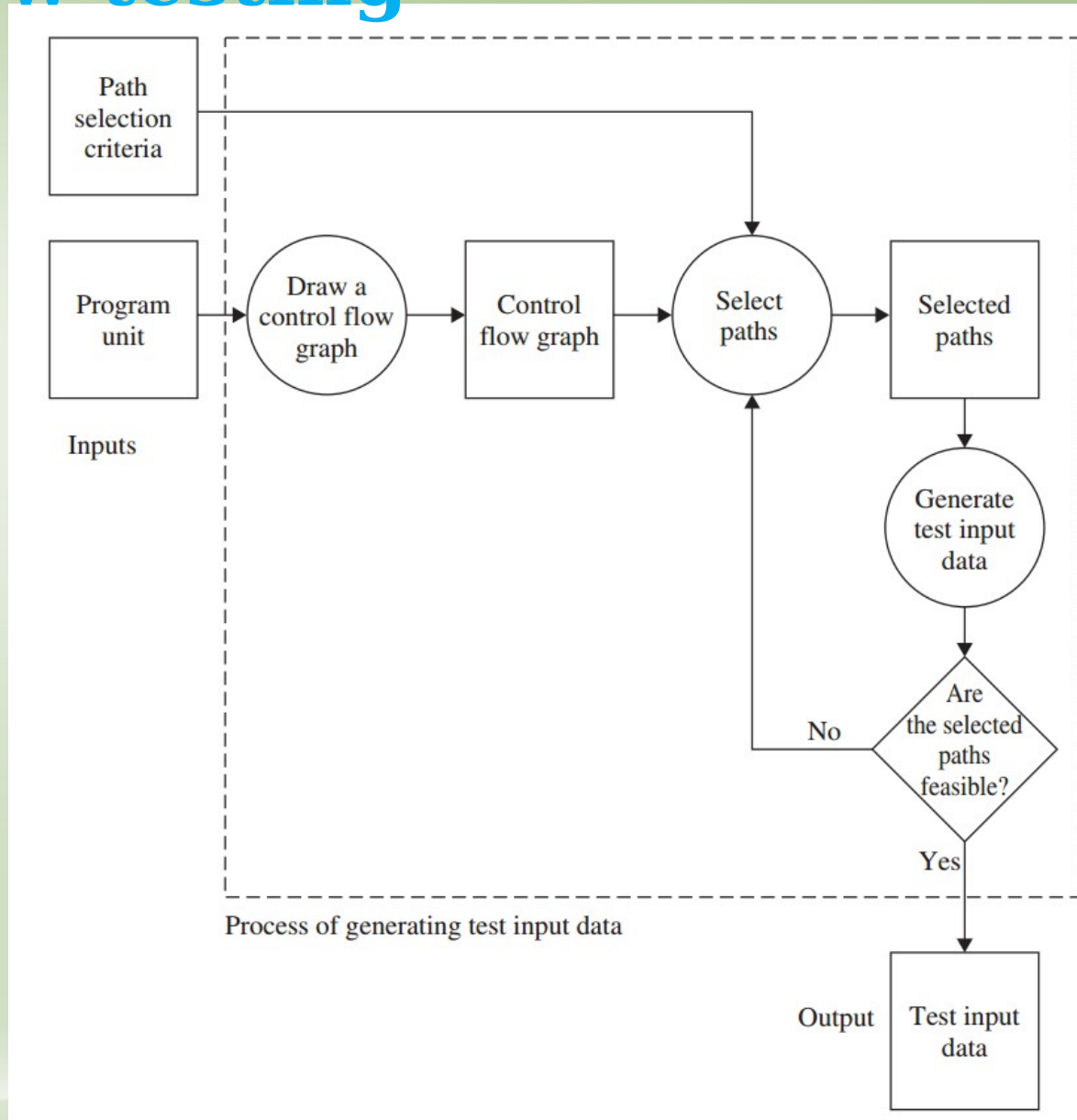
2.1. Control flow testing

The process of generating test input data for control flow testing

Input : source code, path selection criteria

CFG: visualize all paths

Selection of Paths: Paths are selected by considering the structure of the CFG.



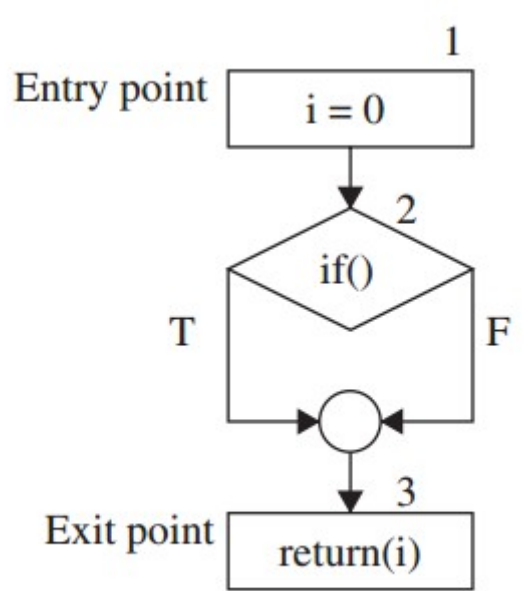
2.1. Control flow testing

Program unit for openfiles()

