1 Overview	2 Life cycle components	3 Infrastructure components	4 Management components	5 Standards and Organizing
6 Static tesing	7 Dynamic testing	8 Test management	9 Tools	

Dynamic techniques (cont.)

References

- Dorothy Grahamet, Erik van Veenendaal, Isabel Evans, Rex Black. *Foundations of software testing: ISTQB Certification*

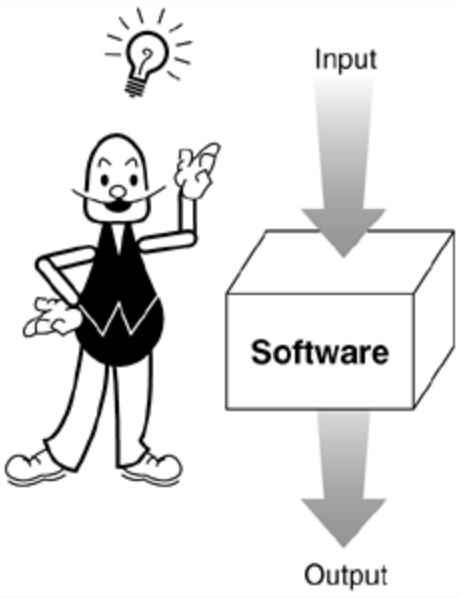
- Lee Copeland (2004). *A Practitioner's Guide to Software Test Design*. Artech House. ISBN:158053791x

Contents

White-box techniques

- Structure-based approach
  - based on the internal structure of a component or system
  - also called glass-box techniques

- What we may be interested in structures?
  - component level: program structures
  - integration level: the way components interact with others
  - system level: how user will interact with the system

