Unit 8.
Software Testing

Contents

- Testing Concepts
- Unit Testing
- Integration Testing
- Object-Oriented Testing
- Black-box Testing
- White-box Testing

Objective of Testing。

- Testing is a process of executing a program with the specific intent of finding errors.

  테스트는 반드시 소스코드로 진행해야한다
  ›› 디자인같은경우 실행 할 수 없기 때문에안됨

- A good test case is one that has a high probability of finding as yet undiscovered error.

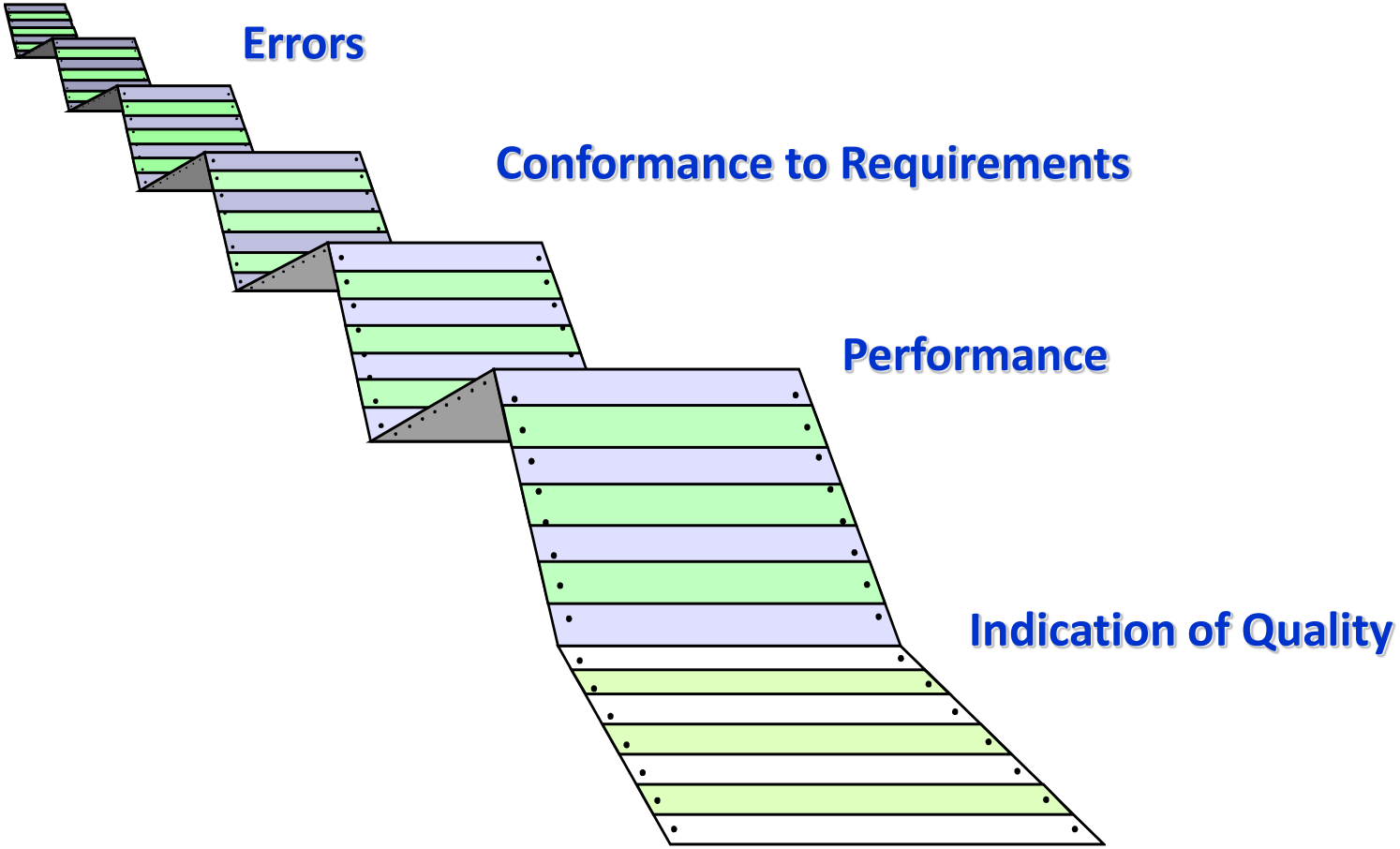
  시간은 언제나 한정되어있기 때문에, 한번의 좋은 테스트를 통해 아직 찾지못한 에러를 발견하는것이 확률이 더 높음

- A successful test is one that uncovers an as yet undiscovered error.

- Testing cannot show the absence of defects, it can only show that software defects are present.

테스트를 하는 이유는 defect(결함)을 찾기위해서다

What Testing Shows



Who should test the Software?

Developer

Understands the system, but will test "gently", and testing is driven by "delivery"

Independent Tester

Must learn about the system, but will attempt to break it, and testing is driven by quality

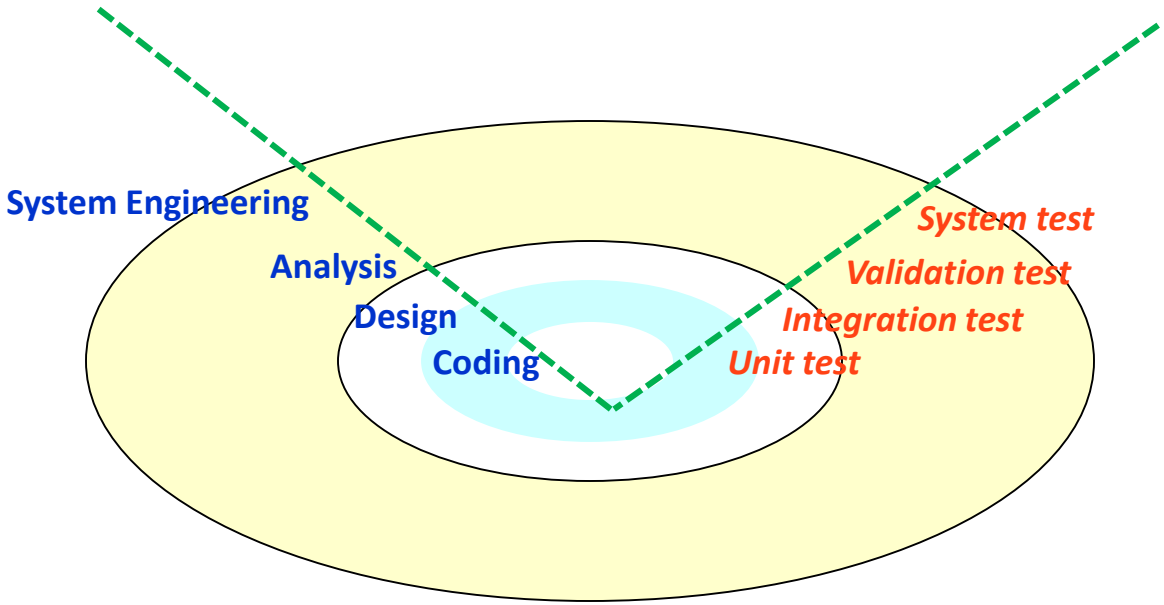
의도적으로 시스템이 멈추도록 테스트케이스를 넣는등
>>예외적인 상황을 테스트케이스로 많이 넣어줌

다음시간부터 이걸 테스트하는 방법들에 대해 배울것임

Verification and Validation (V&V)

- **Software Verification**
  - "Are we building the product right ?"
  - checks that the program conforms to its specification.
- **Software Validation**
  - "Are we building the right product ?"
  - checks that the program as implemented meets the expectations of the clients.

