8. Testing

Choi, Kwanghoon

Dept. of Software Engineering
Chonnam National University

Table of Contents

8.1 Testing Overview

- What is testing?
  - The process of executing a program to discover its defects before it is put into use.

- Why testing?
  - History of worst software bugs (*Wired* article)

https://www.wired.com/2005/11/historys-worst-software-bugs/?currentPage=all

  - Arian 5 rocket launch explosion

    https://www.youtube.com/watch?v=PK_yguLapgA (1:33-)

Verification approaches

- Testing
  - Test case : (input, output)
  - Test suite: set of test cases
- Static verification
  - Considers all possible inputs (executions)
- Inspections (a.k.a. reviews or walkthroughs)
  - Group activity to read codes manually
- Formal proofs of correctness
  - Proves the behavior of program w.r.t. mathematical specifications

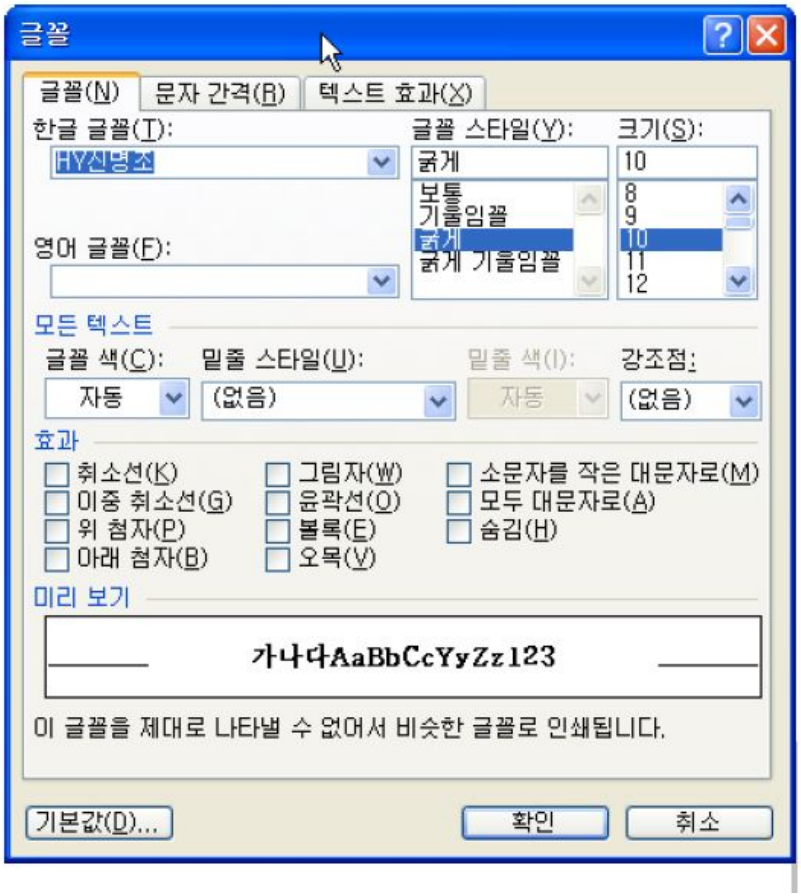
Verification approaches

- Comparison

Approaches	Pros	Cons
Testing	• No false positives	• Incomplete
Static verification	• Considers all program behaviors	• Introduce false positives
Inspections	• Systematic, thorough	• Informal, subjective
Formal proofs of correctness	• Strong guarantees	• Complex, expensive

Exhaustive Testing is Impractical

- E.g., How many cases do we have to check?