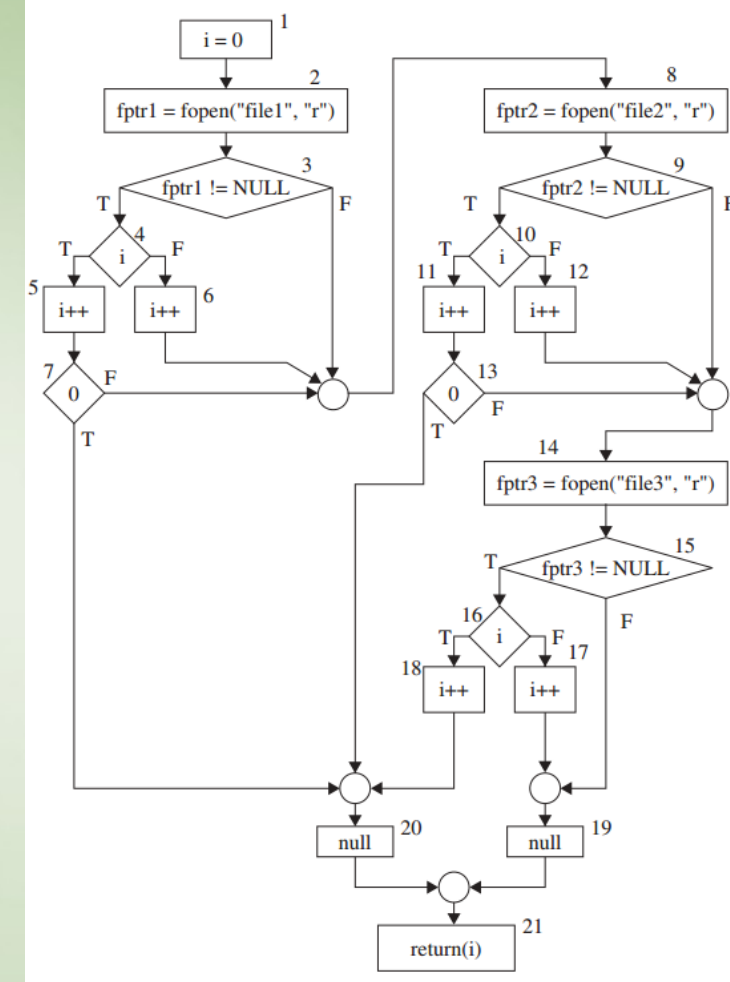
```
FILE *fptr1, *fptr2, *fptr3; /* These are global variables. */

int openfiles(){
    /*
     * This function tries to open files "file1", "file2", and
     * "file3" for read access, and returns the number of files
     * successfully opened. The file pointers of the opened files
     * are put in the global variables.
     */
    int i = 0;
    if(
        ((( fptr1 = fopen("file1", "r")) != NULL) && (i++)
          && (0)) ||
        ((( fptr2 = fopen("file2", "r")) != NULL) && (i++)
          && (0)) ||
        ((( fptr3 = fopen("file3", "r")) != NULL) && (i++))
    );
    return(i);
}
```

- How many paths? How many decision ? sequence ? assignment ?



CFG representation of openfiles()



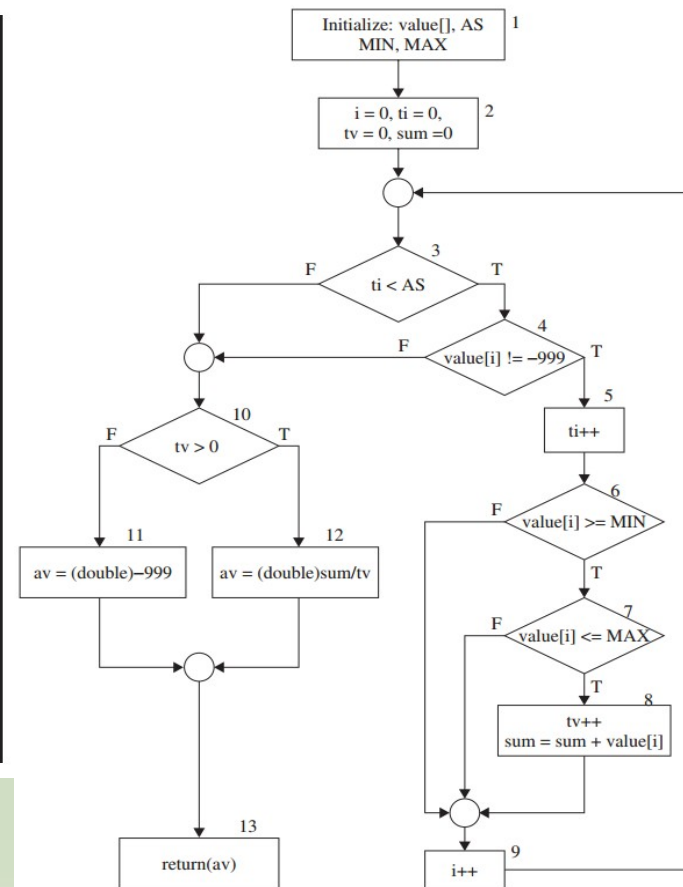
Detailed CFG representation of openfiles()

2.1. Control flow testing

- **Paths In A Control Flow Graph**
- A path is represented as a **sequence of computation and decision nodes** from the **entr node** to the **exit node**

```
public static double ReturnAverage(int value[],
                                   int AS, int MIN, int MAX){
    /*
    Function: ReturnAverage Computes the average
    of all those numbers in the input array in
    the positive range [MIN, MAX]. The maximum
    size of the array is AS. But, the array size
    could be smaller than AS in which case the end
    of input is represented by -999.
    */
    int i, ti, tv, sum;
    double av;
    i = 0; ti = 0; tv = 0; sum = 0;
    while (ti < AS && value[i] != -999) {
        ti++;
        if (value[i] >= MIN && value[i] <= MAX) {
            tv++;
            sum = sum + value[i];
        }
        i++;
    }
    if (tv > 0)
        av = (double)sum/tv;
    else
        av = (double) -999;
    return (av);
}
```

Function to compute average of an array.



CFG Function to compute average of an array.

Path 1	1-2-3(F)-10(T)-12-13
Path 2	1-2-3(F)-10(F)-11-13
Path 3	1-2-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10(T)-12-13
Path 4	1-2-3(T)-4(T)-5-6(T)-7(T)-8-9-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10(T)-12-13

Paths in compute average

Path selection criteria:

- Select all paths.
- Select paths to achieve complete statement coverage.
- Select paths to achieve complete branch coverage.
- Select paths to achieve predicate coverage.

2.1. Control flow testing

All Path coverage criteria

- If *all* the paths in a CFG are selected, then one can detect all faults
- The `openfiles()` function shown above contains more than 25 paths and only eight of all those paths are feasible
- If one selects all possible paths in a program, then we say that the *all-path* selection criterion has been satisfied.

