Software Testing

- What is Testing?

Many people understand many definitions of testing **:**

1. Testing is the process of demonstrating that errors are not

2. The purpose of testing is to show that a program perform functions correctly.

3. Testing is the process of establishing confidence that a p what it is supposed to do.

These definitions are incorrect.

Software Testing

A more appropriate definition is:

"*Testing is the process of executing a prog*
the intent of finding errors."

- Why should We Test ?

Although software testing is itself an expensive activity, yet software without testing may lead to cost potentially much hig of testing, specially in systems where human safety is involvec

In the software life cycle the earlier the errors are discovered the lower is the cost of their removal.

- Who should Do the Testing ?

  - Testing requires the developers to find errors from their
  - It is difficult for software developer to point out erro creations.
  - Many organisations have made a distinction between and testing phase by making different people respons phase.

- What should We Test ?

We should test the program's responses to every possible inp we should test for all valid and invalid inputs. Suppose a prog two 8 bit integers as inputs. Total possible combinations are one second it required to execute one set of inputs, it may tak test all combinations. Practically, inputs are more than two an more than 8 bits. We have also not considered invalid inpu many combinations are possible. Hence, complete testing possible, although, we may wish to do so.

Software Testing

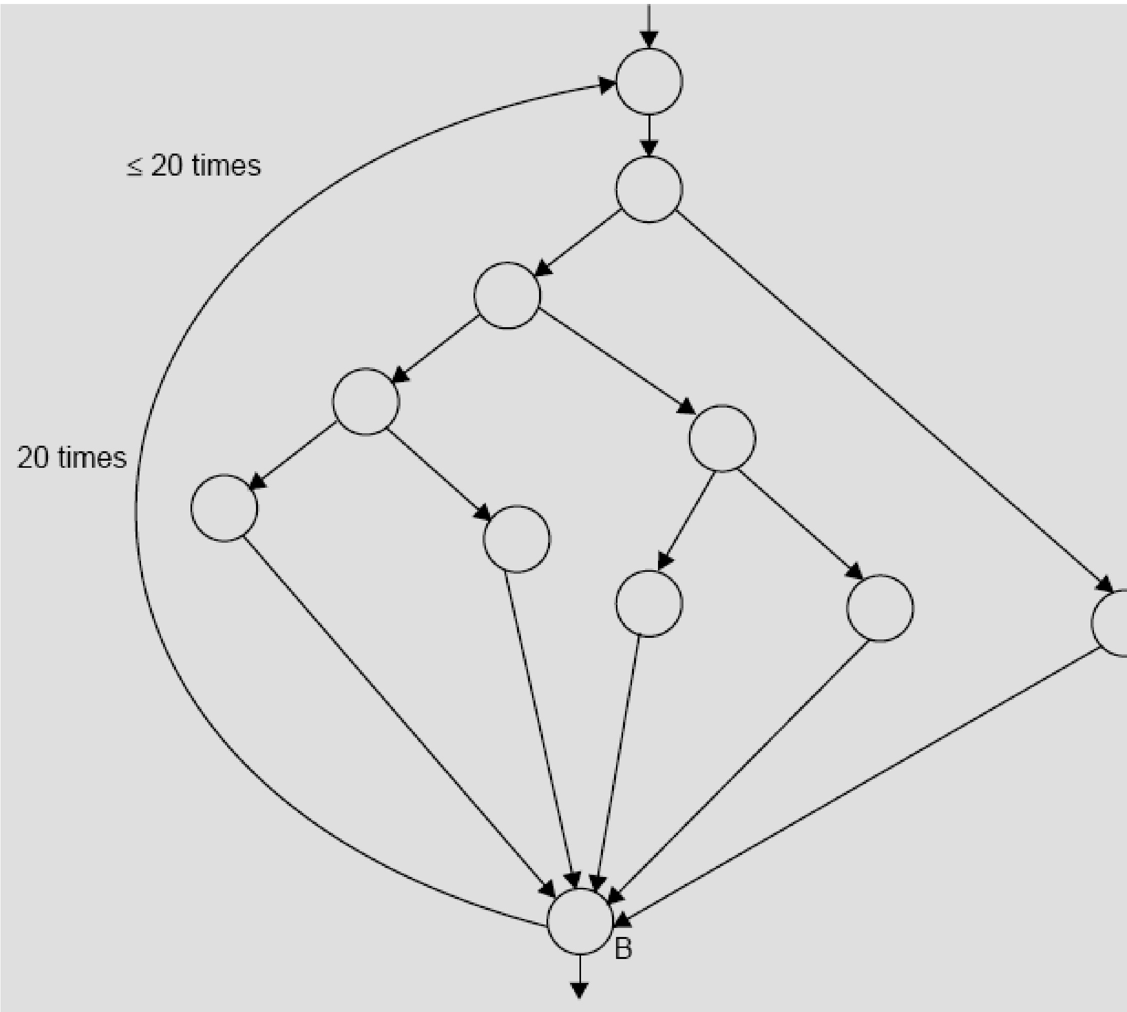


Fig. 1: Control flow graph

The number of paths in the example of Fig. 1 are 10^{14} or 100
computed from $5^{20} + 5^{19} + 5^{18} + \ldots\ldots + 5^{1}$; where 5 is the nu
through the loop body. If only 5 minutes are required to test o
may take approximately one billion years to execute every pat

Some Terminologies

- **Error, Mistake, Bug, Fault and Failure**

People make **errors**. A good synonym is **mistake**. This may error or misunderstanding of specifications. Sometimes, there errors.

When developers make mistakes while coding, we call the "**bugs**".

A **fault** is the representation of an error, where representation of expression, such as narrative text, data flow diagrams, E source code etc. Defect is a good synonym for fault.

A **failure** occurs when a fault executes. A particular fault different failures, depending on how it has been exercised.

➢ Test, Test Case and Test Suite

Test and **Test case** terms are used interchangeably. In practi same and are treated as synonyms. Test case describe description and an expected output description.

Test Case ID	
Section-I (Before Execution)	Section-II (After Execution)
Purpose :	Execution History:
Pre condition: (If any)	Result:
Inputs:	If fails, any possible reason (Optional)
Expected Outputs:	Any other observation:
Post conditions:	Any suggestion:
Written by:	Run by:
Date:	Date:

Fig. 2: Test case template

The set of test cases is called a **test suite**. Hence any combir cases may generate a test suite.

- Verification and Validation

Verification is the process of evaluating a system or co determine whether the products of a given development phas conditions imposed at the start of that phase.

Validation is the process of evaluating a system or componen the end of development process to determine whether it specified requirements .

Testing= Verification+Validation

Alpha, Beta and Acceptance Testing

The term **Acceptance Testing** is used when the software is d
a specific customer. A series of tests are conducted to enable
to validate all requirements. These tests are conducted by th
customer and may range from adhoc tests to well planne
series of tests.

The terms alpha and beta testing are used when the software
as a product for anonymous customers.

Alpha Tests are conducted at the developer's site by so
customers. These tests are conducted in a controlled enviro
testing may be started when formal testing process is near co

Beta Tests are conducted by the customers / end users a
Unlike alpha testing, developer is not present here. Bet
conducted in a real environment that cannot be controlled by th

Functional Testing

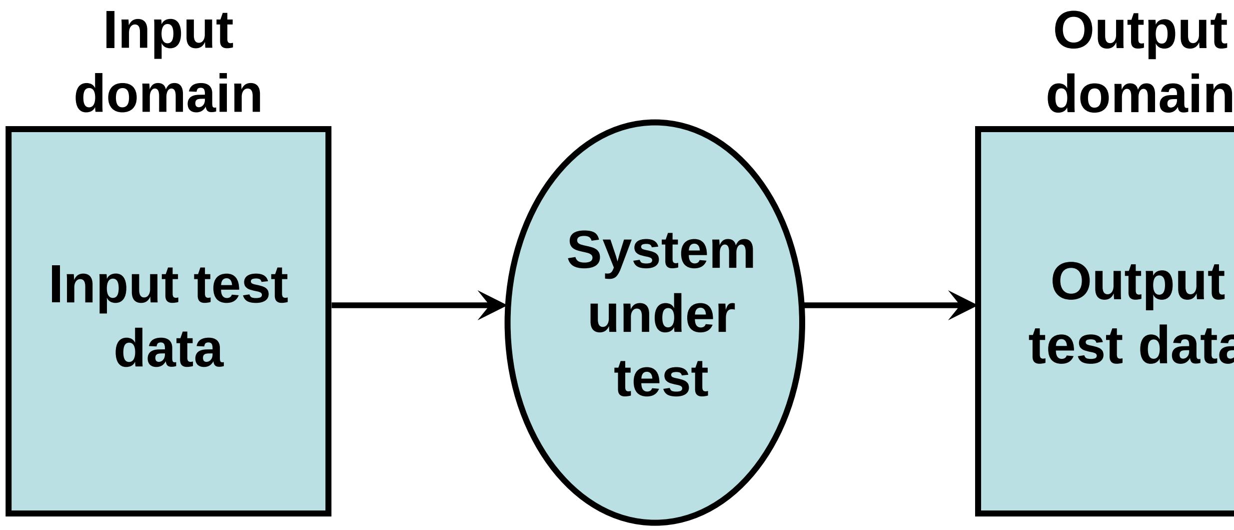


Fig. 3: Black box testing

Boundary Value Analysis

Consider a program with two input variables x and y. These in
have specified boundaries as:

$$a \le x \le b$$

$$c \le y \le d$$

Input domain

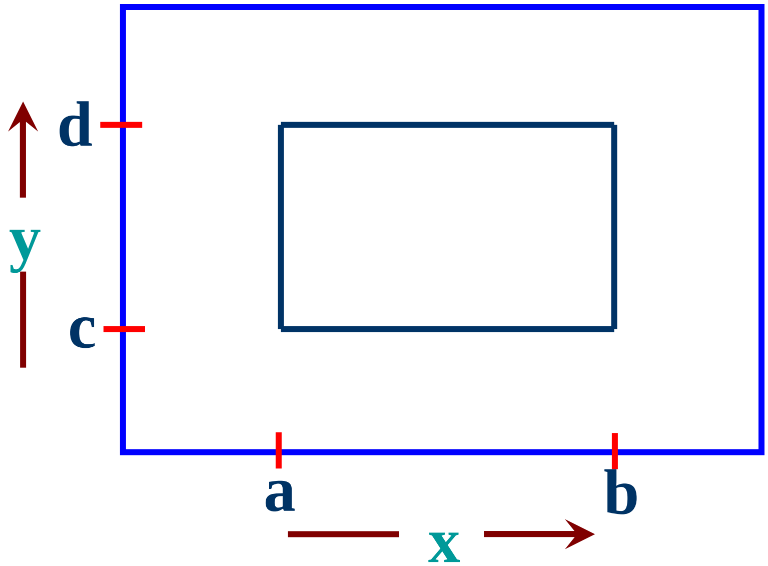


Fig.4: Input domain for program having two input variable

The boundary value analysis test cases for our program wi
variables (x and y) that may have any value from 100 to 300 a
(200,101), (200,200), (200,299), (200,300), (100,200), (101,200),
(300,200). This input domain is shown in Fig. 8.5. Each dot represe
and inner rectangle is the domain of legitimate inputs. Thus, for a
variables, boundary value analysis yield **4n + 1** test cases.

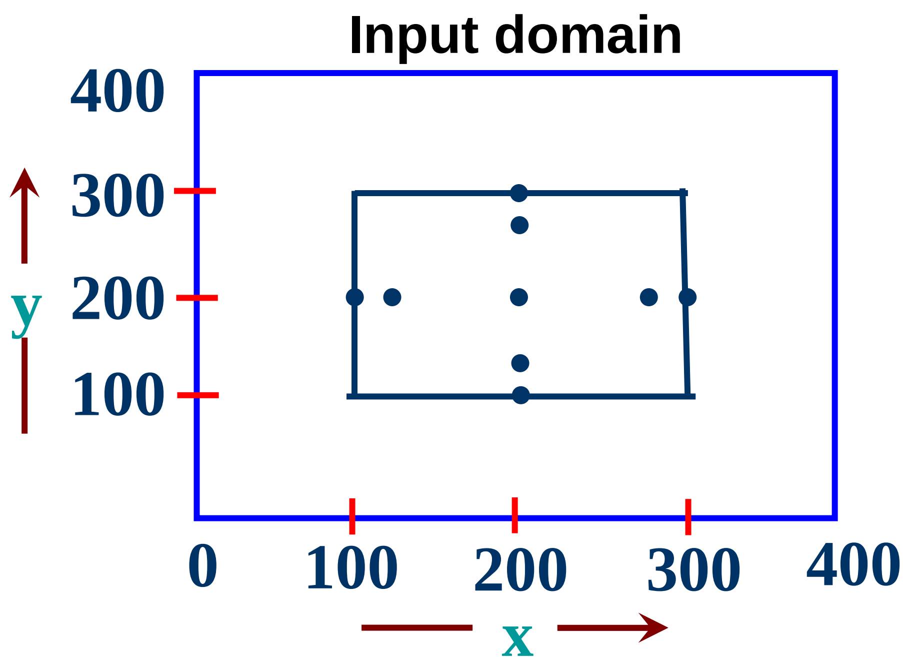


Fig. 5: Input domain of two variables x and y with boundaries [100,300] each

Example- 8.I

Consider a program for the determination of the nature o
quadratic equation. Its input is a triple of positive integers (sa
values may be from interval [0,100]. The program output may
the following words.

[Not a quadratic equation; Real roots; Imaginary roots; Equal ro

Design the boundary value test cases.

Solution

Quadratic equation will be of type:

$$ax^2+bx+c=0$$

Roots are real if $(b^2-4ac)>0$

Roots are imaginary if $(b^2-4ac)<0$

Roots are equal if $(b^2-4ac)=0$

Equation is not quadratic if $a=0$

The boundary value test cases are :

Test Case	a	b	c	Expected
1	0	50	50	Not Qua
2	1	50	50	Real R
3	50	50	50	Imaginary
4	99	50	50	Imaginary
5	100	50	50	Imaginary
6	50	0	50	Imaginary
7	50	1	50	Imaginary
8	50	99	50	Imaginary
9	50	100	50	Equal F
10	50	50	0	Real R
11	50	50	1	Real R
12	50	50	99	Imaginary
13	50	50	100	Imaginary

Example – 8.2

Consider a program for determining the Previous date. Its input
day, month and year with the values in the range

$1 \leq month \leq 12$

$1 \leq day \leq 31$

$1900 \leq year \leq 2025$

The possible outputs would be Previous date or invalid input da
boundary value test cases.

Solution

The Previous date program takes a date as input and checks it
If valid, it returns the previous date as its output.

With single fault assumption theory, 4n+1 test cases can be des
which are equal to 13.

The boundary value test cases are:

Test Case	*Month*	*Day*	*Year*	*Expected*
1	6	15	1900	14 June,
2	6	15	1901	14 June,
3	6	15	1962	14 June,
4	6	15	2024	14 June,
5	6	15	2025	14 June,
6	6	1	1962	31 May, 1
7	6	2	1962	1 June, 1
8	6	30	1962	29 June,
9	6	31	1962	Invalid
10	1	15	1962	14 Januar
11	2	15	1962	14 Februa
12	11	15	1962	14 Novemb
13	12	15	1962	14 Decemb

Example – 8.3

Consider a simple program to classify a triangle. Its inputs positive integers (say x, y, z) and the date type for input param that these will be integers greater than 0 and less than or equ program output may be one of the following words:

[Scalene; Isosceles; Equilateral; Not a triangle]

Design the boundary value test cases.

Solution

The boundary value test cases are shown below:

Test case	***x***	***y***	***z***	***Expected Output***
1	50	50	1	Isosceles
2	50	50	2	Isosceles
3	50	50	50	Equilateral
4	50	50	99	Isosceles
5	50	50	100	Not a triangle
6	50	1	50	Isosceles
7	50	2	50	Isosceles
8	50	99	50	Isosceles
9	50	100	50	Not a triangle
10	1	50	50	Isosceles
11	2	50	50	Isosceles
12	99	50	50	Isosceles
13	100	50	50	Not a triangle

Robustness testing

It is nothing but the extension of boundary value analysis. He
like to see, what happens when the extreme values are exc
value slightly greater than the maximum, and a value sligh
minimum. It means, we want to go outside the legitimate bour
domain. This extended form of boundary value analys
robustness testing and shown in Fig. 6

There are four additional test cases which are outside the leg
domain. Hence total test cases in robustness testing are 6n+
the number of input variables. So, 13 test cases are:

(200,99), (200,100), (200,101), (200,200), (200,299), (200,300)

(200,301), (99,200), (100,200), (101,200), (299,200), (300,200)

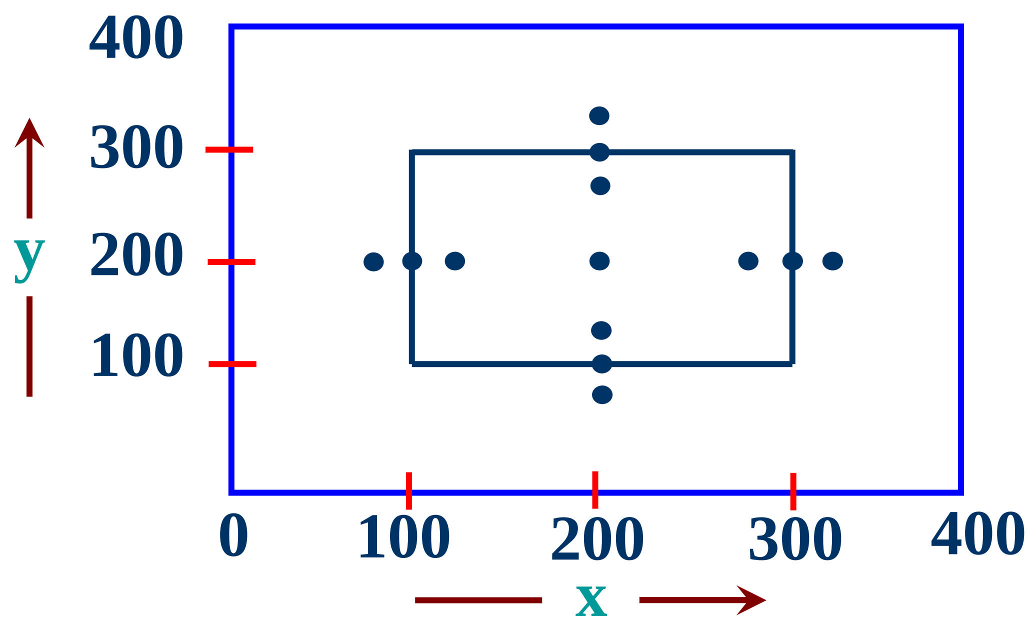


Fig. 8.6: Robustness test cases for two variables x and y with range [100,300] each

Worst-case testing

If we reject "single fault" assumption theory of reliability and m
what happens when more than one variable has an extre
electronic circuits analysis, this is called "worst case analysi
thorough in the sense that boundary value test cases are a p
of worst case test cases. It requires more effort. Worst case
function of *n* variables generate 5^n test cases as opposed
cases for boundary value analysis. Our two variables exam
5^2=25 test cases and are given in table 1.

Table 1: Worst cases test inputs for two variables example

Test case number	Inputs		Test case number	Inputs	
	x	***y***		***x***	***y***
1	100	100	14	200	299
2	100	101	15	200	300
3	100	200	16	299	100
4	100	299	17	299	101
5	100	300	18	299	200
6	101	100	19	299	299
7	101	101	20	299	300
8	101	200	21	300	100
9	101	299	22	300	101
10	101	300	23	300	200
11	200	100	24	300	299
12	200	101	25	300	300
13	200	200	--		

Example - 8.4

Consider the program for the determination of nature of roots
equation as explained in example 8.1. Design the Robust test c
test cases for this program.

