CSE4006 Software Engineering

11. Software Testing

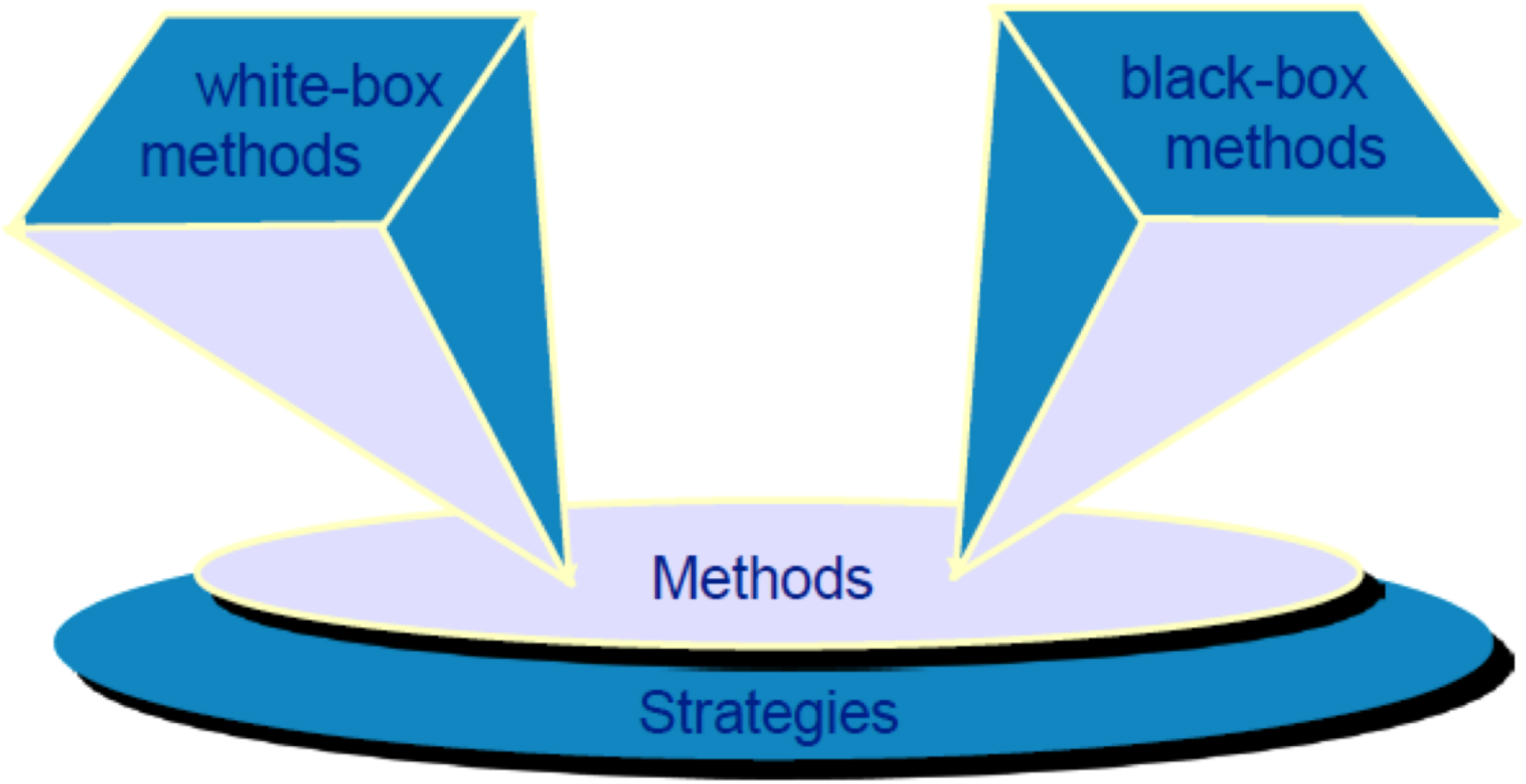
Scott Uk-Jin Lee

Department of Computer Science and Engineering
Hanyang University ERICA Campus

1st Semester 2015

lab(se);

1. Software Testing



lab(se);