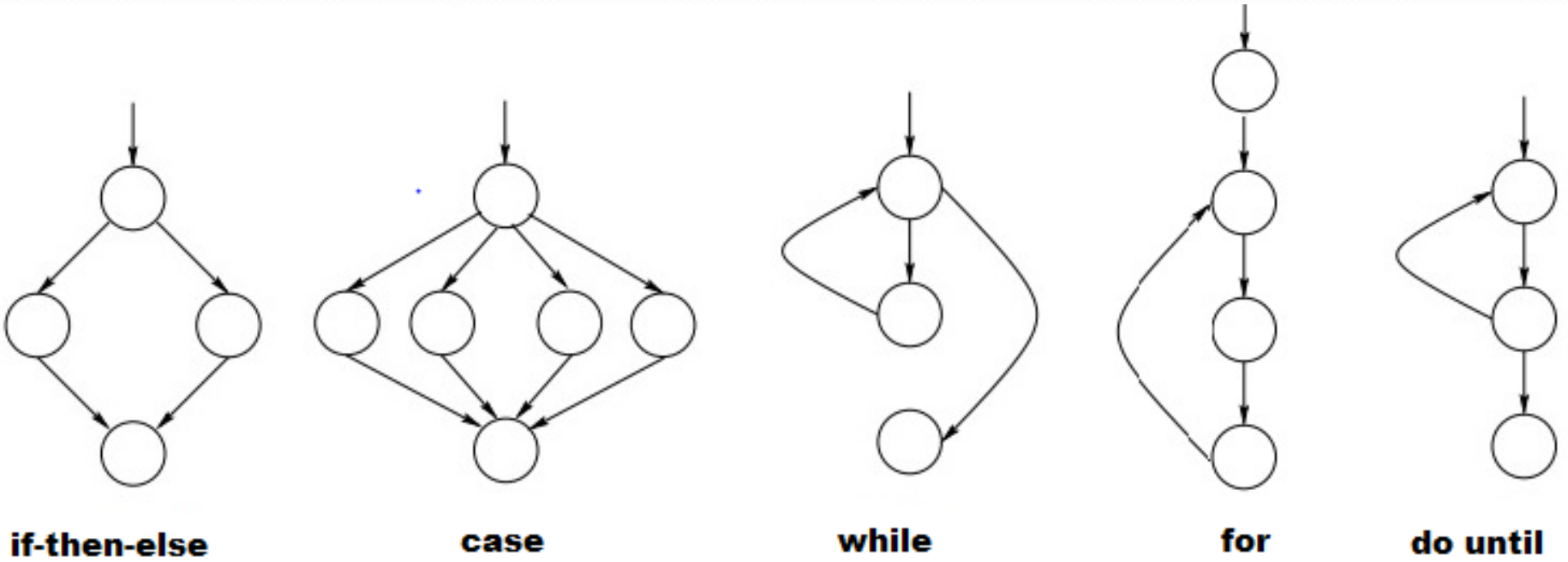


White-box techniques

- Some techniques
  - control flow testing
    - statement testing
    - decision testing
    - condition testing
    - decision/condition testing
    - multiple condition testing
    - path testing
  - data flow testing

