Software Quality Assurance and Testing Technology

2nd Semester, Spring 2022

Haiming Liu

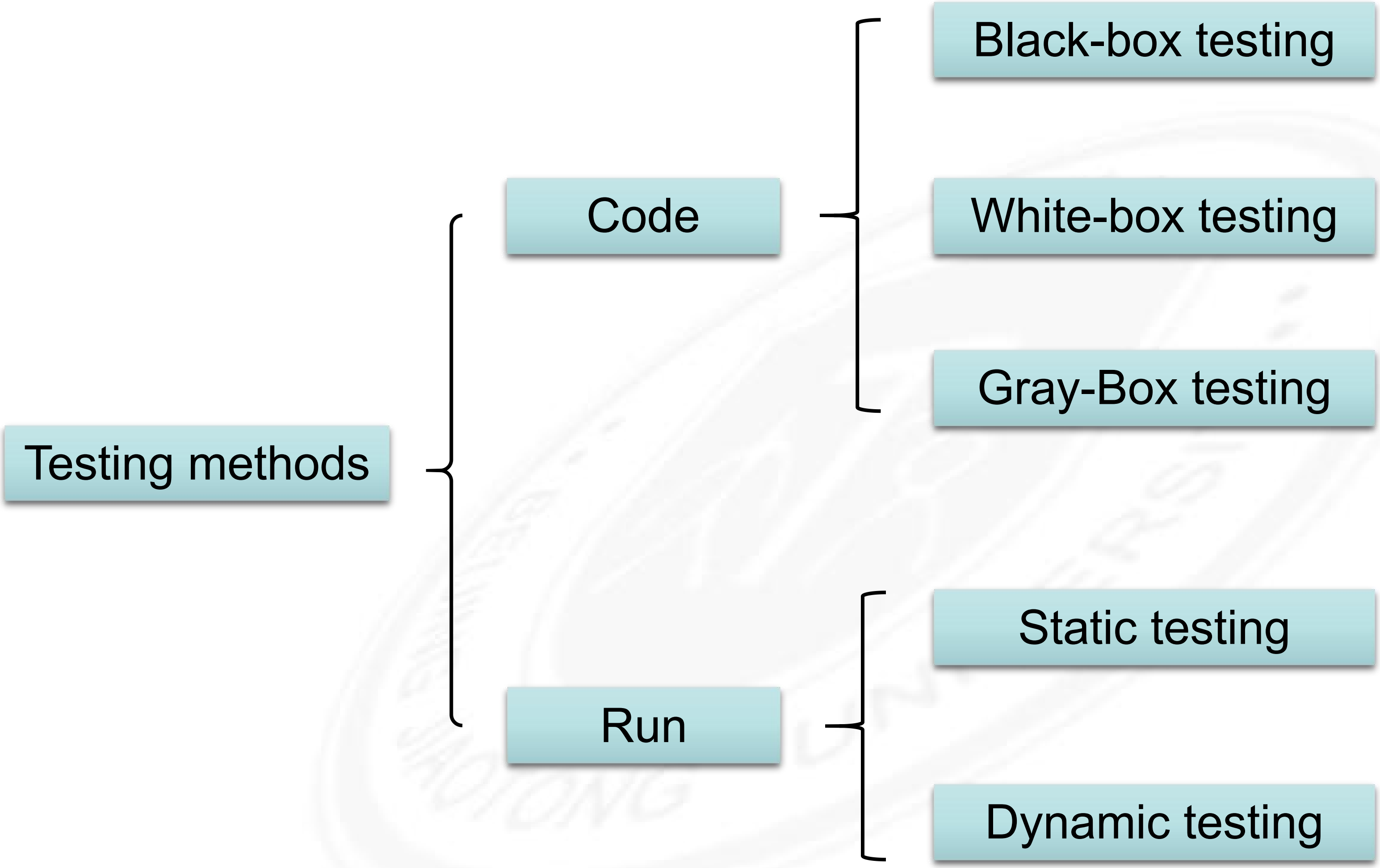
School of Software Engineering

Beijing Jiaotong University

Review

Week 1:	The basic concepts and theories of testing
Week 2-3:	Principles of Testing
Week 4:	Testing the specification
Week 5-6:	Black Box Testing
Week 7-9:	White Box Testing
Week 10:	Integration Testing and System Testing
Week 11:	Usability Testing and Accessibility Testing
Week 12:	Security Testing
Week 13:	Mutation Testing
Week 14:	Software Quality
Week 15:	Review I
Week 16:	Review II

What is Testing



Testing methods

Code

Black-box testing

White-box testing

Gray-Box testing

Run

Static testing

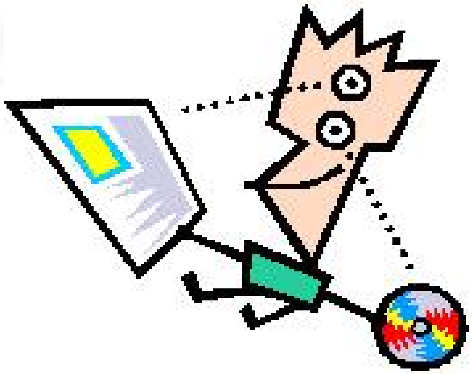
Dynamic testing

TESTING FUNDAMENTALS

EXAMINING THE CODE

Examine the Design Documents & Code

- These are **static, white box** techniques.
- Handling these requires some programming expertise.
- It is best if the testers know the language in which the software is written.
- Consequently, you often find these tests are run by either Programmers with software testers as observers or Software testers with help from the programmers.