1.1 Who Conducts Software Test ?

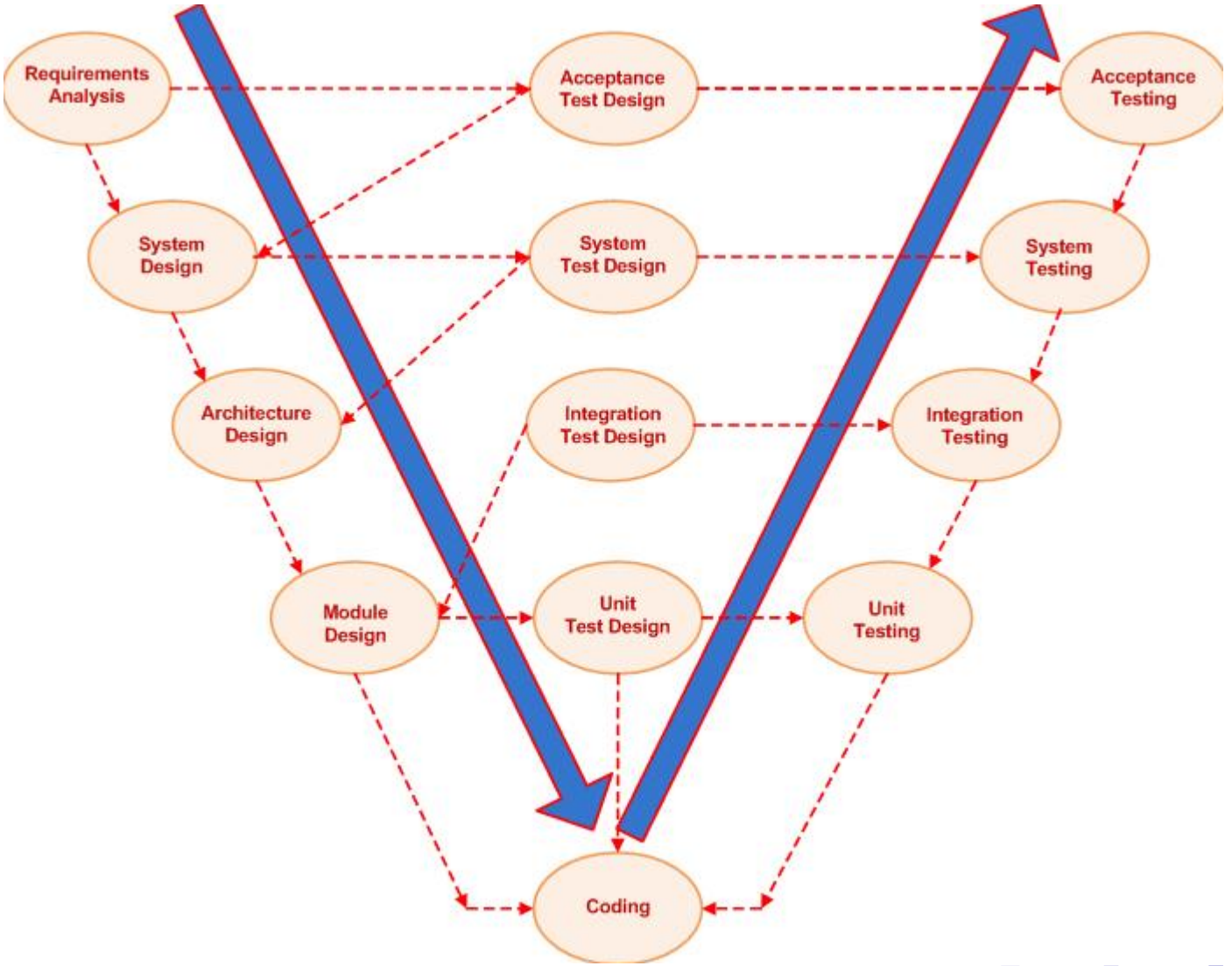
Developer

Test Expert

Understands the system but, will test “gently” and, is driven by “delivery”	Must learn about the system, but, will attempt to break it and, is driven by “quality”

1.2 Testing Strategy

V-Model



lab(se);

1.2 Testing Strategy

- "testing-in-the-small" $\Rightarrow$ "testing-in-the-~~small~~Large"
- Traditional software testing
  - from modules(components) to integration of modules
- Object-Oriented software testing
  - focuses on the class(corresponds to module) which includes attributes and operations

2. Static Analysis of Software

- Static Analysis : method of detecting software defects without an execution
- Types of static analysis
  - Review and Formal Review
  - Code Inspection
  - Formal Method
  - Program Verification

2.1 Review and Formal Review

Review

- self review
- peer review
- **walkthrough** (Fagan inspection by IBM)
  - review conducted by multiple reviewers with predetermined roles & through predefined procedures and preparation process
  - used as the task completion standard for intermediate product since objectivity of task result can be secured
  - used as a means of a project management tool
  - inspection team
    - author, moderator, inspector, reader, recorder
  - often used for the system with high quality requirements
    - since thorough preparation is required in advance
    - due to the size of labour and cost inputted

lab(se);

2.1 Review and Formal Review

Formal Review

- formal review type, procedure, ... are specified in the development plan
- part of review may be imposed as a contract condition
- primarily project milestone management functionality is noticeable
- Types of Formal Review
  - Software Requirements Review (SRR)
  - Preliminary Design Review (PDR)
  - Critical Design Review (CDR)
  - Test Readiness Review
  - Software Acceptance Review

lab(se);

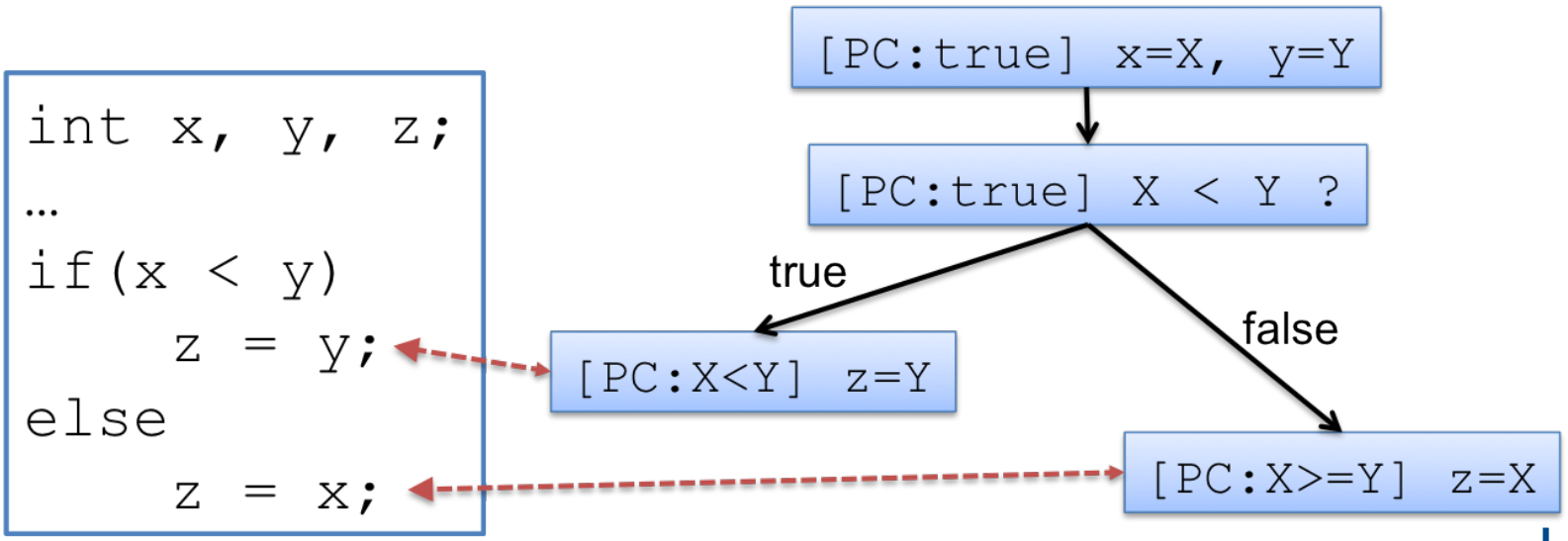
2.2 Code Inspection

- Peer review on code
- Error prevention : reduces the errors propagate to testing stage by 60-80%
- Reduce costs & improve quality and productivity of development and test processs
- 1 good inspection provides the same effectiveness as 30,000 test cases -by Vern Crandell