'V' Model



Unit Testing으로

어떠한 모듈도 유닛이 될 수 있다

- **Concept**
  - Unit testing focuses *verification* effort on the smallest unit of software design - the module.

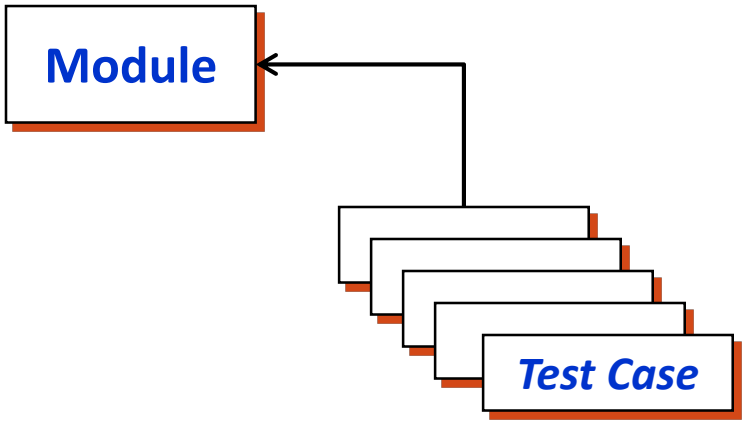
    function이 될 수 도있고, class가 될 수 도있다
  - Uses *detailed design* as a guide, control paths are tested within the boundary of the module.
  - After code has been developed and reviewed, unit test case design begins.
  - *White-box* oriented

Unit Testing

- **What to test**
  - Module Interface
  - Local Data Structure
  - Boundary Conditions
  - Independent Paths
  - Error-Handling Paths

하나의 모듈을 테스트할때, 여러개의 testCase필요

예상되지 않는 인풋을넣어서
시스템이 어떻게 동작하는지 확인해야한다



Unit Testing

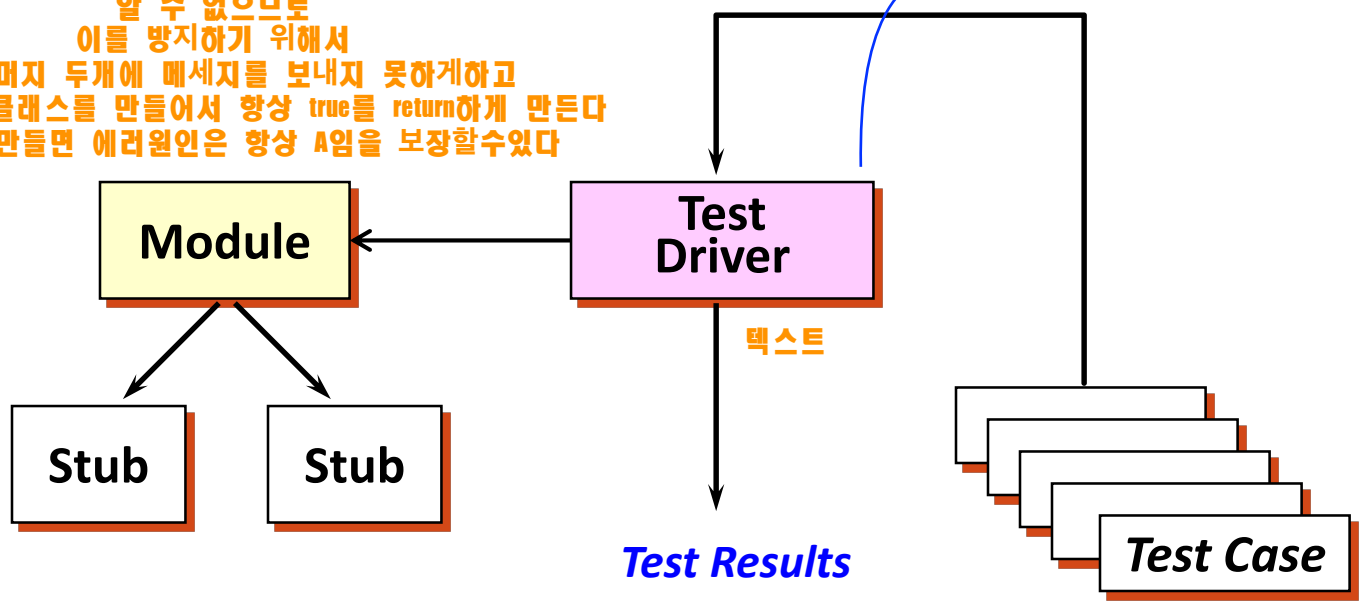
- **Test Driver**
  - A main program that accepts test case data, passes data to the module, and prints the results.
- **Stub**
  - A stub serves to replace a module that is subordinate (called by) the module to be tested.
  - A stub uses the subordinate module's interface, may do minimal data manipulation, prints verification of entry, and returns.

Unit Testing

Unit Testing Environment

원래 A,B,C라는 클래스가 있었을때,
A가 나머지 두개에게 메세지를 날린다면,
유닛하나에 집중한다는 원칙을 깨게된다
이 경우, 어느 클래스가 에러를 발생시켰는지
알 수 없으므로
이를 방지하기 위해서
A가 나머지 두개에 메세지를 보내지 못하게하고
Stub라는 임시클래스를 만들어서 항상 true를 return하게 만든다
>>이렇게 만들면 에러원인은 항상 A임을 보장할수있다

main method와 같은역할,
java에서 test class만드는것과 같음
>>메인에서 다른클래스 객체만들어서 값 출력해보는것처럼



Module Interface
Local Data Structure
Boundary Conditions
Independent Paths
Error-Handling Paths

Unit Testing

응집도, 응집성

- **Unit testing is simplified when a module has a high cohesion.**
  - **Drivers and stubs represent overhead.**
  - **When only one function is addressed by a module, the number of test cases is reduced and errors can be more easily predicted.**

응집도가 높다 〉〉 다른 클래스에 의존도가 낮다.
A라는 클래스가 아무런 다른클래스 필요없이 독자적으로
실행될 수 있다면 〉〉 cohesiveness가 매우높음!