Major Problems with These Tests

- They are often NOT performed!
- These are the hardest to justify to upper management as they are viewed by many as too time consuming.
- Some of the problem is the perception that programmers are not productive if they are not generating code.

Note: The tyranny of the LOC metric, Lines of executable code as a measure of productivity!

FORMAL REVIEWS

These are structured processes for doing static, white box testing.

4 elements are required:

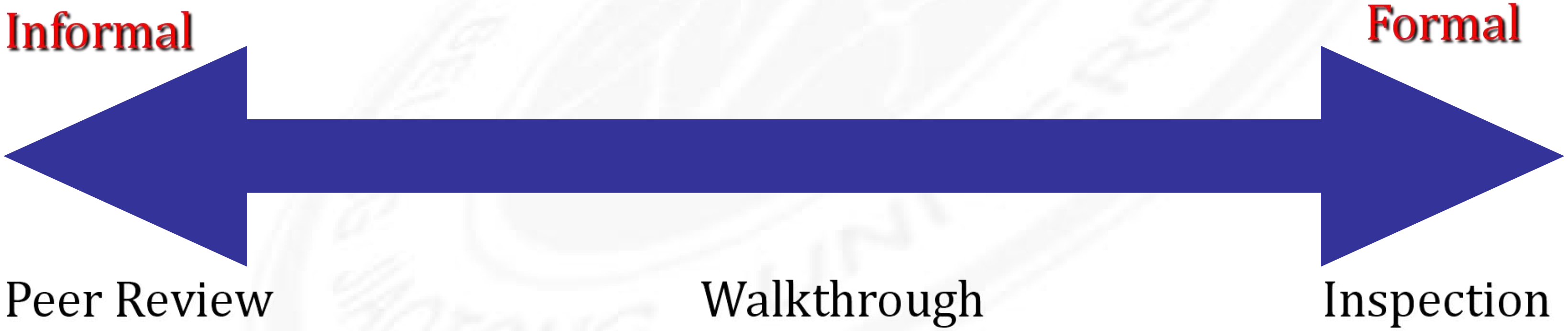
1. Identify problems by directing attention to the code, not who wrote it.
2. Setup and follow rules for the review:
   - How much code should be examined?
   - How long should the review take?
   - What is fair game for the review?
3. Prepare and assign duties to people

   Moderator, recorder, reader, etc.
4. Write a report.

These are not just “get together and go over code” sessions!

Three approaches

- **Peer review**
- **Walkthrough**
- **Formal inspection**



Typically, different levels of formality identify the kind of formal review:

Peer (or Buddy) Review:

- Most **informal.**
- Involves a coder and a few buddies.
- Still be sure all 4 elements are present.

Walkthroughs:

- Next step in formality.
- The programmer works with a small group of ~5 programmers and testers.
- Everyone has copies of the code in advance.
- A presenter “reads” the code line-by-line, function by function, saying what is done and why it is being done.

Example in a Walkthrough

Consider

```
for (i = 1; i < n; i++)
        cout << a[i] << endl;
cout << i << endl;
```

Reader explains

That i is the index of an array named a.

The variable n is initialized elsewhere (and identifies where).

The loop outputs values for a[1], a[2], ..., a[i-1].

Questions raised:

Where does the variable i get a value for the last line?

Does the programmer expect the output value for i to be inside the loop?

Inspections

- Most **formal** of the reviews.
- Very highly structured.
- The agenda and code to consider is available in advance of the meeting.
- The presenter or reader isn’t one of the programmers.
- All the other people are inspectors playing different roles. Examples are

  - User

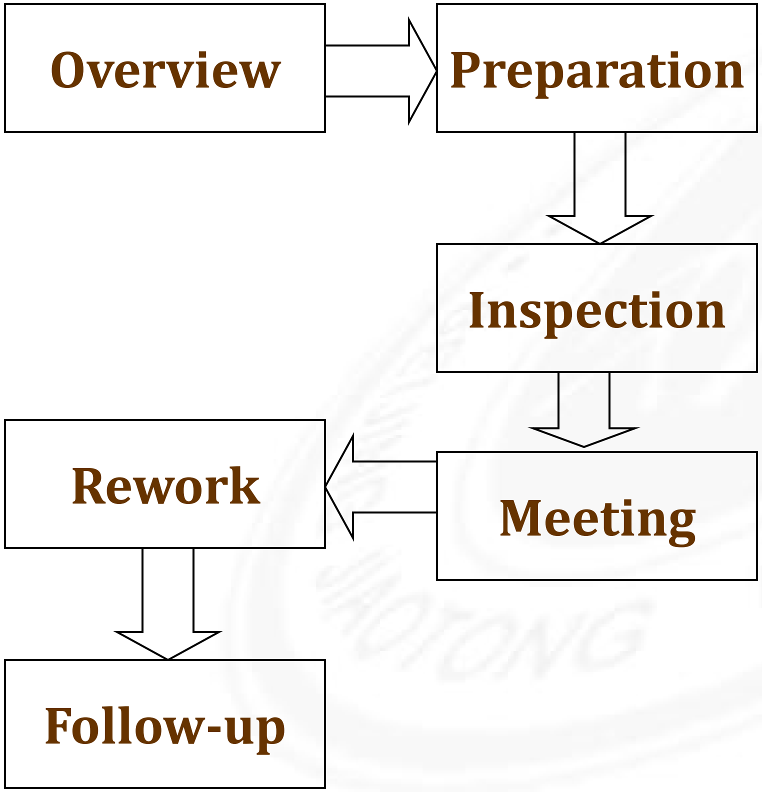
  - Tester

  - Product support person

  - Have a moderator and a recorder.