Solution

Robust test cases are 6n+1. Hence, in 3 variable input cases t
of test cases are 19 as given on next slide:

Test case	a	b	c	Expected Output
1	-1	50	50	Invalid input`
2	0	50	50	Not quadratic equation
3	1	50	50	Real roots
4	50	50	50	Imaginary roots
5	99	50	50	Imaginary roots
6	100	50	50	Imaginary roots
7	101	50	50	Invalid input
8	50	-1	50	Invalid input
9	50	0	50	Imaginary roots
10	50	1	50	Imaginary roots
11	50	99	50	Imaginary roots
12	50	100	50	Equal roots
13	50	101	50	Invalid input
14	50	50	-1	Invalid input
15	50	50	0	Real roots
16	50	50	1	Real roots
17	50	50	99	Imaginary roots
18	50	50	100	Imaginary roots
19	50	50	101	Invalid input

In case of worst test case total test cases are 5^n. Hence, 125 test cas generated in worst test cases. The worst test cases are given below:

Test Case	a	b	c	Expected output
1	0	0	0	Not Quadratic
2	0	0	1	Not Quadratic
3	0	0	50	Not Quadratic
4	0	0	99	Not Quadratic
5	0	0	100	Not Quadratic
6	0	1	0	Not Quadratic
7	0	1	1	Not Quadratic
8	0	1	50	Not Quadratic
9	0	1	99	Not Quadratic
10	0	1	100	Not Quadratic
11	0	50	0	Not Quadratic
12	0	50	1	Not Quadratic
13	0	50	50	Not Quadratic
14	0	50	99	Not Quadratic

Test Case	A	b	c	Expected outpu
15	0	50	100	Not Quadratic
16	0	99	0	Not Quadratic
17	0	99	1	Not Quadratic
18	0	99	50	Not Quadratic
19	0	99	99	Not Quadratic
20	0	99	100	Not Quadratic
21	0	100	0	Not Quadratic
22	0	100	1	Not Quadratic
23	0	100	50	Not Quadratic
24	0	100	99	Not Quadratic
25	0	100	100	Not Quadratic
26	1	0	0	Equal Roots
27	1	0	1	Imaginary
28	1	0	50	Imaginary
29	1	0	99	Imaginary
30	1	0	100	Imaginary
31	1	1	0	Real Roots

Software Testing

Test Case	A	b	C	Expected outpu
32	1	1	1	Imaginary
33	1	1	50	Imaginary
34	1	1	99	Imaginary
35	1	1	100	Imaginary
36	1	50	0	Real Roots
37	1	50	1	Real Roots
38	1	50	50	Real Roots
39	1	50	99	Real Roots
40	1	50	100	Real Roots
41	1	99	0	Real Roots
42	1	99	1	Real Roots
43	1	99	50	Real Roots
44`	1	99	99	Real Roots
45	1	99	100	Real Roots
46	1	100	0	Real Roots
47	1	100	1	Real Roots
48	1	100	50	Real Roots

Test Case	A	b	C	Expected outpu
49	1	100	99	Real Roots
50	1	100	100	Real Roots
51	50	0	0	Equal Roots
52	50	0	1	Imaginary
53	50	0	50	Imaginary
54	50	0	99	Imaginary
55	50	0	100	Imaginary
56	50	1	0	Real Roots
57	50	1	1	Imaginary
58	50	1	50	Imaginary
59	50	1	99	Imaginary
60	50	1	100	Imaginary
61	50	50	0	Real Roots
62	50	50	1	Real Roots
63	50	50	50	Imaginary
64	50	50	99	Imaginary
65	50	50	100	Imaginary

Test Case	A	b	C	Expected outpu
66	50	99	0	Real Roots
67	50	99	1	Real Roots
68	50	99	50	Imaginary
69	50	99	99	Imaginary
70	50	99	100	Imaginary
71	50	100	0	Real Roots
72	50	100	1	Real Roots
73	50	100	50	Equal Roots
74	50	100	99	Imaginary
75	50	100	100	Imaginary
76	99	0	0	Equal Roots
77	99	0	1	Imaginary
78	99	0	50	Imaginary
79	99	0	99	Imaginary
80	99	0	100	Imaginary
81	99	1	0	Real Roots
82	99	1	1	Imaginary

Test Case	A	b	C	Expected outpu
83	99	1	50	Imaginary
84	99	1	99	Imaginary
85	99	1	100	Imaginary
86	99	50	0	Real Roots
87	99	50	1	Real Roots
88	99	50	50	Imaginary
89	99	50	99	Imaginary
90	99	50	100	Imaginary
91	99	99	0	Real Roots
92	99	99	1	Real Roots
93	99	99	50	Imaginary Roots
94	99	99	99	Imaginary
95	99	99	100	Imaginary
96	99	100	0	Real Roots
97	99	100	1	Real Roots
98	99	100	50	Imaginary
99	99	100	99	Imaginary
100	99	100	100	Imaginary

Test Case	A	b	C	Expected outpu
101	100	0	0	Equal Roots
102	100	0	1	Imaginary
103	100	0	50	Imaginary
104	100	0	99	Imaginary
105	100	0	100	Imaginary
106	100	1	0	Real Roots
107	100	1	1	Imaginary
108	100	1	50	Imaginary
109	100	1	99	Imaginary
110	100	1	100	Imaginary
111	100	50	0	Real Roots
112	100	50	1	Real Roots
113	100	50	50	Imaginary
114	100	50	99	Imaginary
115	100	50	100	Imaginary
116	100	99	0	Real Roots
117	100	99	1	Real Roots
118	100	99	50	Imaginary

Test Case	A	b	C	Expected outpu
119	100	99	99	Imaginary
120	100	99	100	Imaginary
121	100	100	0	Real Roots
122	100	100	1	Real Roots
123	100	100	50	Imaginary
124	100	100	99	Imaginary
125	100	100	100	Imaginary

Example – 8.5

Consider the program for the determination of previous date in explained in example 8.2. Design the robust and worst test program.

Solution

Robust test cases are 6*n*+1. Hence total 19 robust test cases a
and are given on next slide.

Test case	Month	Day	Year	Expected Outpu
1	6	15	1899	Invalid date (outside r
2	6	15	1900	14 June, 1900
3	6	15	1901	14 June, 1901
4	6	15	1962	14 June, 1962
5	6	15	2024	14 June, 2024
6	6	15	2025	14 June, 2025
7	6	15	2026	Invalid date (outside ra
8	6	0	1962	Invalid date
9	6	1	1962	31 May, 1962
10	6	2	1962	1 June, 1962
11	6	30	1962	29 June, 1962
12	6	31	1962	Invalid date
13	6	32	1962	Invalid date
14	0	15	1962	Invalid date
15	1	15	1962	14 January, 1962
16	2	15	1962	14 February, 196
17	11	15	1962	14 November, 196
18	12	15	1962	14 December, 196
19	13	15	1962	Invalid date

In case of worst test case total test cases are 5^n. Hence, 125 test cas generated in worst test cases. The worst test cases are given below:

Test Case	Month	Day	Year	Expected output
1	1	1	1900	31 December, 1899
2	1	1	1901	31 December, 1900
3	1	1	1962	31 December, 1961
4	1	1	2024	31 December, 2023
5	1	1	2025	31 December, 2024
6	1	2	1900	1 January, 1900
7	1	2	1901	1 January, 1901
8	1	2	1962	1 January, 1962
9	1	2	2024	1 January, 2024
10	1	2	2025	1 January, 2025
11	1	15	1900	14 January, 1900
12	1	15	1901	14 January, 1901
13	1	15	1962	14 January, 1962
14	1	15	2024	14 January, 2024

Test Case	A	b	c	Expected outpu
15	1	15	2025	14 January, 2025
16	1	30	1900	29 January, 1900
17	1	30	1901	29 January, 1901
18	1	30	1962	29 January, 1962
19	1	30	2024	29 January, 2024
20	1	30	2025	29 January, 2025
21	1	31	1900	30 January, 1900
22	1	31	1901	30 January, 1901
23	1	31	1962	30 January, 1962
24	1	31	2024	30 January, 2024
25	1	31	2025	30 January, 2025
26	2	1	1900	31 January, 1900
27	2	1	1901	31 January, 1901
28	2	1	1962	31 January, 1962
29	2	1	2024	31 January, 2024
30	2	1	2025	31 January, 2025
31	2	2	1900	1 February, 1900

Software Testing

Test Case	***Month***	***Day***	***Year***	***Expected outpu***
32	2	2	1901	1 February, 1901
33	2	2	1962	1 February, 1962
34	2	2	2024	1 February, 2024
35	2	2	2025	1 February, 2025
36	2	15	1900	14 February, 190
37	2	15	1901	14 February, 190
38	2	15	1962	14 February, 196
39	2	15	2024	14 February, 202
40	2	15	2025	14 February, 202
41	2	30	1900	Invalid date
42	2	30	1901	Invalid date
43	2	30	1962	Invalid date
44	2	30	2024	Invalid date
45	2	30	2025	Invalid date
46	2	31	1900	Invalid date
47	2	31	1901	Invalid date
48	2	31	1962	Invalid date

Test Case	Month	Day	Year	Expected outpu
49	2	31	2024	Invalid date
50	2	31	2025	Invalid date
51	6	1	1900	31 May, 1900
52	6	1	1901	31 May, 1901
53	6	1	1962	31 May, 1962
54	6	1	2024	31 May, 2024
55	6	1	2025	31 May, 2025
56	6	2	1900	1 June, 1900
57	6	2	1901	1 June, 1901
58	6	2	1962	1 June, 1962
59	6	2	2024	1 June, 2024
60	6	2	2025	1 June, 2025
61	6	15	1900	14 June, 1900
62	6	15	1901	14 June, 1901
63	6	15	1962	14 June, 1962
64	6	15	2024	14 June, 2024
65	6	15	2025	14 June, 2025

Test Case	Month	Day	Year	Expected outpu
66	6	30	1900	29 June, 1900
67	6	30	1901	29 June, 1901
68	6	30	1962	29 June, 1962
69	6	30	2024	29 June, 2024
70	6	30	2025	29 June, 2025
71	6	31	1900	Invalid date
72	6	31	1901	Invalid date
73	6	31	1962	Invalid date
74	6	31	2024	Invalid date
75	6	31	2025	Invalid date
76	11	1	1900	31 October, 1900
77	11	1	1901	31 October, 1901
78	11	1	1962	31 October, 1962
79	11	1	2024	31 October, 2024
80	11	1	2025	31 October, 2025
81	11	2	1900	1 November, 190
82	11	2	1901	1 November, 190

Test Case	Month	Day	Year	Expected outpu
83	11	2	1962	1 November, 196
84	11	2	2024	1 November, 202
85	11	2	2025	1 November, 202
86	11	15	1900	14 November, 190
87	11	15	1901	14 November, 190
88	11	15	1962	14 November, 196
89	11	15	2024	14 November, 202
90	11	15	2025	14 November, 202
91	11	30	1900	29 November, 190
92	11	30	1901	29 November, 190
93	11	30	1962	29 November, 196
94	11	30	2024	29 November, 202
95	11	30	2025	29 November, 202
96	11	31	1900	Invalid date
97	11	31	1901	Invalid date
98	11	31	1962	Invalid date
99	11	31	2024	Invalid date
100	11	31	2025	Invalid date

Test Case	Month	Day	Year	Expected outpu
101	12	1	1900	30 November, 190
102	12	1	1901	30 November, 190
103	12	1	1962	30 November, 196
104	12	1	2024	30 November, 202
105	12	1	2025	30 November, 202
106	12	2	1900	1 December, 190
107	12	2	1901	1 December, 190
108	12	2	1962	1 December, 196
109	12	2	2024	1 December, 202
110	12	2	2025	1 December, 202
111	12	15	1900	14 December, 190
112	12	15	1901	14 December, 190
113	12	15	1962	14 December, 196
114	12	15	2024	14 December, 202
115	12	15	2025	14 December, 202
116	12	30	1900	29 December, 190
117	12	30	1901	29 December, 190
118	12	30	1962	29 December, 196

Test Case	Month	Day	Year	Expected outpu
119	12	30	2024	29 December, 202
120	12	30	2025	29 December, 202
121	12	31	1900	30 December, 190
122	12	31	1901	30 December, 190
123	12	31	1962	30 December, 196
124	12	31	2024	30 December, 202
125	12	31	2025	30 December, 202

Example – 8.6

Consider the triangle problem as given in example 8.3. Generat
worst test cases for this problem.

Solution

Robust test cases are given on next slide.

`	x	y	z	Expected Output
1	50	50	0	Invalid input`
2	50	50	1	Isosceles
3	50	50	2	Isosceles
4	50	50	50	Equilateral
5	50	50	99	Isosceles
6	50	50	100	Not a triangle
7	50	50	101	Invalid input
8	50	0	50	Invalid input
9	50	1	50	Isosceles
10	50	2	50	Isosceles
11	50	99	50	Isosceles
12	50	100	50	Not a triangle
13	50	101	50	Invalid input
14	0	50	50	Invalid input
15	1	50	50	Isosceles
16	2	50	50	Isosceles
17	99	50	50	Isosceles
18	100	50	50	Not a triangle
19	100	50	50	Invalid input

Worst test cases are 125 and are given below:

Test Case	x	y	z	Expected output
1	1	1	1	Equilateral
2	1	1	2	Not a triangle
3	1	1	50	Not a triangle
4	1	1	99	Not a triangle
5	1	1	100	Not a triangle
6	1	2	1	Not a triangle
7	1	2	2	Isosceles
8	1	2	50	Not a triangle
9	1	2	99	Not a triangle
10	1	2	100	Not a triangle
11	1	50	1	Not a triangle
12	1	50	2	Not a triangle
13	1	50	50	Isosceles
14	1	50	99	Not a triangle

Test Case	A	b	c	Expected outpu
15	1	50	100	Not a triangle
16	1	99	1	Not a triangle
17	1	99	2	Not a triangle
18	1	99	50	Not a triangle
19	1	99	99	Isosceles
20	1	99	100	Not a triangle
21	1	100	1	Not a triangle
22	1	100	2	Not a triangle
23	1	100	50	Not a triangle
24	1	100	99	Not a triangle
25	1	100	100	Isosceles
26	2	1	1	Not a triangle
27	2	1	2	Isosceles
28	2	1	50	Not a triangle
29	2	1	99	Not a triangle
30	2	1	100	Not a triangle
31	2	2	1	Isosceles

Software Testing

Test Case	A	b	C	Expected outpu
32	2	2	2	Equilateral
33	2	2	50	Not a triangle
34	2	2	99	Not a triangle
35	2	2	100	Not a triangle
36	2	50	1	Not a triangle
37	2	50	2	Not a triangle
38	2	50	50	Isosceles
39	2	50	99	Not a triangle
40	2	50	100	Not a triangle
41	2	99	1	Not a triangle
42	2	99	2	Not a triangle
43	2	99	50	Not a triangle
44	2	99	99	Isosceles
45	2	99	100	Scalene
46	2	100	1	Not a triangle
47	2	100	2	Not a triangle
48	2	100	50	Not a triangle

Test Case	A	b	C	Expected outpu
49	2	100	50	Scalene
50	2	100	99	Isosceles
51	50	1	100	Not a triangle
52	50	1	1	Not a triangle
53	50	1	2	Isosceles
54	50	1	50	Not a triangle
55	50	1	99	Not a triangle
56	50	2	100	Not a triangle
57	50	2	1	Not a triangle
58	50	2	2	Isosceles
59	50	2	50	Not a triangle
60	50	2	99	Not a triangle
61	50	50	100	Isosceles
62	50	50	1	Isosceles
63	50	50	2	Equilateral
64	50	50	50	Isosceles
65	50	50	99	Not a triangle

Software Testing

Test Case	A	B	C	Expected outpu
66	50	99	1	Not a triangle
67	50	99	2	Not a triangle
68	50	99	50	Isosceles
69	50	99	99	Isosceles
70	50	99	100	Scalene
71	50	100	1	Not a triangle
72	50	100	2	Not a triangle
73	50	100	50	Not a triangle
74	50	100	99	Scalene
75	50	100	100	Isosceles
76	50	1	1	Not a triangle
77	99	1	2	Not a triangle
78	99	1	50	Not a triangle
79	99	1	99	Isosceles
80	99	1	100	Not a triangle
81	99	2	1	Not a triangle
82	99	2	2	Not a triangle

Test Case	A	b	C	Expected outpu
83	99	2	50	Not a triangle
84	99	2	99	Isosceles
85	99	2	100	Scalene
86	99	50	1	Not a triangle
87	99	50	2	Not a triangle
88	99	50	50	Isosceles
89	99	50	99	Isosceles
90	99	50	100	Scalene
91	99	99	1	Isosceles
92	99	99	2	Isosceles
93	99	99	50	Isosceles
94	99	99	99	Equilateral
95	99	99	100	Isosceles
96	99	100	1	Not a triangle
97	99	100	2	Scalene
98	99	100	50	Scalene
99	99	100	99	Isosceles
100	99	100	100	Isosceles

Test Case	A	b	C	Expected outpu
101	100	1	1	Not a triangle
102	100	1	2	Not a triangle
103	100	1	50	Not a triangle
104	100	1	99	Not a triangle
105	100	1	100	Isosceles
106	100	2	1	Not a triangle
107	100	2	2	Not a triangle
108	100	2	50	Not a triangle
109	100	2	99	Scalene
110	100	2	100	Isosceles
111	100	50	1	Not a triangle
112	100	50	2	Not a triangle
113	100	50	50	Not a triangle
114	100	50	99	Scalene
115	100	50	100	Isosceles
116	100	99	1	Not a triangle
117	100	99	2	Scalene
118	100	99	50	Scalene

Test Case	A	b	C	Expected outpu
119	100	99	99	Isosceles
120	100	99	100	Isosceles
121	100	100	1	Isosceles
122	100	100	2	Isosceles
123	100	100	50	Isosceles
124	100	100	99	Isosceles
125	100	100	100	Equilateral

Equivalence Class Testing

In this method, input domain of a program is partitioned into a fi equivalence classes such that one can reasonably assume absolutely sure, that the test of a representative value of equivalent to a test of any other value.

Two steps are required to implementing this method:

1. The equivalence classes are identified by taking each input partitioning it into valid and invalid classes. For example condition specifies a range of values from 1 to 999, we ide equivalence class [1<item<999]; and two invalid equival [item<1] and [item>999].

2. Generate the test cases using the equivalence classes ide previous step. This is performed by writing test cases coveri equivalence classes. Then a test case is written for each invali class so that no test contains more than one invalid class. Th that no two invalid classes mask each other.

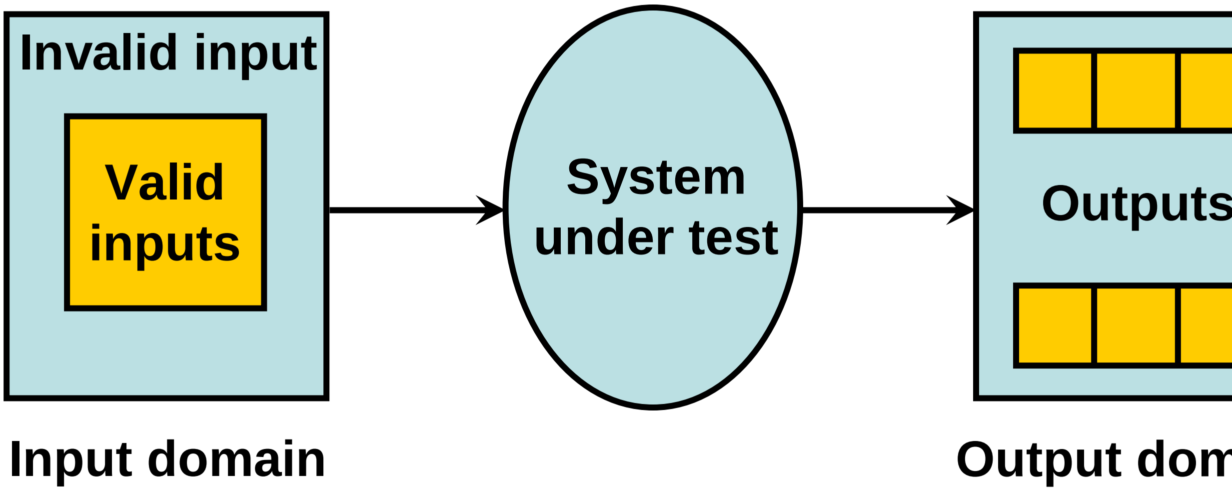


Fig. 7: Equivalence partitioning

Most of the time, equivalence class testing defines classes of the
However, equivalence classes should also be defined for ou
Hence, we should design equivalence classes based on inpu
domain.

Example 8.7

Consider the program for the determination of nature of roots o
equation as explained in example 8.1. Identify the equivalen
cases for output and input domains.

Solution

Output domain equivalence class test cases can be identified a

$O_1=\{<a,b,c>:\text{Not a quadratic equation if } a = 0\}$

$O_1=\{<a,b,c>:\text{Real roots if } (b^2-4ac)>0\}$

$O_1=\{<a,b,c>:\text{Imaginary roots if } (b^2-4ac)<0\}$

$O_1=\{<a,b,c>:\text{Equal roots if } (b^2-4ac)=0\}$`

The number of test cases can be derived form above relations a below:

Test case	*a*	*b*	*c*	*Expected output*
1	0	50	50	Not a quadratic equati
2	1	50	50	Real roots
3	50	50	50	Imaginary roots
4	50	100	50	Equal roots

We may have another set of test cases based on input domain.

$I_1 = \{a: a = 0\}$

$I_2 = \{a: a < 0\}$

$I_3 = \{a: 1 \le a \le 100\}$

$I_4 = \{a: a > 100\}$

$I_5 = \{b: 0 \le b \le 100\}$

$I_6 = \{b: b < 0\}$

$I_7 = \{b: b > 100\}$

$I_8 = \{c: 0 \le c \le 100\}$

$I_9 = \{c: c < 0\}$

$I_{10} = \{c: c > 100\}$

Test Case	a	b	c	Expecte
1	0	50	50	Not a quadr
2	-1	50	50	Invalid
3	50	50	50	Imaginary
4	101	50	50	invalid
5	50	50	50	Imaginary
6	50	-1	50	invalid
7	50	101	50	invalid
8	50	50	50	Imaginary
9	50	50	-1	invalid
10	50	50	101	invalid

Here test cases 5 and 8 are redundant test cases. If we choose ar than nominal, we may not have redundant test cases. Hence total t 10+4=14 for this problem.

Example 8.8

Consider the program for determining the previous date in a ca
explained in example 8.3. Identify the equivalence class test ca
& input domains.

Solution

Output domain equivalence class are:

O_1={<D,M,Y>: Previous date if all are valid inputs}

O_1={<D,M,Y>: Invalid date if any input makes the date invalid}

Test case	M	D	Y	Expected ou
1	6	15	1962	14 June, 19
2	6	31	1962	Invalid dat

Software Testing

We may have another set of test cases which are based on input dom

I_1={month: $1 \le m \le 12$}

I_2={month: $m < 1$}

I_3={month: $m > 12$}

I_4={day: $1 \le D \le 31$}

I_5={day: $D < 1$}

I_6={day: $D > 31$}

I_7={year: $1900 \le Y \le 2025$}

I_8={year: $Y < 1900$}

I_9={year: $Y > 2025$}

Inputs domain test cases are :

Test Case	M	D	Y	Expect
1	6	15	1962	14 Jun
2	-1	15	1962	Invalid
3	13	15	1962	invalid
4	6	15	1962	14 June
5	6	-1	1962	invalid
6	6	32	1962	invalid
7	6	15	1962	14 June
8	6	15	1899	invalid input (V
9	6	15	2026	invalid input (V

Example – 8.9

Consider the triangle problem specified in a example 8.3.
equivalence class test cases for output and input domain.

Solution

Output domain equivalence classes are:

O_1={<x,y,z>: Equilateral triangle with sides x,y,z}
O_1={<x,y,z>: Isosceles triangle with sides x,y,z}
O_1={<x,y,z>: Scalene triangle with sides x,y,z}
O_1={<x,y,z>: Not a triangle with sides x,y,z}

The test cases are:

Test case	*x*	*y*	*z*	*Expected Output*
1	50	50	50	Equilateral
2	50	50	99	Isosceles
3	100	99	50	Scalene
4	50	100	50	Not a triangle

Input domain based classes are:

$I_1=\{x: x < 1\}$

$I_2=\{x: x > 100\}$

$I_3=\{x: 1 \le x \le 100\}$

$I_4=\{y: y < 1\}$

$I_5=\{y: y > 100\}$

$I_6=\{y: 1 \le y \le 100\}$

$I_7=\{z: z < 1\}$

$I_8=\{z: z > 100\}$

$I_9=\{z: 1 \le z \le 100\}$

Some inputs domain test cases can be obtained using the relationship
and z.