2.3 Formal Method

- Method of finding program defects through mathematical and logical analysis without an execution

- Symbolic execution
  - verification technique which executes native program with special symbolic variables (without specific data value)



lab(se);

2.4 Program Verification

- Verifies the correctness of a program by showing that the program is consistent with its specification through the application of mathematical laws and logical reasoning
  - typical static analysis techniques proves the correctness of a program indirectly by showing that the program has defects
  - formal specification of a program is required
- Axiomatic Method
  - input output assetion technique
    - configure set of assertions from a program $\rightarrow$ prove each one of them using previously proven assertions + rules and axioms related to program statements and operations

lab(se);

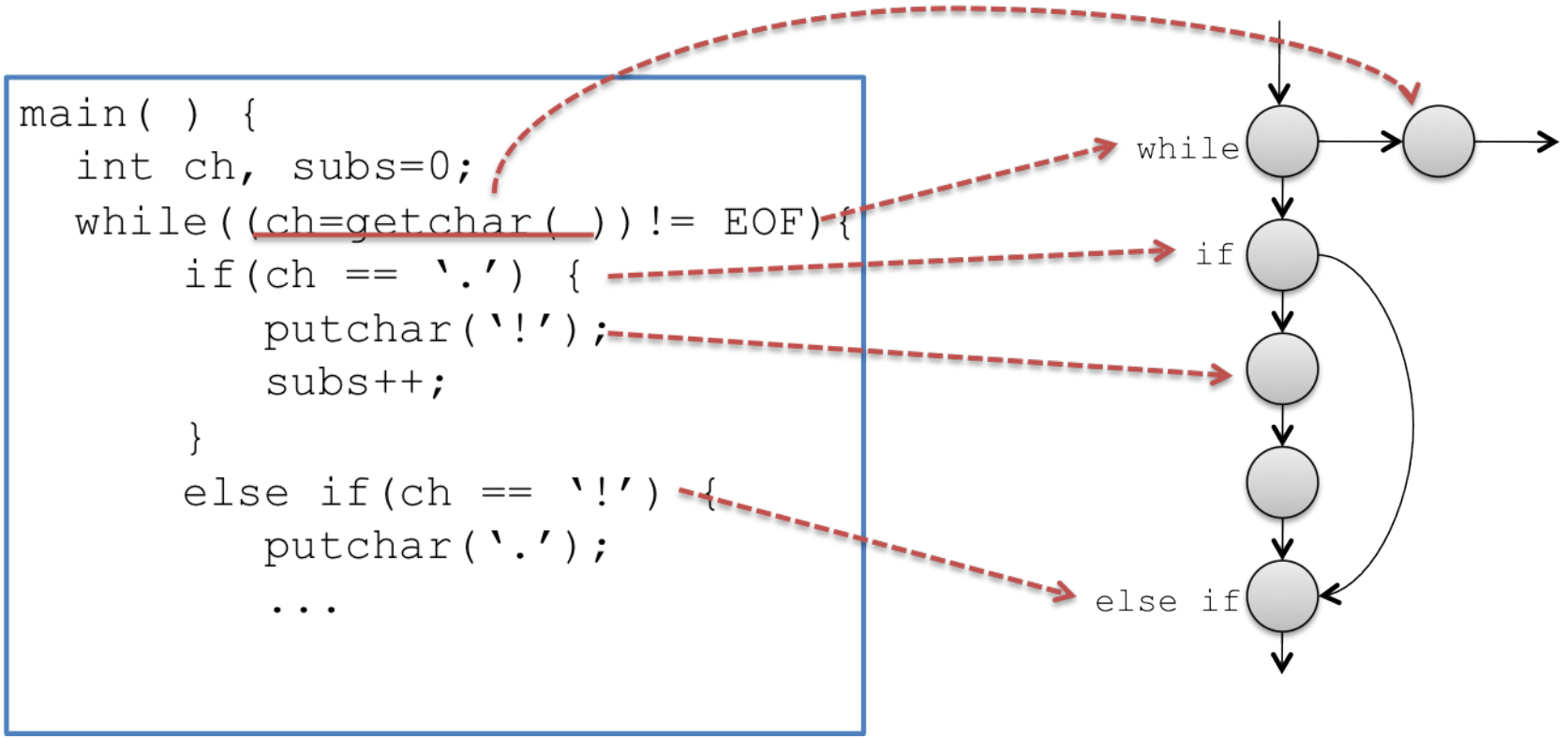
3. Code Based Test

- For effective code based test
  - limit the number of path to examine since it is difficult to examine all execution paths of a program
  - as criteria for selecting execution paths to be examined, use program statements, decision condition, ...
    - **Statement test**
    - **Decision test**
    - Condition test (a type of decision test)
    - **Path test**
    - Data flow test (a type of a path test)

lab(se);