Formal Reviews - Formal inspection

1. Well-defined roles and responsibilities
2. Well-defined steps



Formal Reviews

- Look for problems and omissions in the code.
- May check also to see if the code is written to adhere to pre-specified **standards** or **guidelines**.
- There is a lot of literature on how formal reviews should be conducted.
- Most companies that use them develop their own checklists.

One list to check while doing formal reviews (from the text) :

- Data reference errors.
- Data declaration errors.
- Computation errors.
- Comparison errors.
- Control flow errors.
- Subroutine (or function) parameter errors.
- I/O errors
- Miscellaneous

Standards and Guidelines

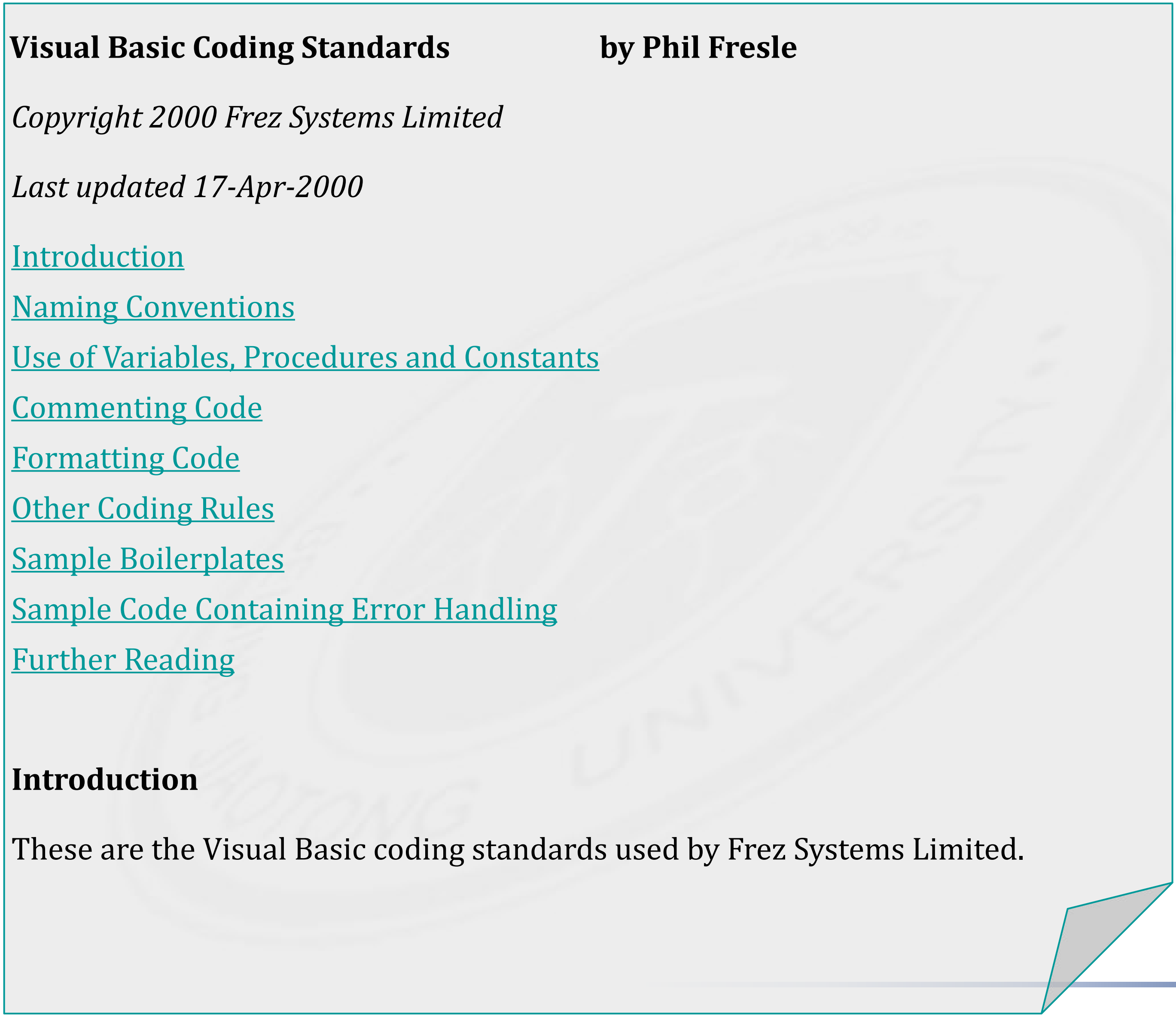
Be careful not confuse these with style considerations.

✓ Indenting rules are about style, not something that affects whether a program is correct or not.

Examples of standards or guidelines:

- Don’t use GOTOs
- Use WHILE loops, instead of DO-WHILE loops except in rare cases.

Examples



Visual Basic Coding Standards **by Phil Fresle**

Copyright 2000 Frez Systems Limited

Last updated 17-Apr-2000

Introduction

These are the Visual Basic coding standards used by Frez Systems Limited.

Another Example

1. Foreword
2. Release Note
3. Commentary in file
4. Constant
5. Variable
6. Struct/Enum definition
7. Expression and code blocks
8. Some good habit

Why Use Standards or Guidelines

Studies show they increase

- Reliability
- Readability and, hence, maintainability
- Portability

Some contractors require that certain standards be used when developing software for them.