$I_{10}=\{< x,y,z >: x = y = z\}$

$I_{11}=\{< x,y,z >: x = y, x \neq z\}$

$I_{12}=\{< x,y,z >: x = z, x \neq y\}$

$I_{13}=\{< x,y,z >: y = z, x \neq y\}$

$I_{14}=\{< x,y,z >: x \neq y, x \neq z, y \neq z\}$

$I_{15}=\{< x,y,z >: x = y + z\}$

$I_{16}=\{< x,y,z >: x > y +z\}$

$I_{17}=\{< x,y,z >: y = x +z\}$

$I_{18}=\{< x,y,z >: y > x + z\}$

$I_{19}=\{< x,y,z >: z = x + y\}$

$I_{20}=\{< x,y,z >: z > x +y\}$

Test cases derived from input domain are:

Test case	x	y	z	Expected Output
1	0	50	50	Invalid input
2	101	50	50	Invalid input
3	50	50	50	Equilateral
4	50	0	50	Invalid input
5	50	101	50	Invalid input
6	50	50	50	Equilateral
7	50	50	0	Invalid input
8	50	50	101	Invalid input
9	50	50	50	Equilateral
10	60	60	60	Equilateral
11	50	50	60	Isosceles
12	50	60	50	Isosceles
13	60	50	50	Isosceles

Test case	x	y	z	Expected Output
14	100	99	50	Scalene
15	100	50	50	Not a triangle
16	100	50	25	Not a triangle
17	50	100	50	Not a triangle
18	50	100	25	Not a triangle
19	50	50	100	Not a triangle
20	25	50	100	Not a triangle

Decision Table Based Testing

Condition Stub		Entry					
	C_1	True					
	C_2	True		False		True	
	C_3	True	False	True	False	True	
Action Stub	a_1	X	X			X	
	a_2	X		X			
	a_3		X			X	
	a_4				X		

Table 2: Decision table terminology

Test case design

C_1:x,y,z are sides of a triangle?	N	Y					
C_2:x = y?	--	Y					
C_3:x = z?	--	Y		N		Y	
C_4:y = z?	--	Y	N	Y	N	Y	
a_1: Not a triangle	X						
a_2: Scalene							
a_3: Isosceles					X		X
a_4: Equilateral		X					
a_5: Impossible			X	X		X	

Table 3: Decision table for triangle problem

Conditions									
C_1 : x < y + z ?	F	T	T	T	T	T	T	T	T
C_2 : y < x + z ?	--	F	T	T	T	T	T	T	T
C_3 : z < x + y ?	--	--	F	T	T	T	T	T	T
C_4 : x = y ?	--	--	--	T	T	T	T	F	F
C_5 : x = z ?	--	--	--	T	T	F	F	T	T
C_6 : y = z ?	--	--	--	T	F	T	F	T	F
a_1 : Not a triangle	X	X	X						
a_2 : Scalene									
a_3 : Isosceles							X		X
a_4 : Equilateral				X					
a_5 : Impossible					X	X		X	

Table 4: Modified decision table

Example 8.10

Consider the triangle program specified in example 8.3. Id
test cases using the decision table of Table 4.

Solution

There are eleven functional test cases, three to fail triangle pro impossible cases, one each to get equilateral, scalene triangle three to get on isosceles triangle. The test cases are given in Tab

Test case	*x*	*y*	*z*	*Expected Output*
1	4	1	2	Not a triangle
2	1	4	2	Not a triangle
3	1	2	4	Not a triangle
4	5	5	5	Equilateral
5	?	?	?	Impossible
6	?	?	?	Impossible
7	2	2	3	Isosceles
8	?	?	?	Impossible
9	2	3	2	Isosceles
10	3	2	2	Isosceles
11	3	4	5	Scalene

Test cases of triangle problem using decision table

Example 8.11

Consider a program for the determination of Previous date. Its input is a
month and year with the values in the range

$1 \leq month \leq 12$

$1 \leq day \leq 31$

$1900 \leq year \leq 2025$

The possible outputs are "Previous date" and "Invalid date". Design the
using decision table based testing.

Solution

The input domain can be divided into following classes:

I_1= {M_1: month has 30 days}

I_2= {M_2: month has 31 days except March, August and Janua

I_3= {M_3: month is March}

I_4= {M_4: month is August}

I_5= {M_5: month is January}

I_6= {M_6: month is February}

I_7= {D_1: day = 1}

I_8= {D_2: $2 \leq$ day ≤ 28}

I_9= {D_3: day = 29}

I_{10}={D_4: day = 30}

I_{11}={D_5: day = 31}

I_{12}={Y_1: year is a leap year}

I_{13}={Y_2: year is a common year}

The decision table is given below:

Sr.No.	1	2	3	4	5	6	7	8	9	10	11
C_1: Months in	M_1	M_1	M_1	M_1	M_1	M_1	M_1	M_1	M_1	M_1	M_2
C_2: days in	D_1	D_1	D_2	D_2	D_3	D_3	D_4	D_4	D_5	D_5	D_1
C_3: year in	Y_1	Y_2	Y_1	Y_2	Y_1	Y_2	Y_1	Y_2	Y_1	Y_2	Y_1
a_1: Impossible									X	X	
a_2: Decrement day			X	X	X	X	X	X			
a_3: Reset day to 31	X	X									
a_4: Reset day to 30											X
a_5: Reset day to 29											
a_6: Reset day to 28											
a_7: decrement month	X	X									X
a_8: Reset month to December											
a_9: Decrement year											

Sr.No.	16	17	18	19	20	21	22	23	24	25	26
C_1: Months in	M_2	M_2	M_2	M_2	M_2	M_3	M_3	M_3	M_3	M_3	M_3
C_2: days in	D_3	D_4	D_4	D_5	D_5	D_1	D_1	D_2	D_2	D_3	D_3
C_3: year in	Y_2	Y_1	Y_2	Y_1	Y_2	Y_1	Y_2	Y_1	Y_2	Y_1	Y_2
a_1: Impossible											
a_2: Decrement day	X	X	X	X	X			X	X	X	X
a_3: Reset day to 31											
a_4: Reset day to 30											
a_5: Reset day to 29						X					
a_6: Reset day to 28							X				
a_7: decrement month						X	X				
a_8: Reset month to December											
a_9: Decrement year											

Sr.No.	31	32	33	34	35	36	37	38	39	40	41
C_1: Months in	M_4	M_4	M_4	M_4	M_4	M_4	M_4	M_4	M_4	M_4	M_5
C_2: days in	D_1	D_1	D_2	D_2	D_3	D_3	D_4	D_4	D_5	D_5	D_1
C_3: year in	Y_1	Y_2	Y_1	Y_2	Y_1	Y_2	Y_1	Y_2	Y_1	Y_2	Y_1
a_1: Impossible											
a_2: Decrement day			X	X	X	X	X	X	X	X	
a_3: Reset day to 31	X	X									X
a_4: Reset day to 30											
a_5: Reset day to 29											
a_6: Reset day to 28											
a_7: decrement month	X	X									
a_8: Reset month to December											X
a_9: Decrement year											X

Sr.No.	46	47	48	49	50	51	52	53	54	55	56
C_1: Months in	M_5	M_5	M_5	M_5	M_5	M_6	M_6	M_6	M_6	M_6	M_6
C_2: days in	D_3	D_4	D_4	D_5	D_5	D_1	D_1	D_2	D_2	D_3	D_3
C_3: year in	Y_2	Y_1	Y_2	Y_1	Y_2	Y_1	Y_2	Y_1	Y_2	Y_1	Y_2
a_1: Impossible											X
a_2: Decrement day	X	X	X	X	X			X	X	X	
a_3: Reset day to 31						X	X				
a_4: Reset day to 30											
a_5: Reset day to 29											
a_6: Reset day to 28											
a_7: decrement month						X	X				
a_8: Reset month to December											
a_9: Decrement year											

Test case	Month	Day	Year	Expected out
1	June	1	1964	31 May, 196
2	June	1	1962	31 May, 196
3	June	15	1964	14 June, 19
4	June	15	1962	14 June, 19
5	June	29	1964	28 June, 19
6	June	29	1962	28 June, 19
7	June	30	1964	29 June, 19
8	June	30	1962	29 June, 19
9	June	31	1964	Impossible
10	June	31	1962	Impossible
11	May	1	1964	30 April, 196
12	May	1	1962	30 April, 196
13	May	15	1964	14 May, 196
14	May	15	1962	14 May, 196
15	May	29	1964	28 May, 196

Test case	Month	Day	Year	Expected out
16	May	29	1962	28 May, 196
17	May	30	1964	29 May, 196
18	May	30	1962	29 May, 196
19	May	31	1964	30 May, 196
20	May	31	1962	30 May, 196
21	March	1	1964	29 February, 1
22	March	1	1962	28 February, 1
23	March	15	1964	14 March, 19
24	March	15	1962	14 March, 19
25	March	29	1964	28 March, 19
26	March	29	1962	28 March, 19
27	March	30	1964	29 March, 19
28	March	30	1962	29 March, 19
29	March	31	1964	30 March, 19
30	March	31	1962	30 March, 19

Software Testing

Test case	Month	Day	Year	Expected out
31	August	1	1964	31 July, 196
32	August	1	1962	31 July, 196
33	August	15	1964	14 August, 19
34	August	15	1962	14 August, 19
35	August	29	1964	28 August, 19
36	August	29	1962	28 August, 19
37	August	30	1964	29 August, 19
38	August	30	1962	29 August, 19
39	August	31	1964	30 August, 19
40	August	31	1962	30 August, 19
41	January	1	1964	31 December,
42	January	1	1962	31 December,
43	January	15	1964	14 January, 1
44	January	15	1962	14 January, 1
45	January	29	1964	28 January, 1

Test case	Month	Day	Year	Expected out
46	January	29	1962	28 January, 1
47	January	30	1964	29 January, 1
48	January	30	1962	29 January, 1
49	January	31	1964	30 January, 1
50	January	31	1962	30 January, 1
51	February	1	1964	31 January, 1
52	February	1	1962	31 January, 1
53	February	15	1964	14 February, 1
54	February	15	1962	14 February, 1
55	February	29	1964	28 February, 1
56	February	29	1962	Impossible
57	February	30	1964	Impossible
58	February	30	1962	Impossible
59	February	31	1964	Impossible
60	February	31	1962	Impossible

Cause Effect Graphing Technique

- Consider single input conditions
- do not explore combinations of input circumstances

Steps

1. Causes & effects in the specifications are identified.

   A cause is a distinct input condition or an equivalence class of inp

   An effect is an output condition or a system transformation.

2. The semantic content of the specification is analysed and tran boolean graph linking the causes & effects.

3. Constraints are imposed

4. graph – limited entry decision table

   Each column in the table represent a test case.

5. The columns in the decision table are converted into test cases.

The basic notation for the graph is shown in fig. 8

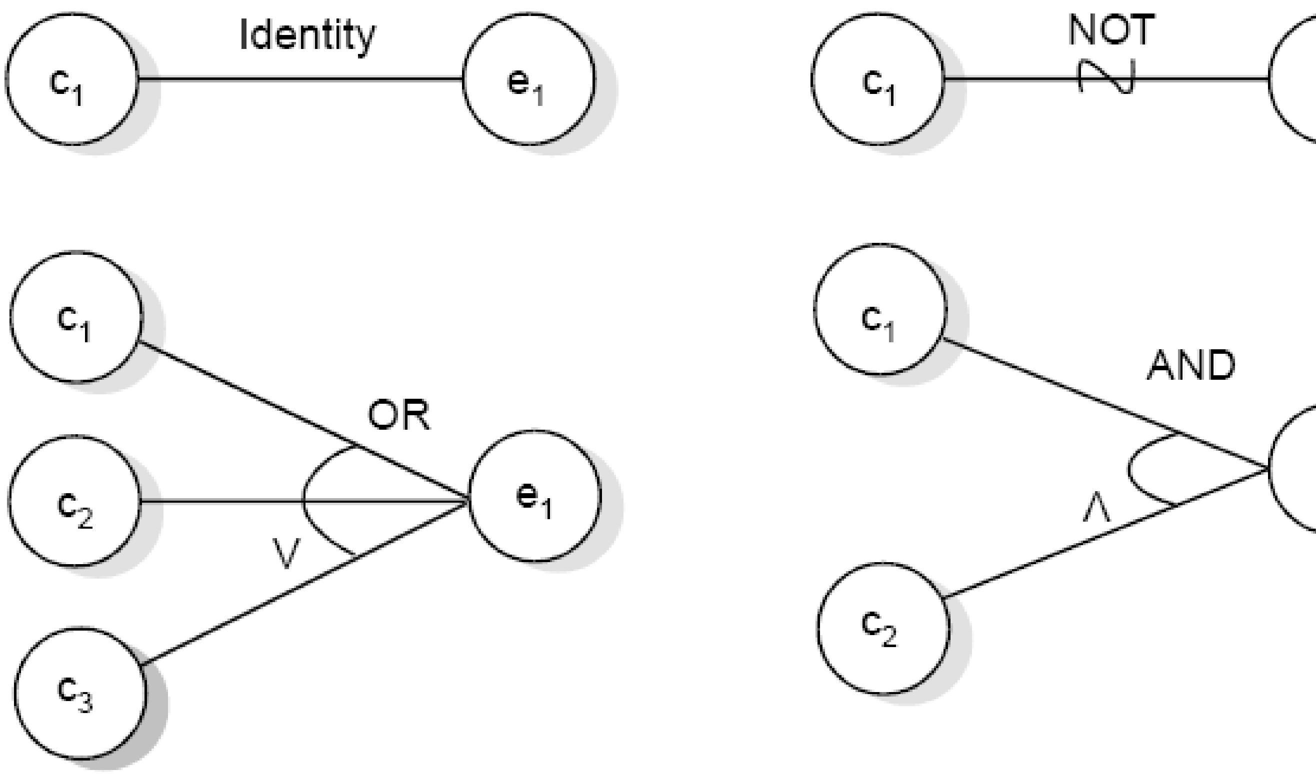


Fig.8. 8 : Basic cause effect graph symbols

Myers explained this effectively with following example. "The characte must be an A or B. The character in column 2 must be a digit. In this file update is made. If the character in column 1 is incorrect, message the character in column 2 is not a digit, message y is issued".

The causes are

c_1: character in column 1 is A

c_2: character in column 1 is B

c_3: character in column 2 is a digit

and the effects are

e_1: update made

e_2: message *x* is issued

e_3: message *y* is issued

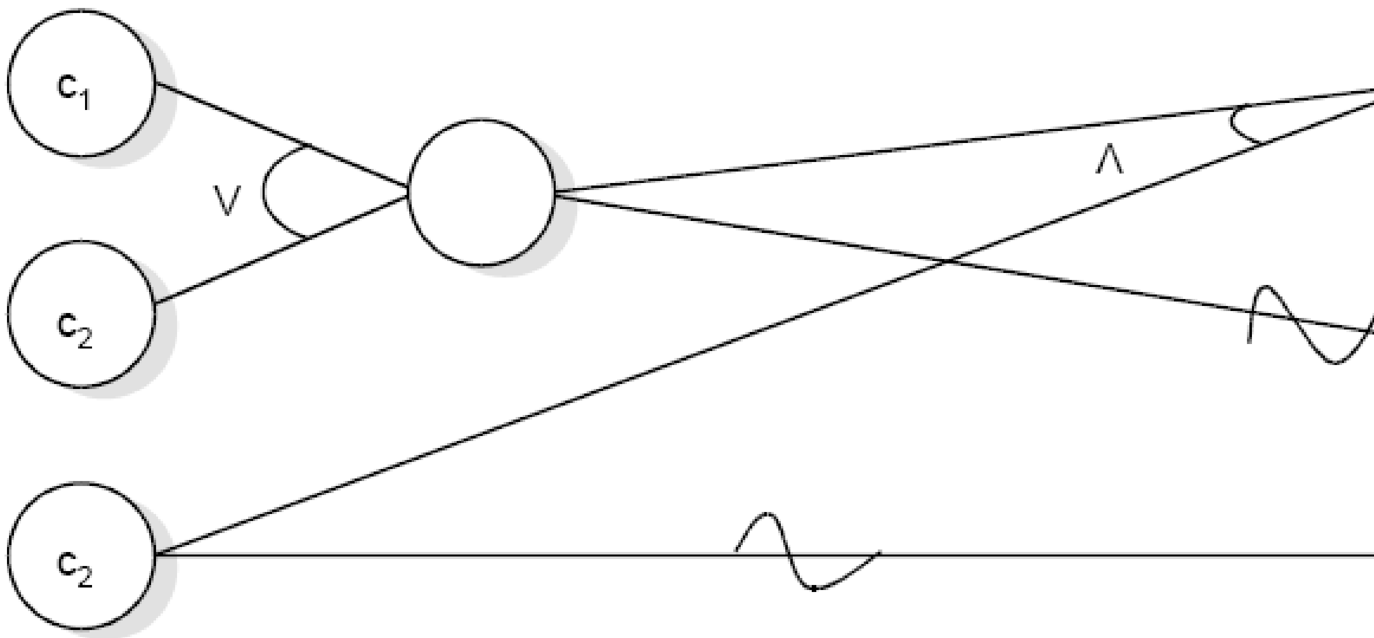


Fig. 9: Sample cause effect graph

The **E** constraint states that it must always be true th
one of c_1 or c_2 can be 1 (c_1 or c_2 cannot be 1 simultane
I constraint states that at least one of c_1, c_2 and c_3 mus
1 (c_1, c_2 and c_3 cannot be 0 simultaneously). The **O**
states that one, and only one, of c_1 and c_2 must
constraint **R** states that, for c_1 to be 1, c_2 must be
impossible for c_1 to be 1 and c_2 to be 0),