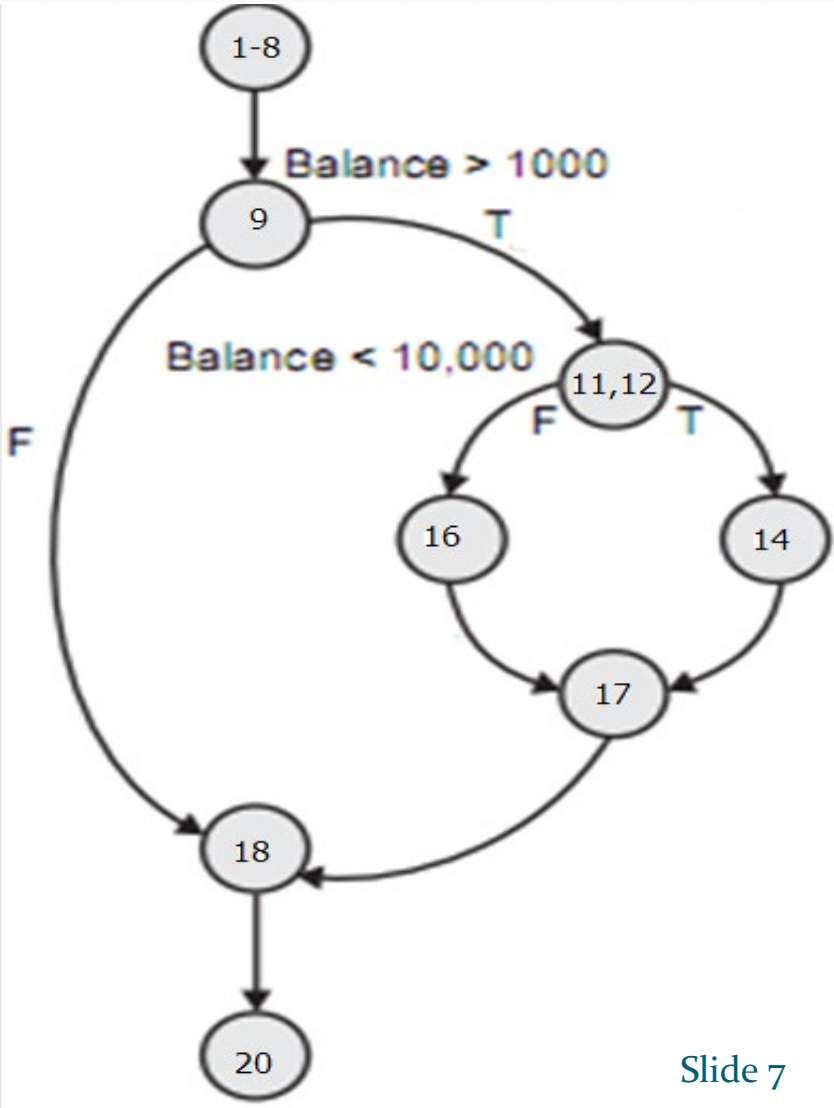
Control-flow graph

- Flow graphs for control structures



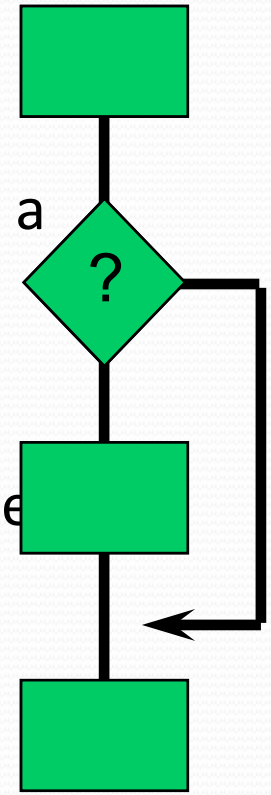
Control-flow graph - Example

```
1    Program BestInterest
2    Interest, Base Rate, Balance: Real
3
4    Begin
5    Base Rate = 0.035
6    Interest = Base Rate
7
8    Read (Balance)
9    If Balance > 1000
10   Then
11       Interest = Interest + 0.005
12       If Balance < 10000
13       Then
14           Interest = Interest + 0.005
15       Else
16           Interest = Interest + 0.010
17       Endif
18   Endif
19
20   Balance = Balance × (1 + Interest)
21
22   End
```



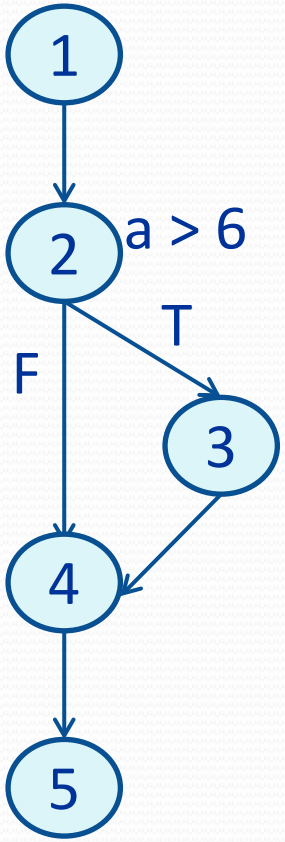
Statement testing (Level 1)

- *A test design technique in which test cases are designed to execute statements* [ISTQB Glossary]
- Statement coverage
  - the percentage of executable statements exercised by a test suite
    - $= \frac{Number\ of\ statements\ exercised}{Total\ number\ of\ statements} x100\%$
  - black-box testing: only 60% to 75% statement coverage
  - typical ad hoc testing achieves 30%
- How to get 100% statement coverage?
  - Find out the shortest number of paths which all the nodes will be covered



Statement coverage example 1

```
1   read(a)
2   IF a > 6 THEN
3      b = a
4   ENDIF
5   print b
```



How many test cases to get 100% statement coverage?

#	Condition	Input	Line number excuted	Expected result
1	a>6	7	1,2,3,4,5	7

Statement coverage example 2

```
1. public int MaxAndMean(int A, int B, int C, out double Mean)
2. {
3.     Mean = (A + B + C) / 3.0;
4.     int Maximum;
5.     if (A > B)
6.         if (A > C)
7.             Maximum = A;
8.         else
9.             Maximum = B;
10.    else
11.        if (B > C)
12.            Maximum = B;
13.        else
14.            Maximum = C;
15.  return Maximum;
16. }
```

A program for calculating the mean and maximum of three integers.

a. How many test cases will you need to achieve 100% statement coverage?

b. What will the test cases be?