Good example- the government

Organizations Producing Various Standards and Guidelines

ANSI – **American National Standards Institute**

IEC – **International Engineering Consortium**

ISO – **International Organization for Standardization**

NCITS **– National Committee for Information Technology Standards**

Plus various professional organizations

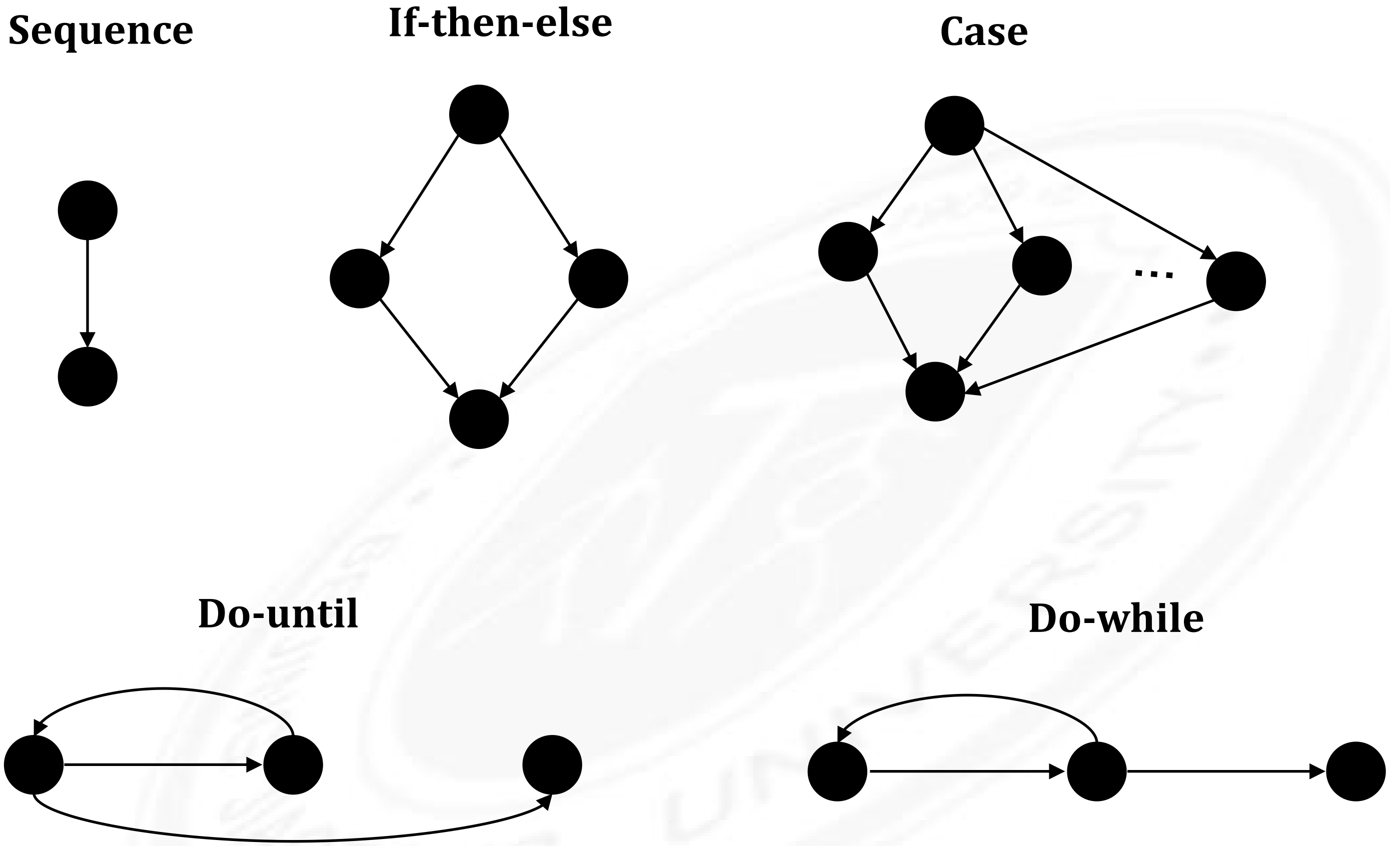
- ACM – **Association for Computing Machinery**
- IEEE – **Institute of Electrical and Electronic Engineering**

TESTING FUNDAMENTALS

TESTING THE SOFTWARE WITH X-RAY GLASSES

BEIJING JIAOTONG UNIVERSITY

Flow graph from code



Sequence

If-then-else

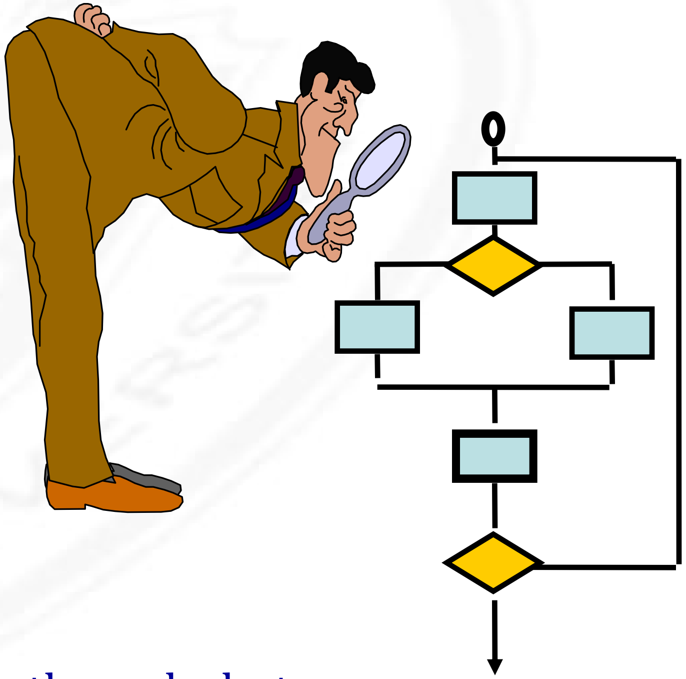
Case

Do-until

Do-while

Dynamic, White Box Testing

- Control Flow Testing also called **structural testing**.