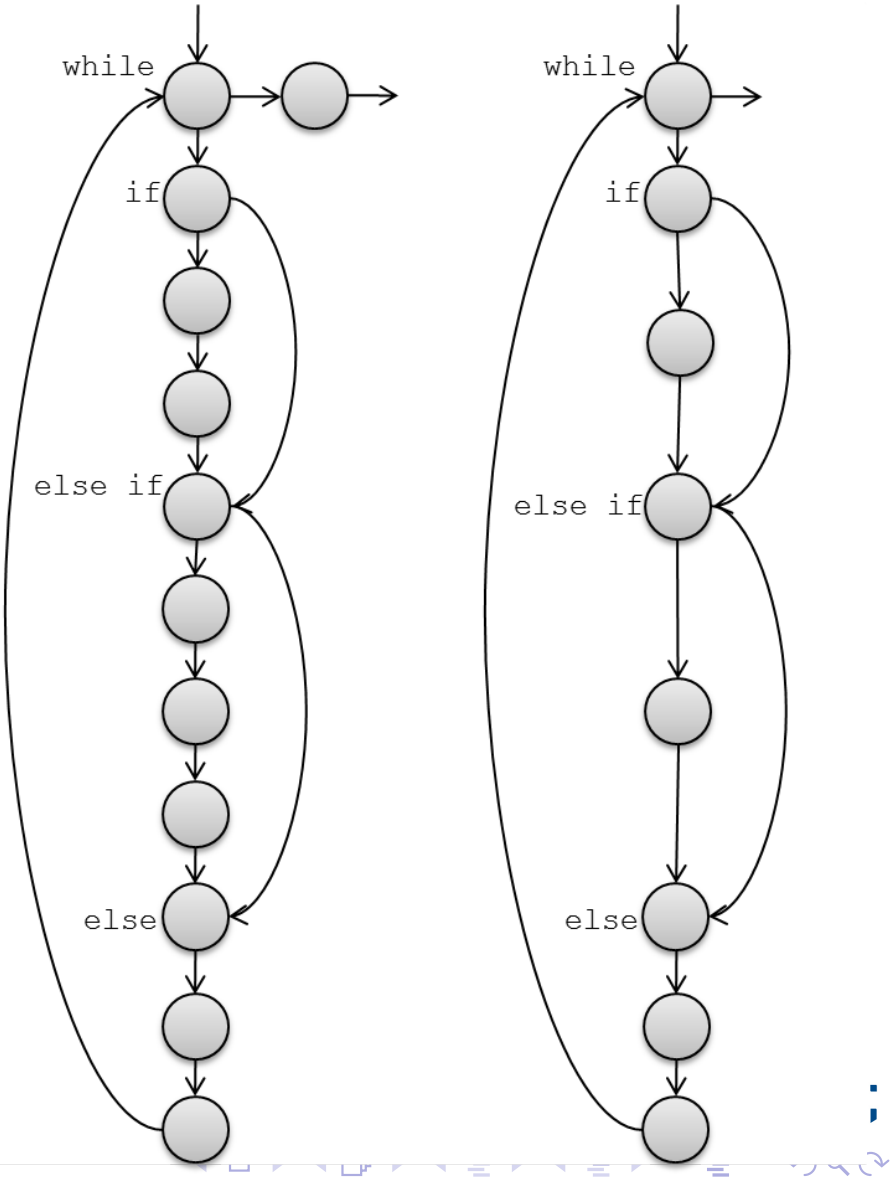
3.1 Control Flow Graph

- Directed graph
  - nodes : executable program statements
  - edges : the flow of control between statements



3.1 Control Flow Graph

```
main( ) {
   int ch, subs=0;
   while((ch=getchar( ))!=EOF){
        if(ch == '.') {
            putchar('!');
            subs++;
        }
        else if(ch == '!'){
            putchar('.');
            putchar('!');
            subs++;
       else
            putchar(ch);
       print("%d sub.\n",subs);
   }
}
```



3.2 Statement Test

- Each statement in a program is tested at least once in any one of test cases
  - In a control flow graph, sentence test is criteria that cover all nodes
  - In the unit testing standard of IEEE, sentence testing is defined as the minimum standard of testing

```
  main( ) {
     int x, y, z;
1)    z=1;
2)    while(y!=0)   {
3)       if(y%2 != 0) z=z*x;
4)       y = y/2;
5)       x = x*x;
     }
6)   print("x**y=%5d",z);
  }
```

Test Data		Program Sentence					
x	y	1	2	3	4	5	6
5	0	O	O	X	X	X	O
5	2	O	O	X	O	O	O
5	3	O	O	O	O	O	O

3.3 Decision Test

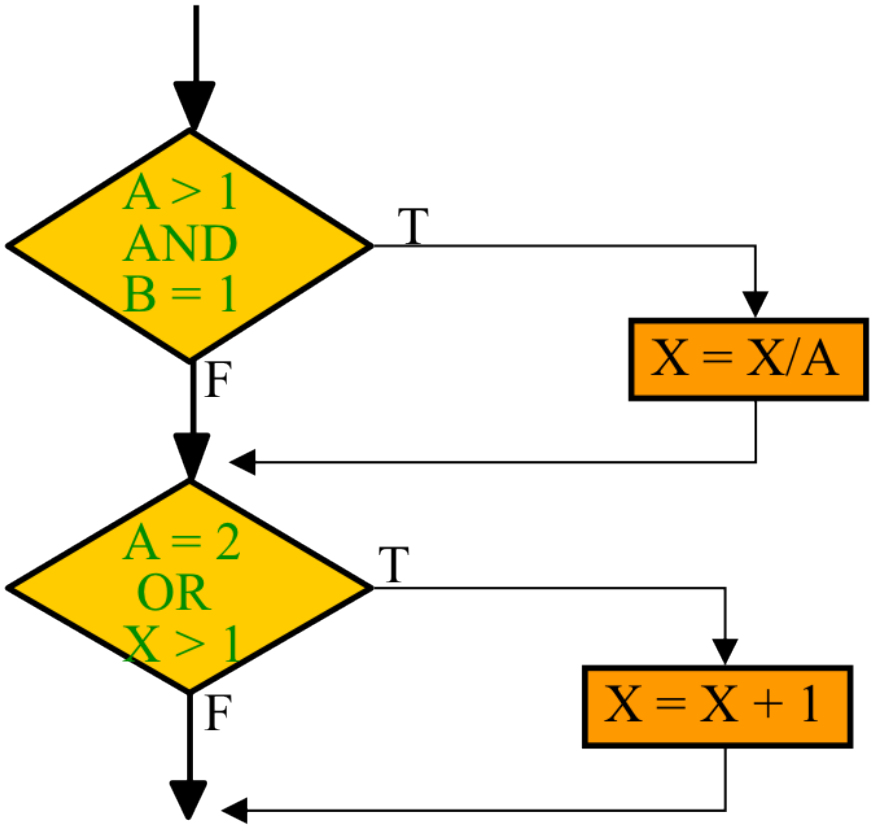
- Depending on a test case, sentence test may not be able to reveal the difference between two different programs

`if(c)`
`  S1;`
`S2;`
	`if(c) {`
	`  S1;`
	`  S2;`
	`}`

- Decision test can cover a significant part of the sentence test's weakness
- Decision test : every decision has to be executed at least once

lab(se);

3.3 Decision Test



Test Case 1 (Path TT)

T: A > 1 and B = 1
(A = 2, B = 1, any X)
T: A = 2 or X > 1 (true)

Test Case 2 (Path FF)

F: A > 1 and B = 1
(A = 1, X = 1)
F: A = 2 or X > 1 (false)

Test Case 3 (Path TF)

T: A > 1 and B = 1
(A = 2, B = 1, X = 1)
F: A = 2 or X > 1 (false)