cohesiveness가 높을경우
〉〉테스트할때 stub만들필요없이 바로 할 수 있으니
매우편리함

Unit testing이 끝난뒤 통합테스트(Integration testing)을 진행한다

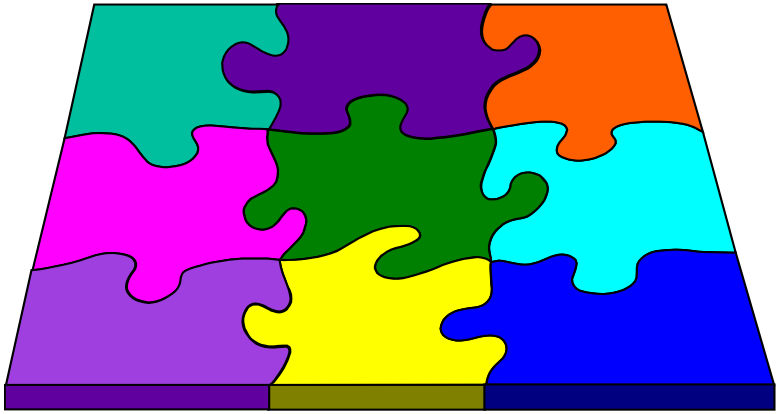
Integration Testing

각각의 class가 멀쩡하다고 해도
interface에 문제가있어서 합쳤을때 문제가 발생 할 수 있으므로
>>통합테스트를 진행해야한다.

Concept

- Once all modules have been unit-tested, why test more ?
  - There might be problems on putting them together - interfacing.
- Integration testing is a systematic technique
  - for constructing the whole program structure while at the same time testing the interface.

목적
1.모든 모듈을 하나의 시스템에 통합한다.
2.interface에 문제가 존재하는지 찾는다.



Integration Testing

- **The "big bang" approach**
- **Incremental Integration**
  - **The program is constructed and tested in small segments, where errors are easier to isolate and correct.**
    - **Top-Down Integration**
    - **Bottom-Up Integration**

Integration Testing

- **Top-Down Testing**
  - Modules are integrated by moving downward through the control hierarchy, beginning with the main control module.
  - Modules subordinate (and ultimately subordinate) to the main control module are incorporated into the structure
    - in either a depth-first or breadth-first manner.
- **Bottom-Up Testing**
  - Begins construction and testing with atomic modules.
  - Modules are integrated from the bottom up.

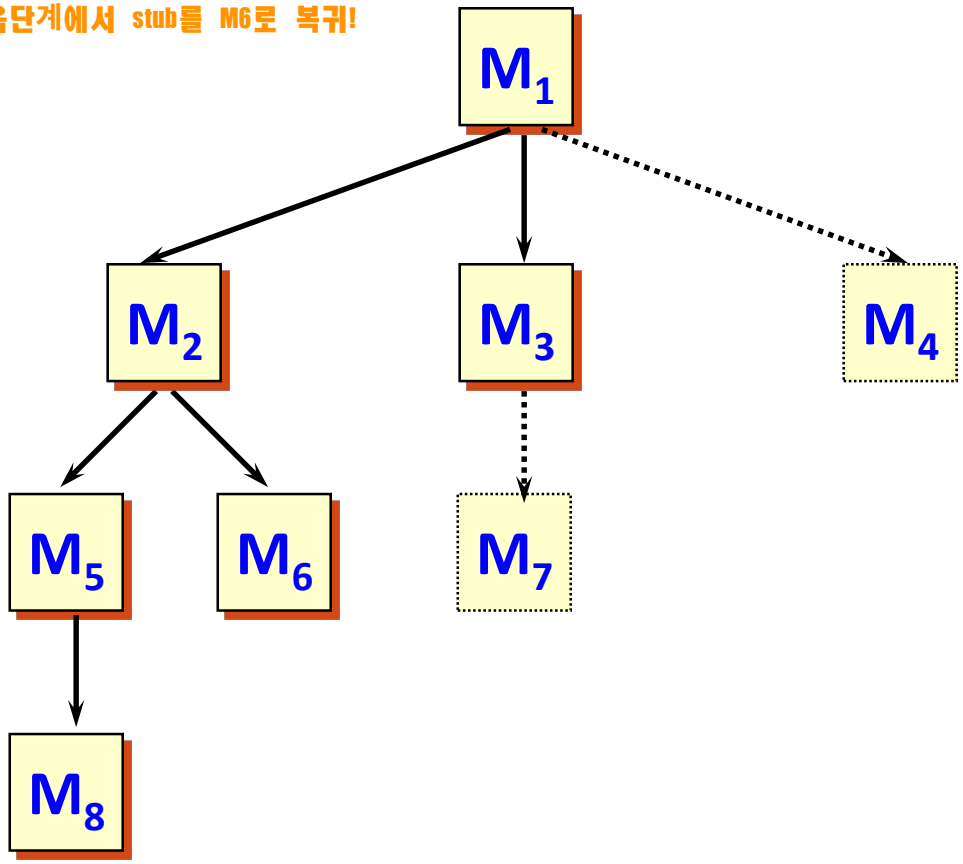
Integration Testing

- **Top-Down Testing with Depth-First**
  - M1, M2, M5, M8 DFS로 포함
  - M6
  - M3, M7
  - M4

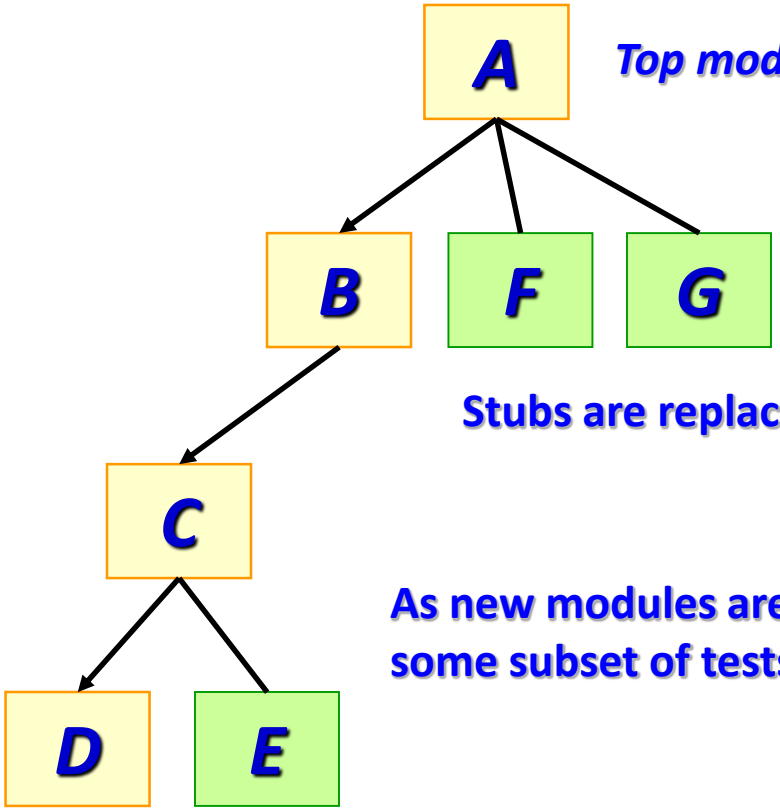
그럼 dfs가 되었든, bfs가 되었든 시작할때
stub자리를 모두포함하면 모든 클래스가 포함된것?