Statement coverage example 2 Solution

```
1. public int MaxAndMean(int A, int B, int C, out double Mean)
2. {
3.     Mean = (A + B + C) / 3.0;
4.     int Maximum;
5.     if (A > B)
6.         if (A > C)
7.             Maximum = A;
8.         else
9.             Maximum = B;
10.    else
11.        if (B > C)
12.            Maximum = B;
13.        else
14.            Maximum = C;
15.  return Maximum;
16. }
```

Statement coverage example 2 - Solution

- Design test case

Statement coverage exercise

```
1    Program Grading
2
3    StudentScore: Integer
4    Result: String
5
6    Begin
7
8    Read StudentScore
9
10   If StudentScore > 79
11   Then Result = "Distinction"
12   Else
13        If StudentScore > 59
14        Then Result = "Merit"
15        Else
16             If StudentScore > 39
17             Then Result = "Pass"
18             Else Result = "Fail"
19             Endif
20        Endif
21   Endif
22   Print ("Your result is", Result)
23   End
```

A program evaluates grades for students:

- **If StudentScore<=39, print "Fail".**
- **If StudentScore between 39 and 60, print "Pass"**
- **If StudentScore between 59 and 80, print "Merit"**
- **If StudentScore>79, print "Distinction"**

a. How many test cases will you need to achieve 100% statement coverage? Design test case.

b. Suppose we ran two test cases: StudentScore = 50 and StudentScore = 30, which lines of pseudo code will not be exercised? How much is statement coverage?

Statement coverage exercise: Solution

Statement coverage problems

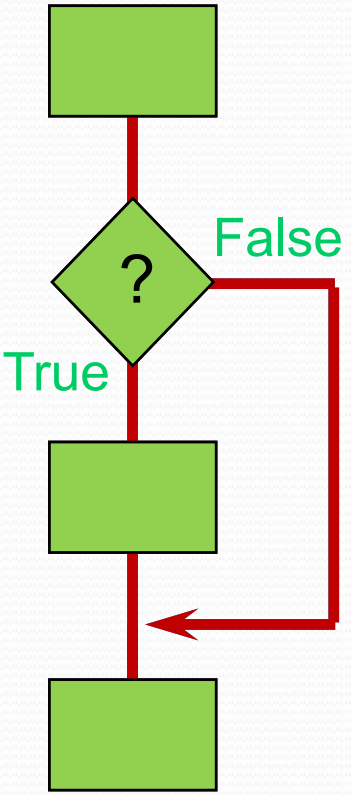
- Statement coverage can be achieved without branch coverage
  - important cases may be missed

Decision testing (Level 2)

- *A test design technique in which test cases are designed to execute **decision outcomes** [ISTQB Glossary]*
  - decision: a logical expression which can be composed of several logical operators like "or", "and", "xor"
  - decision outcome: each exit from a decision
    - two or more possible decision outcomes

IF A > 1 ← **decision**

IF A > 1 AND X = 2



- Known as 'Branch testing', 'Basis path testing'

Decision testing (cont'd)

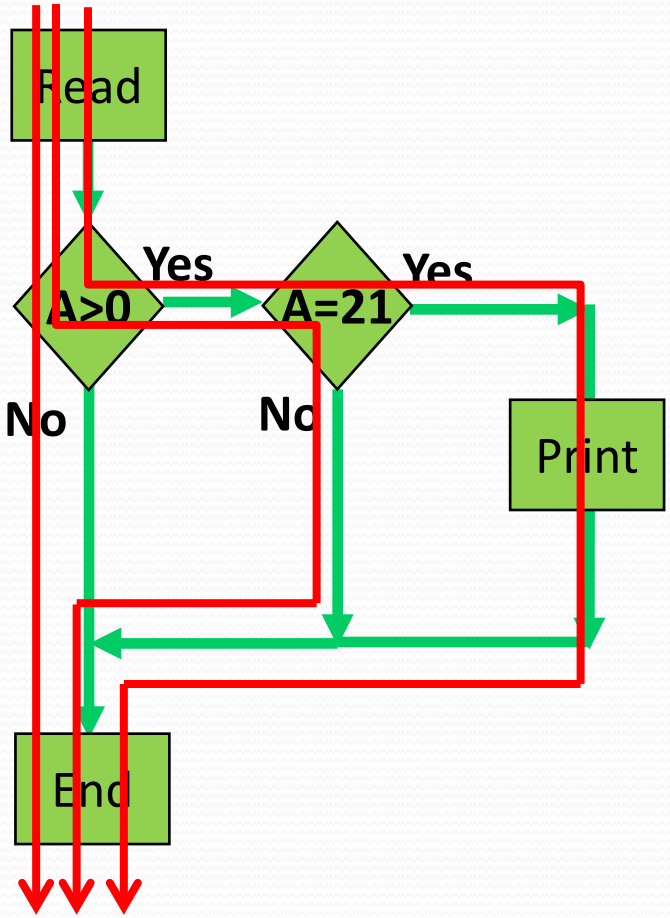
- Decision coverage
  - percentage of decision outcomes exercised