Test Case 4 (Path FT)

F: A > 1 and B = 1
(A = 1, X = 2)
T: A = 2 or X > 1 (true)

lab(se);

3.4 Path Test

- Problems with path test effectiveness : number of path increases significantly as the size of software increases
  - even identifying possible paths become difficult
  - complete path test is impossible if loop is included
  - need adequate coverage and ways to reduce the number of paths

반복문의 경우 테스트가 어려워 만든거

- Independent path : if a path contains a partial path that is not covered before, it is said to be independent from other path
  - can check and examine all independent paths

lab(se);

3.4.1 Independent Execution Path

- McCabe's program complexity
  - high complexity increases the possibility of mistakes in the implementation process
  - test effort must be increased proportionally to the complexity
  - proposes cyclomatic number v(G) as a quantitative measure of mplexity

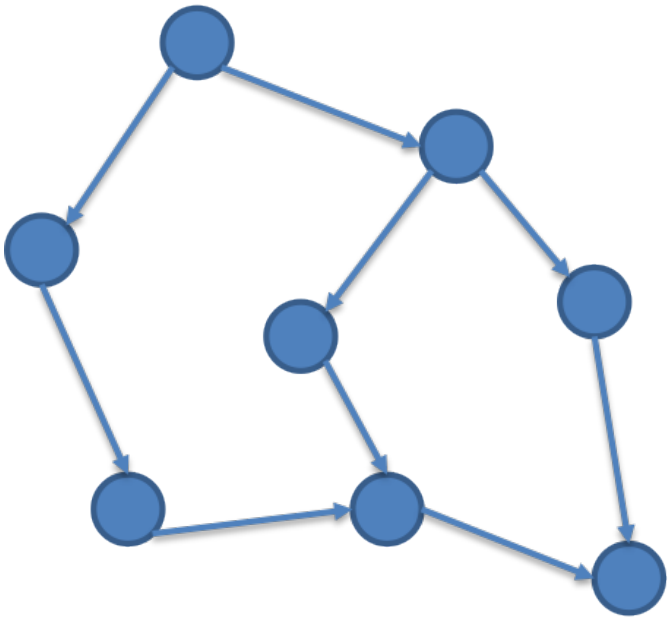
    v(G) = e - n + 2

    - e: number of edges, n: number of nodes (excludes edges that corresponds to entrance and exit to a program)
- McCabe proved v(G) = the number of independent execution path

lab(se);

3.4.1 Independent Execution Path

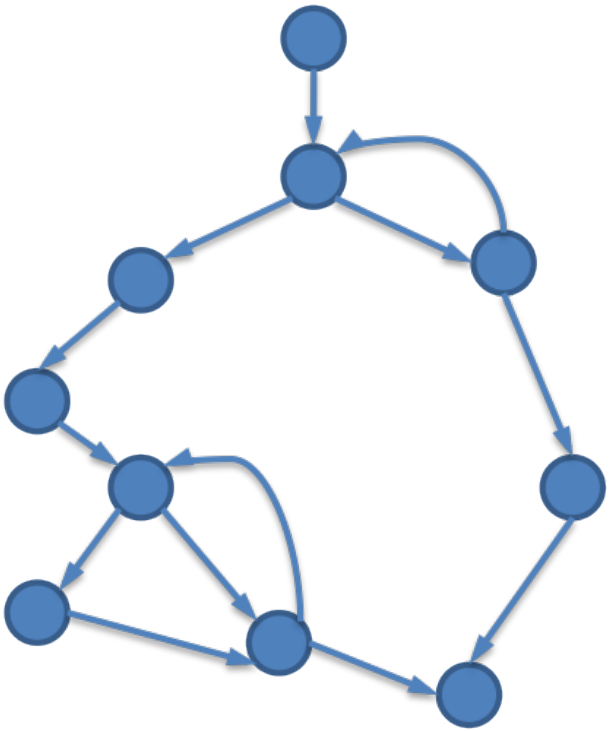
Calculatin cyclomatic number



Control Flow Graph: G1

$$v(G1) = e - n + 2$$
$$= 9 - 8 + 2$$
$$= 3$$

➔ independent path = 3



Control Flow Graph G2

$$v(G2) = e - n + 2$$
$$= 13 - 10 + 2$$
$$= 5$$

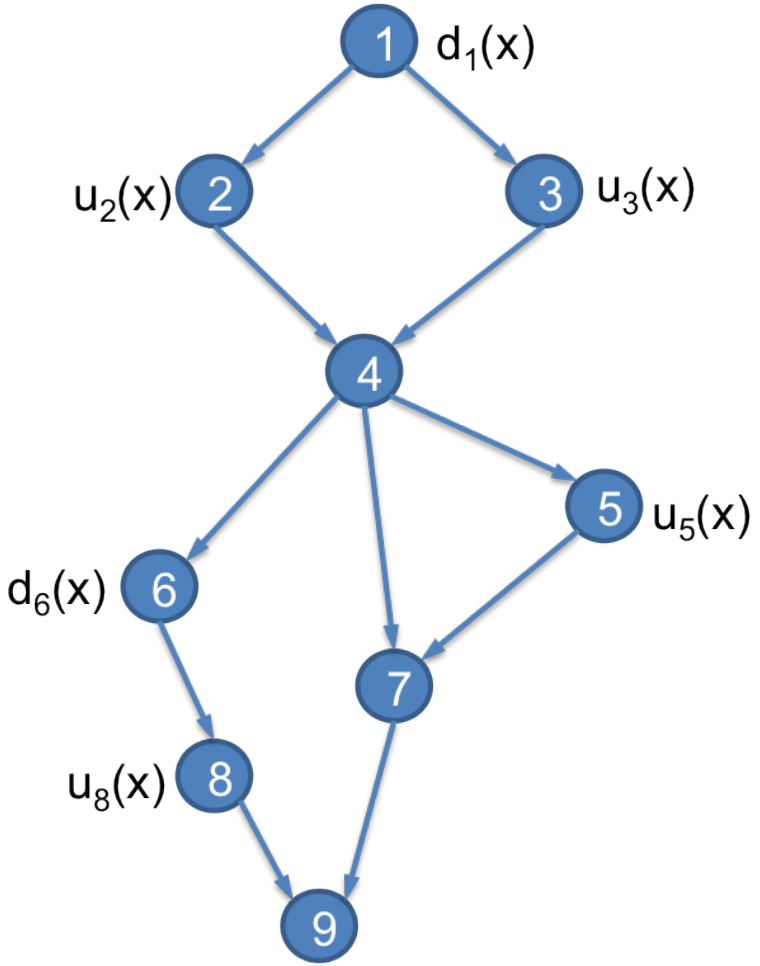
➔ Independent path = 5

lab(se);