Example of the `openfiles()` function tries to open the three files **file1**, **file2**, and **file3**. A file is said to be successfully opened with “read” access if the **file exists**. The existence of a file is either “**yes**” or “**no**.” Thus, the input domain of the function consists of eight combinations of the existence of the three files.

Existence of file1	Existence of file2	Existence of file3
No	No	No
No	No	Yes
No	Yes	No
No	Yes	Yes
Yes	No	No
Yes	No	Yes
Yes	Yes	No
Yes	Yes	Yes

Input	Path
< No, No, No >	1-2-3(F)-8-9(F)-14-15(F)-19-21
< Yes, No, No >	1-2-3(T)-4(F)-6-8-9(F)-14-15(F)-19-21
< Yes, Yes, Yes >	1-2-3(T)-4(F)-6-8-9(T)-10(T)-11-13(F)-14-15(T)-16(T)-18-20-21

Example for three Inputs and Paths in `openfiles()`

Input Domain of
openfiles()



2.1. Control flow testing

Statement Coverage Criterion

- Refers to executing individual program statements and observing the outcome
- Complete statement coverage is the **weakest coverage** criterion in program testing.

- Any test suite that achieves complete statement coverage for new

The software is considered to be tested.
example

```
i    = 0;  
ti   = 0;  
tv   = 0;  
sum  = 0;
```

Node 2

```
(ti < AS && value[i] != -999)
```

statement coverage for new
table. **Node 3** and **Node 4**

SCPath 1	1-2-3(F)-10(F)-11-13
SCPath 2	1-2-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10(T)-12-13

Paths for Statement **Coverage of CFG compute Average**



2.1. Control flow testing

Branch Coverage Criterion

- Covering a branch means selecting a path that includes the outgoing edge

BCPath 1	1-2-3(F)-10(F)-11-13
BCPath 2	1-2-3(T)-4(T)-5-6(T)-7(T)-8-9-3(F)-10(T)-12-13
BCPath 3	1-2-3(T)-4(F)-10(F)-11-13
BCPath 4	1-2-3(T)-4(T)-5-6(F)-9-3(F)-10(F)-11-13
BCPath 5	1-2-3(T)-4(T)-5-6(T)-7(F)-9-3(F)-10(F)-11-13

Paths for Branch Coverage of CFG Compute average

Predicate Coverage Criterion

- If all possible combinations of **truth values** of the conditions affecting a selected path have been explored under some tests, then we say that **predicate coverage** is satisfied.



Reading

- Generating Test Input
- What is Infeasible Path



2.2. Data flow testing

- Testing *paths* that include pairs of **definition** and **use of variables** in programs

Data Flow Anomaly

- An abnormal situation to successively assign two values to a variable without using the first value a.k.a ***Defined and then Defined Again.***
- It is abnormal to use a value of a variable before assigning a value to the variable. a.k.a ***Undefined but Referenced.***
- Abnormal situation is to generate a data value and never use it a.k.a ***Defined but Not Referenced.***



2.2. Data flow testing

Data flow graph

- A **data flow graph** is drawn with the objective of identifying **data definitions** and their **uses**.
- Each occurrence of a data variable is classified as
 - **Definition**: This occurs when a value is moved into the memory location of the variable.
 - **Undefinition or Kill**: This occurs when the value and the location become unbound.
 - **Use**: This occurs when the value is fetched from the memory location of the variable.
 - **Computation use(c-use)**: to produce new variable value or the same variable of different value.
 - **Predicate use (p-use)**: controlling the flow of execution.
- A data flow graph is a directed graph constructed as follows:
 - A sequence of **definitions** and **c-uses** is associated with each **node** of the graph.
 - A set of **p-uses** is associated with each **edge** of the graph.
 - The entry node has a definition of each parameter and each nonlocal variable which occurs in the subprogram.
 - The **exit** node has an **undefinition** of each local variable