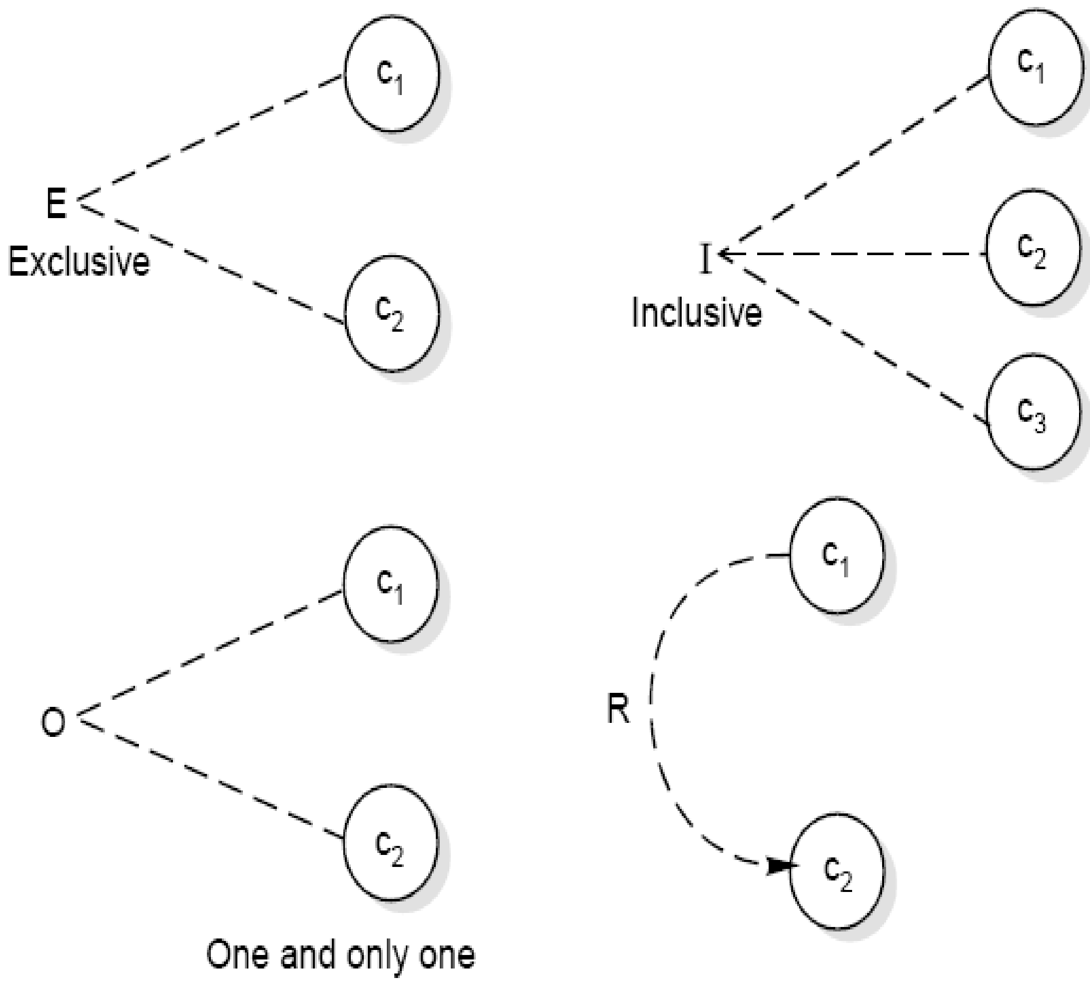


Fig. 10: Constraint symbols

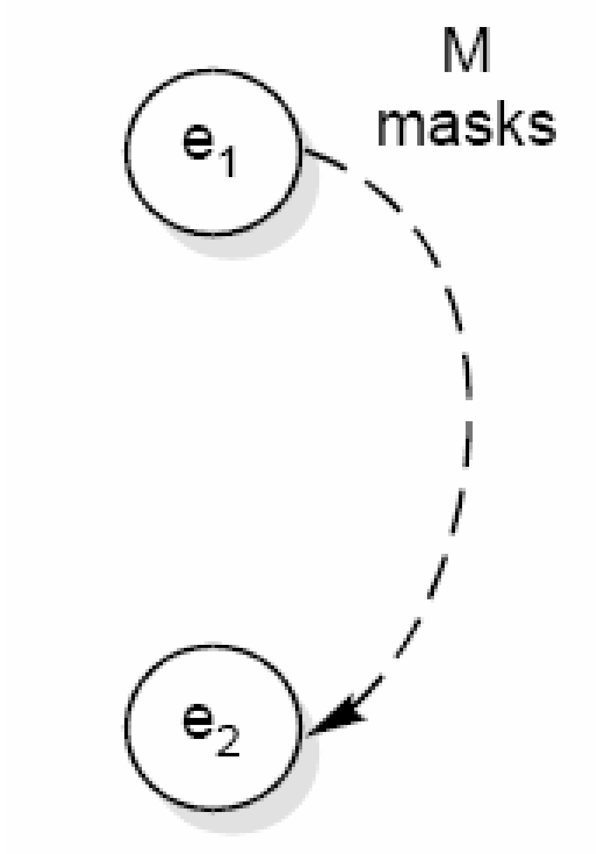


Fig. 11: Symbol for masks constraint

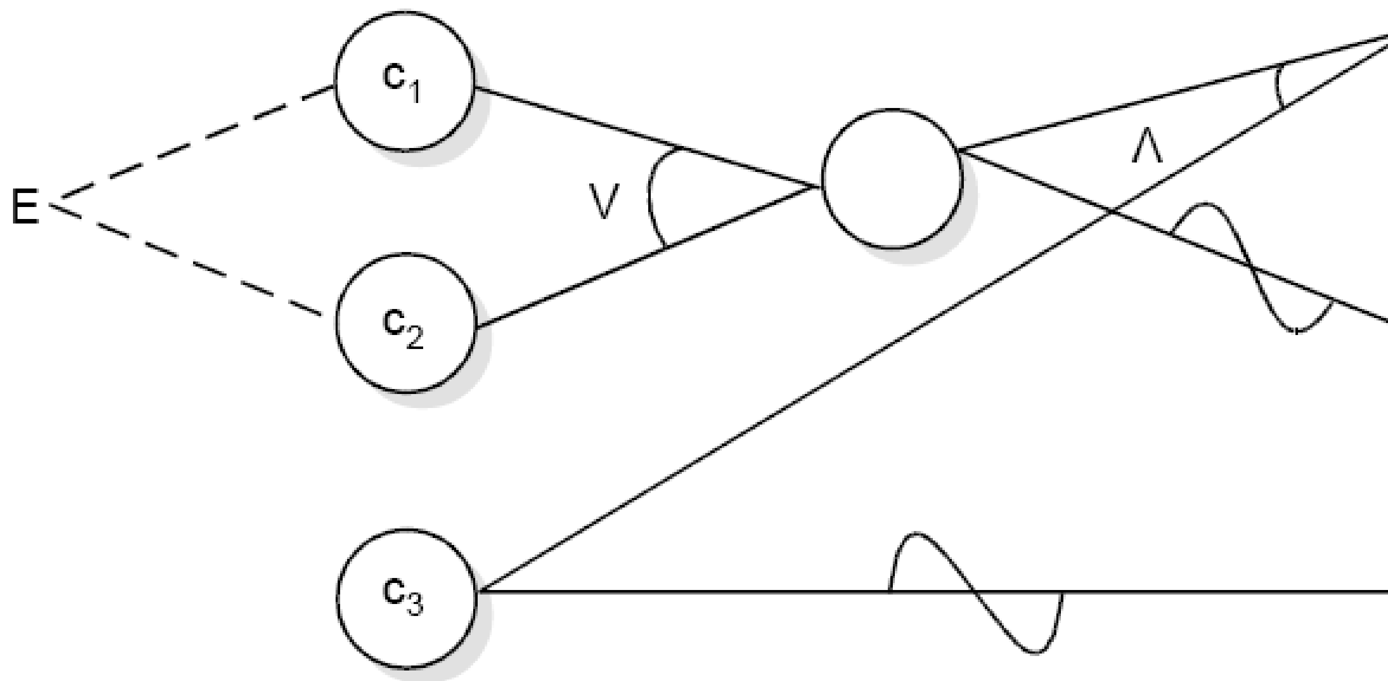


Fig. 12 : Sample cause effect graph with exclusive constra

Example 8.12

Consider the triangle problem specified in the example 8.3. Dra
effect graph and identify the test cases.

Solution

The causes are

c_1: side x is less than sum of sides y and z

c_2: side y is less than sum of sides x and y

c_3: side z is less than sum of sides x and y

c_4: side x is equal to side y

c_5: side x is equal to side z

c_6: side y is equal to side z

and effects are

e_1: Not a triangle

e_2: Scalene triangle

e_3: Isosceles triangle

e_4: Equilateral triangle

e_5: Impossible stage

The cause effect graph is shown in fig. 13 and decision table is show
The test cases for this problem are available in Table 5.

Conditions									
C_1: x < y + z ?	0	1	1	1	1	1	1	1	1
C_2: y < x + z ?	X	0	1	1	1	1	1	1	1
C_3: z < x + y ?	X	X	0	1	1	1	1	1	1
C_4: x = y ?	X	X	X	1	1	1	1	0	0
C_5: x = z ?	X	X	X	1	1	0	0	1	1
C_6: y = z ?	X	X	X	1	0	1	0	1	0
e_1: Not a triangle	1	1	1						
e_2: Scalene									
e_3: Isosceles							1		1
e_4: Equilateral				1					
e_5: Impossible					1	1		1	

Table 6: Decision table

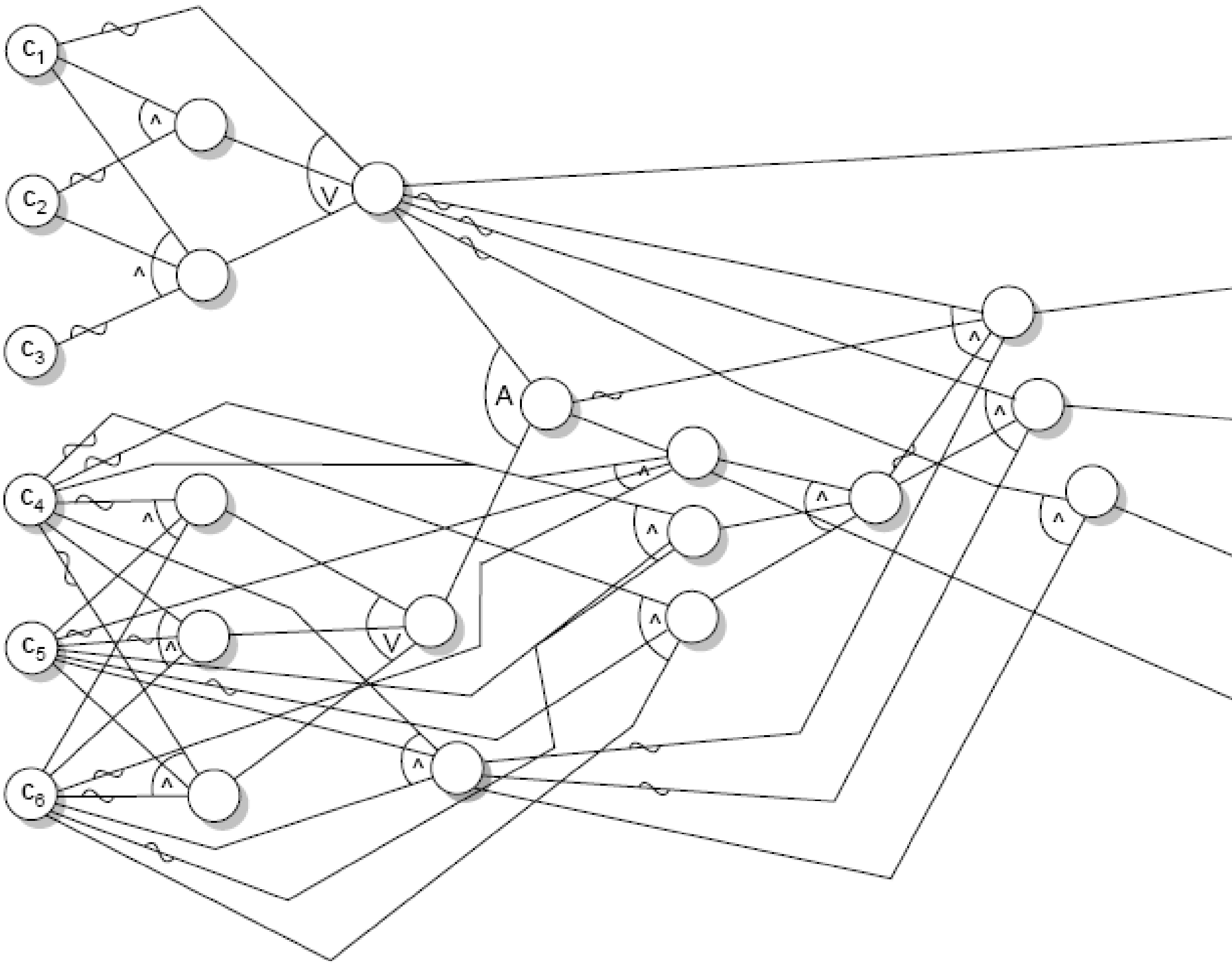


Fig. 13: Cause effect graph of triangle problem

Structural Testing

A complementary approach to functional testing is called structura
testing. It permits us to examine the internal structure of the program.

Path Testing

Path testing is the name given to a group of test techniques based
selecting a set of test paths through the program. If the set of pat
chosen, then it means that we have achieved some measure of test th

This type of testing involves:

1. generating a set of paths that will cover every branch in the progra
2. finding a set of test cases that will execute every path in the s
   paths.

Flow Graph

The control flow of a program can be analysed using a graphical r
known as flow graph. The flow graph is a directed graph in which no
entire statements or fragments of a statement, and edges represents f

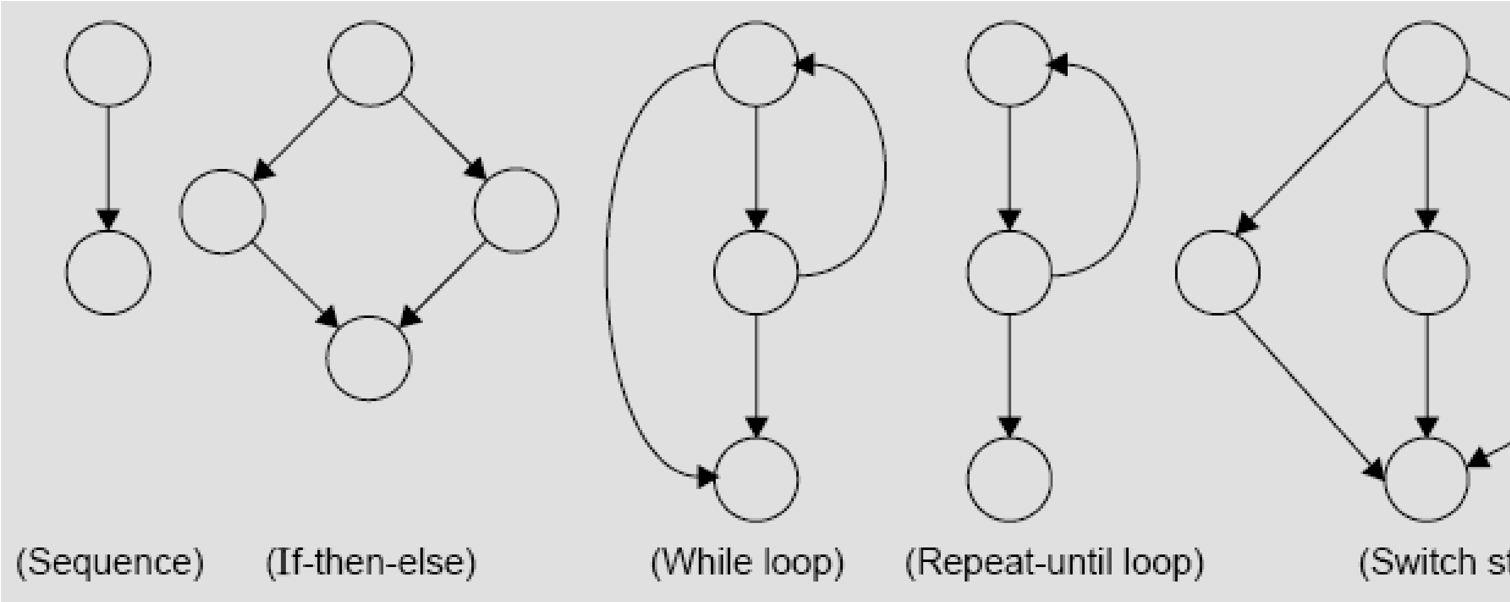


Fig. 14: The basic construct of the flow graph

```
/* Program to generate the previous date given a date, a
given as dd mm yyyy separated by space and performs error ch
validity of the current date entered. */

#include <stdio.h>
#include <conio.h>

int main()
{
  int day, month, year, validDate = 0;
/*Date Entry*/
  printf(“Enter the day value: ”);
  scanf(“%d”, &day);
  printf(“Enter the month value: ”);
  scanf(“%d”, &month);
  printf(“Enter the year value: ”);
  scanf(“%d”,&year);
/*Check Date Validity */
  if (year >= 1900 && year <= 2025) {
    if (month == 1 || month == 3 || month == 5 || month
      month == 8 || month == 10 || month == 12) {
```

```
    if (day >= 1 && day <= 31) {
        validDate = 1;
    }
    else {
      validDate = 0;
    }
  }
  else if (month == 2) {
    int rVal=0;
    if (year%4 == 0) {
      rVal=1;
      if ((year%100)==0 && (year % 400) !=0) {
        rVal=0;
      }
    }
    if (rVal ==1 && (day >=1 && day <=29) ) {
      validDate = 1;
    }
    else if (day >=1 && day <= 28 ) {
      validDate = 1;
    }
```

```
        else {
          validDate = 0;
        }
      }
      else if ((month >= 1 && month <= 12) && (day >= 1 && day
        validDate = 1;
      }
      else {
        validDate = 0;
      }
    }
    /*Prev Date Calculation*/
    if (validDate) {
      if (day == 1) {
        if (month == 1) {
          year--;
          day=31;
          month=12;
        }
        else if (month == 3) {
          int rVal=0;
```

```
            if (year%4 == 0) {
              rVal=1;
              if ((year%100)==0 && (year % 400) !=0) {
            rVal=0;
          }
            }
            if (rVal ==1) {
              day=29;
              month--;
            }
            else {
              day=28;
              month--;
            }
          }
          else if (month == 2 || month == 4 || month == 6 ||
          month == 11) {
            day = 31;
            month--;
```

```
            }
            else {
              day=30;
              month--;
            }
          }
          else {
            day--;
          }
          printf("The next date is: %d-%d-%d",day,month,year
        }
        else {
          printf("The entered date ( %d-%d-%d ) is invalid",da
        }
        getche ();
        return 1;
      }
```

Fig. 15: Program for previous date problem

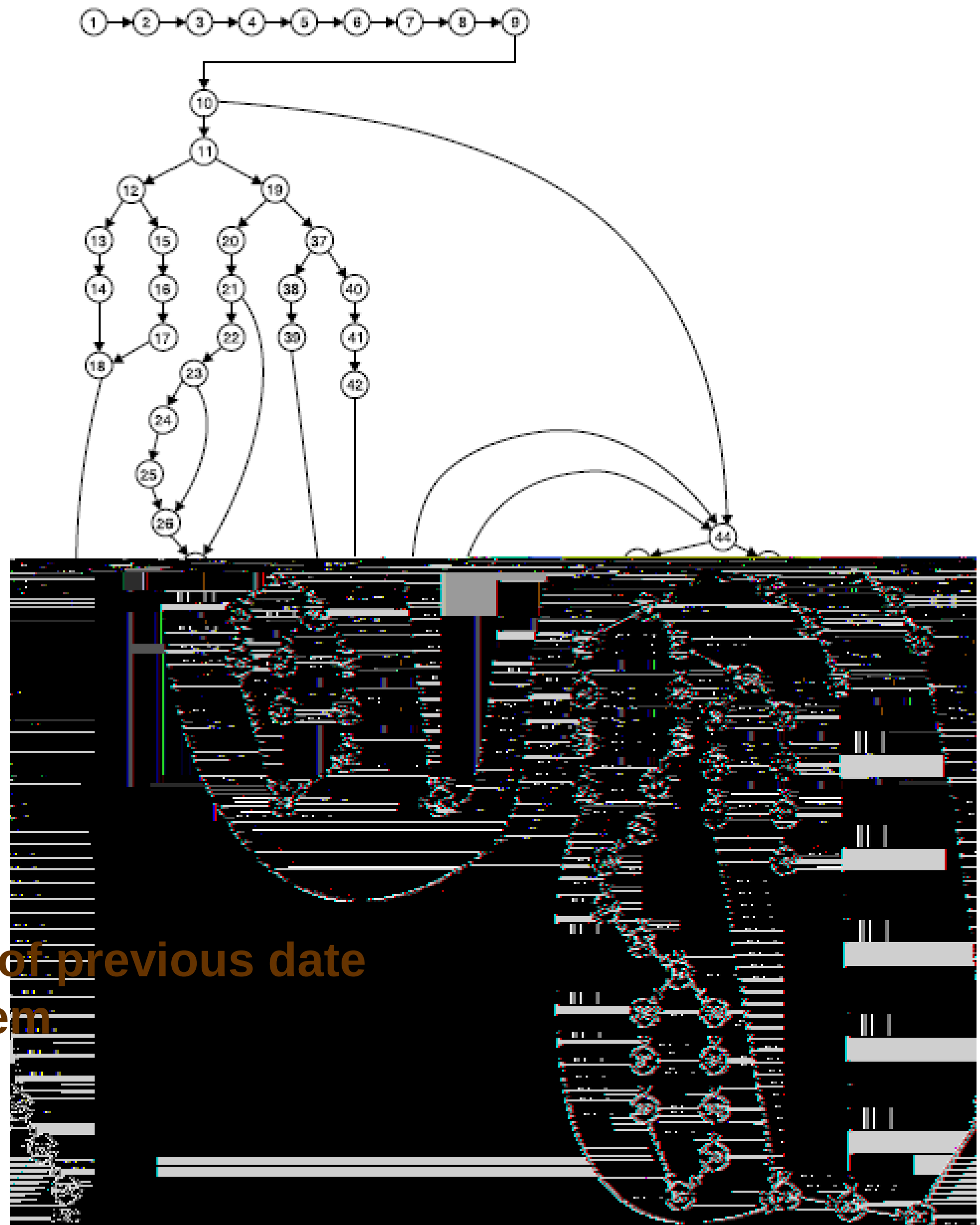


Fig. 16: Flow graph of previous date problem

DD Path Graph

Table 7: Mapping of flow graph nodes and DD path no

Flow graph nodes	DD Path graph corresponding node	Remarks
1 to 9	n_1	There is a sequential flow from node 1 to 9
10	n_2	Decision node, if true go to 13 else go to 44
11	n_3	Decision node, if true go to 12 else go to 19
12	n_4	Decision node, if true go to 13 else go to 15
13,14	n_5	Sequential nodes and are combined to form new node n_5
15,16,17	n_6	Sequential nodes
18	n_7	Edges from node 14 to 17 are terminated here
19	n_8	Decision node, if true go to 20 else go to 37
20	n_9	Intermediate node with one input edge and one output ed
21	n_{10}	Decision node, if true go to 22 else go to 27
22	n_{11}	Intermediate node
23	n_{12}	Decision node, if true go to 24 else go to 26

Flow graph nodes	DD Path graph corresponding node	Remarks
24,25	n_{13}	Sequential nodes
26	n_{14}	Two edges from node 25 & 23 are terminated here
27	n_{15}	Two edges from node 26 & 21 are terminated here. Also
28,29	n_{16}	Sequential nodes
30	n_{17}	Decision node, if true go to 31 else go to 33
31,32	n_{18}	Sequential nodes
33,34,35	n_{19}	Sequential nodes
36	n_{20}	Three edge from node 29,32 and 35 are terminated here
37	n_{21}	Decision node, if true go to 38 else go to 40
38,39	n_{22}	Sequential nodes
40,41,42	n_{23}	Sequential nodes
43	n_{24}	Three edge from node 36,39 and 42 are terminated here

Flow graph nodes	DD Path graph corresponding node	Remarks
44	n_{25}	Decision node, if true go to 45 else go to 82. Three edges are also terminated here.
45	n_{26}	Decision node, if true go to 46 else go to 77
46	n_{27}	Decision node, if true go to 47 else go to 51
47,48,49,50	n_{28}	Sequential nodes
51	n_{29}	Decision node, if true go to 52 else go to 68
52	n_{30}	Intermediate node with one input edge & one output ege
53	n_{31}	Decision node, if true go to 54 else go to 59
54	n_{32}	Intermediate node
55	n_{33}	Decision node, if true go to 56 else go to 58
56,57	n_{34}	Sequential nodes
58	n_{35}	Two edge from node 57 and 55 are terminated here
59	n_{36}	Decision node, if true go to 60 else go to 63. Two edge fr 53 are terminated.