3.4.2 Data Flow Test

- Examine what happen to data elements in the program during execution
  - variable is defined : when a variable is initialized or when a value is stored in a variable by assignment, read, input, ... statements
  - variable is used : when the value of a variable is used to determine some treatment of IF statement ... or used in calculating other variable or to be outputted
- Definition of data : test method of detecting defects by finding execution path where problems within the usage relation can be looked at

3.4.2 Data Flow Test



Location of data definition and use

- Execution path 1-2-4-6-8-9 which consists of the locations of definition 1 and 6
- Test the path that passes 2, 3, 5, 8 at least once to examine every locations of use
  - 1-2-4-6-8-9 & 1-3-4-5-7-9
- Path that contains definition-use relation
  - 1-2-4-6-8-9
  - 1-3-4-6-8-9
  - 1-2-4-5-7-9
  - 1-3-4-5-7-9

lab(se);

3.4.3 Test Coverage

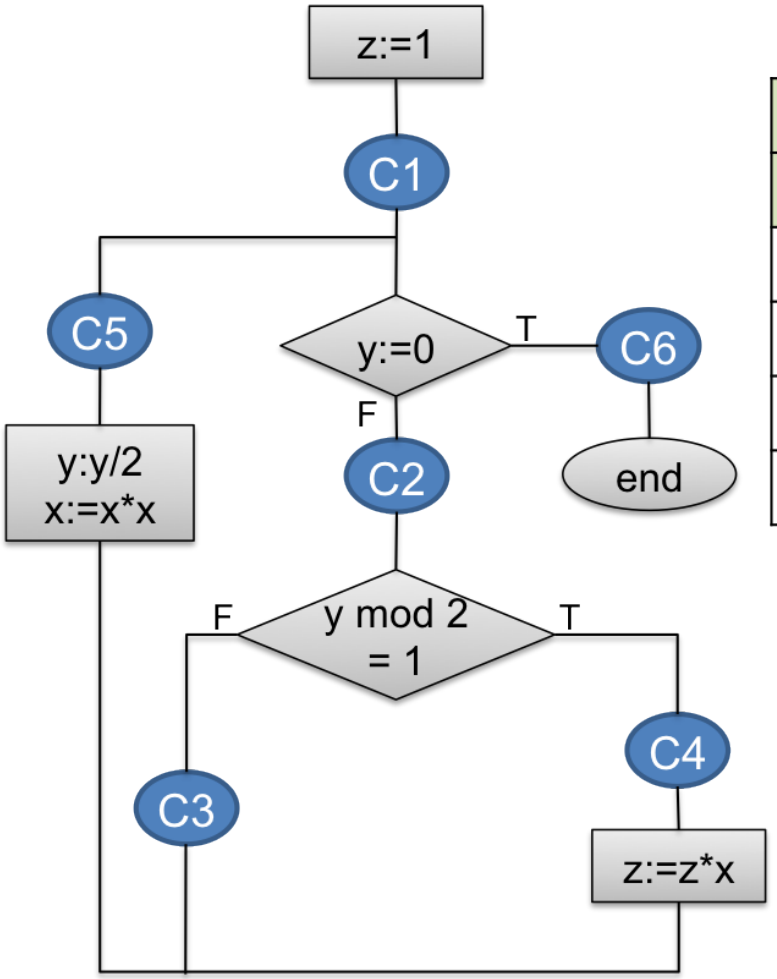
- Test is examining of a code as thorough as possible from diverse point of view
  - better to have as few test cases as possible
  - there should be no overlap between test cases
  - minimize the part missing from the test

  $\rightarrow$ need to measure which part is covered and which is not
  $\Rightarrow$ Test Coverage

- Instrumentation
  - insert a probe at the strategic points on a program path
  - probe is a sentence added to a program that provides simple functionality of indicating whether a path is executed or not

lab(se);

3.4.3 Test Coverage



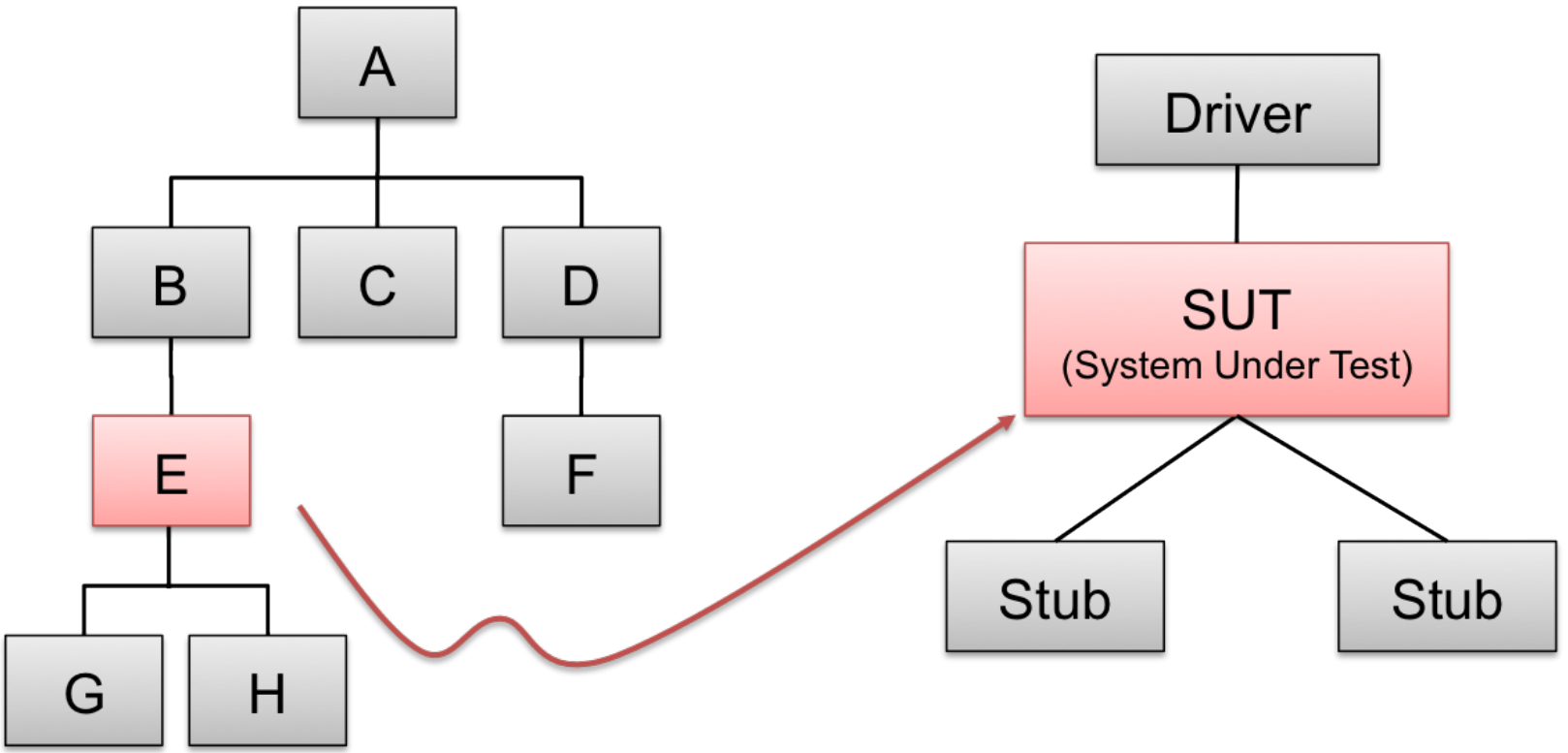
Test cases		Counter values					
x	y	C1	C2	C3	C4	C5	C6
10	0	1	0	0	0	0	1
20	1	1	1	0	1	1	1
5	2	1	2	1	1	2	1
40	4	1	3	2	1	3	1