이녀석을 테스트 할 때, M6는 아직 포함되지 않았음
>>but, M2는 M6를 필요로 하므로 M6를 stub로 대체해서
진행 후 다음단계에서 stub를 M6로 복귀!

- **Top-Down Testing with Breath-First** BFS로 포함
  - M1
  - M2, M3, M4
  - M5, M6, M7
  - M8



Top Down Integration

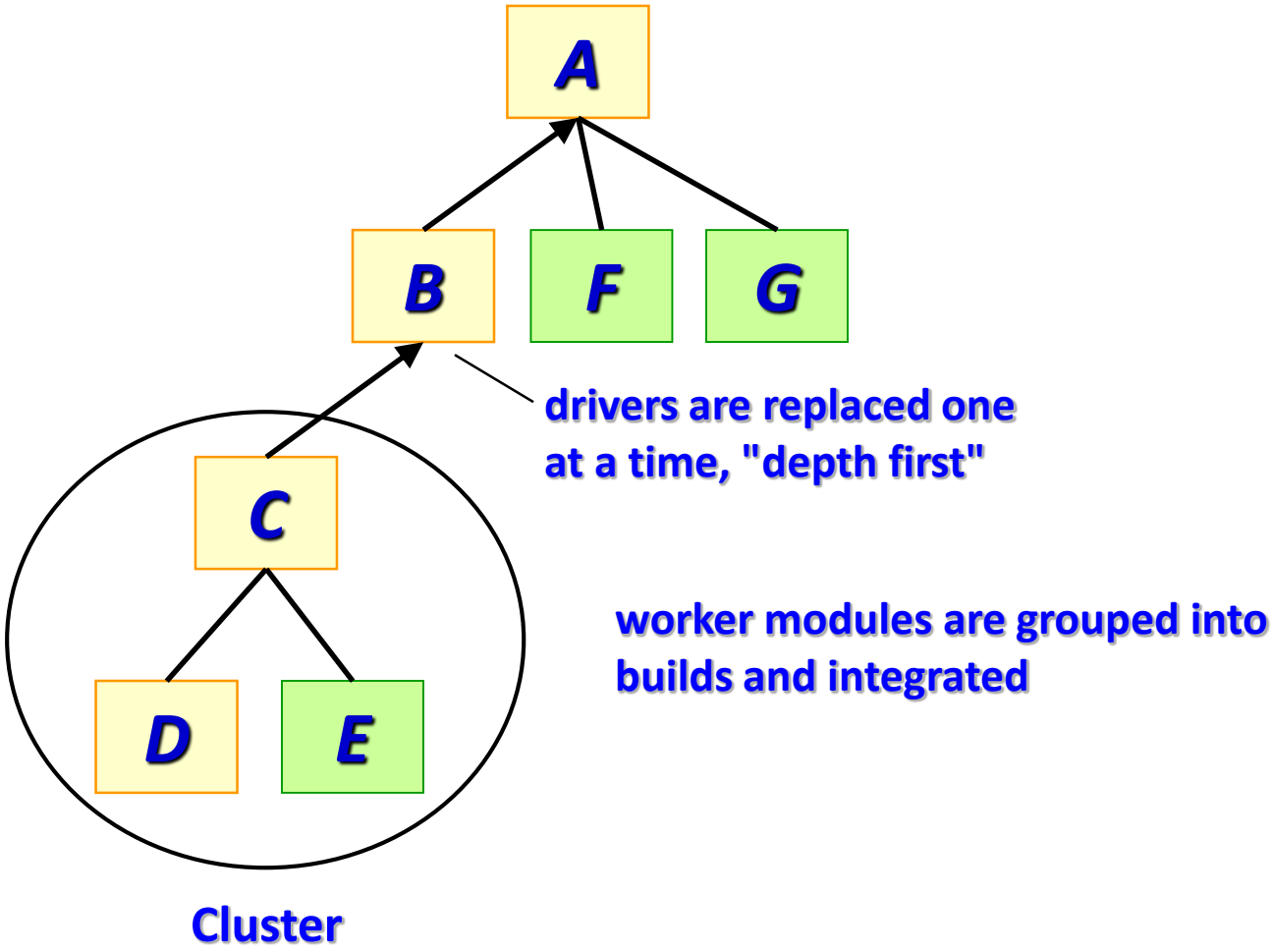


Top module is tested with stubs.

Stubs are replaced one at a time, "depth first"

As new modules are integrated,
some subset of tests is re-run.

Bottom-Up Integration



High Order Testing

- **Validation testing**
  - Focus is on software requirements
- **System testing**
  - Focus is on system integration
- **Alpha/Beta testing**
  - Focus is on customer usage
- **Recovery testing**
  - Forces the software to fail in a variety of ways and verifies that recovery is properly performed
- **Security testing**
  - Verifies that protection mechanisms built into a system will, in fact, protect it from improper penetration
- **Stress testing**
  - Executes a system in a manner that demands resources in abnormal quantity, frequency, or volume
- **Performance Testing**
  - Test the run-time performance of software within the context of an integrated system

Alpha and Beta Testings

- **Alpha Testing**
  - Conducted at the developer's site by a customer.

클라이언트 & 커스터머가 만족도, 승인을 보내 줄 수 있음(그정도의 역할이 다임)

클라이언트가 개발자옆으로와서 이게 맞게 개발한건지 확인하는 용도

- **Beta Testing**
  - Conducted at one or more customer sites by the end-users.
  - Developers generally are not present.

일반유저가 사용해보면서 테스트를 직접 해보는것!
>>개발환경과 실제사용환경이 서로 다를 수 있음
보통 개발환경이 훨씬 풍부한 자원을 가지고있음
>>다양한 환경에서 테스트 해볼 수 있는 기회가 생긴다는게 장점이다.