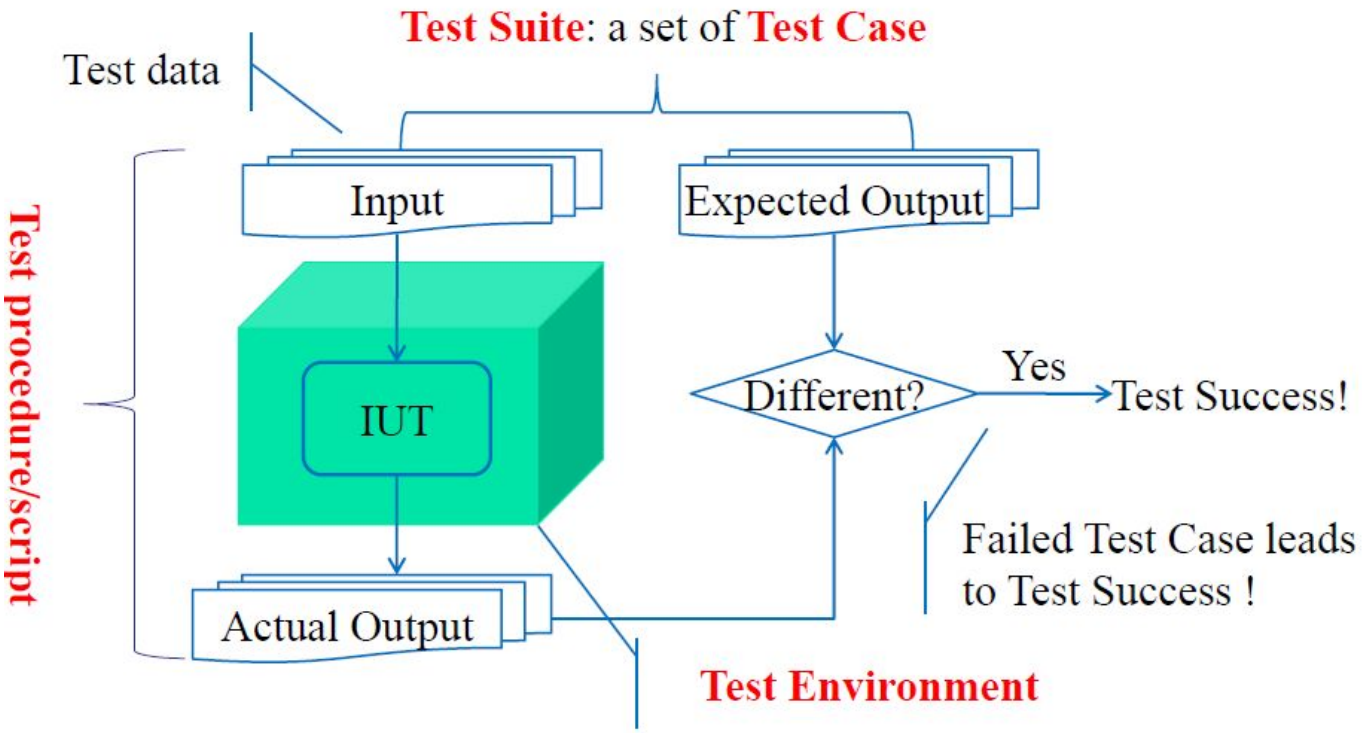
[참고: 채흥석 교수 강의]

Testing in Reality

- Testing shows the presence, not the absence of bugs – Edsger Wybe. Dijkstra

- Testing is the last chance that quality can be assessed and defects can be uncovered, and is not a safety net!