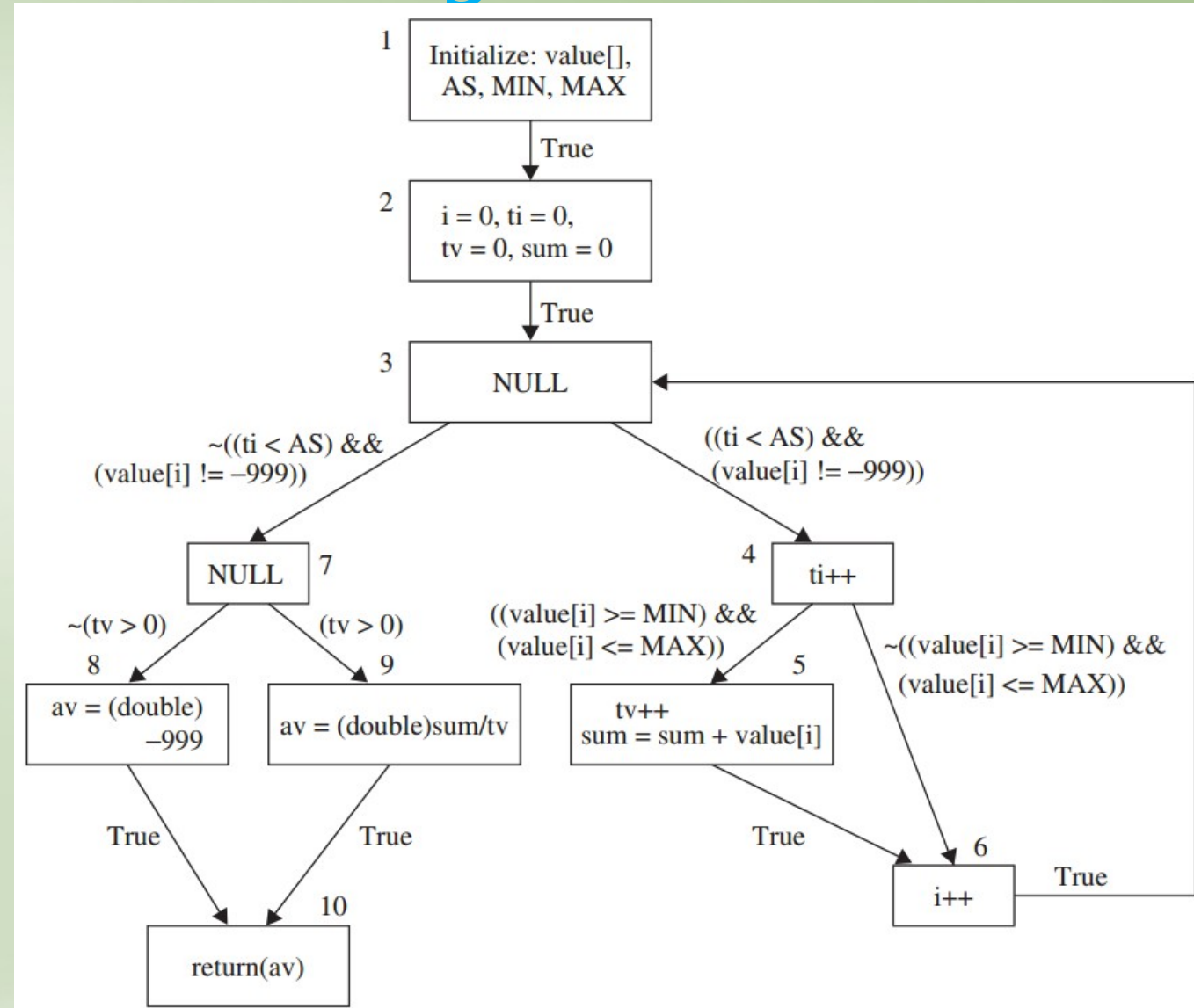


2.2. Data flow testing

- **Node 1**, represents initialization of the input vector.
- **Node 2**, represents the initialization of the four local variables
- **Node 3**, NULL node indicator of the control exiting from the loop,
- The predicate associated with **edge (3, 4)** is the condition part of the while loop
- Finally, the return(av) statement is represented by node **10**.

Data Flow Terms

- A **variable defined** in a statement is *used* in another statement which may occur immediately or several statements after the definition.



Data flow graph of ReturnAverage()
example.

2.2. Data flow testing

Data Flow Terms...

- We are interested in **finding paths** that include pairs of **definition** and **use** of variables.
- **Global c-use**: A c-use of a variable x in node i is said to be a *global c-use* if x has been defined before in a node other than node i .
 - The **c-use** of variable **tv** in node **9** is a global c-use since **tv** has been defined in nodes **2** and **5** in the above DFG.
- **Definition Clear Path**: A path $(i - \dots - j)$, $m \geq 0$, is called a definition clear path (def-clear path) with respect to variable x
 - from node i to node j and
 - from node i to edge (nm, j)
- For example The paths **2-3-4-5** and **2-3-4-6** are def-clear paths with respect to variable **tv** from node **2** to **5** and from node **2** to **6**, respectively.
- **Global Definition**: A node i has a global definition of a variable x if node i has a definition of x and there is a def-clear path with respect to x from

2.2. Data flow testing

- **Simple Path:** is a path in which all nodes, except possibly the first and the last, are distinct.
- $\text{def}(i)$ denotes the set of variables which have global definitions in node i . Similarly, $\text{c-use}(i)$ denotes the set of variables which have global c-uses in node i .

Nodes i	$\text{def}(i)$	$\text{c-use}(i)$
1	{value, AS, MIN, MAX}	{}
2	{i, ti, tv, sum}	{}
3	{}	{}
4	{ti}	{ti}
5	{tv, sum}	{tv, i, sum, value}
6	{i}	{i}
7	{}	{}
8	{av}	{}
9	{av}	{sum, tv}
10	{}	{av}

Global definitions and global c-uses appearing in the **DFG**

Edges (i, j)	$\text{predicate}(i, j)$	$\text{p-use}(i, j)$
(1, 2)	True	{}
(2, 3)	True	{}
(3, 4)	$(ti < AS) \ \&\& \ (value[i] \neq -999)$	{i, ti, AS, value}
(4, 5)	$(value[i] \leq MIN) \ \&\& \ (value[i] \geq MAX)$	{i, MIN, MAX, value}
(4, 6)	$\sim((value[i] \leq MIN) \ \&\& \ (value[i] \geq MAX))$	{i, MIN, MAX, value}
(5, 6)	True	{}
(6, 3)	True	{}
(3, 7)	$\sim((ti < AS) \ \&\& \ (value[i] \neq -999))$	{i, ti, AS, value}
(7, 8)	$\sim(tv > 0)$	{tv}
(7, 9)	$(tv > 0)$	{tv}
(8, 10)	True	{}
(9, 10)	True	{}

Predicates and p-uses appearing in the data flow graph

- **Du-path:** A path $(n_1 - n_2 - \dots - n_j - n_k)$ is a definition-use path (du-path) with respect to (w.r.t) variable x if node n_1 has a global definition of x and *either* node n_k has a global c-use of x and $(n_1 - n_2 - \dots - n_j - n_k)$ is a def-clear simple path w.r.t. x *or* edge (n_j, n_k) has a p-use of x and $(n_1 - n_2 - \dots - n_j)$ is a