$$= \frac{Number\ of\ decision\ outcomes\ exercised}{Total\ number\ of\ decision\ outcomes} x100\%$$

  - specification-based may achieve only 40% to 60% decision coverage
  - typical ad hoc testing achieves 20%
  - 100% decision coverage guarantees 100% statement coverage, but not vice versa
- How to get 100% decision corverage?
  - Find out the minimum number of paths which will ensure covering of all the edges

Decision testing example 1

```
Read A
IF A > 0 THEN
    IF A  = 21 THEN
        Print "Key"
    ENDIF
ENDIF
```



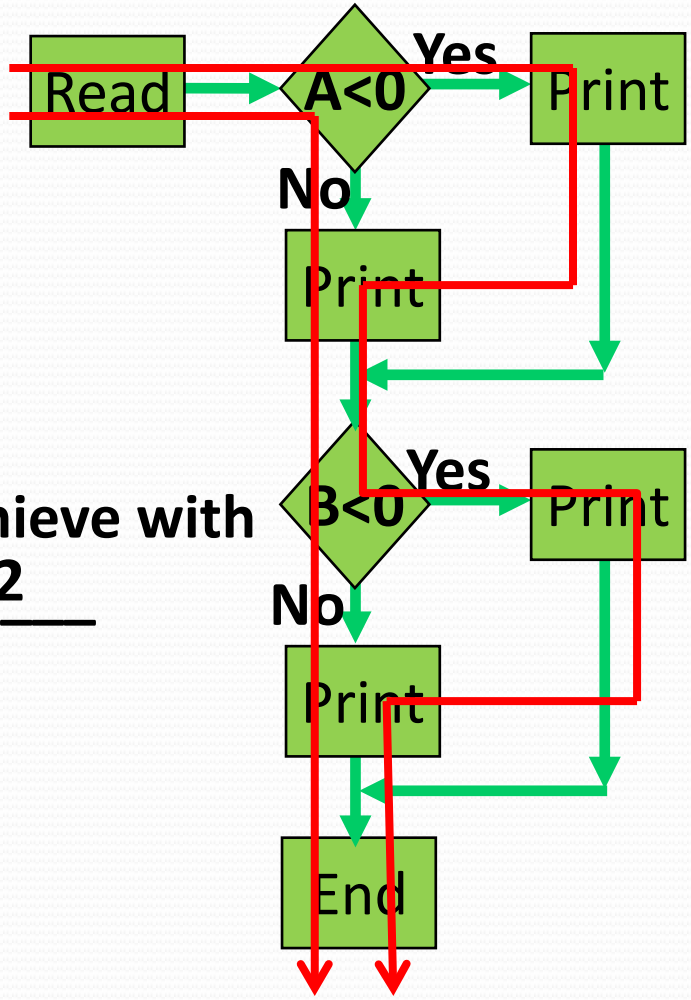
- **Minimum tests to achieve with decision coverage: 3**

#	Cases	Inputs	Expected result
1	A>0(F)	A=-5	No message
2	A>0(T) and A=21(F)	A=10	No message
3	A>0(T) and A=21(T)	A=21	Message "Key"

Decision testing example 2

```
Read A
Read B
IF A < 0 THEN
    Print  “A negative”
ELSE
    Print  “A positive”
ENDIF
IF B < 0 THEN
    Print  “B negative”
ELSE
    Print  “B positive”
ENDIF
```

- **Minimum tests to achieve with decision coverage: 2**



Decision testing exercise 1

```
1    Read(CandidateAge)
2    If  CandidateAge < 18
3    Then
4       Print (“Candidate is too young”)
5    Else
6     If  CandidateAge > 30
7     Then
8       Print (“Candidate is too old”)
9     Else
10      Print(“Candidate may join Club 18–30”)
11    Endif
12   Endif
```

How many test cases for 100% decision coverage?