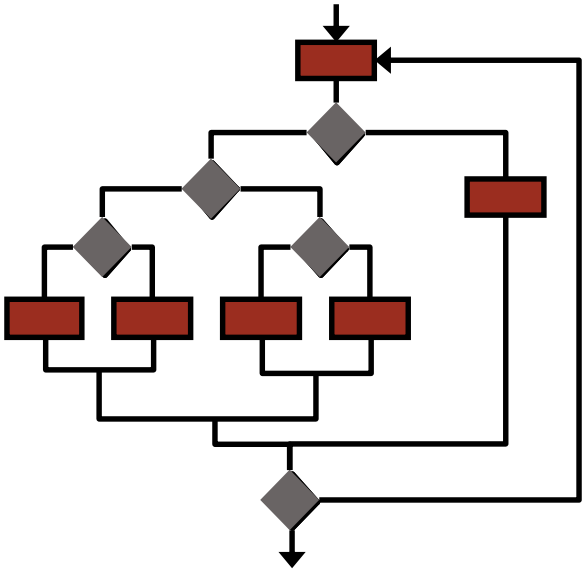
개발자는 이 테스트에 관여하지 않는다는 뜻임

Exhaustive Testing

- **Exhaustive Testing is impossible.**

멍청하게 모든경우의 수를 다 돌려보는 것

  - Testing by executing every statement and every possible path is impossible in practice.
  - Therefore, testing must be based on a subset of possible test cases.

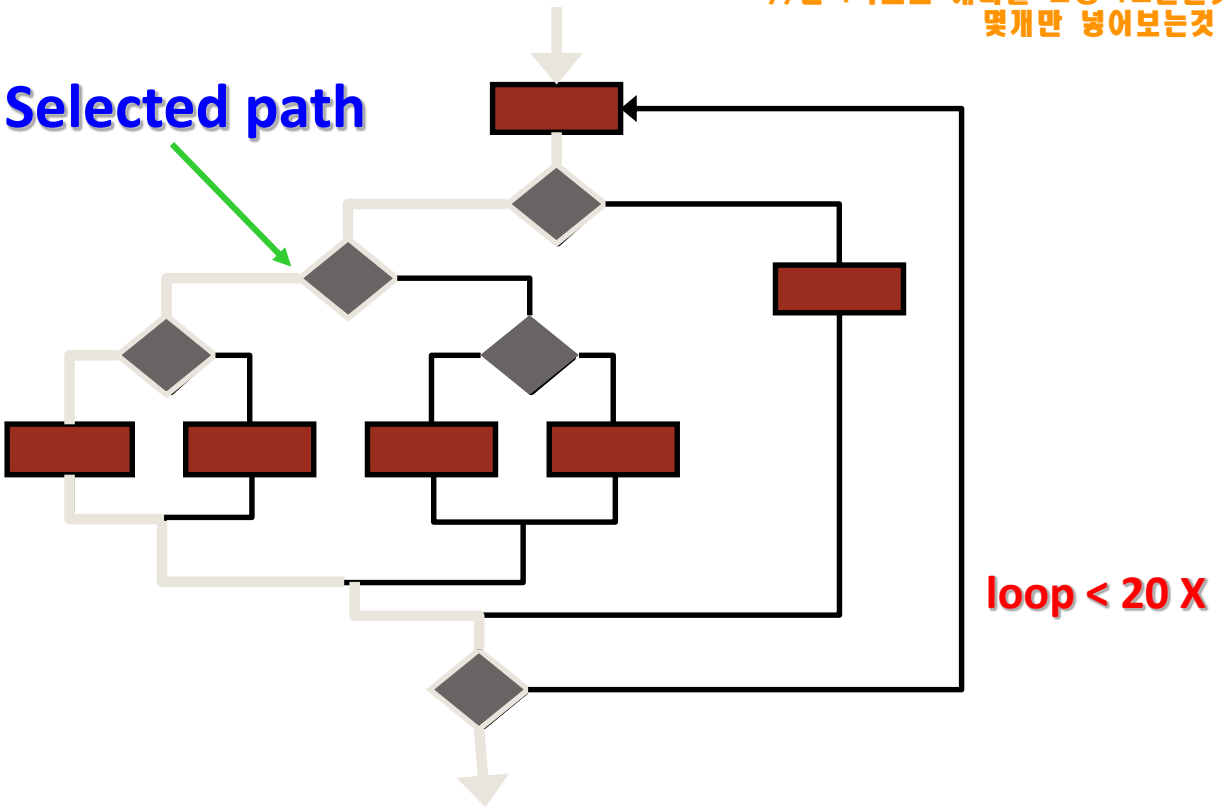


loop < 20 X1

  - There are 1014 possible paths!
    If we execute one test per millisecond,
    it would take 3,170 years to test this program!!

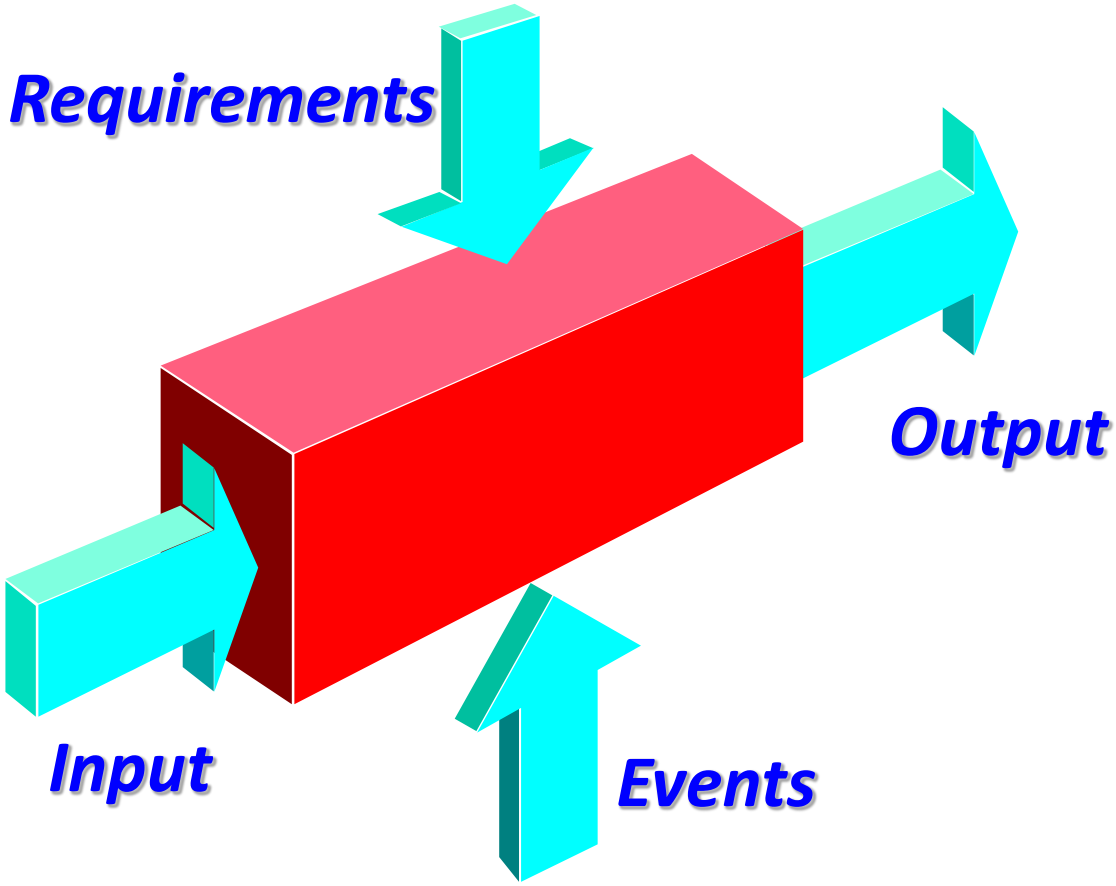
Selective Testing

멍처하게 모든 경우의 수를 다 해볼 수 없으니까
>>논리적으로 에러를 발생시킬만한것을 미리 정해서
몇개만 넣어보는것



Black-Box Testing

보통 전형적인 프로젝트에서 시간할애
40 - 20 - 40
디자인 - 개발 - 테스트
>>요즘은
50 - 20 - 30
도 많이씀
>>디자인이 좋을경우 더 좋은
결과물이 나오기 때문에!