Terminology in Testing

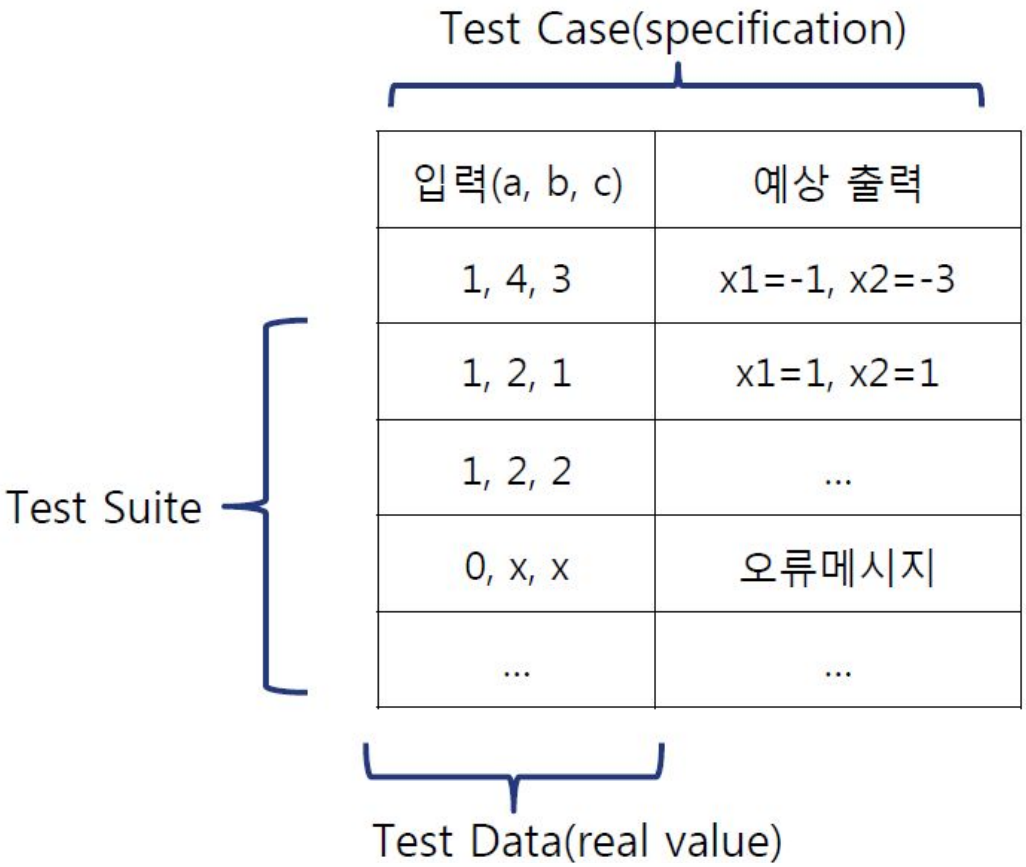
- 

(*) IUT (Implementation Under Test) : Function/Class/Component/System

[참고: 채흥석 교수 강의]

Test case / Test Suite / Test Data

•



[참고: 채흥석 교수 강의]