Solution exercise 1

```
1    Read(CandidateAge)
2    If  CandidateAge < 18
3    Then
4        Print (“Candidate is too young”)
5    Else
6      If  CandidateAge > 30
7      Then
8          Print (“Candidate is too old”)
9      Else
10         Print(“Candidate may join Club 18–30”)
11     Endif
12   Endif
```

Decision testing exercise 2

```
1    Begin
2    Read Time
3    If Time < 12 Then
4        Print(Time, "am")
5    Endif
6    If Time > 12 Then
7        Print(Time -12, "pm")
8    Endif
9    If Time = 12 Then
10       Print (Time, "noon")
11   Endif
12   End
```

a. How many test cases are needed to achieve 100% decision coverage?

b. If the test cases Time = 11 and Time = 15 were input, what level of decision coverage would be achieved?

Solution exercise 2

Decision testing exercise 3

```
7    Index = 0
8    Sum = 0
9    Read (Count)
10   Read (New)
11
12   While Index <= Count
13   Do
14       If New < 0
15       Then
16           Sum = Sum + 1
17       Endif
18       Index = Index + 1
19       Read (New)
20   Enddo
21
22   Print ("There were", Sum, "negative numbers in the input
     stream")
23
24   End
```

What test case for 100% decision coverage?

Solution exercise 3

Test

Suppose you are developing a software unit that will convert a non-signed 16 bit binary number (in string format) to a decimal integer. For example, BinaryToDecimal("0000000000001111") = 15. Draw CFG for this function.

```
1. public long BinaryToDecimal(string sbin)
2. {
3.     int sum = 0;
4.     int tpow = 1;
5.     const int MAX_BITS = 16;
6.     if (sbin.Length > MAX_BITS)
7.        throw new OverflowException("The number is too big.");
8.     for (int i = sbin.Length - 1; i >= 0; i++)
9.     {
10.       if (sbin[i] == '1')
11.          sum = sum + tpow;
12.       else if (sbin[i] != '0')
13.          throw new FormatException("Invalid binary format.");
14.       tpow = 2 * tpow;
15.    }
16.    return sum;
17. }
```

Decision testing problems

- Some branching decisions in programs are made not based on a single condition but on **multiple conditions**
  - Decision coverage does not ensure that all entry-exit paths are executed
- A compound predicate is treated as a single statement
  - If n clauses, 2^n combinations, but only 2 are tested
  - Example

```
if (condition1 && (condition2 || function1()))
        statement1;
else
        statement2;
```

Condition testing (Level 3)

- Design test cases based on Boolean sub-expression (BsE): **each condition** to be evaluated as **true and false at least once**
- Condition coverage
  - $= \frac{Number\ of\ BsE\ outcomes\ exercised}{Total\ number\ of\ BsE\ outcomes} x100\%$

Decision/Condition testing (Level 4)

- Full condition coverage does not guarantee full decision coverage
  - Example: if (x&&y) {conditionedStatement;}
  - Using condition coverage, if we choose two test cases (**x=TRUE, y=FALSE** and **x=FALSE, y=TRUE**), the conditionedStatement will never be executed

- Decision/Condition testing: Test cases are created for **every condition** and **every decision**

Multiple condition testing (Level 5)