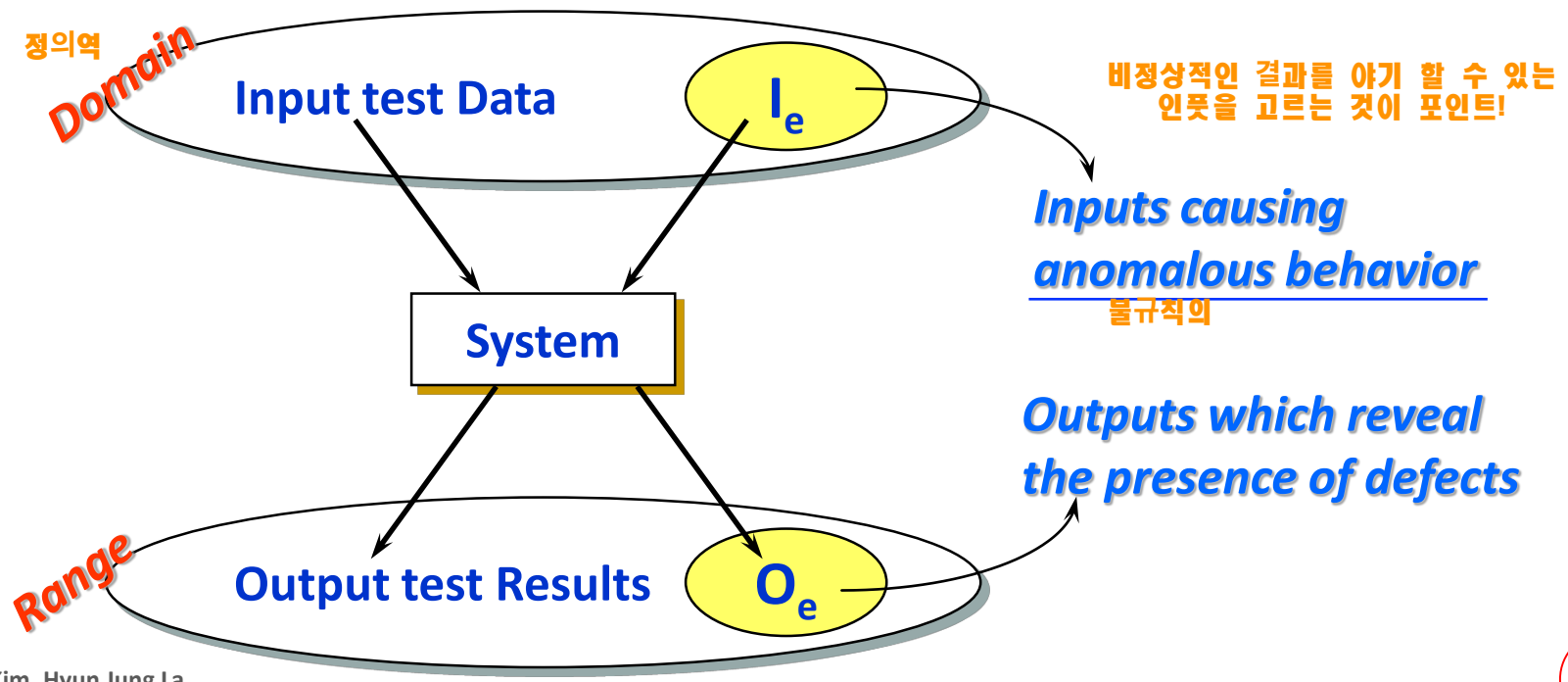
Black-Box Testing

- **Concept**
  - **It relies on the specification of the system or component which is being tested.**
  - **The system is blackbox whose behavior can only be determined by studying its inputs and the related outputs.**
  - **It is also called *functional* testing.**

Black-Box Testing

- **Key Problem**
  - To select inputs that have a high probability of being members of the set Ie.
  - In many cases, use the previous experience and domain knowledge to identify test cases.

정상적이지 않은 인풋을 넣어서 확인하는 것
〉〉다음시간부터 이런 인풋을 정하는 방법을 알려주실것임



Black-Box Testing

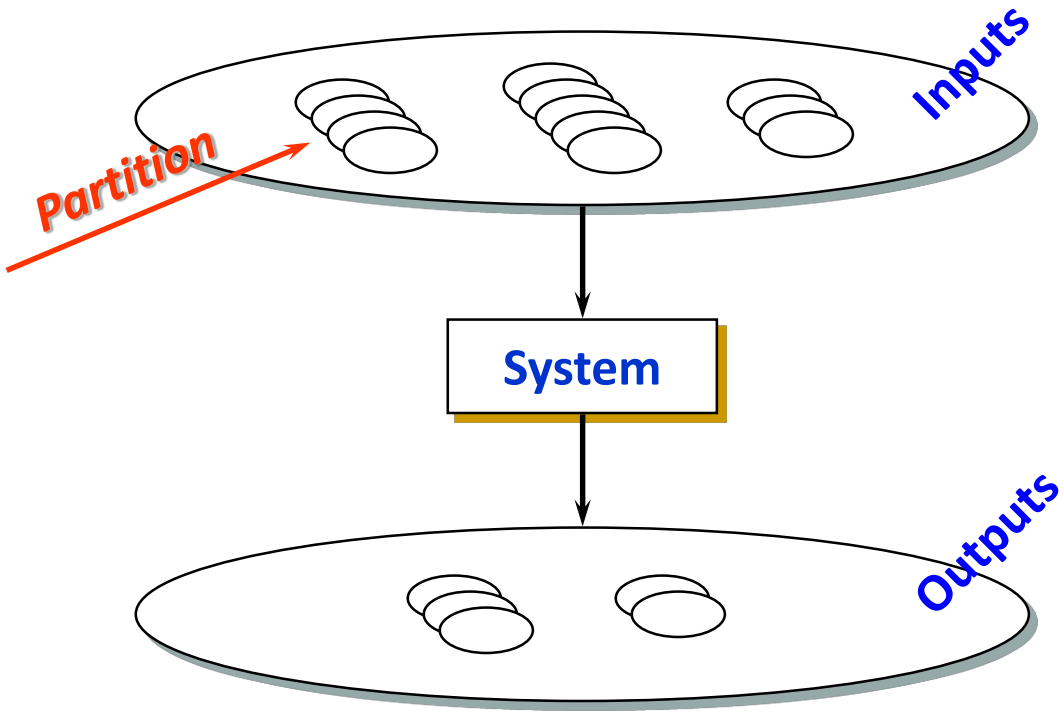
균등분할

- **Equivalence Partitioning**
  - Input data to a program usually fall into a number of different classes or partitions.
  - These classes have common characteristics.
    - Positive Numbers, Negative Numbers, Strings without blanks
  - Identify a set of these equivalence partitions which must be handled by a program.