Test Procedure / Test Script

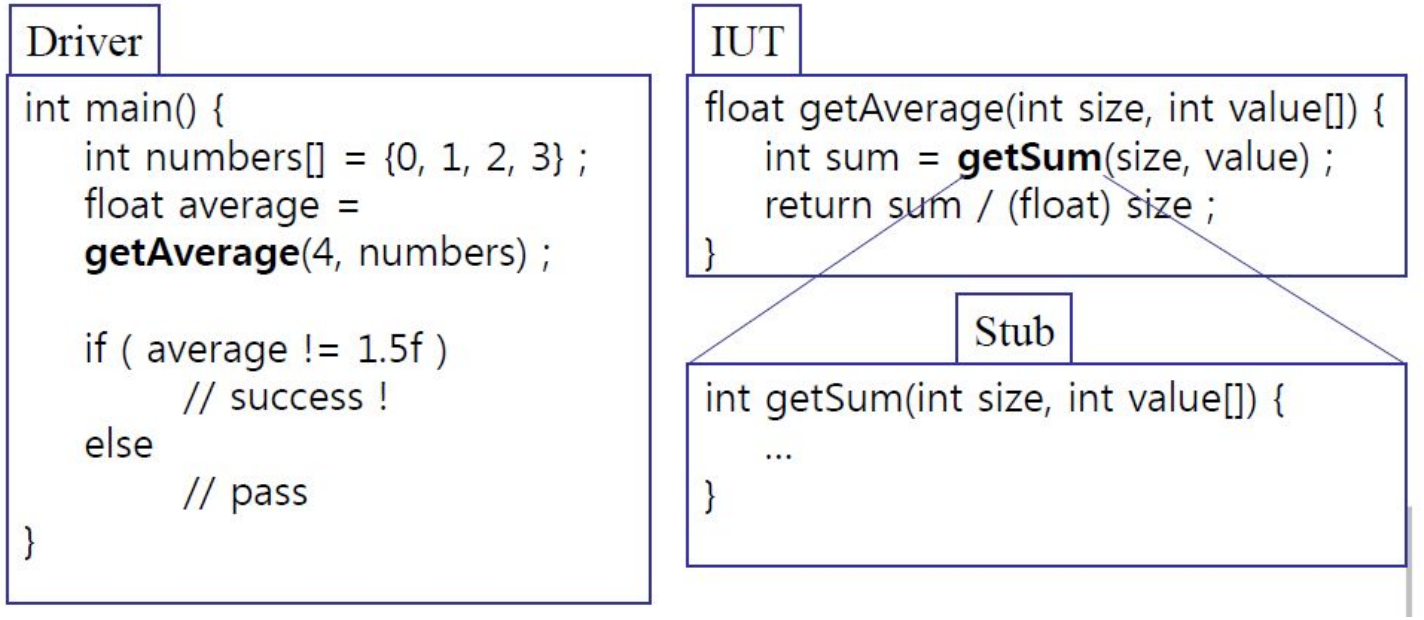
- Example

항목	예시
준비 절차	1. HTML 문서를 준비한다. HTML 문서는 웹 상에서 다운로드 하여 저장한다. 또는 다른 전문적인 웹 편집기를 이용하여 저장된 HTML 문서를 활용한다. 2. 워드 프로그램을 실행시킨다.
시작 절차	1. 워드에서 준비된 HTML 문서를 읽는다.
상세 절차	1. 워드에서 읽힌 HTML 문서를 편집한다. 즉 텍스트를 추가/변경/삭제한다. 일정 부분을 선택하여 글꼴을 수정하도록 한다. 2. HTML 문서를 저장한다. 3. 저장된 HTML 문서를 웹 브라우저를 이용하여 읽는다.
결과 측정 방법	1. 시작 절차에서 즉 HTML 문서를 처음에 워드에서 읽었을 때 화면이 웹 브라우저에서의 화면과 동일한지를 확인한다. 2. 상세 절차 2)에서 저장된 HTML 문서가 워드에서 출력된 화면과 상세 절차 3)에서 웹 브라우저에 동일하게 출력되는 지 확인한다.

[참고: 최은만 교재]

Test Environment

- Unit test needs test harness (driver and stub)



[참고: 채흥석 교수 강의]