path containing the probe C1 is executed 1 time

lab(se);

4. Unit Test & Integration Test

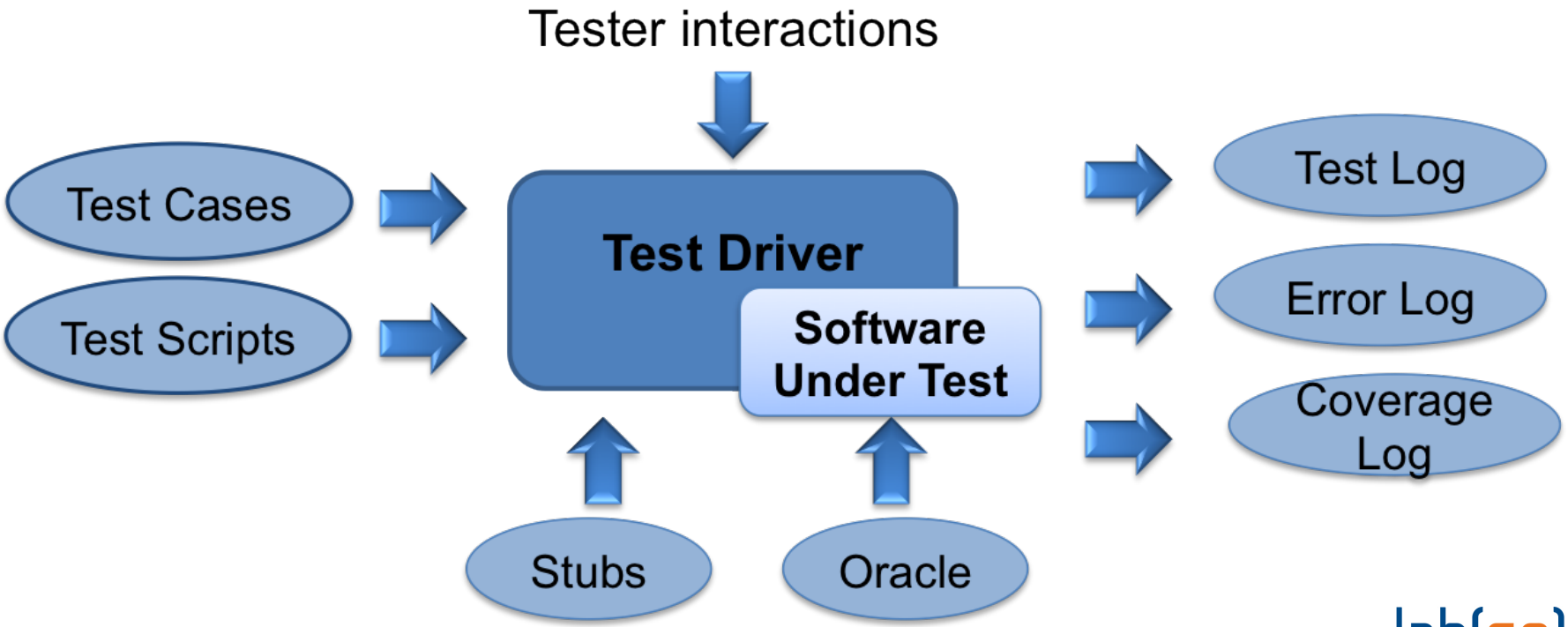
- The purpose of unit test : checks each part that that make up a system (module, function, ...) is implemented as specified in the detailed design specification



lab(se);

4. Unit Test & Integration Test

- Unit test driver
  - setup required for test, execution of test cases, evaluate test results
  - automated tool : Codescroll, Junit Java, CppUnit C, C++, etc



lab(se);

4. Unit Test & Integration Test

- Integration test is to verify interfaces and interactions between unit tested system components
- Big Bang Integration
  - Integration is done after implementing all the components so the procedure is simple but the start of the test is delayed and difficult to debug
- Incremental Integration
  - Top-down, bottom-up, build-based integration