Flow graph nodes	DD Path graph corresponding node	Remarks
60,61,62	n_{37}	Sequential nodes
63,64,65,66	n_{38}	Sequential nodes
67	n_{39}	Two edge from node 62 and 66 are terminated here
68	n_{40}	Decision node, if true go to 69 else go to 72
69,70,71	n_{41}	Sequential nodes
72,73,74,75	n_{42}	Sequential nodes
76	n_{43}	Four edges from nodes 50, 67, 71 and 75 are terminated
77,78,79	n_{44}	Sequential nodes
80	n_{45}	Two edges from nodes 76 & 79 are terminated here
81	n_{46}	Intermediate node
82,83,84	n_{47}	Sequential nodes
85	n_{48}	Two edges from nodes 81 and 84 are terminated here
86,87	n_{49}	Sequential nodes with exit node

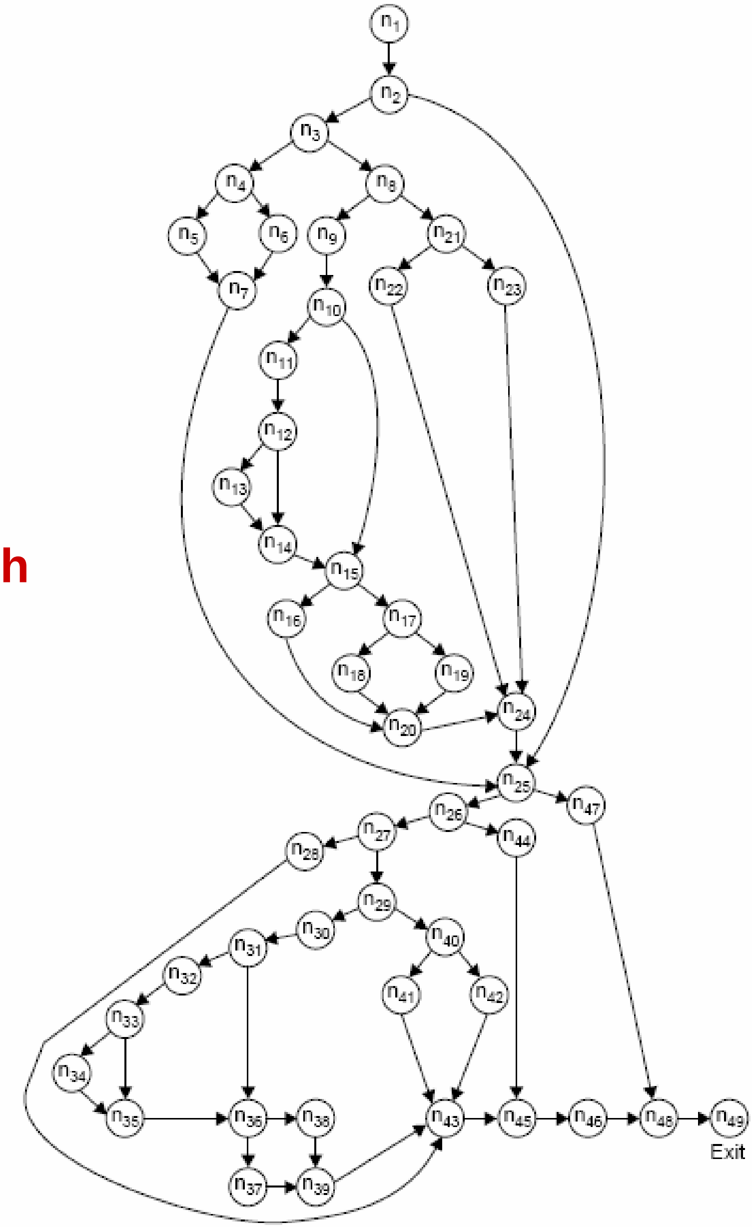


Fig. 17: DD path graph of previous date problem

	Independent paths of previous date problem
1	$n_1, n_2, n_{25}, n_{47}, n_{48}, n_{49}$
2	$n_1, n_2, n_3, n_4, n_5, n_7, n_{25}, n_{47}, n_{48}, n_{49}$
3	$n_1, n_2, n_3, n_4, n_6, n_7, n_{25}, n_{47}, n_{48}, n_{49}$
4	$n_1, n_2, n_3, n_8, n_{21}, n_{22}, n_{24}, n_{25}, n_{47}, n_{48}, n_{49}$
5	$n_1, n_2, n_3, n_8, n_{21}, n_{23}, n_{24}, n_{25}, n_{47}, n_{48}, n_{49}$
6	$n_1, n_2, n_3, n_8, n_9, n_{10}, n_{15}, n_{17}, n_{19}, n_{20}, n_{24}, n_{25}, n_{47}, n_{48}, n_{49}$
7	$n_1, n_2, n_3, n_8, n_9, n_{10}, n_{15}, n_{17}, n_{18}, n_{20}, n_{24}, n_{25}, n_{47}, n_{48}, n_{49}$
8	$n_1, n_2, n_3, n_8, n_9, n_{10}, n_{11}, n_{12}, n_{13}, n_{14}, n_{15}, n_{17}, n_{18}, n_{20}, n_{24}, n_{25}, n_{47}, n_{48}, n_{49}$
9	$n_1, n_2, n_3, n_8, n_9, n_{10}, n_{11}, n_{12}, n_{14}, n_{15}, n_{17}, n_{18}, n_{20}, n_{24}, n_{25}, n_{47}, n_{48}, n_{49}$
10	$n_1, n_2, n_3, n_8, n_9, n_{10}, n_{15}, n_{16}, n_{20}, n_{24}, n_{25}, n_{47}, n_{48}, n_{49}$
11	$n_1, n_2, n_3, n_8, n_9, n_{10}, n_{15}, n_{16}, n_{20}, n_{24}, n_{25}, n_{26}, n_{44}, n_{45}, n_{46}, n_{48}, n_{49}$
12	$n_1, n_2, n_3, n_8, n_9, n_{11}, n_{12}, n_{14}, n_{15}, n_{16}, n_{20}, n_{24}, n_{25}, n_{26}, n_{27}, n_{28}, n_{43}, n_{45}, n_{46}, n_{48},$
13	$n_1, n_2, n_3, n_8, n_9, n_{10}, n_{11}, n_{12}, n_{14}, n_{15}, n_{16}, n_{20}, n_{24}, n_{25}, n_{26}, n_{27}, n_{29}, n_{40}, n_{41}, n_{43},$ n_{48}, n_{49}
14	$n_1, n_2, n_3, n_8, n_9, n_{10}, n_{11}, n_{12}, n_{14}, n_{15}, n_{16}, n_{20}, n_{24}, n_{25}, n_{26}, n_{27}, n_{29}, n_{40}, n_{42}, n_{43},$ n_{48}, n_{49}
15	$n_1, n_2, n_3, n_8, n_{21}, n_{22}, n_{24}, n_{25}, n_{26}, n_{27}, n_{29}, n_{30}, n_{31}, n_{36}, n_{38}, n_{39}, n_{43}, n_{45}, n_{46}, n_{48},$
16	$n_1, n_2, n_3, n_8, n_{21}, n_{22}, n_{24}, n_{25}, n_{26}, n_{27}, n_{29}, n_{30}, n_{31}, n_{36}, n_{37}, n_{39}, n_{43}, n_{45}, n_{46}, n_{48},$
17	$n_1, n_2, n_3, n_8, n_{21}, n_{22}, n_{24}, n_{25}, n_{26}, n_{27}, n_{29}, n_{30}, n_{31}, n_{32}, n_{33}, n_{34}, n_{35}, n_{36}, n_{37}, n_{39},$ n_{46}, n_{48}, n_{49}
18	$n_1, n_2, n_3, n_8, n_{21}, n_{22}, n_{24}, n_{25}, n_{26}, n_{27}, n_{29}, n_{30}, n_{31}, n_{32}, n_{33}, n_{35}, n_{36}, n_{37}, n_{39}, n_{43},$ n_{48}, n_{49}

Fig. 18: Independent paths of previous date problem

Example 8.13

Consider the problem for the determination of the nature of roots
equation. Its input a triple of positive integers (say a,b,c) and value
interval [0,100].

The program is given in fig. 19. The output may have one of the follow

[Not a quadratic equation; real roots; Imaginary roots; Equal roots]

Draw the flow graph and DD path graph. Also find independent path
Path graph.

```
#include <conio.h>
#include <math.h>
    int main()
  {
    int  a,b,c,validInput=0,d;
    double D;
    printf(“Enter the ‘a’ value: ”);
    scanf(“%d”,&a);
    printf(“Enter the ‘b’ value: ”);
    scanf(“%d”,&b);
    printf(“Enter the ‘c’ value: ”);
    scanf(“%d”,&c);
    if ((a >= 0) && (a <= 100) && (b >= 0) && (b <= 100) && (c
      && (c <= 100)) {
      validInput = 1;
      if (a == 0) {
        validInput = -1;
      }
    }
    if (validInput==1) {
      d = b*b - 4*a*c;
      if (d == 0) {
        printf(“The roots are equal and are r1 = r2 = %f\n”,
               -b/(2*(float) a));
```

```
    }
    else if ( d > 0 ) {
      D=sqrt(d);
      printf("The roots are real and are r1 = %f and r2
              (-b-D)/(2* a), (-b+D)/(2* a));
    }
    else {
      D=sqrt(-d)/(2*a);
      printf("The roots are imaginary and are r1 = (%f,%
              r2 = (%f,%f)\n", -b/(2.0*a),D,-b/(2.0*a),-
    }
  }
  else if (validInput == -1) {
    printf("The vlaues do not constitute a Quadratic e
  }
  else {
    printf("The inputs belong to invalid range.");
  }
  getche();
  return 1;
}
```

Fig. 19: Code of quadratic equation problem

Solution

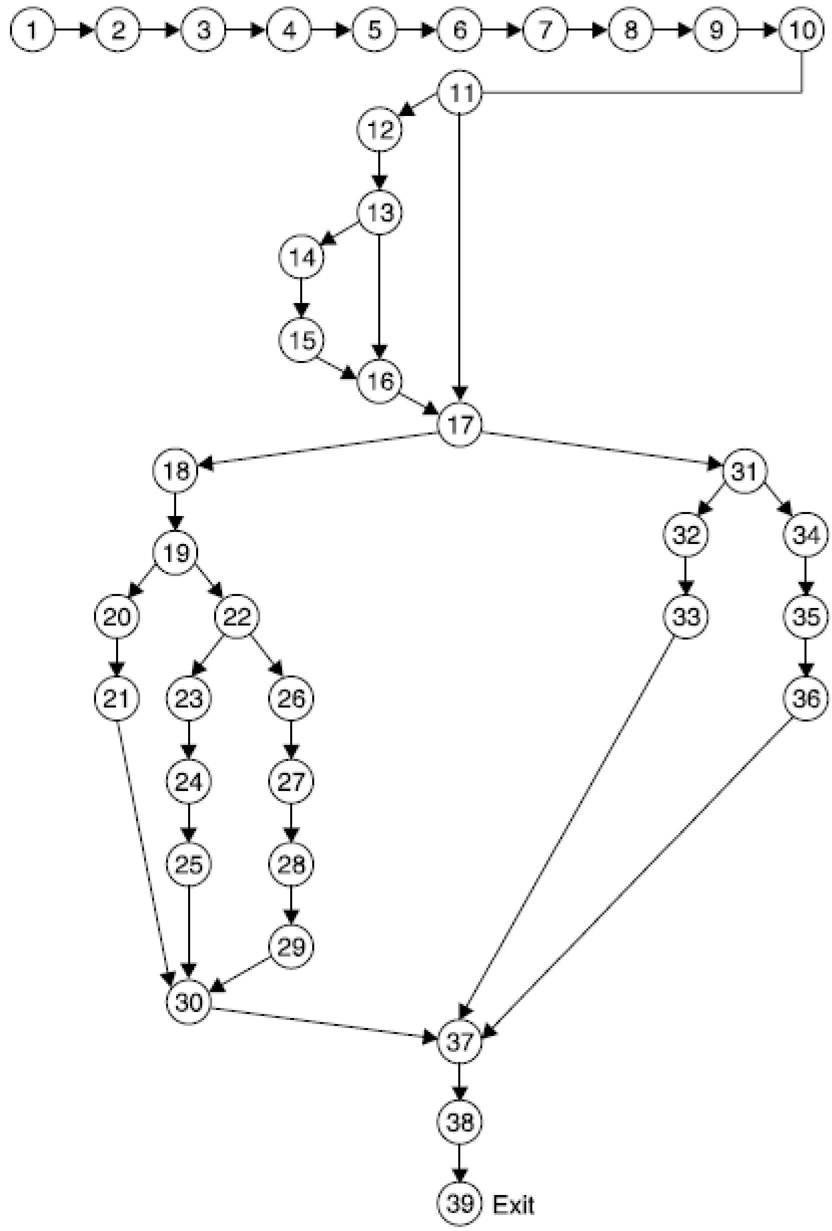


Fig. 19 (a) :
g

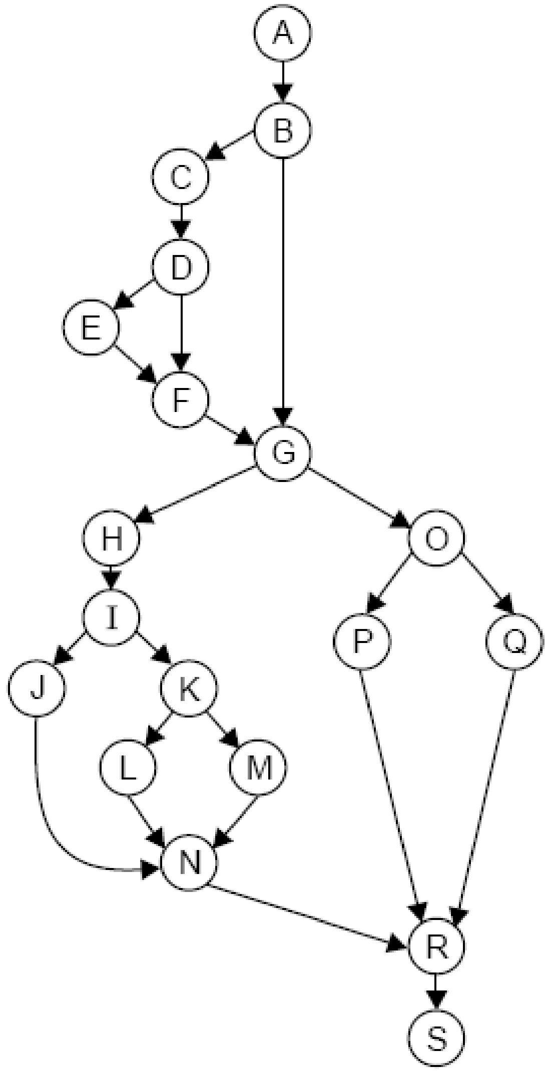


Fig. 19 (b) : DD Path graph

The mapping table for DD path graph is:

Flow graph nodes	DD Path graph corresponding node	Remarks
1 to 10	A	Sequential nodes
11	B	Decision node
12	C	Intermediate node
13	D	Decision node
14,15	E	Sequential node
16	F	Two edges are combined here
17	G	Two edges are combined and decision node
18	H	Intermediate node
19	I	Decision node
20,21	J	Sequential node
22	K	Decision node
23,24,25	L	Sequential node

Flow graph nodes	DD Path graph corresponding node	Remarks
26,27,28,29	M	Sequential nodes
30	N	Three edges are combined
31	O	Decision node
32,33	P	Sequential node
34,35,36	Q	Sequential node
37	R	Three edges are combined here
38,39	S	Sequential nodes with exit node

Independent paths are:

(i) ABGOQRS
(ii) ABGOPRS
(iii) ABCDFGOQRS
(iv) ABCDEFGOPRS
(v) ABGHIJNRS
(vi) ABGHIKLNRS
(vi) ABGHIKMNRS

Example 8.14

Consider a program given in Fig.8.20 for the classification of a triangl triple of positive integers (say a,b,c) from the interval [1,100]. The [Scalene, Isosceles, Equilateral, Not a triangle].

Draw the flow graph & DD Path graph. Also find the independent path Path graph.

```
#include <stdio.h>
#include <conio.h>
int main()
{
  int a,b,c,validInput=0;
  printf("Enter the side 'a' value: ");
  scanf("%d",&a);
  printf("Enter the side 'b' value: ")
  scanf("%d",&b);
  printf("Enter the side 'c' value:");
  scanf("%d",&c);
  if ((a > 0) && (a <= 100) && (b > 0) && (b <= 100) && (c
      && (c <= 100)) {
    if ( (a + b) > c) && ((c + a) > b) && ((b + c) > a))
      validInput = 1;
    }
  }
  else {
    validInput = -1;
  }
  If (validInput==1) {
    If ((a==b) && (b==c)) {
      printf("The trinagle is equilateral");
    }
    else if ( (a == b) || (b == c) || (c == a) ) {
```

```
      printf("The triangle is isosceles");
    }
    else {
      printf("The trinagle is scalene");
    }
  }
  else if (validInput == 0) {
    printf("The values do not constitute a Triangle");
  }
  else {
    printf("The inputs belong to invalid range");
  }
  getche();
  return 1;
}
```

Fig. 20 : Code of triangle classification problem

Solution :

Flow graph of triangle problem is:

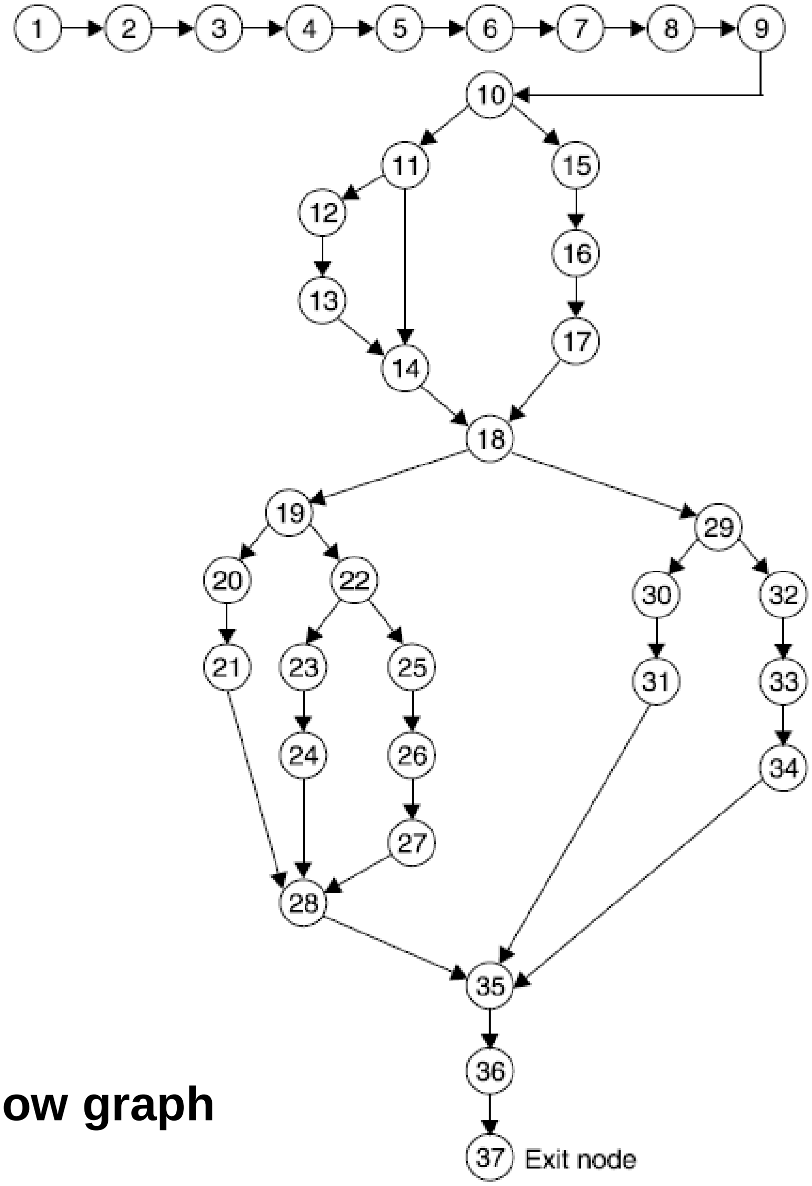


Fig.8. 20 (a): Program flow graph

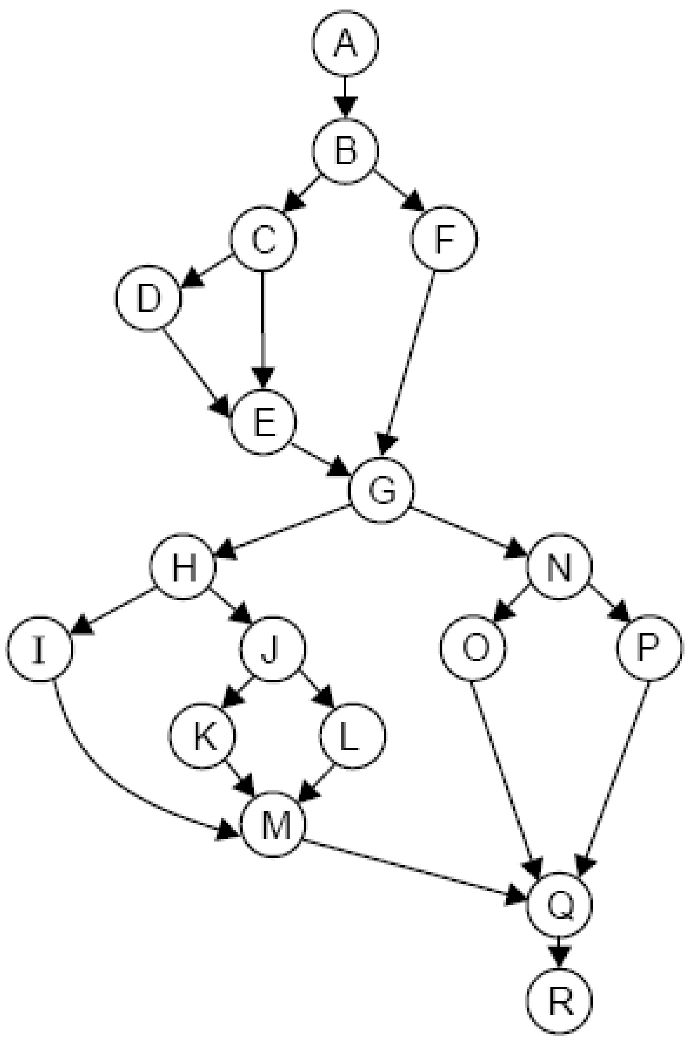
The mapping table for DD path graph is:

Flow graph nodes	DD Path graph corresponding node	Remarks
1 TO 9	A	Sequential nodes
10	B	Decision node
11	C	Decision node
12, 13	D	Sequential nodes
14	E	Two edges are joined here
15, 16, 17	F	Sequential nodes
18	G	Decision nodes plus joining of two edges
19	H	Decision node
20, 21	I	Sequential nodes
22	J	Decision node
23, 24	K	Sequential nodes
25, 26, 27	L	Sequential nodes

Flow graph nodes	DD Path graph corresponding node	Remarks
28	M	Three edges are combined here
29	N	Decision node
30, 31	O	Sequential nodes
32, 33, 34	P	Sequential nodes
35	Q	Three edges are combined here
36, 37	R	Sequential nodes with exit node

Fig. 20 (b): DD Path graph

DD Path graph is given in Fig. 20 (b)



Independent pa

(i) ABFGNI

(ii) ABFGNO

(iii) ABCEGI

(iv) ABCDEO

(v) ABFGHI

(vi) ABFGHJ

(vii)ABFGHJ

Fig. 20 (b): DD Path graph

Cyclomatic Complexity

McCabe's cyclomatic metric V(G) = $e - n + 2P$.