2.2. Data flow testing

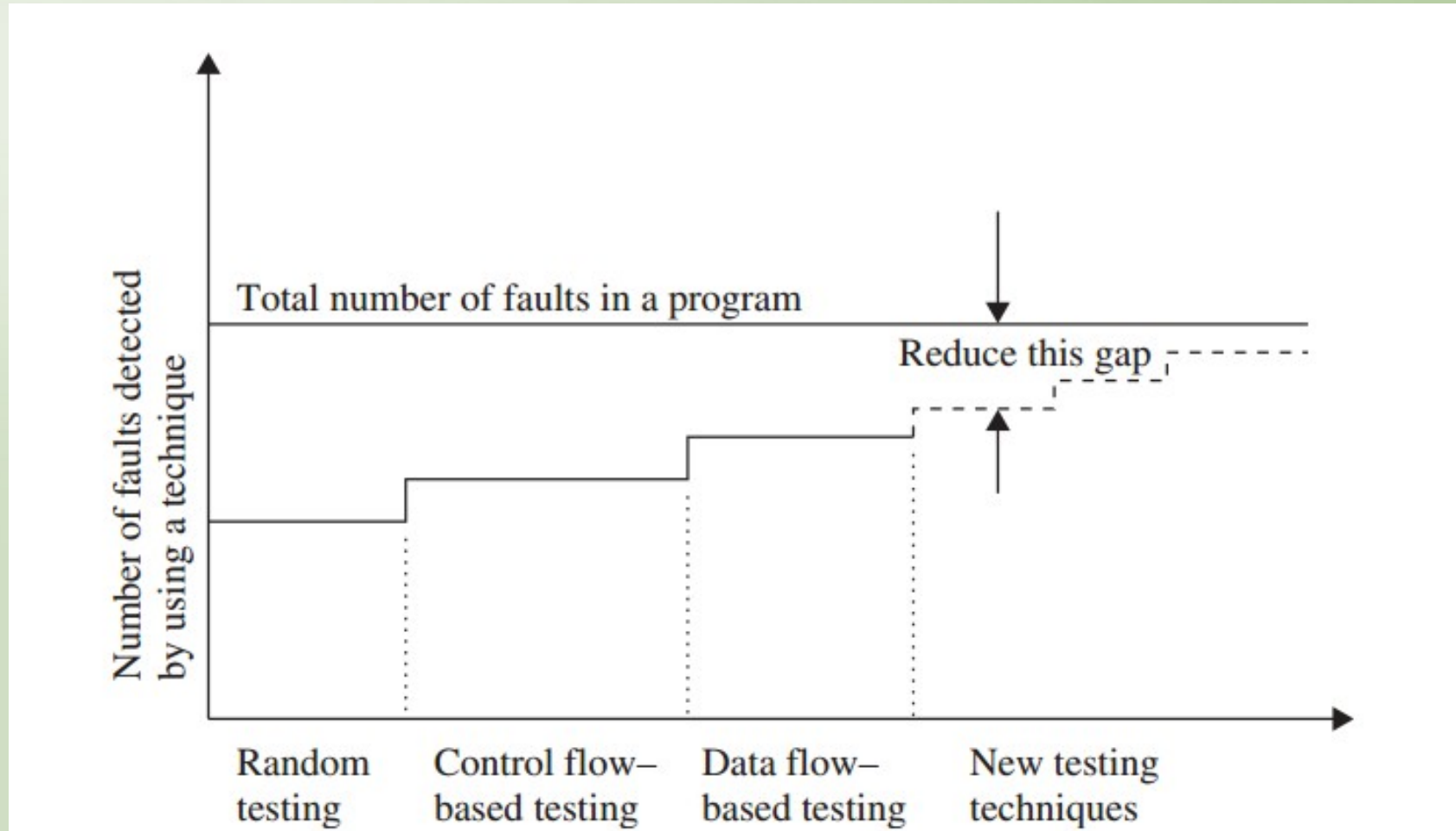
- There are seven **data flow testing criterion** based on definitions and uses —both c-uses and p-uses—of variables.

- 1. All-defs:** For each variable x and for each node i such that x has a global definition in node i , select a complete path which includes a def-clear path from node i to
 - node j having a global c-use of x or
 - edge (j, k) having a p-use of x .
- 2. All-c-uses:** For each variable x and for each node i , such that x has a global definition in node i , select complete paths which include def-clear paths from node i to *all* nodes j such that there is a global c-use of x in j .
- 3. All-p-uses:** For each variable x and for each node i such that x has a global definition in node i , select complete paths which include def-clear paths from node i to *all* edges (j, k) such that there is a p-use of x on edge



2.2. Data flow testing

Limitation of different fault detection techniques.



2.2. Data flow testing

Self- assessment questions

1. Compare
 - random test selection,
 - test selection based on control flow, and
 - test selection based on data flow.
2. Explain feasible testing paths, and feasible path selection criterion with concrete example.



2.3. Domain Testing

- Two fundamental elements of a computer program are ***input domain*** and ***program paths***.
- The input domain of a program is the set of **all input data** to the program.
- A program path is a sequence of instructions from the start of the program to some point of *interest* in the program.
- A path is said to be ***feasible*** if there exists an input data which causes the program to execute the path. Otherwise, the path is said to be *infeasible*.

Two common domain errors

- ***Computation Error***: A computation error occurs when a specific input data causes the program to execute the correct, i.e., desired path, but the output value is wrong. e.g. function result = $f(a, b)$ is executed as result = $f(b, a)$.
- ***Domain Error***: An incorrect path can be selected by a program if there is a fault in one or more of the *conditional* statements in the program. e.g. when p is wrong in *if (p) then f1() else f2()*, a wrong function will be called.