정의역으로 넣어야하는 집합을
특성으로 나누는 것
int 〉 양수, 음수, 0
string 〉 null, 1문자스트링, 2문자이상스트링

Black-Box Testing

하나의 partition에 대해서
적어도 3개의 case가 존재하게 될 것이다
(경계값 2개, 중간값 1개)



Black-Box Testing

- **Test Cases for Each Partition**
  - Choose particular test cases from each partition.
  - Choose test cases on the boundaries of the partitions and test cases close to the mid-point of the partition.
    - Boundary values are often atypical and so they are overlooked.
    - Designers and programmers tend to consider typical values of inputs. And, these are tested by the mid-point cases.

중간값, 경계값을 찾아서 사용
>>대표적인 성격을 지니게 된다

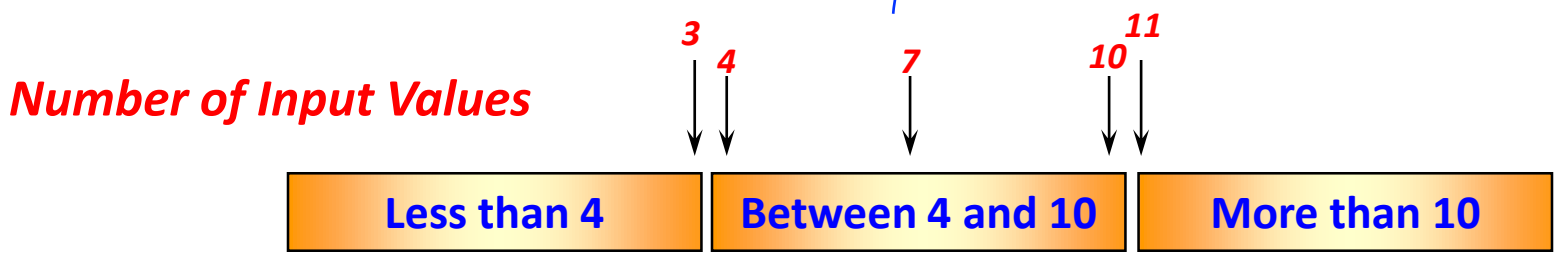
보통 중간값의 경우 handling이 된 경우가 많지만
대부분의 개발자는 경계값을 무시하는 경향이 있다

Black-Box Testing: Example

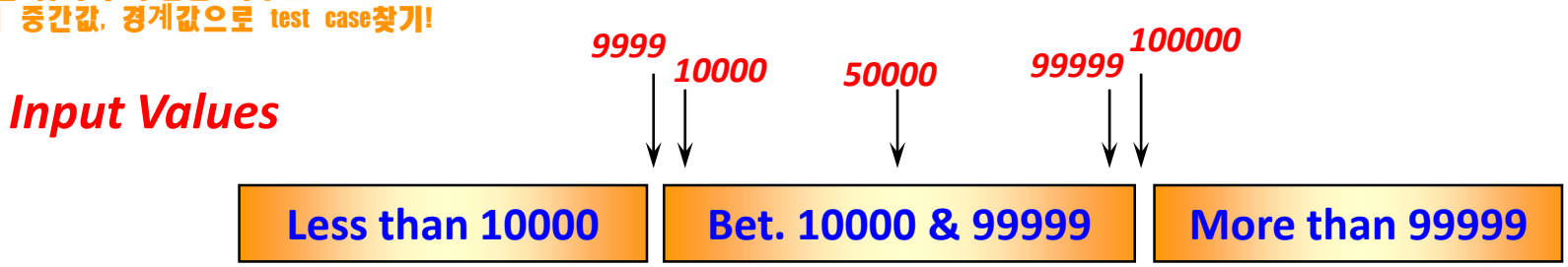
- **Program accepting 4 to 10 inputs which are 5 digit integers greater than 10000.**

criteria(결정조건)을 찾아야 한다(partitioning)에 대한

결정조건으로 3개의 partition으로 나누고
>>경계값과, 중간값을 찾아서 5개의 input을 찾았다



순서!
결정조건 찾아서 부분을 나누고
각 부분에 대해서 중간값, 경계값으로 test case찾기!



- **Test each partition with all instances of partitions in other classes.**

Black-Box Testing: Example

- **Test Cases**

	Test Case ID	n1	n2	n3
# of Inputs = 3	1	9,999	9,999	9,999
	2			10,000
	3			50,000
	4			99,999
	5			100,000
	6	9,999	10,000	9,999
	7			10,000
	8			50,000
	9			99,999
	10			100,000
	11	9,999	50,000	9,999
	12			10,000
	13			50,000
	14			99,999
	15			100,000
	16	9,999	99,999	9,999
	17			10,000
	18			50,000
	19			99,999
	20			100,000
	21	9,999	100,000	9,999
	22			10,000
	23			50,000
	24			99,999
	25			100,000
	26~50	10,000		
	51~75	50,000		
	76~100	99,999		
	101~125	100,000		
	is 5*5*5 = 125			
# of Inputs = 4				
	625			
# of Inputs = 7				
	78125			
# of Inputs = 10				
	9765625			
# of Inputs = 11				
	48828125			

위에서 알아낸 조건들에 의해서
test case를 만들경우
5^3 + 5^4 + 5^7 + 5^10 + 5^11
>>확실히 많이 줄어들었지만 아직 많다

testing coverage
>이렇게 input table을 만들었을때
이를 얼마나 실행 할 수 있는지 척도
이 지수가 높다는 것은 table에 있는 대부분의 경우를
test 해 볼 수 있다는 것을 의미함.