For example, a flow graph shown in in Fig. 21 with entry node '*a*' an

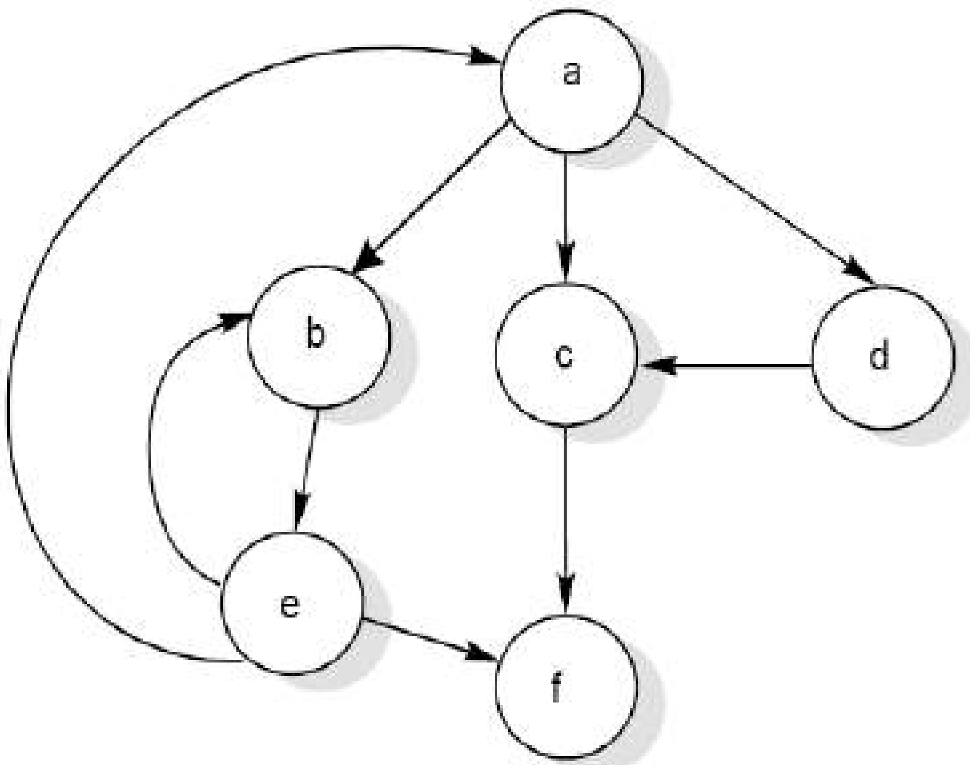


Fig. 21: Flow graph

The value of cyclomatic complexity can be calculated as :

$$V(G) = 9 - 6 + 2 = 5$$

Here $e = 9$, $n = 6$ and $P = 1$

There will be five independent paths for the flow graph illustrated in

Path 1 : *a c f*

Path 2 : *a b e f*

Path 3 : *a d c f*

Path 4 : *a b e a c f* **or** *a b e a b e f*

Path 5 : *a b e b e f*

Software Testing

Several properties of cyclomatic complexity are stated below:

1. $V(G) \geq 1$
2. V (G) is the maximum number of independent paths in gr
3. Inserting & deleting functional statements to G does not a
4. G has only one path if and only if V(G)=1.
5. Inserting a new row in G increases V(G) by unity.
6. V(G) depends only on the decision structure of G.

The role of P in the complexity calculation V(G)=e-n+2P is required to
correctly. We define a flow graph with unique entry and exit noc
reachable from the entry, and exit reachable from all nodes. This d
result in all flow graphs having only one connected component. One c
imagine a main program M and two called subroutines A and B havin
shown in Fig. 22.

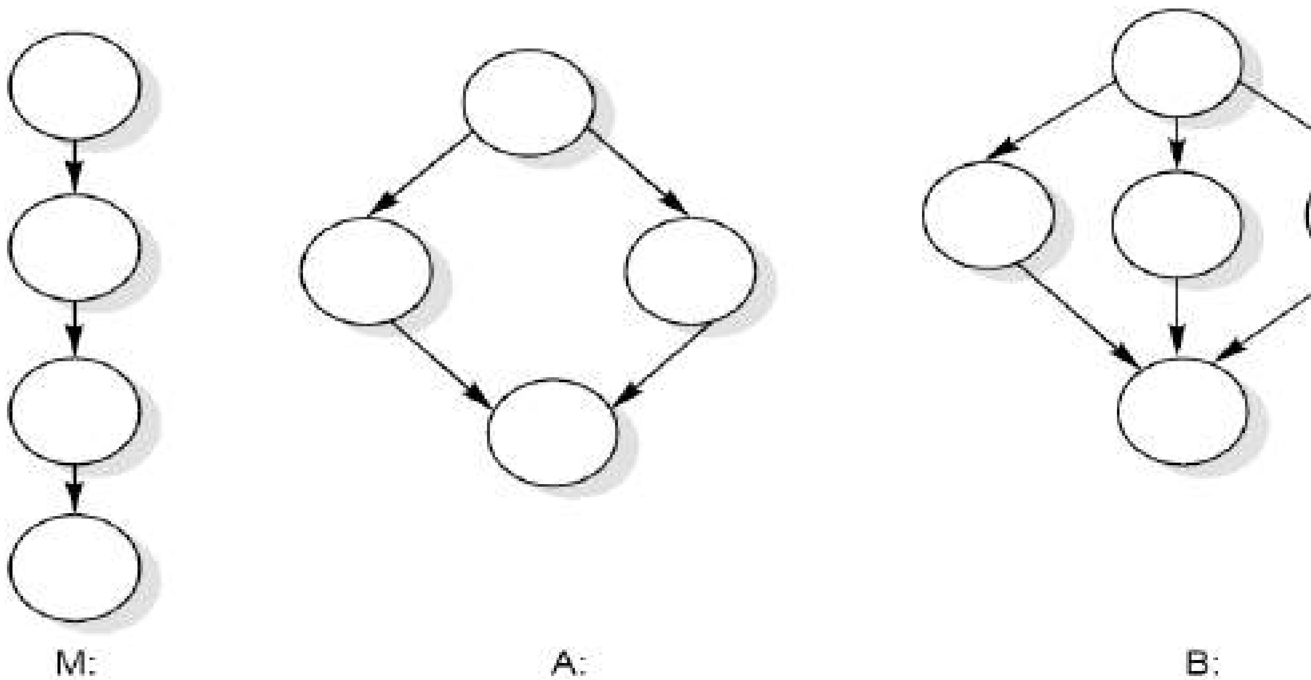


Fig. 22

Let us denote the total graph above with 3 connected compone

$$V(M \cup A \cup B) = e - n + 2P$$

= 13-13+2*3

= 6

This method with P $\neq$ 1 can be used to calculate the co
collection of programs, particularly a hierarchical nest of subrou

Notice that $V(M \cup A \cup B) = V(M) + V(A) + V(B) = 6$. In
complexity of a collection C of flow graphs with K connected c
equal to the summation of their complexities. To see this le
denote the k distinct connected component, and let e_i and n_i be the nu
and nodes in the ith-connected component. Then

$$V(C) = e - n + 2p = \sum_{i=1}^{k} e_i - \sum_{i=1}^{k} n_i + 2K$$

$$= \sum_{i=1}^{k} (e_i - n_i + 2) = \sum_{i=1}^{k} V(C_i)$$

Two alternate methods are available for the complexity calcula

1. Cyclomatic complexity V(G) of a flow graph G is equal to th predicate (decision) nodes plus one.

   $V(G)=\prod+1$

   Where $\prod$ is the number of predicate nodes contained in th G.

2. Cyclomatic complexity is equal to the number of region graph.

Example 8.15

Consider a flow graph given in Fig. 23 and calculate the complexity by all three methods.

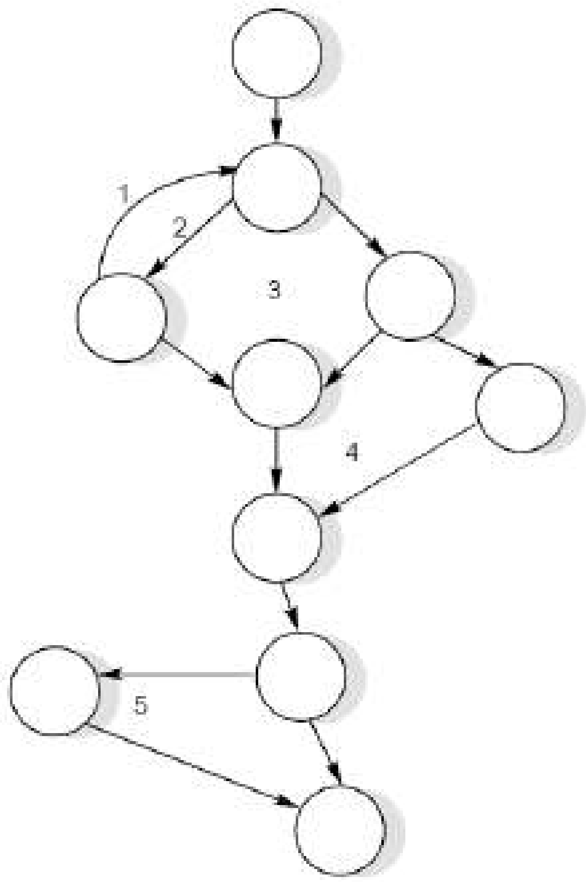


Fig. 23

Solution

Cyclomatic complexity can be calculated by any of the three me

1. V(G) $= e - n + 2P$

   $= 13 - 10 + 2 = 5$

2. V(G) $= \pi + 1$

   $= 4 + 1 = 5$

3. V(G) = number of regions

   = 5

Therefore, complexity value of a flow graph in Fig. 23 is 5.

Example 8.16

Consider the previous date program with DD path graph give
Find cyclomatic complexity.

Solution

Number of edges (e) = 65

Number of nodes (n) =49

(i) V(G) = $e - n + 2P = 65 - 49 + 2 = 18$

(ii) V(G) = $\pi + 1 = 17 + 1 = 18$

(iii) V(G) = Number of regions = 18

The cyclomatic complexity is 18.

Example 8.17

Consider the quadratic equation problem given in example 8.1
Path graph. Find the cyclomatic complexity:

Solution

Number of nodes (n) = 19

Number of edges (e) = 24

(i) V(G) = $e - n + 2P = 24 - 19 + 2 = 7$

(ii) V(G) = $\pi + 1 = 6 + 1 = 7$

(iii) V(G) = Number of regions = 7

Hence cyclomatic complexity is 7 meaning ther
independent paths in the DD Path graph.

Example 8.18

Consider the classification of triangle problem given in examp
the cyclomatic complexity.

Solution

Number of edges (e) = 23

Number of nodes (n) =18

(i) $V(G) = e - n + 2P = 23 - 18 + 2 = 7$

(ii) $V(G) = \pi + 1 = 6 + 1 = 7$

(iii) V(G) = Number of regions = 7

The cyclomatic complexity is 7. Hence, there are seven indepe
as given in example 8.14.

Graph Matrices

A graph matrix is a square matrix with one row and one column for ev
graph. The size of the matrix (i.e., the number of rows and columns)
number of nodes in the flow graph. Some examples of graphs a
matrices are shown in fig. 24.

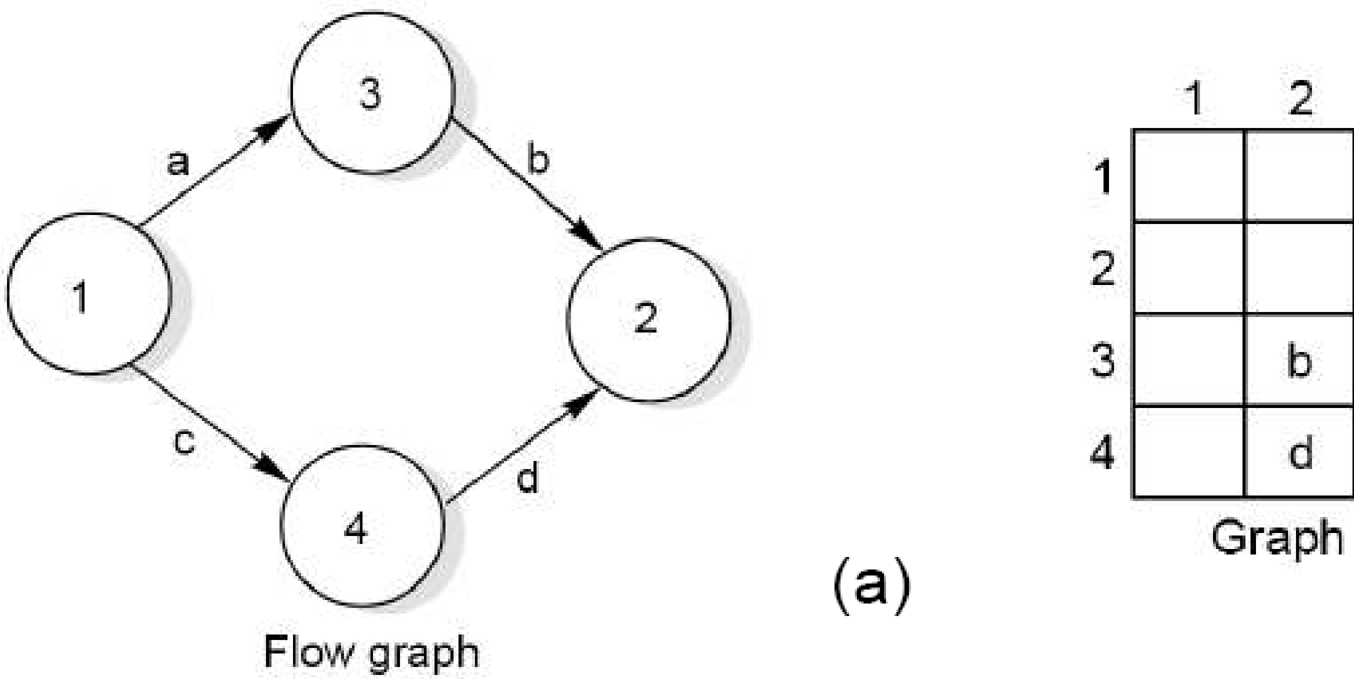
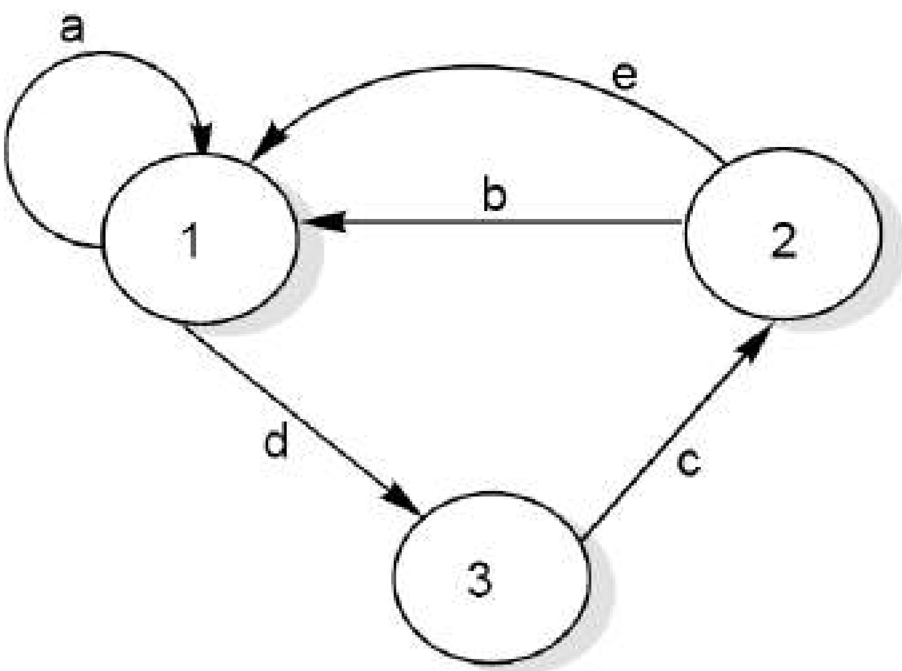


Fig. 24 (a): Flow graph and graph matrices



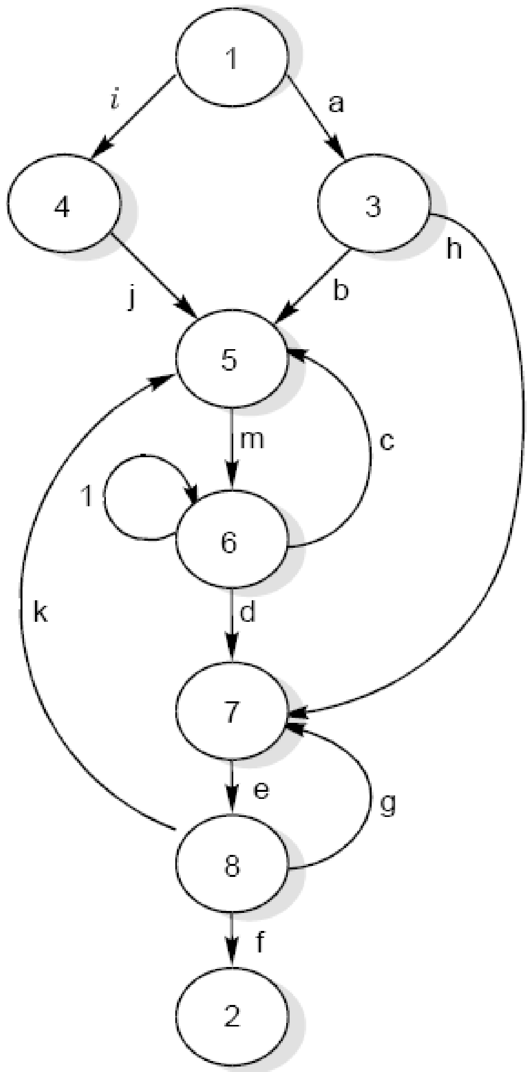
Flow graph

	1	
1	a	
2	b + e	
3		

Graph

(b)

Fig. 24 (b): Flow graph and graph matrices



Flow graph

	1	2	3	4	5
1			a	i	
2					
3					b
4					j
5					
6					c
7					
8		f			k

Graph Matrix

(c)

Fig. 24 (c): Flow graph and graph matrices

	1	2	3	4	5	6	7	8	Con
1			1	1					2
2									
3					1		1		2
4					1				1
5						1			1
6					1	1	1		3
7								1	1
8		1			1		1		3
									6

Fig. 25 : Connection matrix of flow graph shown in Fig. 24 (

	1	2	3	4
1			a	c
2				
3		b		
4		d		

[A]

	1	2
1		ab + cd
2		
3		
4		

$[A]^2$

The square matrix represent that there are two path *ab* and *cd* fro
node 2.

Example 8.19

Consider the flow graph shown in the Fig. 26 and draw the graph matrices. Find out cyclomatic complexity and two / three link paths f any other node.

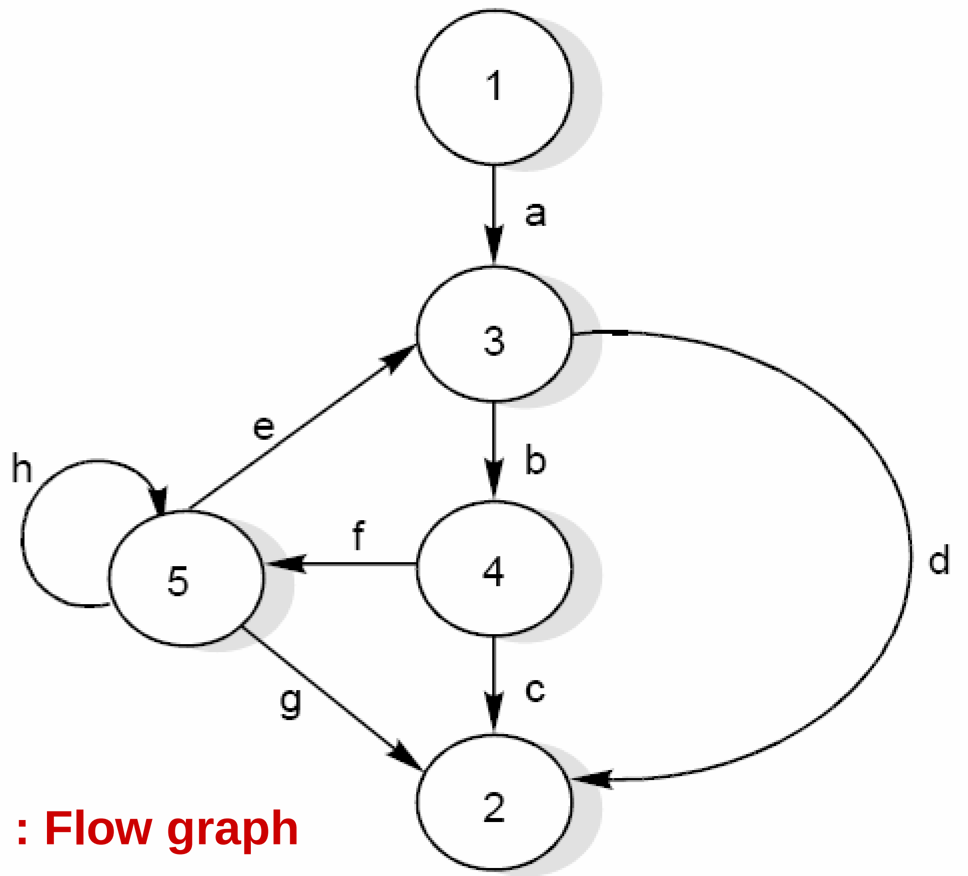


Fig. 26 : Flow graph

Solution

The graph & connection matrices are given below :

	1	2	3	4	5
1			a		
2					
3		d		b	
4		c			f
5		g	e		h

Graph Matrix (A)

	1	2	3	4	5
1			1		
2					
3		1		1	
4		1			1
5		1	1		1

Connection Matrix

To find two link paths, we have to generate a square of graph matrix [A
link paths, a cube of matrix [A] is required.

	1	2	3	4	5
1		ad		ab	
2					
3		bc			bf
4		fg	fe		fh
5		ed + hg	he	eb	h^2

$[A^2]$

	1	2	3
1		abc	
2			
3		bfg	bfe
4		fed + fhg	fhe
5		ebc + hed + h^2g	h^2e

$[A^3]$

Data Flow Testing

Data flow testing is another from of structural testing. It has nothing t flow diagrams.

i. Statements where variables receive values.

ii. Statements where these values are used or reference

As we know, variables are defined and referenced throughout the may have few define/ reference anomalies:

i. A variable is defined but not used/ referenced.

ii. A variable is used but never defined.

iii. A variable is defined twice before it is used.

Definitions

The definitions refer to a program P that has a program graph G(P program variables V. The G(P) has a single entry node and a single set of all paths in P is PATHS(P)

(i) **Defining Node:** Node n ϵ G(P) is a defining node of the va written as DEF (*v*, *n*), if the value of the variable *v* is defined at th fragment corresponding to node *n*.