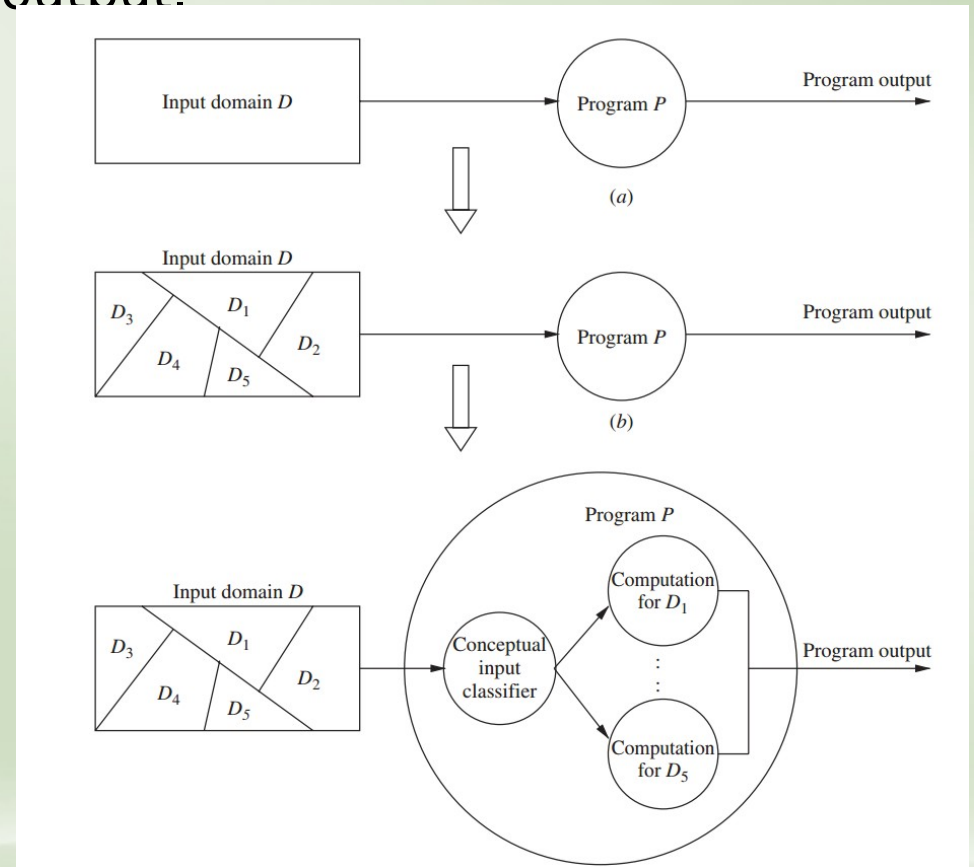
2.3. Domain Testing

- A **domain** is a set of input values for which the program performs the same computation for every member of the set. We are interested in **maximal domains** such that the program performs different computations on *adjacent* domains.
- A program is said to have a *domain error* if the program incorrectly performs input classification. Assuming that adjacent domains perform different computations, a domain error will cause the program to produce incorrect output.

(a) The set D is the entire input set of a program P

(b) P performs five different computations, one for each subset $D_1, \dots, D_5$.

(c) Five different computations, computation for D_1 through computation for D_5 , for subsets $D_1, \dots, D_5$, respectively



2.3. Domain Testing

Testing For Domain Errors

- The difference between **flow graph-based** (control and data flow graph) testing techniques and **domain** testing.
 - Select paths from a control flow graph or a data flow graph to satisfy certain *coverage criteria*.
 - Domain testing defines a category of faults, called *domain errors*, and selects test data to detect those faults.

Source of Domains

- Domains can be identified from both **specifications** and **programs** using the steps.
 - Draw a control flow graph from the given **source code**.
 - Find all possible **interpretations** of the **predicates**.
 - Analyze the interpreted **predicates** to identify **domains**.



2.3. Domain Testing

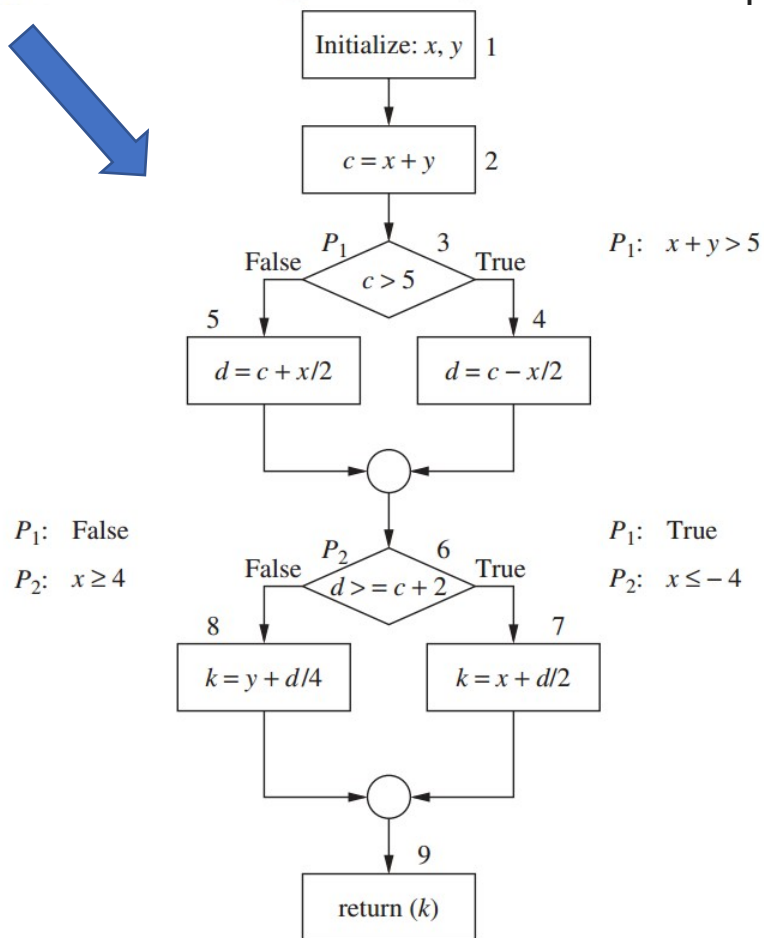
A function to explain program domains.

```
int codedomain(int x, int y){  
    int c, d, k  
    c = x + y;  
    if (c > 5) d = c - x/2;  
    else      d = c + x/2;  
    if (d >= c + 2) k = x + d/2;  
    else      k = y + d/4;  
    return(k)  
}
```

The first **if()** statement has just one interpretation

$$P_1: x + y > 5$$

The second **if()** statement gets two interpretations $P_1 = \text{True}$ and $P_1 = \text{False}$ leads us to P_2



2.3. Domain Testing

Domains and its *boundaries*

- **Closed Boundary:** A boundary is said to be closed if the points on the boundary are included in the domain of interest.
- **Open Boundary:** A boundary is said to be open if the points on the boundary do not belong to the domain of interest.
- **Closed Domain:** A domain is said to be closed if all of its boundaries are closed.
- **Open Domain:** A domain is said to be open if some of its boundaries are open.
- **Extreme Point:** An extreme point is a point where two or more boundaries cross.
- **Adjacent Domains:** Two domains are said to be adjacent if they have a boundary inequality in common.