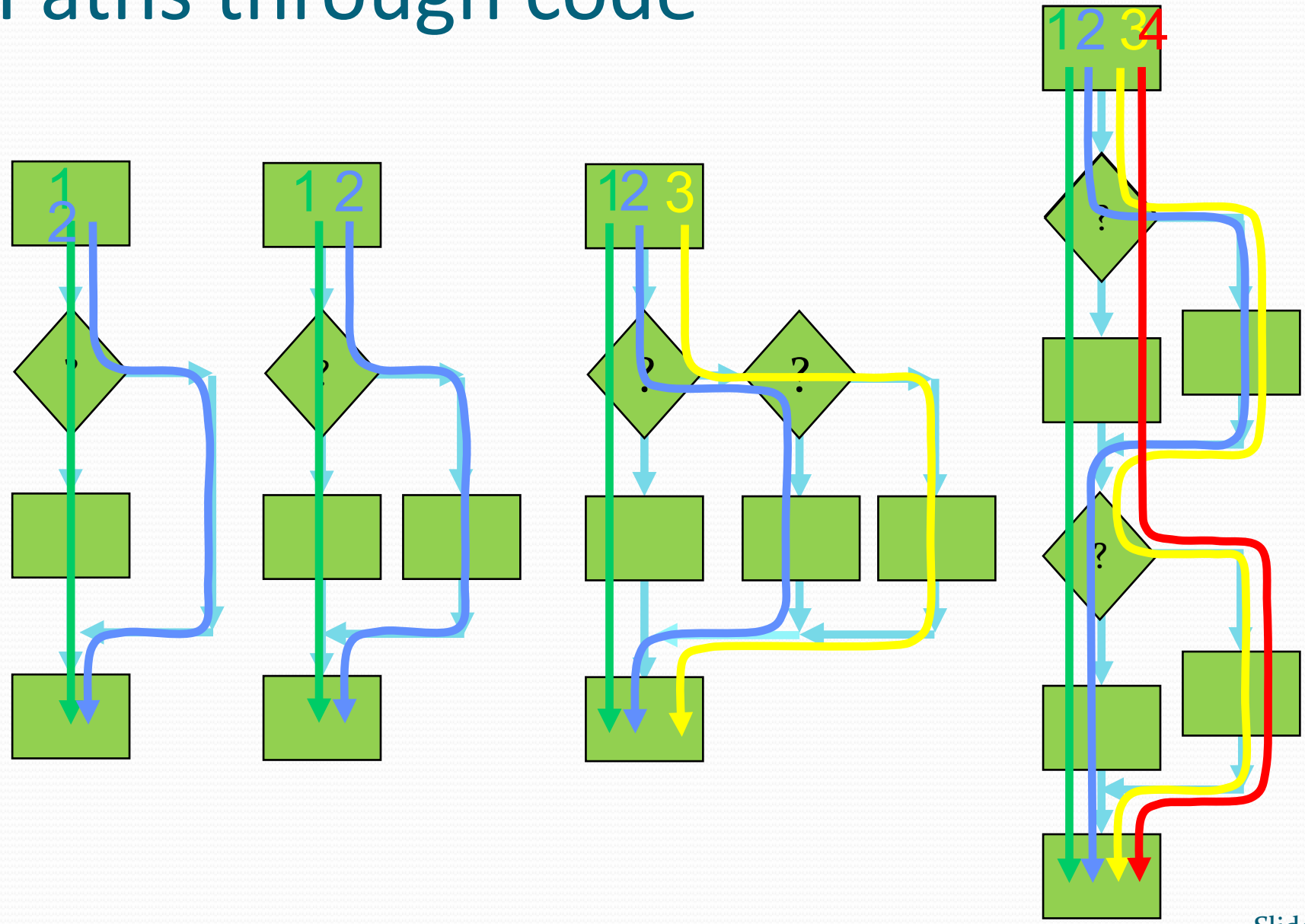
- Known as ‘condition combination testing'
- Requiring 2^n test cases to achieve 100% coverage of a decision containing n boolean operands
- Example: A or (B and C)