(ii) **Usage Node:** Node n ϵ G(P) is a usage node of the variable *v* USE (*v*, *n*), if the value of the variable *v* is used at staten corresponding to node *n*. A usage node USE (*v*, n) is a predicat as p) if statement n is a predicate statement otherwise US computation use (denoted as *c*).

(iii) Definition use: A definition use path with respect to a variab du-path) is a path in PATHS(P) such that, for some *v* ∈ V, there a usage nodes DEF(v, m) and USE(v, n) such that m and n are i nodes of the path.

(iv) Definition clear : A definition clear path with respect to a varia dc-path) is a definition use path in PATHS(P) with initial and fi (*v*, m) and USE (*v*, n), such that no other node in the path is a de

The du-paths and dc-paths describe the flow of data across source s points at which the values are defined to points at which the values du-paths that are not definition clear are potential trouble spots.

Hence, our objective is to find all du-paths and then identity those du-
not dc-paths. The steps are given in Fig. 27. We may like to generate
cases for du-paths that are not dc-paths.

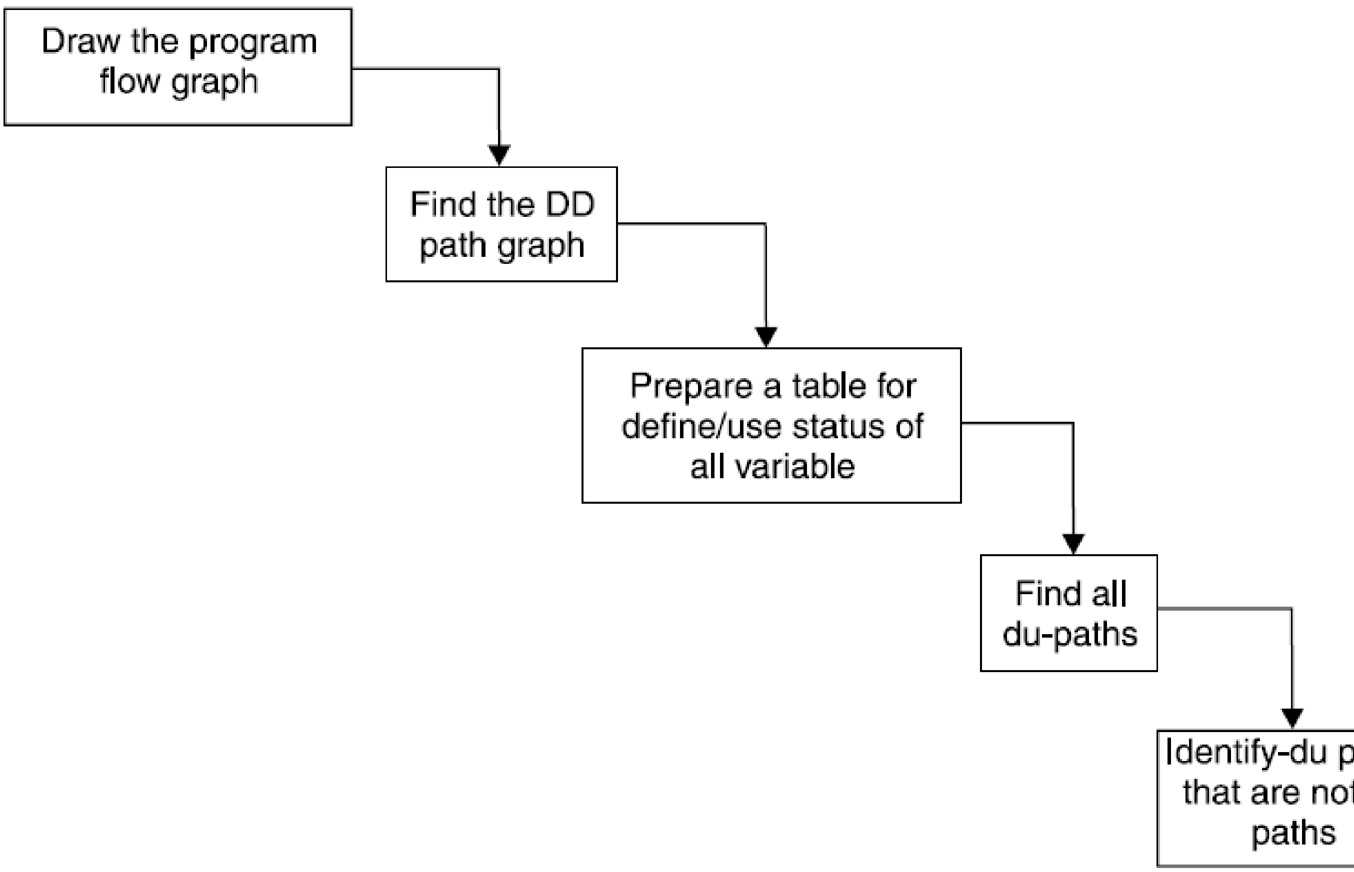


Fig. 27 : Steps for data flow testing

Example 8.20

Consider the program of the determination of the nature of roots equation. Its input is a triple of positive integers (say a,b,c) and valu these may be from interval [0,100]. The program is given in Fig. 19. T have one of the option given below:

(i) Not a quadratic program

(ii) real roots

(iii) imaginary roots

(iv) equal roots

(v) invalid inputs

Find all du-paths and identify those du-paths that are definition clear.

Solution

Step I: The program flow graph is given in Fig. 19 (a). The variabl program are a,b,c,d, validinput, D.

Step II: DD Path graph is given in Fig. 19(b). The cyclomatic complexi is 7 indicating there are seven independent paths.

Step III: Define/use nodes for all variables are given below:

Variable	Defined at node	Used at node
a	6	11,13,18,20,24,27,28
b	8	11,18,20,24,28
c	10	11,18
d	18	19,22,23,27
D	23, 27	24,28
Validinput	3, 12, 14	17,31

Step IV: The du-paths are identified and are named by their beginni
nodes using Fig. 19 (a).

Variable	Path (beginning, end) nodes	Definition clea
a	6, 11 6, 13 6, 18 6, 20 6, 24 6, 27 6, 28	Yes Yes Yes Yes Yes Yes Yes
b	8, 11 8, 18 8, 20 8, 24 8, 28	Yes Yes Yes Yes Yes

(C

Variable	Path (beginning, end) nodes	Definition cle
c	10, 11 10, 18	Yes Yes
d	18, 19 18, 22 18, 23 18, 27	Yes Yes Yes Yes
D	23, 24 23, 28 27, 24 27, 28	Yes Path not poss Path not poss Yes
validinput	3, 17 3, 31 12, 17 12, 31 14, 17 14, 31	no no no no yes yes

Example 8.21

Consider the program given in Fig. 20 for the classification of
input is a triple of positive integers (say a,b,c) from the interva
output may be:

[Scalene, Isosceles, Equilateral, Not a triangle, Invalid inputs].

Find all du-paths and identify those du-paths that are definition

Solution

Step I: The program flow graph is given in Fig. 20 (a). The vari
the program are a,b,c, valid input.

Step II: DD Path graph is given in Fig. 20(b). The cyclomatic
this graph is 7 and thus, there are 7 independent paths.

Step III: Define/use nodes for all variables are given below:

Variable	Defined at node	Used at node
a	6	10, 11, 19, 22
b	7	10, 11, 19, 22
c	9	10, 11, 19, 22
valid input	3, 13, 16	18, 29

Step IV: The du-paths are identified and are named by their beginni
nodes using Fig. 20 (a).

Variable	Path (beginning, end) nodes	Definition clea
a	5, 10 5, 11 5, 19 5, 22	Yes Yes Yes Yes
b	7, 10 7, 11 7, 19 7, 22	Yes Yes Yes Yes

(C

Variable	Path (beginning, end) nodes	Definition cle
c	9, 10 9, 11 9, 19 9, 22	Yes Yes Yes Yes
valid input	3, 18 3, 29 12, 18 12, 29 16, 18 16, 29	no no no no Yes Yes

Hence total du-paths are 18 out of which four paths are not definiti

Mutation Testing

Mutation testing is a fault based technique that is similar to fault seed mutations to program statements are made in order to determine p test cases. it is basically a fault simulation technique.

Multiple copies of a program are made, and each copy is altered; this called a mutant. Mutants are executed with test data to determine w data are capable of detecting the change between the original pr mutated program.

A mutant that is detected by a test case is termed "killed" and the g procedure is to find a set of test cases that are able to kill gro programs.

When we mutate code there needs to be a way of measuring the degr code has been modified. For example, if the original expression mutant for that expression is x+2, that is a lesser change to the origi mutant such as (c*22), where both the operand and the operator ar may have a ranking scheme, where a first order mutant is a single expression, a second order mutant is a mutation to a first order mut High order mutants becomes intractable and thus in practice only low are used.

One difficulty associated with whether mutants will be killed is t reaching the location; if a mutant is not executed, it cannot be kille cases are to be designed to reach a mutant. For example, suppos code.

Read (a,b,c);

If(a>b) and (b=c) then

x:=a*b*c; (make mutants; m_1, m_2, m_3 …….)

To execute this, input domain must contain a value such that a is gre
b equals c. If input domain does not contain such a value, then all m
this location should be considered equivalent to the original prograr
statement x:=a*b*c is dead code (code that cannot be reached durir
we make the mutant x+y for x+1, then we should take care about
which should not be equal to 1 for designing a test case.

The manner by which a test suite is evaluated (scored) via mutatio
follows: for a specified test suite and a specific set of mutants, the
types of mutants in the code i.e., killed or dead, live, equivalent. T
number of live, killed, and equivalent mutants will be the total num
created. The score associated with a test suite T and mutants M is sir

$$\frac{\#killed}{\#total - \#equivalent} \quad 100\%$$

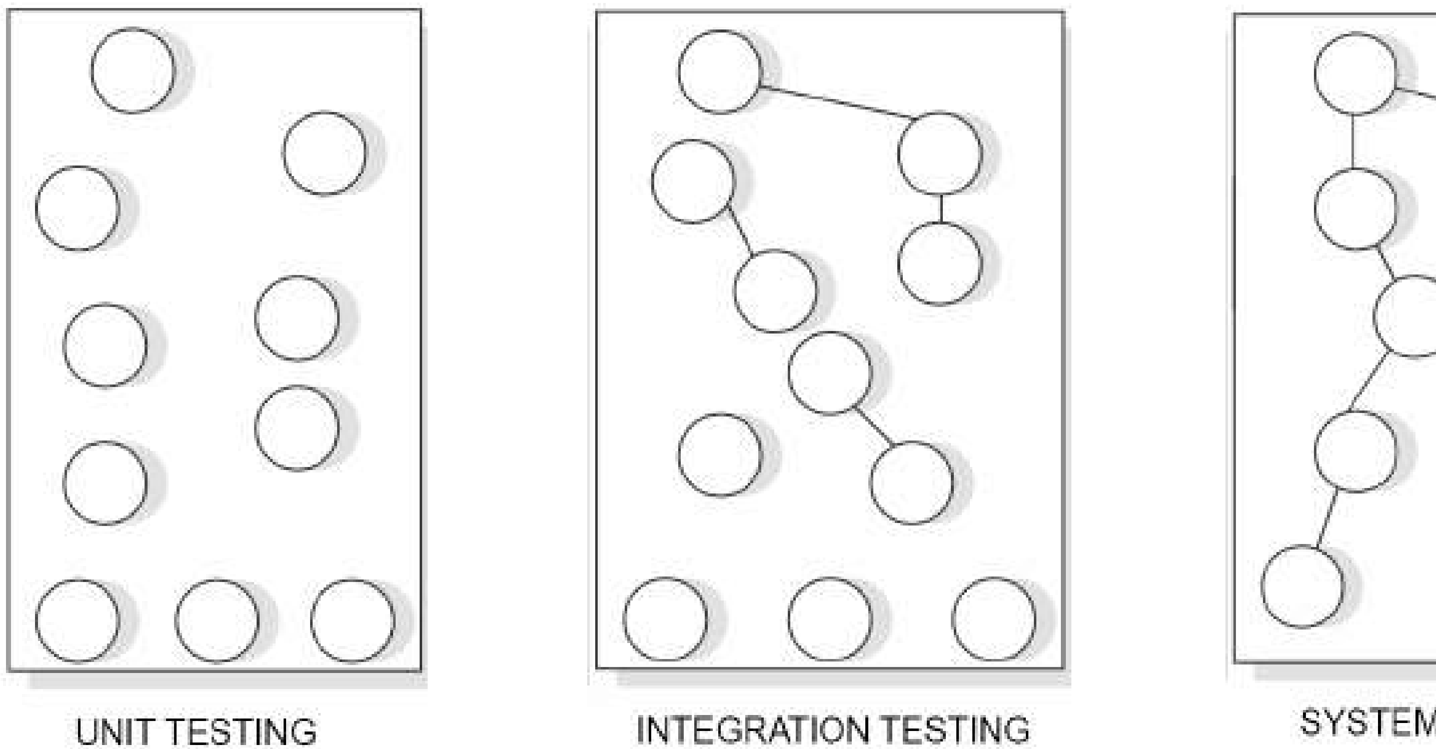
Levels of Testing

There are 3 levels of testing:

i. Unit Testing

ii. Integration Testing

iii. System Testing



Unit Testing

There are number of reasons in support of unit testing than testing the

1. The size of a single module is small enough that we can lo fairly easily.

2. The module is small enough that we can attempt to te demonstrably exhaustive fashion.

3. Confusing interactions of multiple errors in widely differen software are eliminated.

There are problems associated with testing a module in isolation. Ho
module without anything to call it, to be called by it or, poss
intermediate values obtained during execution? One approach is t
appropriate driver routine to call if and, simple stubs to be called by
output statements in it.

Stubs serve to replace modules that are subordinate to (called by) th
tested. A stub or dummy subprogram uses the subordinate module's
do minimal data manipulation, prints verification of entry, and returns.

This overhead code, called scaffolding represents effort that is impor
does not appear in the delivered product as shown in Fig. 29.

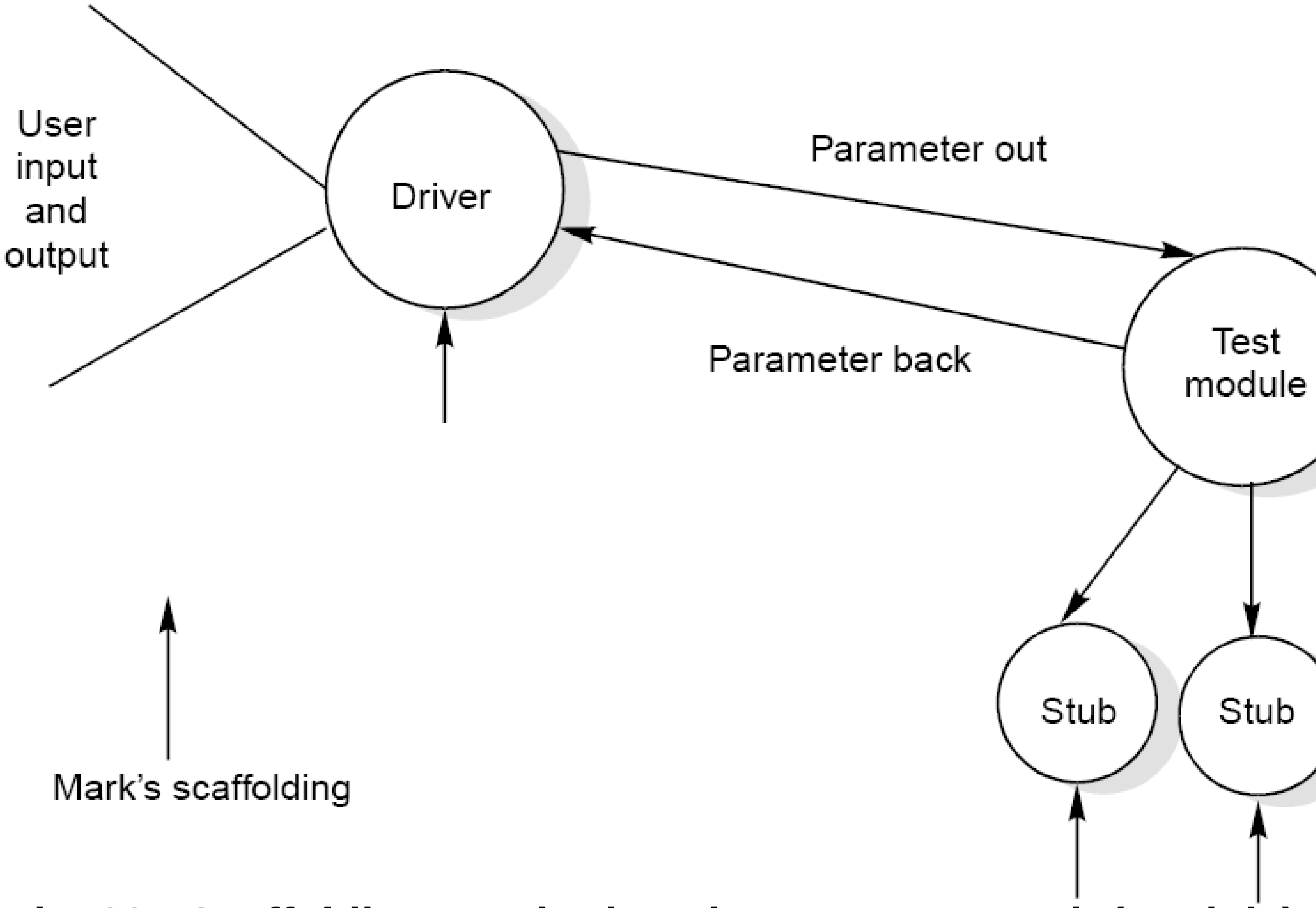


Fig. 29 : Scaffolding required testing a program unit (module)

Integration Testing

The purpose of unit testing is to determine that each independ correctly implemented. This gives little chance to determine tha between modules is also correct, and for this reason integration t performed. One specific target of integration testing is the inte parameters match on both sides as to type, permissible ranges utilization.

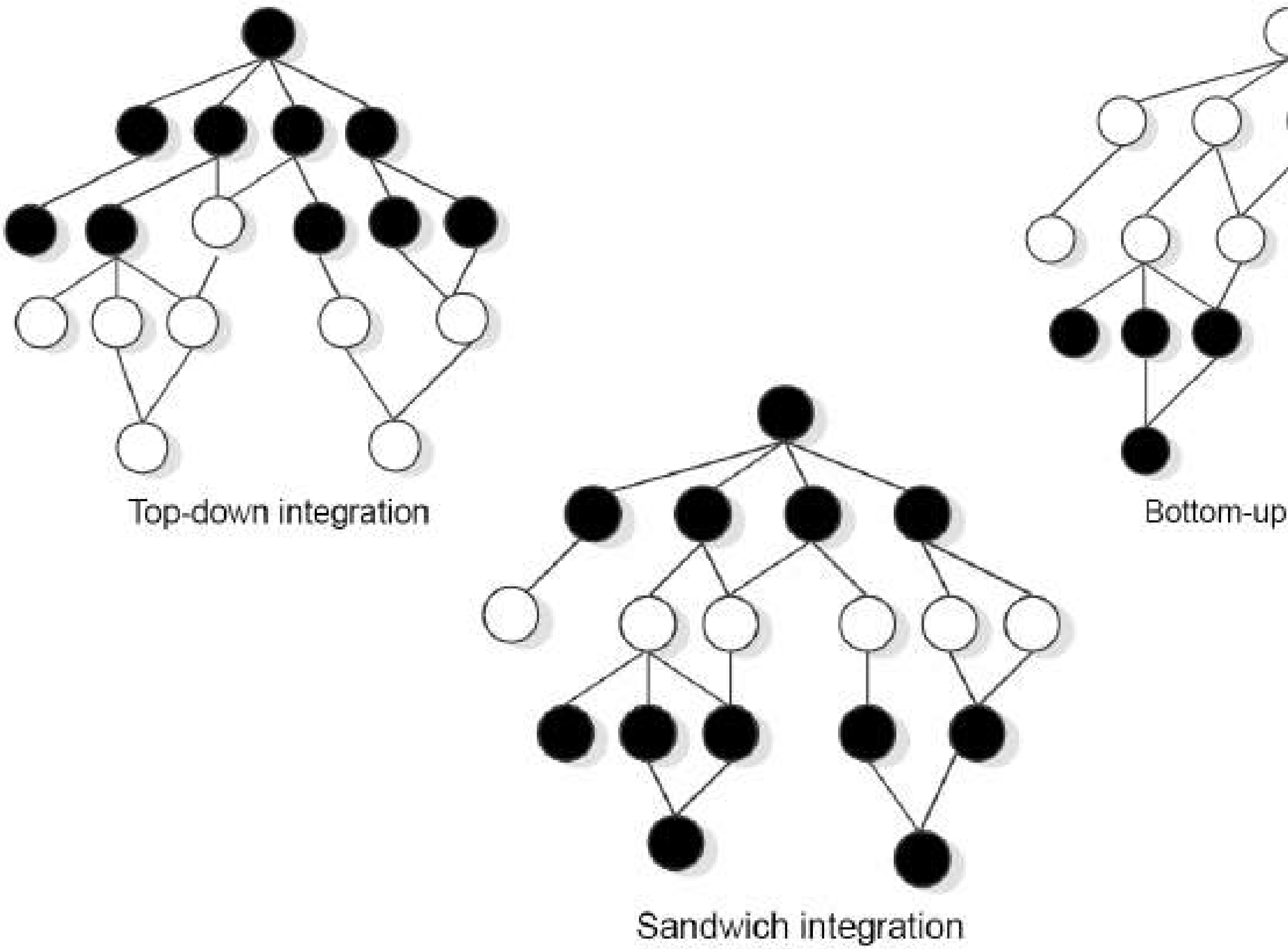


Fig. 30 : Three different integration approaches

System Testing

Of the three levels of testing, the system level is closet to everyda
We test many things; a used car before we buy it, an on-line
service before we subscribe, and so on. A common pattern in
forms is that we evaluate a product in terms of our expecta
respect to a specification or a standard. Consequently, goal is no
but to demonstrate performance. Because of this we tend to ap
testing from a functional standpoint rather than from a structural
so intuitively familiar, system testing in practice tends to be less
might be, and is compounded by the reduced testing interva
remains before a delivery deadline.

Petschenik gives some guidelines for choosing test cases during s

During system testing, we should evaluate a number of attr software that are vital to the user and are listed in Fig. 31. These operational correctness of the product and may be part of specifications.