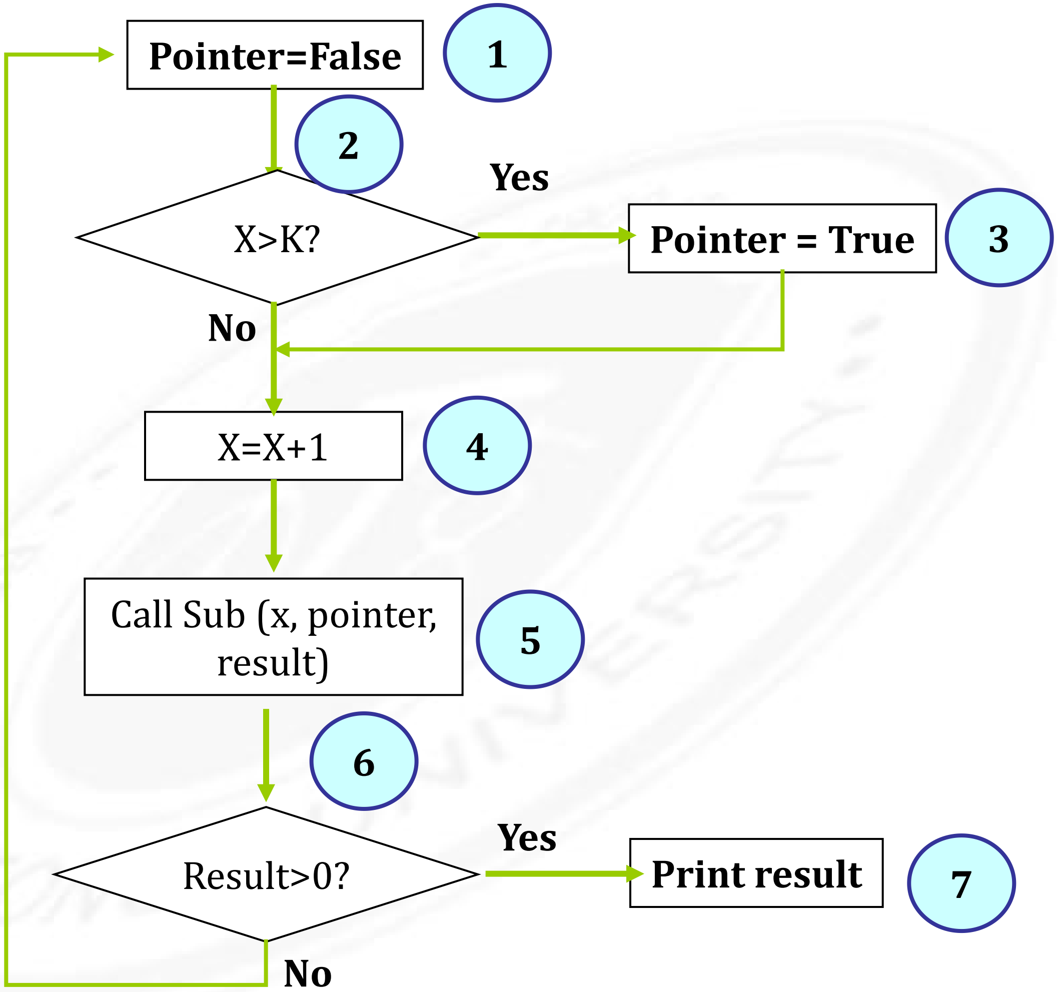


- Control-flow testing techniques are based on judiciously selecting a set of test paths through the program.
- The set of paths chosen is used to achieve a certain measure of testing thoroughness.

  E.g., pick enough paths to assure that every source statement is executed as least once.

- Techniques here are not limited to just examining the code, but involve directly controlling the software.

Examples


Pointer=False
1
2
X>K?
Yes
Pointer = True
3
No
X=X+1
4
Call Sub (x, pointer, result)
5
6
Result>0?
Yes
Print result
7
No

Overview of the Areas of Dynamic, White Box Testing

- Directly test the **pieces**--- the **low-level** functions, procedures, subroutines or libraries.
- Do **top level** testing of the completed program, but choose test cases by knowledge of the code.
- Directly access **variables** and **state information** and force the software to do things.
- **Measure** how much of the code has been tested and be able to adjust your tests to remove redundant test cases and add missing ones.

CAUTION: Be careful to not confuse testing with debugging!
When you try to correct bugs, you are debugging. Normally, programmers debug.