Case	A	B	C
1	FALSE	FALSE	FALSE
2	FALSE	FALSE	TRUE
3	FALSE	TRUE	FALSE
4	FALSE	TRUE	TRUE
5	TRUE	FALSE	FALSE
6	TRUE	FALSE	TRUE
7	TRUE	TRUE	FALSE
8	TRUE	TRUE	TRUE

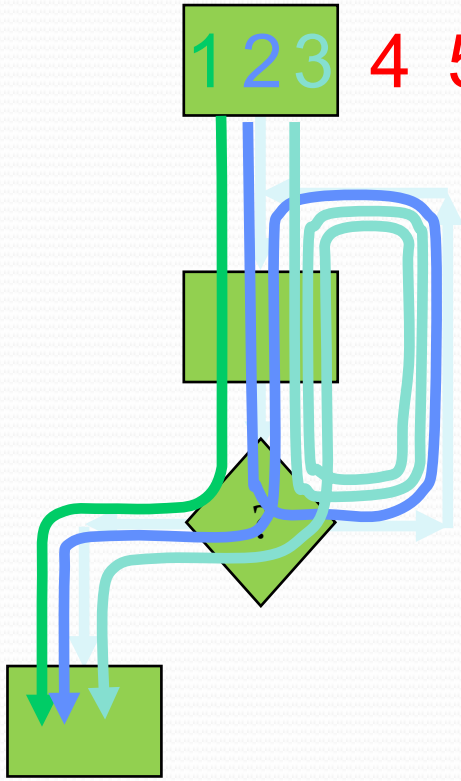
Path testing (Level 6)

- Design test cases based on what **paths** which should be exercised
- Path coverage
  - $= \frac{number\ of\ paths\ exercised}{total\ number\ of\ paths} x100\%$

Paths through code



Paths through code with loops



for as many times as it is possible to go round the loop (this can be unlimited, i.e. infinite)

White box techniques - Advantages

- It permits direct checking of processing paths and algorithms
- It provides line coverage follow-up that delivers lists of lines of code that have not yet been executed
- It is capable of testing the quality of coding work

White box techniques - Disadvantages

- It requires vast resources, much above those required for black box testing
- It cannot test the performance of software in terms of availability, reliability, stress, etc.
- The tester must have sufficient programming skill to understand the code and its control flow
- Control flow testing can be very time consuming because of all the modules and basis paths that comprise a system

Contents

Non-systematic test techniques

- Based on a person's knowledge, experience, imagination and intuition
- Some techniques
  - error guessing
  - exploratory testing

It is true that testing should be rigorous, thorough and systematic

This is not all there is to testing

Error guessing

- No rules, no script
- Think of situations in which the software may not be able to cope
  - division by zero
  - blank (or no) input
  - empty files
  - wrong kind of data (e.g. alphabetic characters where numeric are required)...
- After systematic techniques have been used
- Supplements systematic techniques

Not a good approach to start testing with

Exploratory testing

This testing helps improving quality

"A style of testing in which you **explore the software** while simultaneously designing and executing tests, using feedbacks from the last test to inform the next." (Elisabeth Hendrickson)

Exploratory testing as 'an interactive process of simultaneous learning, test design, and test execution' (James Bach)

Exploratory testing (cont.)

- The test design and test execution activities are performed in parallel typically without formally documenting the test conditions, test cases or test scripts
- Test logging is undertaken as test execution is performed, documenting the key aspects of what is tested, any defects found and any thoughts about possible further testing
- No limits on test techniques that explorers can use
  - combines with formal testing techniques
- Most useful when there are no or poor specifications and when time is severely limited

Contents

Choosing test techniques

- Each technique is good for certain things, and not as good for other things
- The best testing technique is no single testing technique: each testing technique is good at finding one specific class of defect
- How can we choose the most appropriate testing techniques to use? - Depend on internal and external factors

Choosing test techniques (cont'd)

- Internal factors
  - models used
  - tester knowledge or experience
  - test objective
  - documentation
  - life cycle model

Choosing test techniques (cont'd)

- External factors
  - risk
  - customer or contractual requirements
  - type of system
  - time and budget
  - regulatory requirements