2.3. Domain Testing

Types of Domain Errors

- 1. Closure Error:** A closure error occurs if a boundary is open when the intention is to have a closed boundary, or vice versa. e.g. **The operator $\leq$ is implemented as $<$ or vice versa**
- 2. Shifted-Boundary Error:** A shifted-boundary error occurs when the implemented boundary is parallel to the intended boundary. **e.g. $P1:x+y>5$ is used instead of $P1:x+y>4$.**
- 3. Tilted-Boundary Error:** If the constant coefficients of the variables in a predicate defining a boundary take up wrong values, then the tilted-boundary error occurs. e.g. **$P1:x+y>5$ instead of $P1 : x + 0.5y > 5$.**



2.3. Domain Testing

Test selection Criterion

1. Closed inequality boundary
 - a. Boundary shift resulting in a reduced domain
 - b. Boundary shift resulting in an enlarged domain
 - c. Boundary tilt
 - d. Closure error
2. Open inequality boundary
 - a. Boundary shift resulting in a reduced domain
 - b. Boundary shift resulting in an enlarged domain
 - c. Boundary tilt
 - d. Closure error
3. Equality boundary



2.3. Domain Testing

Self assessments

1. The domain testing strategy requires us to select test points on and/or very close to domain boundaries. Why don't we select test points far from the boundaries?
2. Explain the difference between control flow-based testing and domain error-based testing
3. explain the test selection criterion in domain-based testing and show the closed inequality error (boundary shift resulting in a reduced domain) is detected by the test points chosen by the selection criterion
4. Identify some difficulties in applying the concept of domain testing to actual program testing.



2.4. System Integration Testing

- A software module, or component, is a self-contained element of a system. Modules have well-defined interfaces with other modules.
- A system is a collection of modules interconnected in a certain way to accomplish a tangible objective.
- system integration testing is a systematic technique for assembling a software system while conducting tests to uncover errors associated with interfacing.
- Integration testing is said to be complete when the system is fully integrated together, all the test cases have been executed, all the severe and moderate defects found have been fixed, and the system is retested



2.4. System Integration Testing

Different Types Of Interfaces

- Modules are interfaced with other modules to realize the system's functional requirements.
 - **Procedure Call Interface** the caller passes on control to the called module.
 - **Shared Memory** A block of memory is shared between two modules.
 - **Message Passing Interface:** One module prepares a message by initializing the fields of a data structure and sending the message to another module.

Interface Errors

- Construction
- Inadequate Functionality
- Location of Functionality
- Added Functionality
- Misuse of Interface
- Misunderstanding of Interface and etc...



2.4. System Integration Testing

The major advantages of conducting system integration testing are as follows:

- Defects are detected early.
- It is easier to fix defects detected earlier.
- We get earlier feedback on the health and acceptability of the individual modules and on the overall system.
- Scheduling of defect fixes is flexible, and it can overlap with development.

Granularity of System Integration Testing

- ***Intra-system testing*** combining the modules together to build a cohesive system.
- ***Inter-system testing*** Integrating a call control system and a billing system in a telephone network is another example of intersystem testing.
- ***Pairwise system testing*** For example, in testing communication between a network element (radio node) and the element management systems



2.4. System Integration Testing

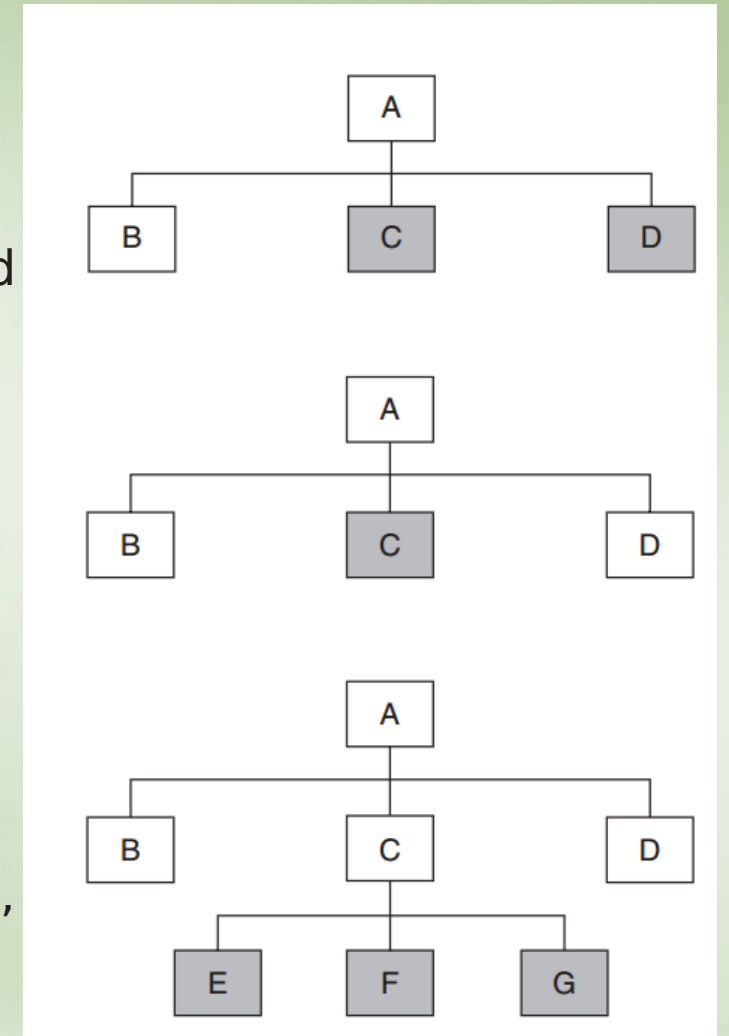
System Integration Techniques

- **Incremental** a series of test cycles created to add new modules in existing build.
- **Top down testing** top level modules are tested and down in the hierarchy.