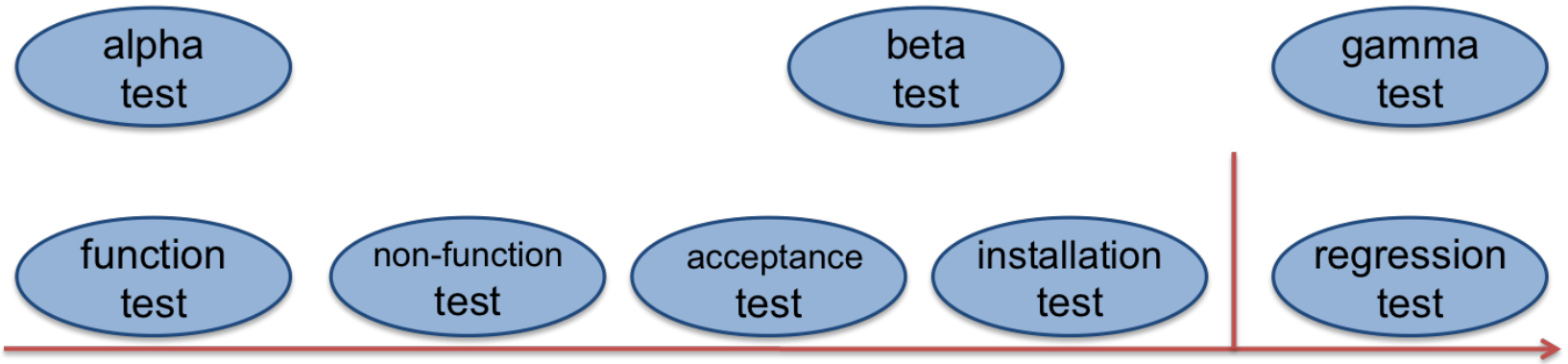
4. Unit Test & Integration Test

- Top-down
  - since sub-modules are replaced with stubs, although process is difficult to check but operation status can be determined
- Bottom-up
  - system cannot be operational until the integration of top-level units are complete
  - since sub-modules are weaved first, suits situation where precise calculation or data processing
- Build-based integration
  - since build is set of modules that are functionally related within the system, integration is conducted after the unit tests of modules that belongs to the same build

lab(se);

5. System Level Test

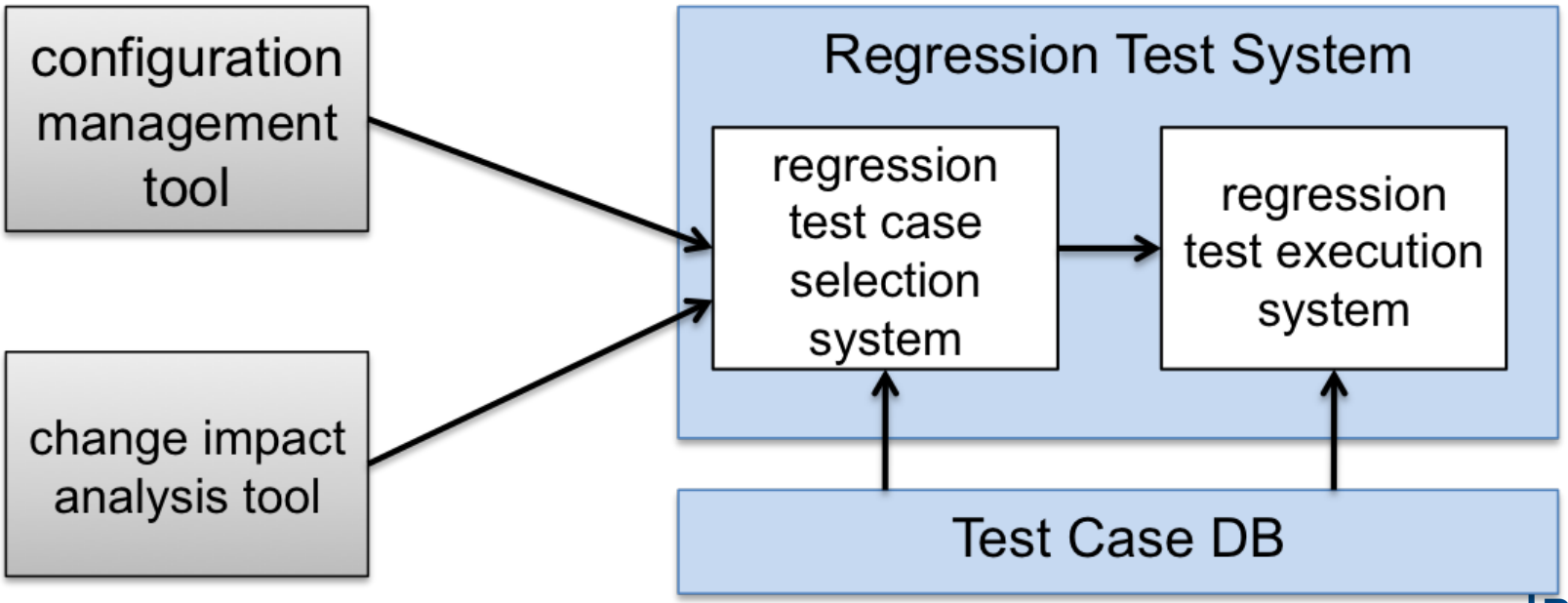
- Functional test
  - examines whether functions specified in the software specification is correctly implemented
- Non-functional test
  - examines whether a system equipped with all functionality satisfies system properties specified in the requirements



lab(se);

6. Regression Test

- To determine whether software modification has been made correctly and whether modification has any bad affect on other parts of the software
  - demonstrate that there is no problem with modified part and there is no regressed part in system functionalities



lab(se);

7. Object-Oriented Software Testing

- Problems of object-oriented system in the testing point of view
  - Non-procedural : sentence order $\neq$ execution order
  - No obvious structure to be referred during integration
  - Components are diverse compared to procedural system
  - Inheritance, encapsulation, dynamic binding, polymorphism, ... makes dependencies between components relatively complex
  - procedural modules have same characteristics as functions, but object-oriented modules are related to states
- Testing dimension
  - methods, class, class clusters, system

lab(se);

7. Object-Oriented Software Testing

- Class testing

  After individually testing methods in the class, check the correctness of connection between methods and check the correctness of relationship between methods through data members

  - for testing errors caused by interactions between methods, every possible combination of methods must be checked

    e.g., if there are 11 methods, 11! sequences must be tested
    $\Rightarrow$ ineffective

- Class testing techniques
  - slicing technique
  - MM path (Method/Message path) technique
  - state-based testing technique

lab(se);