Test Case Design

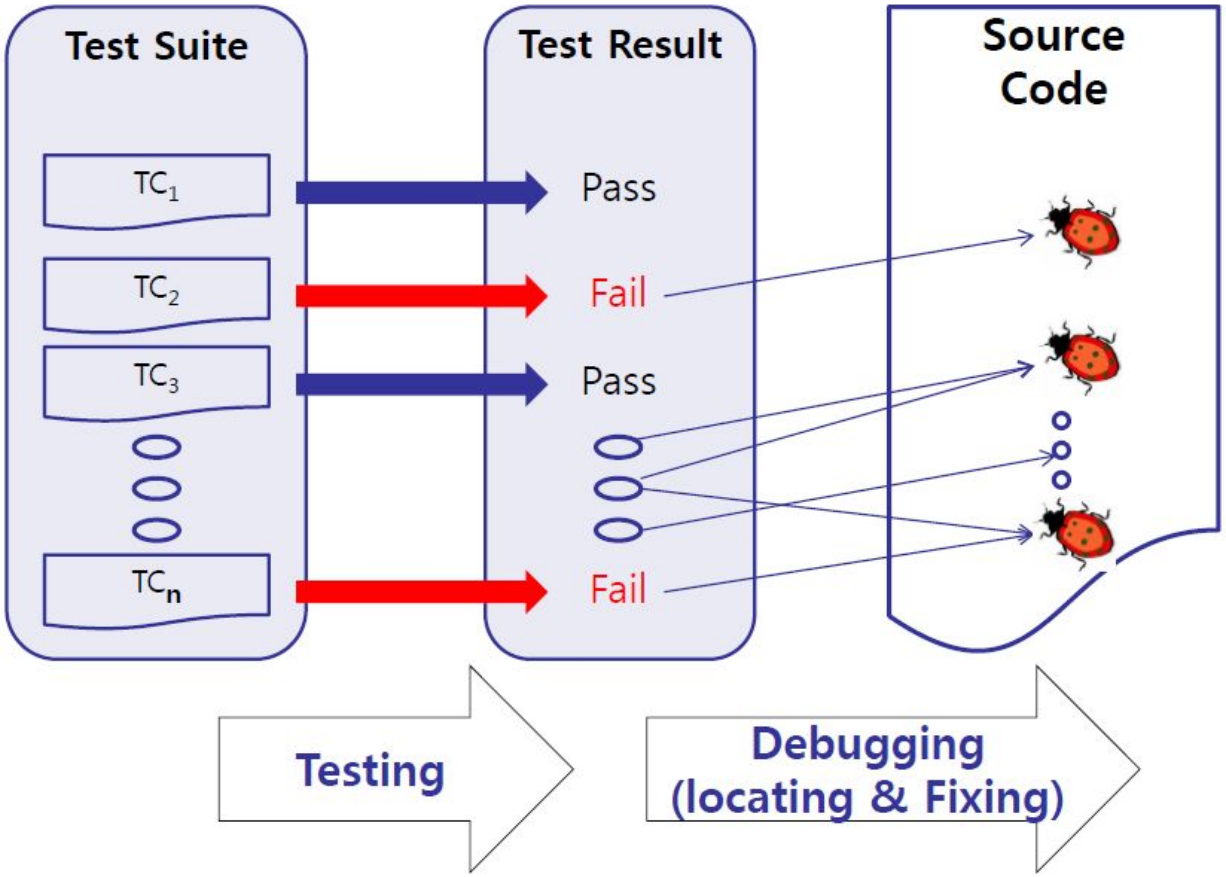
- Designing test cases
  - to uncover defects (Objective),
  - in a complete manner (Criteria),
  - with a minimum of test case (Effort and time)

- Two Tips
  - The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section.
  - Invalid or unexpected input data may cause crash
    - (Negative testing vs. Positive testing)