Usable	Is the product convenient, clear, and predictable?
Secure	Is access to sensitive data restricted to those with autho
Compatible	Will the product work correctly in conjunction with existin software, and procedures?
Dependable	Do adequate safeguards against failure and methods fo exist in the product?
Documented	Are manuals complete, correct, and understandable?

Fig. 31 : Attributes of software to be tested during system tes

Validation Testing

- It refers to test the software as a complete product.
- This should be done after unit & integration testing.
- Alpha, beta & acceptance testing are nothing but the various wa customer during testing.

Validation Testing

- IEEE has developed a standard (IEEE standard 1059-1993) entitl for software verification and validation " to provide specific g planning and documenting the tasks required by the standa customer may write an effective plan.
- Validation testing improves the quality of software product in term capabilities and quality attributes.

The Art of Debugging

The goal of testing is to identify errors (bugs) in the program. T testing generates symptoms, and a program's failure is a clear sy presence of an error. After getting a symptom, we begin to investig and place of that error. After identification of place, we examine identify the cause of the problem. This process is called debugging

Debugging Techniques

Pressman explained few characteristics of bugs that provide some clue

1. "The symptom and the cause may be geographically remote symptom may appear in one part of a program, while the cause m located in other part. Highly coupled program structures may c situation.

2. The symptom may disappear (temporarily) when another error is co

3. The symptom may actually be caused by non errors (e.g. round of

4. The symptom may be caused by a human error that is not easily tr

5. The symptom may be a result of timing problems rather th problems.

6. It may be difficult to accurately reproduce input conditions (e application in which input ordering is indeterminate).

7. The symptom may be intermittent. This is particularly commo system that couple hardware with software inextricably.

8. The symptom may be due to causes that are distributed across a running on different processors".

Induction approach

- Locate the pertinent data
- Organize the data
- Devise a hypothesis
- Prove the hypothesis

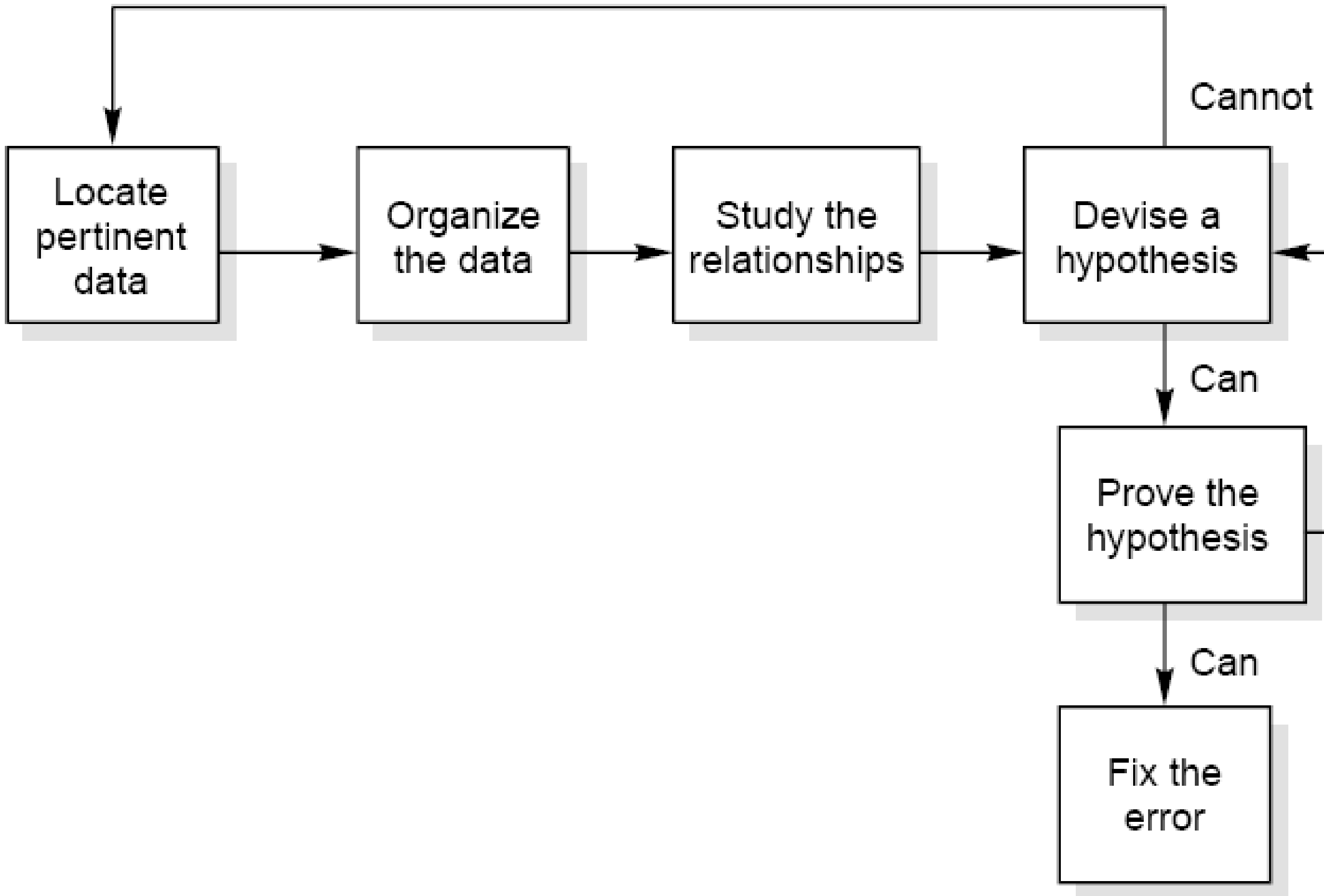


Fig. 32 : The inductive debugging process

Deduction approach

- Enumerate the possible causes or hypotheses
- Use the data to eliminate possible causes
- Refine the remaining hypothesis
- Prove the remaining hypothesis

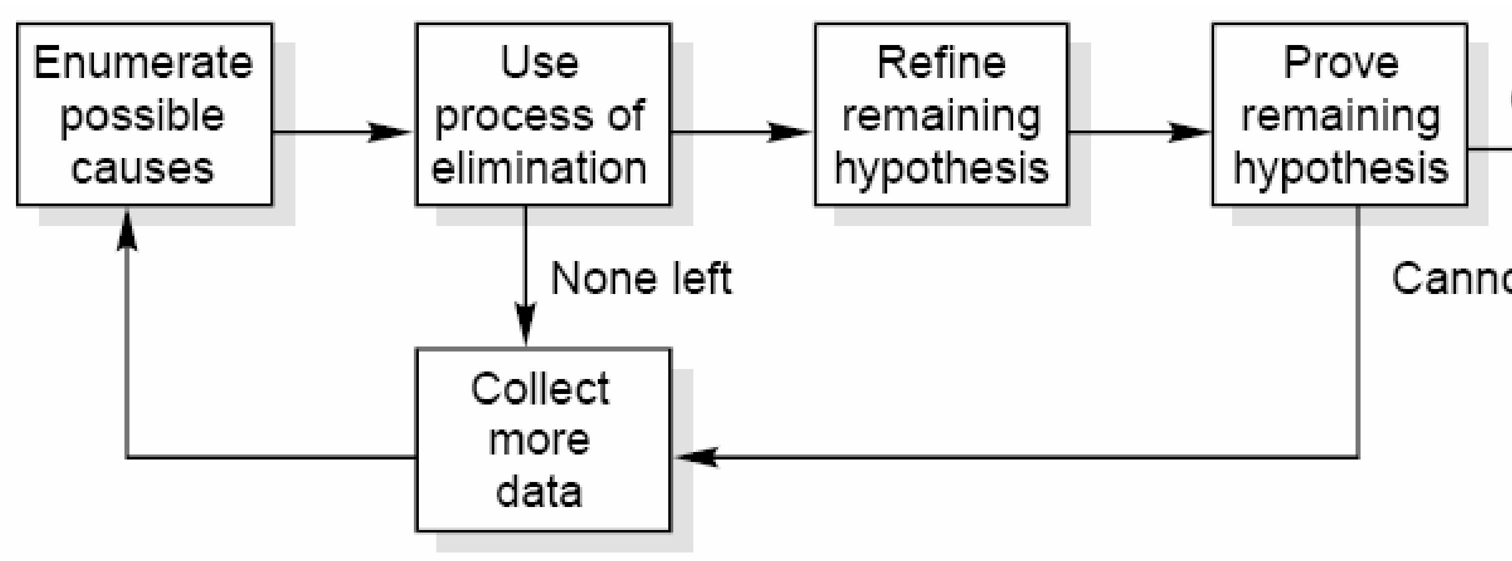


Fig. 33 : The inductive debugging process

Testing Tools

One way to improve the quality & quantity of testing is to make pleasant as possible for the tester. This means that tools should powerful & natural as possible.

The two broad categories of software testing tools are :

- Static
- Dynamic

Software Testing

There are different types of tools available and some are listed below:

1. Static analyzers, which examine programs systematically and auto

2. Code inspectors, who inspect programs automatically to make su to minimum quality standards.

3. standards enforcers, which impose simple rules on the developer.

4. Coverage analysers, which measure the extent of coverage.

5. Output comparators, used to determine whether the output in appropriate or not.

6. Test file/ data generators, used to set up test inputs.

7. Test harnesses, used to simplify test operations.

8. Test archiving systems, used to provide documentation about prog

Multiple Choice Questions

Note: Choose most appropriate answer of the following q

8.1 Software testing is:

(a) the process of demonstrating that errors are not present

(b) the process of establishing confidence that a program does wh
to do

(c) the process of executing a program to show it is working as pe

(d) the process of executing a program with the intent of finding e

8.2 Software mistakes during coding are known as:

(a) failures (b) defects

(c) bugs (d) errors

8.3 Functional testing is known as:

(a) Structural testing (b) Behavior testing

(c) Regression testing (d) None of the above

8.4 For a function of n variables, boundary value analysis yields:

(a) 4n+3 test cases (b) 4n+1 test cases

(c) n+4 test cases (d) None of the above

Multiple Choice Questions

8.5 For a function of two variables, how many cases will be generated robustness testing?

(a) 9 (b) 13

(c) 25 (d) 42

8.6 For a function of n variables robustness testing of boundary value

(a) 4n+1 (b) 4n+3

(c) 6n+1 (d) None of the above

8.7 Regression testing is primarily related to:

(a) Functional testing (b) Data flow testing

(c) Development testing (d) Maintenance testing

8.8 A node with indegree=0 and out degree ≠ 0 is called

(a) Source node (b) Destination node

(c) Transfer node (d) None of the above

8.9 A node with indegree ≠ 0 and out degree=0 is called

(a) Source node (b) Predicate node

(c) Destination node (d) None of the above

Multiple Choice Questions

8.10 A decision table has

(a) Four portions (b) Three portions

(c) Five portions (d) Two portions

8.11 Beta testing is carried out by

(a) Users (b) Developers

(c) Testers (d) All of the above

8.12 Equivalence class partitioning is related to

(a) Structural testing (b) Blackbox testing

(c) Mutation testing (d) All of the above

8.13 Cause effect graphing techniques is one form of

(a) Maintenance testing (b) Structural testing

(c) Function testing (d) Regression testing

8.14 During validation

(a) Process is checked (b) Product is checked

(c) Developer's performance is evaluated (d) The customer chec

Multiple Choice Questions

8.15 Verification is
- (a) Checking the product with respect to customer's expectation
- (b) Checking the product with respect to specifications
- (c) Checking the product with respect to the constraints of the pr
- (d) All of the above

8.16 Validation is
- (a) Checking the product with respect to customer's expectation
- (b) Checking the product with respect to specifications
- (c) Checking the product with respect to the constraints of the pro
- (d) All of the above

8.17 Alpha testing is done by
- (a) Customer
- (b) Tester
- (c) Developer
- (d) All of the above

8.18 Site for Alpha testing is
- (a) Software company
- (b) Installation place
- (c) Any where
- (d) None of the above

8.19 Site for Beta testing is
(a) Software company (b) User's site
(c) Any where (d) All of the above

8.20 Acceptance testing is done by
(a) Developers (b) Customers
(c) Testers (d) All of the above

8.21 One fault may lead to
(a) One failure (b) No failure
(c) Many failure (d) All of the above

8.22 Test suite is
(a) Set of test cases (b) Set of inputs
(c) Set of outputs (d) None of the above

8.23 Behavioral specification are required for:
(a) Modeling (b) Verification
(c) Validation (d) None of the above

Multiple Choice Questions

8.24 During the development phase, the following testing approach is no

(a) Unit testing (b) Bottom up testing

(c) Integration testing (d) Acceptance testing

8.25 Which is not a functional testing technique?

(a) Boundary value analysis (b) Decision table

(c) Regression testing (d) None of the above

8.26 Decision table are useful for describing situations in which:

(a) An action is taken under varying sets of conditions.

(b) Number of combinations of actions are taken under varying se

(c) No action is taken under varying sets of conditions

(d) None of the above

8.27 One weakness of boundary value analysis and equivalence partit

(a) They are not effective

(b) They do not explore combinations of input circumstances

(c) They explore combinations of input circumstances

(d) None of the above

Multiple Choice Questions

8.28 In cause effect graphing technique, cause & effect are related to

(a) Input and output (b) Output and input

(c) Destination and source (d) None of the above

8.29 DD path graph is called as

(a) Design to Design Path graph (b) Defect to Defect Pat

(c) Destination to Destination Path graph (d) Decision to decisio

8.30 An independent path is

(a) Any path through the DD path graph that introduce at least on processing statements or new conditions

(b) Any path through the DD path graph that introduce at most or processing statements or new conditions

(c) Any path through the DD path graph that introduce at one and set of processing statements or new conditions

(d) None of the above

8.31 Cyclomatic complexity is developed by

(a) B.W.Boehm (b) T.J.McCabe

(c) B.W.Lettlewood (d) Victor Basili

Multiple Choice Questions

8.32 Cyclomatic complexity is denoted by

(a) V(G)=e-n+2P (b) V(G)= $\prod$ +1

(c) V(G)=Number of regions of the graph (d) All of the above

8.33 The equation V(G)= $\prod$ +1 of cyclomatic complexity is applicabl
every predicate node has

(a) two outgoing edges (b) three or more outgoin

(c) no outgoing edges (d) none of the above

8.34 The size of the graph matrix is

(a) Number of edges in the flow graph

(b) Number of nodes in the flow graph

(c) Number of paths in the flow graph

(d) Number of independent paths in the flow graph

Multiple Choice Questions

8.35 Every node is represented by
 (a) One row and one column in graph matrix
 (b) Two rows and two columns in graph matrix
 (c) One row and two columns in graph matrix
 (d) None of the above

8.36 Cyclomatic complexity is equal to
 (a) Number of independent paths (b) Number of paths
 (c) Number of edges (d) None of the above

8.37 Data flow testing is related to
 (a) Data flow diagrams (b) E-R diagrams
 (c) Data dictionaries (d) none of the above

8.38 In data flow testing, objective is to find
 (a) All dc-paths that are not du-paths (b) All du-paths
 (c) All du-paths that are not dc-paths (d) All dc-paths

Multiple Choice Questions

8.39 Mutation testing is related to

(a) Fault seeding (b) Functional testing
(c) Fault checking (d) None of the above

8.40 The overhead code required to be written for unit testing is called

(a) Drivers (b) Stubs
(c) Scaffolding (d) None of the above

8.41 Which is not a debugging techniques

(a) Core dumps (b) Traces
(c) Print statements (d) Regression testing

8.42 A break in the working of a system is called

(a) Defect (b) Failure
(c) Fault (d) Error

8.43 Alpha and Beta testing techniques are related to

(a) System testing (b) Unit testing
(c) acceptance testing (d) Integration testing

8.44 Which one is not the verification activity

(a) Reviews (b) Path testing

(c) Walkthrough (d) Acceptance testing

8.45 Testing the software is basically

(a) Verification (b) Validation

(c) Verification and validation (d) None of the above

8.46 Integration testing techniques are

(a) Topdown (b) Bottom up

(c) Sandwich (d) All of the above

8.47 Functionality of a software is tested by

(a) White box testing (b) Black box testing

(c) Regression testing (d) None of the above

8.48 Top down approach is used for

(a) Development (b) Identification of fau

(c) Validation (d) Functional testing

8.49 Thread testing is used for testing

(a) Real time systems
(b) Object oriented syst
(c) Event driven systems
(d) All of the above

8.50 Testing of software with actual data and in the actual environmen

(a) Alpha testing
(b) Beta testing
(c) Regression testing
(d) None of the above

Exercises

8.1 What is software testing? Discuss the role of software software life cycle and why is it so difficult?

8.2 Why should we test? Who should do the testing?

8.3 What should we test? Comment on this statement. importance of testing

8.4 Defined the following terms:

(i) fault	(ii) failure
(iii) bug	(iv) mistake

8.5 What is the difference between

(i) Alpha testing & beta testing

(ii) Development & regression testing

(iii) Functional & structural testing

8.6 Discuss the limitation of testing. Why do we say that comp impossible?

Exercises

8.7 Briefly discuss the following

(i) Test case design, Test & Test suite

(ii) Verification & Validation

(iii) Alpha, beta & acceptance testing

8.8 Will exhaustive testing (even if possible for every sm guarantee that the program is 100% correct?

8.9 Why does software fail after it has passed from accep Explain.

8.10 What are various kinds of functional testing? Describe any

8.11 What is a software failure? Explain necessary and suffici for software failure. Mere presence of faults means softwa true? If not, explain through an example, a situation in w will definitely occur.

8.12 Explain the boundary value analysis testing techniques w an example.

Exercises

8.13 Consider the program for the determination of next date
Its input is a triple of day, month and year with the followi

$1 \leq \text{month} \leq 12$

$1 \leq \text{day} \leq 31$

$1900 \; 1 \leq \text{year} \leq 2025$

The possible outputs would be Next date or invalid boundary value, robust and worst test cases for this progra

8.14 Discuss the difference between worst test case and ac performance evaluation by means of testing. How can we b real worst case has actually been observed?

8.15 Describe the equivalence class testing method. Comp boundary value analysis techniques

8.16 Consider a program given below for the selection of numbers

```
main()
  {
    float A,B,C;
    printf("Enter three values\n");
    scanf("%f%f%f", &A,&B,&c);
    printf("\n Largest value is");
    if (A>B)
        {
            if(A>C)
                printf("%f\n",A);
            else
                printf("%f\n",c);
        }
        else
        {
            if(C>B)
                printf("%f\n",C);
            else
                printf("%f\n",B);
        }
    }
```

Exercises

(i) Design the set of test cases using boundary value analysis equivalence class testing technique.

(ii) Select a set of test cases that will provide 100% statement c

(iii) Develop a decision table for this program.

8.17 Consider a small program and show, why is it practically do exhaustive testing?

8.18 Explain the usefulness of decision table during testin effective? Justify your answer.

8.19 Draw the cause effect graph of the program given in exerc

8.20 Discuss cause effect graphing technique with an example.

8.21 Determine the boundary value test cases the extended tri that also considers right angle triangles.

Exercises

8.22 Why does software testing need extensive planning? Expla

8.23 What is meant by test case design? Discuss its objective the steps involved in test case design.

8.24 Let us consider an example of grading the students in institution. The grading is done according to the following r

Marks obtained	*Grade*
80–100	Distinction
60–79	First division
50–59	Second division
40–49	Third division
0–39	Fail

Generate test cases using equivalence class testing technique

Exercises

8.25 Consider a program to determine whether a number is 'o
and print the message

NUMBER IS EVEN

Or

NUMBER IS ODD

The number may be any valid integer.

Design boundary value and equivalence class test cases.

8.26 Admission to a professional course is subject to t
conditions:

(a)	Marks in Mathematics >=	60
(b)	Marks in Physics >=	50
(c)	Marks in Chemistry >=	40
(d)	Total in all three subjects >=	200
	Or	
	Total in Mathematics and Physics >=	150

Exercises

If aggregate marks of an eligible candidate are more than 225, eligible for honors course, otherwise he/she will be eligible fo The program reads the marks in the three subjects and following outputs:

(a) Not Eligible

(b) Eligible to Pass Course

(c) Eligible to Honors Course

Design test cases using decision table testing technique.

8.27 Draw the flow graph for program of largest of three num in exercise 8.16. Find out all independent paths that will all statements in the program have been tested.

8.28 Explain the significance of independent paths. Is it necess a tool for flow graph generation, if program size increase source lines?

8.29 Discuss the structure testing. How is it different form func

Exercises

8.30 What do you understand by structural testing? Illustr
structural testing techniques.

8.31 Discuss the importance of path testing during structural tes

8.32 What is cyclomatic complexity? Explain with the help of a

8.33 Is it reasonable to define "thresholds" for software m
example, is a module acceptable if its V(G) ≤ 10? Justify yo

8.34 Explain data flow testing. Consider an example and show
Also identify those "du" paths that are not "dc" paths.

8.35 Discuss the various steps of data flow testing.

8.36 If we perturb a value, changing the current value of 100 b
is the effect of this change? What precautions are re
designing the test cases?

Exercises

8.37 What is the difference between white and black bo
determining test cases easier in back or white box testing?
claim that if white box testing is done properly, it will ac
100% path coverage?

8.38 What are the objectives of testing? Why is the psycholog
person important?

8.39 Why does software fail after it has passed all testing phase
software, unlike hardware does not wear out with time.

8.40 What is the purpose of integration testing? How is it done?

8.41 Differentiate between integration testing and system testin

8.42 Is unit testing possible or even desirable in all circumsta
examples to Justify your answer?

8.43 Peteschenik suggested that a different team than the
integration testing should carry out system testing. What a
reasons for this?

Exercises

8.44 Test a program of your choice, and uncover several pr
Localise the main route of these errors, and explain how t
courses. Did you use the techniques of Table 8? Explain wh

8.45 How can design attributes facilitate debugging?

8.46 List some of the problem that could result from addi
statements to code. Discuss possible solutions to these prob

8.47 What are various debugging approaches? Discuss them w
examples.

8.48 Researchers and practitioners have proposed several
strategies intended to combine advantages of variou
discussed in this chapter. Propose your own combination
using some kind of random testing at selected points.

8.49 Design a test set for a spell checker. Then run it on a w
having a spell checker, and report on possible inadequacie
to your requirements.

Exercises

8.50 4 GLs represent a major step forward in the developmen program generation. Explain the major advantage & disadv use of 4 GLs. What are the cost impact of applications of te do you justify expenditures for these activities.