TESTING

- The **operation of a system or application** under controlled conditions and the **evaluation of results** with the intent of finding errors.
  - Should include normal and abnormal conditions
- **Testing intentionally attempts to make things go wrong to determine:**
  - if things happen when they shouldn’t
  - if things don’t happen when they should
- Oriented towards “**detection**”

DEBUGGING

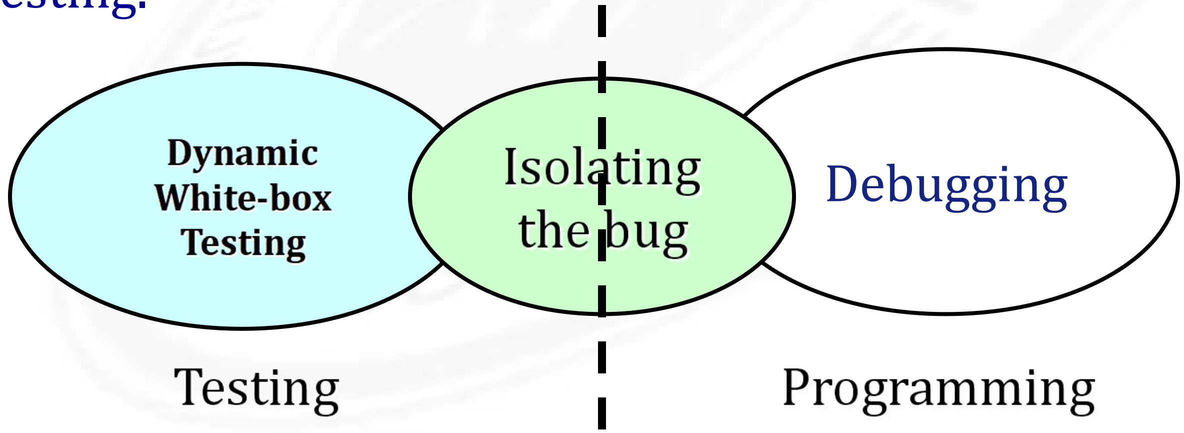
- DEBUGGING **starts with an identified** error and is the process of locating what is causing the bug and correcting the flaw.

- It is NOT the process of showing that a bug exists.

- Oriented towards "**correction**".

Debugging plays a role

- We see this even with **compiler** errors.
- Consequently, testing requires that **debugging** be done quickly after some bugs are found.
- Remember, the earlier a bug is found, the cheaper it is to fix. Bugs often mask other bugs
- Always remember that creating black-box test cases based on specs is important as these expose misinterpreted ideas, which can't be found by white-box testing.



White-Box Testing

1. **Basic Path Testing**
   exercise each independent path at least once

2. **Condition Testing**
   exercise all logical conditions on their true and false sides

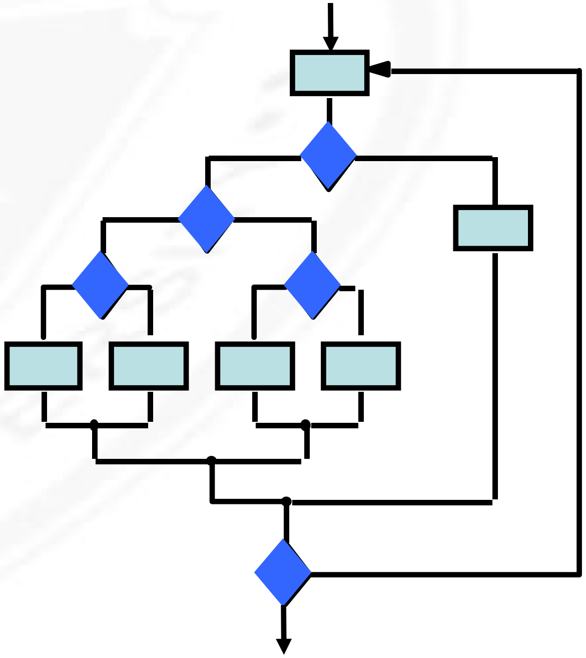
3. **Loop Testing**
   execute all loops at their boundaries and within their bounds

4. **Data Flow Testing**
   exercise all data structures to ensure their validity

Basic Path Testing

Goal: exercise each independent path at least once.

1. Using the code, draw a corresponding **flow graph**
2. Determine the **cyclomatic complexity** of the flow graph.
3. Determine a **basis set** of linearly independent paths.
4. Prepare **test cases** that **force** the **execution of each path** in the basis set.



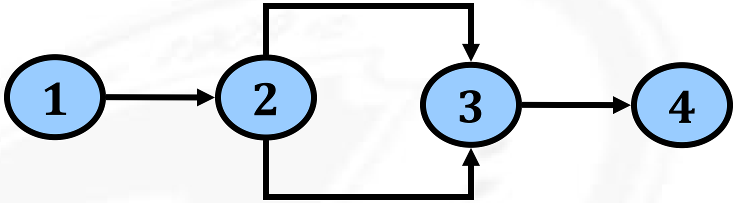
Flow graph from code

Flow graphs Consist of Three Primitives

- A **decision** is a program point at which the control can diverge.
  - (*e.g.*, if and case statements).
- A **junction** is a program point where the control flow can merge.
  - (*e.g.*, end if, end loop, goto label)
- A **process block** is a sequence of program statements uninterrupted by either decisions or junctions. (*i.e.*, straight-line code).
  - A process has one entry and one exit.
  - A program does not jump into or out of a process.

Basis path test

- A **path** through a program is a sequence of statements that starts at an entry, junction, or decision and ends at another (possible the same), junction, decision, or exit.



- A path may go through several junctions, processes, or decisions, one or more times.
- Paths consist of **segments**.
- The smallest segment is a link. A **link** is a single process that lies between 2 nodes.
- The length of a path is the number of links in a path.
- An entry/exit path or a complete path is a path that starts at a routine's entry and ends at the same routine's exit.

Basis path test

- Complete paths are useful for testing because:
  - It is difficult to set up and execute paths that start at an arbitrary statement.
  - It is difficult to stop at an arbitrary statement without changing the code being tested.
  - We think of routines as input/output paths.