Top-down integration of modules A and B
C and D are test drivers

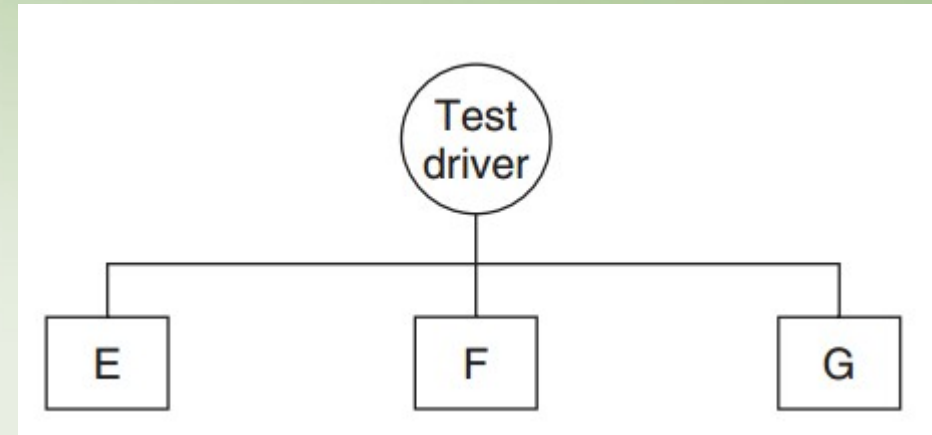
Top-down integration of modules A, B, and D

Top-down integration of modules A, B, D, and C



2.4. System Integration Testing

Bottom Up integration begins with the integration of lowest level modules that invokes no other modules.



Bottom-up integration of modules E, F, and G

Sandwich and Big Bang

- In Sandwich mix of the top-down and bottom-up approaches.
- In Big Bang first, all modules are individually tested. then, all those modules are put together to construct the entire system which is also tested as a whole.



2.4. System Integration Testing

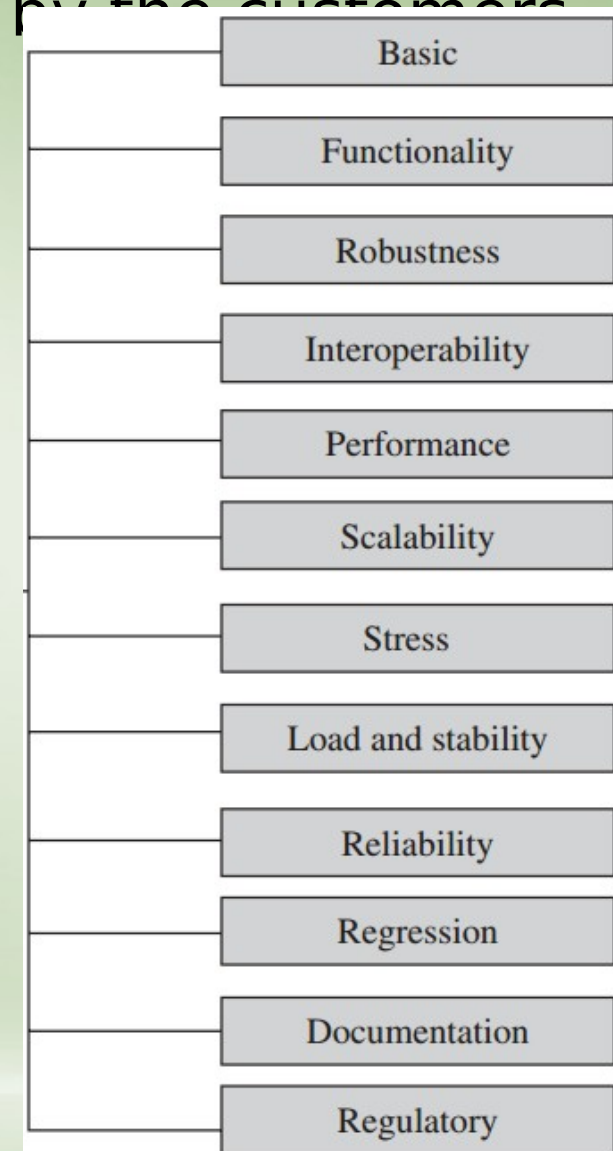
Self Assessment

1. Describe the difference between black-box and white-box testing techniques?
2. If a program passes all the black-box tests, it means that the program should work properly. Then, in addition to black-box testing, why do you need to perform white-box testing?
3. Describe the difference between unit testing and integration testing? Why should integration testing be performed? What types of errors can this phase of testing reveal?
4. Discuss the advantages and disadvantages of top-down and bottom-up approaches to integration testing.
5. Does automation of integration tests help the verification of the daily build process? Justify your answer.



2.5. System Testing

- The objective of system-level testing is to establish whether an implementation conforms to the requirements specified by the customers.
- **Basic tests** provide an evidence that the system can be installed, configured, and brought to an operational state.
- **Functionality tests** provide comprehensive testing of requirements within the capabilities of the system.
- **Robustness tests** determine how well the system recovers from various input errors and other failure situations.
- **Interoperability tests** determine whether the system can interoperate with other third-party products.
- **Performance tests** measure the throughput and response time, under various conditions.
- **Scalability tests** determine user scaling, geographic scaling, and resource scaling.
- **Stress tests** put a system under stress in order to determine the limitations of a system and, when it fails, to determine the manner in which the failure occurs.
- **Regression tests** determine that the system remains stable as it cycles through the integration of other subsystems and through



2.5. System Testing

1. What are the differences between
i, configuration, compatibility, and interoperability testing? ii, performance, stress, and scalability testing? iii, load testing and stress testing? iii, performance and speed? iv, safety and reliability v, safety testing and security testing?
2. Discuss the importance of regression testing when developing a new software release. What test cases from the test suite would be more useful in performing a regression test?
3. What is the similarity between software safety and fault tolerance?
4. For each of the following situations, explain whether it is a hazard or a mishap:
 - (a) Water in a swimming pool becomes electrified.
 - (b) A room fills with carbon dioxide.
 - (c) A car stops abruptly.
 - (d) A long-distance telephone company suffers an outage.
 - (e) A nuclear weapon is destroyed in an unplanned manner



2.6. Acceptance Testing

Reading assignment



QUESTIONS PLEASE □