White-box Testing

- **Concept**
  - A complementary approach to blackbox testing, also called structural or glass-box testing.
  - Analyze the code and use the knowledge about the program structure to derive test data.

black box에션
내부가 어떻게 진행되는지 보지 못하고
인풋과 아웃풋만 확인하게 된다

이 경우는 직접우리가 내부를 볼 수 있다

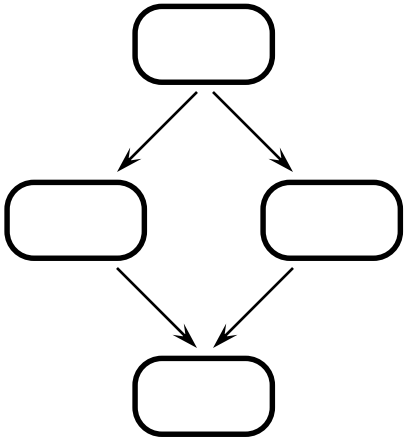
아웃풋에 에러가 없다고 할 지라도
내부적으로 문제가 있을 수 도 있다
>>이러한 부분을 확인하기위해 반드시
필요한 테스트이다.

White-box Testing

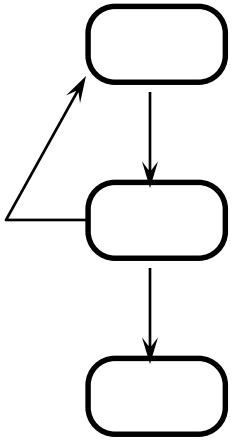
- **Path Testing**
  - **A whitebox testing strategy whose objective is to exercise every independent *execution path* through the component.**
  - **Use *program flow graph*, which is a skeletal model of all paths through the program.**

White-box Testing

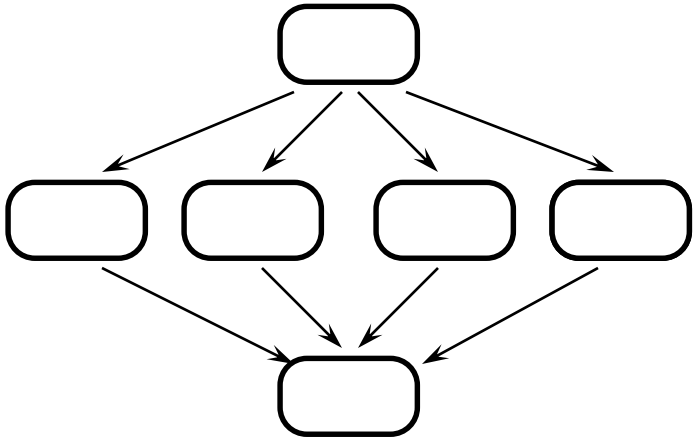
- **Flow Graph Representations**



if-then-else



loop-while

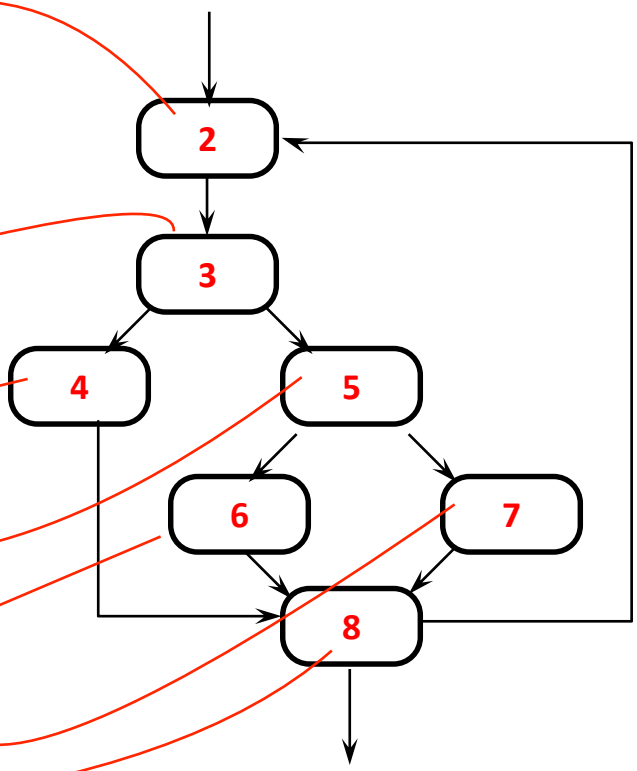


case-of

White-box Testing

- void Binary_Search (elem key, elem*t, int size, boolean &found, int &L) {

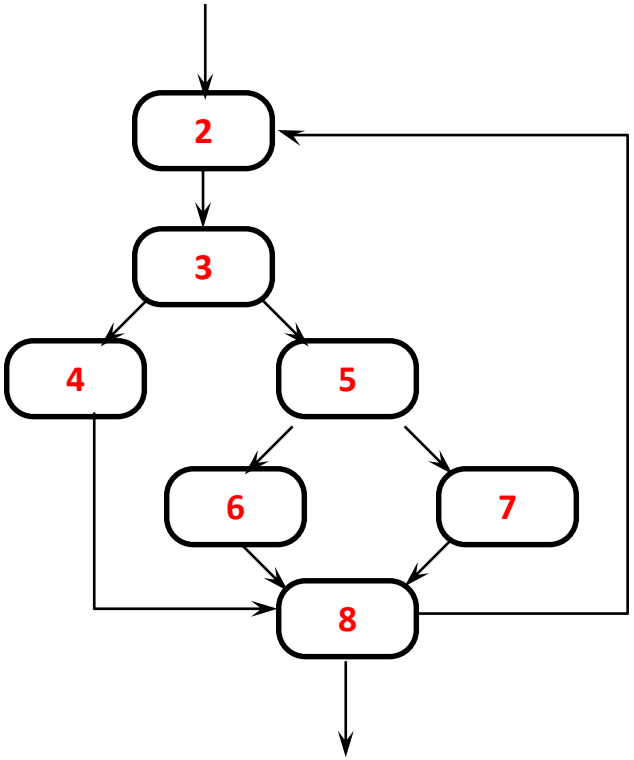
```
        int bott, top, mid;
        bott = 0;
        top = size - 1;
        L = (top + bott) / 2;
        if (T[L] == key)
                found = true;
        else
                found = false;
        while (bott <= top && !found) {
                mod = (top + bott) /2;
                if (T[mid] == key) {
                        found = true;
                        L = mid;
                } else
                        if (T[mid] < key)
                                bott = mid + 1;
                        else
                                bott = mid - 1;
        } // while
}
```



White-box Testing

- **Independent Program Path**
  - A path which traverses at least one new edge in the flow graph.
  - 2, 3, 4, 8
  - 2, 3, 5, 6, 8
  - 2, 3, 5, 7, 8

각각의 패스는 서로 독립적이기 때문에
independent path라고 부른다



- **Execute all these paths**
  - Every statement in the routine has been executed at least once and
  - every branch has been exercised for true and false conditions.

black-box testing의 경우
내부적인 부분을 볼 필요가 없기때문에
>>end user가 테스트 할 수 있다.

하지만
white box testring의 경우
내부적인 부분을 확인하면서 테스트해야하므로
>>tester가 해야한다.


Thank You !