- Path Selection Criteria
  - There are many paths between the entry and exit points of a typical routine.
  - Even a small routine can have a large number of paths.

Example

Procedure: process records

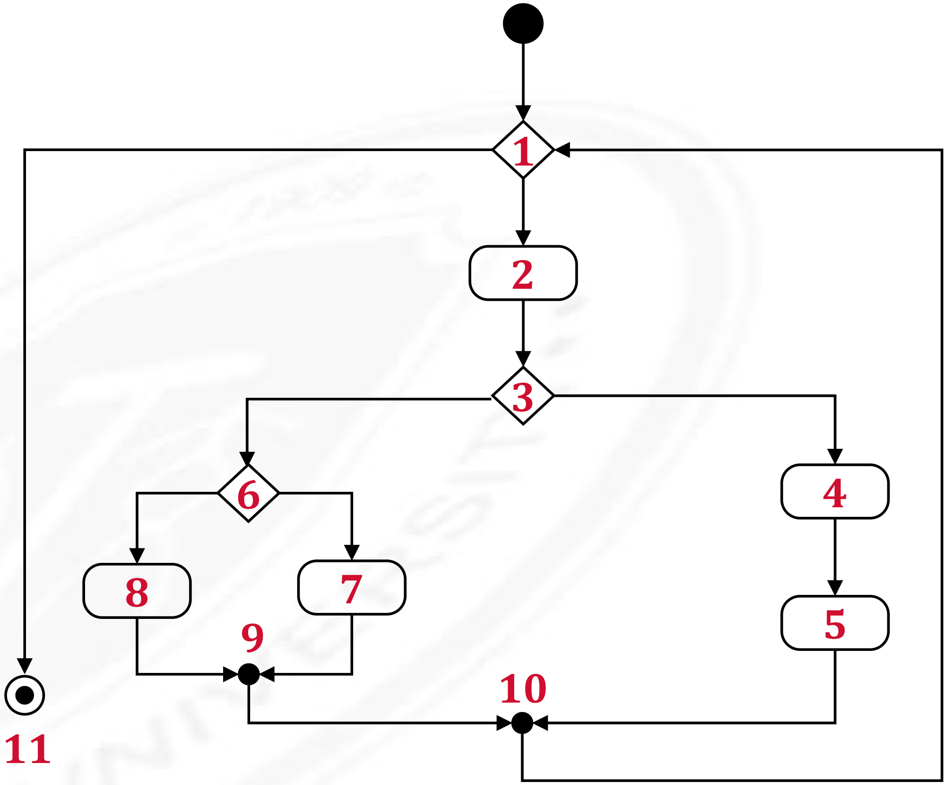
```
Do While records remain
    Read record;
    If record field 1 = 0 Then
        store in buffer;
        increment counter;
    Else If record field 2 = 0 Then
        reset counter;
        Else store in file;
        End If
    End If
End Do
End
```

```
    Do While records remain
        Read record;
        If record field 1 = 0 Then
            store in buffer;
            increment counter;
        Else If record field 2 = 0 Then
            reset counter;
            Else store in file;
            End If
        End If
    End Do
```

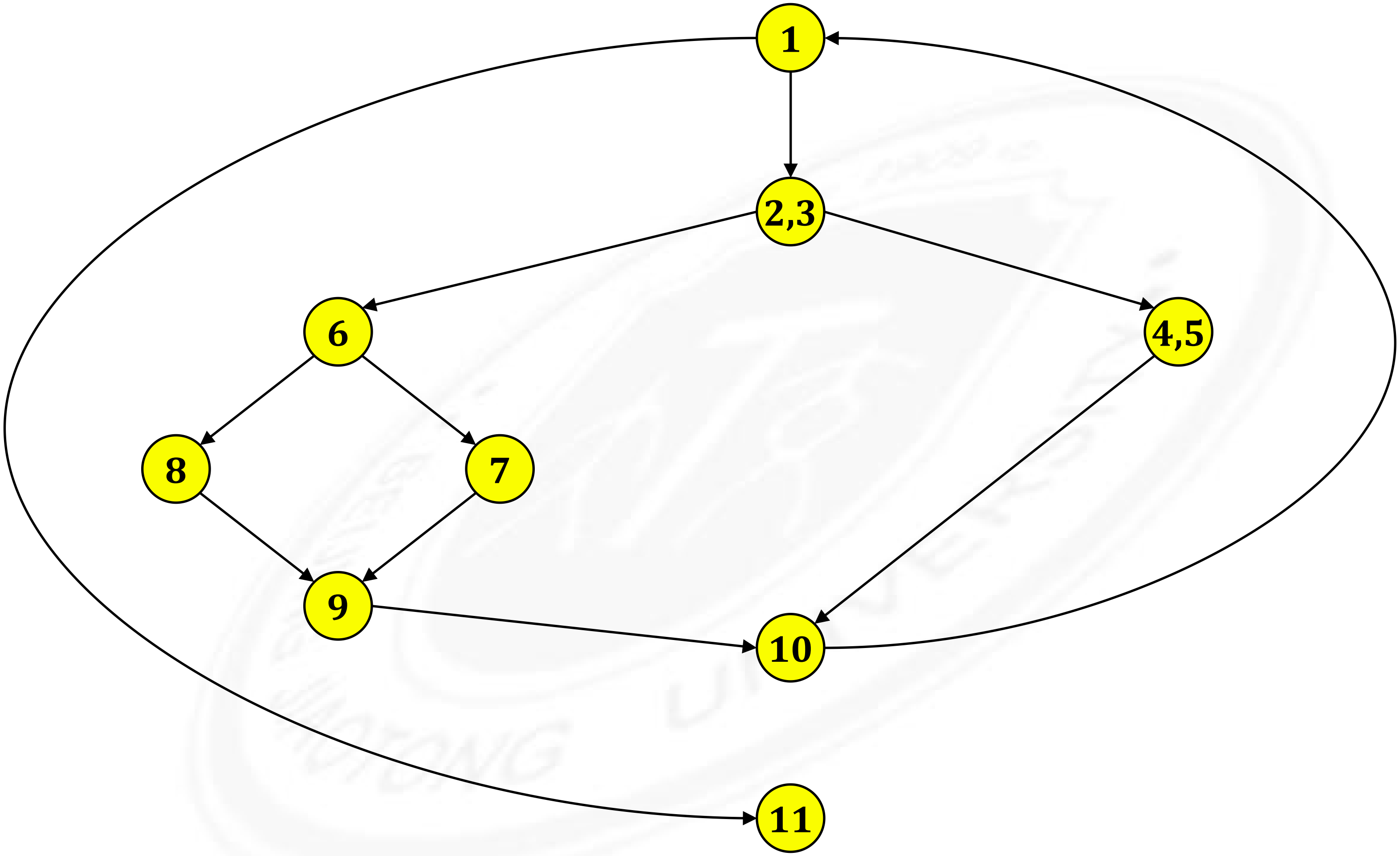
Example (continued)

Procedure: process records

```
1.     Do While records remain
2.         Read record;
3.         If record field 1 = 0 Then
4.                 store in buffer;
5.                 increment counter;
6.         Else If record field 2 = 0 Then
7.                 reset counter;
8.                 Else store in file;
9.                 End If
10.        End If
11.    End Do
End
```

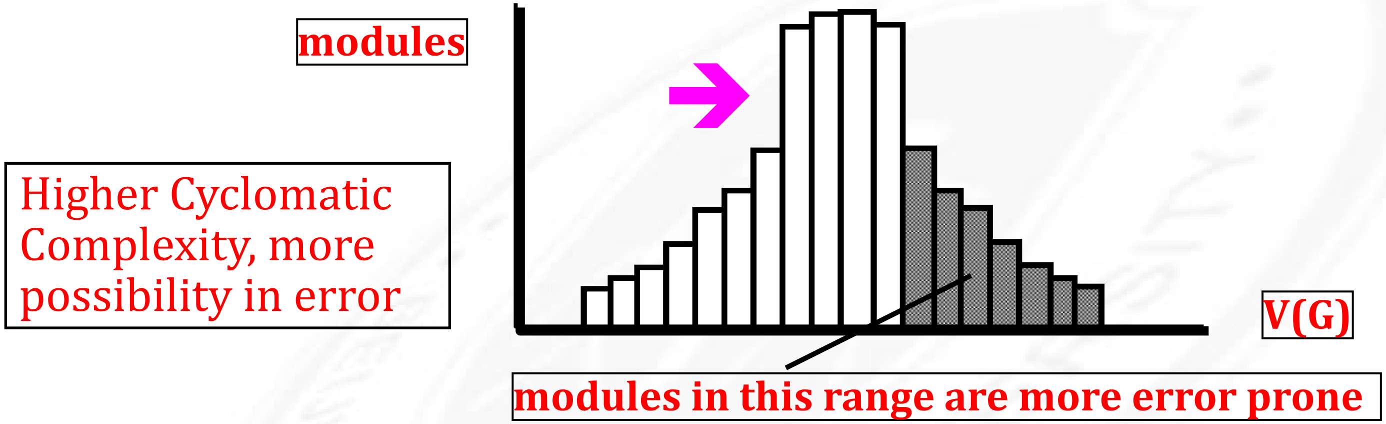


Basis path test example: FLOW GRAPH



Determine cyclomatic complexity of flow graph

Cyclomatic complexity: a quantitative measure of the logical complexity of code, provides an upper bound on the number of paths that need to be tested in the code



- V(G) =number of regions (areas bounded by nodes and edges—area outside the graph is also a region)
- V(G) = number of edges - the number of nodes + 2
- V(G) = number of (simple) predicate nodes + 1

Determine a basis set of linearly independent paths

- Independent path ® any path that introduces at least one new set of processing statements or a new condition

- Basis set ® set of independent paths through the code

- Test cases derived from a basis set are guaranteed to execute every statement at least one time during testing

- Basis set is not unique

Basis path test example 1

```
scanf("%d %d",&x, &y)
if (y < 0)
    pow = -y;
else
    pow = y;
z = 1.0;
while (pow != 0) {
     z = z * x;
    pow = pow - 1;
     }
if (y < 0)
     z = 1.0 / z;
printf ("%f",z);
```

1. Please draw the control flow graph of the following code and provide the cyclomatic complexity V(G) of the control flow graph;
2. Please provide the Basis Path set of the control flow graph;

【腾讯文档】Basis path test example 1
https://docs.qq.com/form/page/DSFdHVlIleEZnQnVa

Basis path test example 2

```
for (j=1; j<N; j++)
 {
   last = N - j + 1;
   for (k=1; k<last; k++)
     {
       if (list[k] > list[k+1])
            {
             temp = list[k];
             list[k] = list[k+1];
              list[k+1] = temp;
            }
         }
     }
print(“Done\n”);
```

【腾讯文档】Basis path test example 2
https://docs.qq.com/form/page/DSEF5cXRYeUdnTlhY

To be continued…

See you next week