Testing vs. Debugging

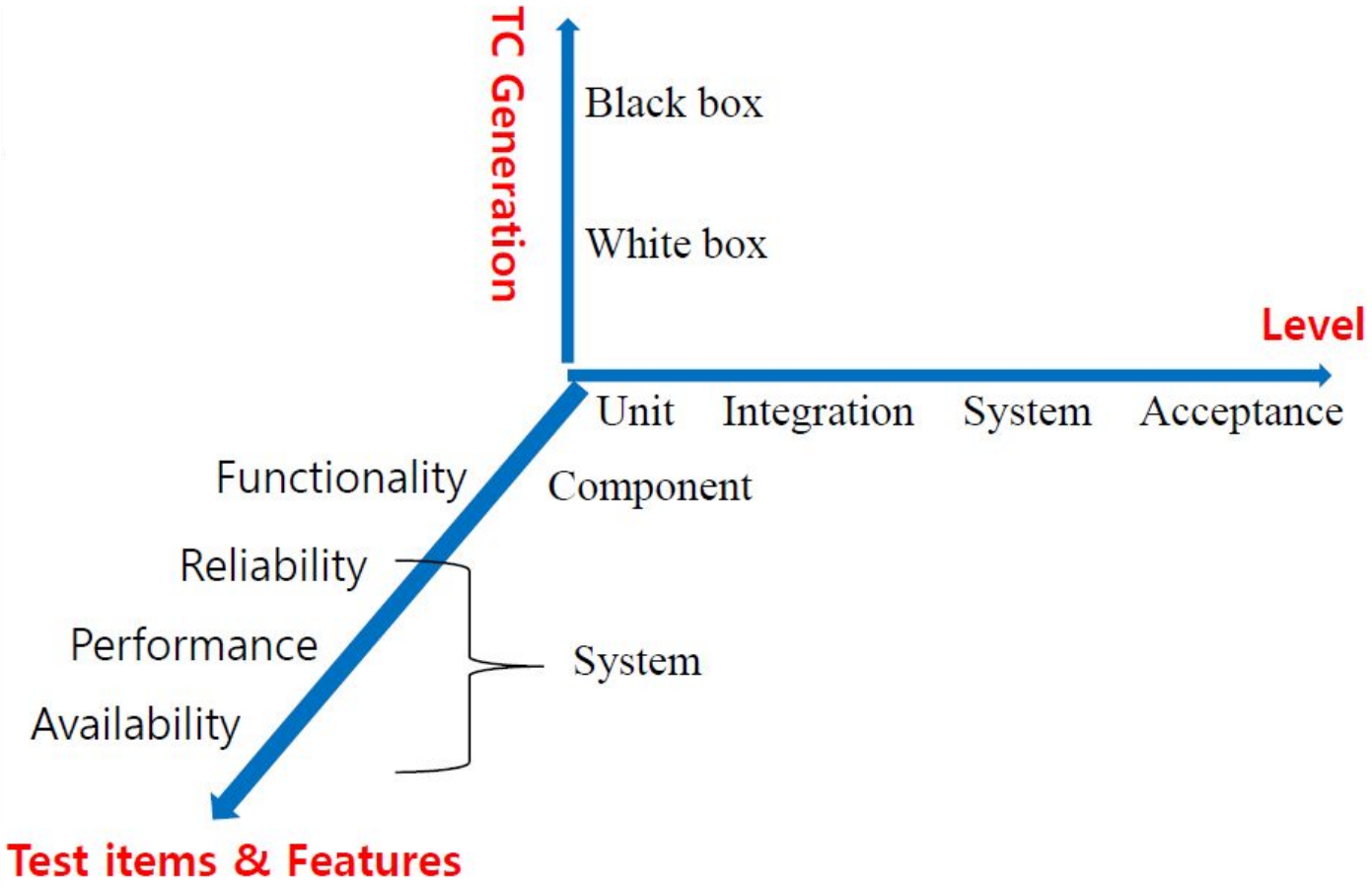
- Defects are introduced during the development, but we don’t know their existence.
- Testing can determine software has some defects, but not what/where they are.
- Debugging locates the location of defects and removes them.

Testing vs. Debugging (cont.)

- 

[참고: 채흥석 교수 강의]

Testing Approaches

- 

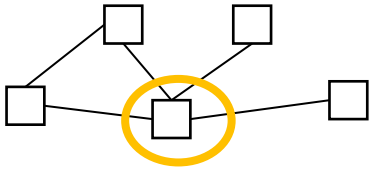
[참고: 채흥석 교수 강의]

Testing Approaches

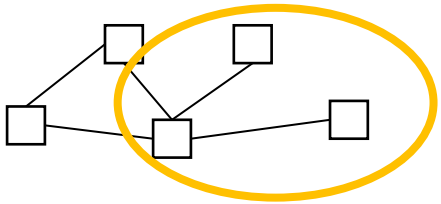
- Granularity levels

- Functional testing
- Nonfunctional testing

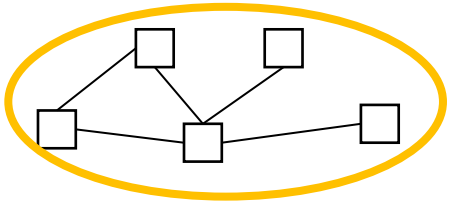
[Unit testing]
(단위 테스트)



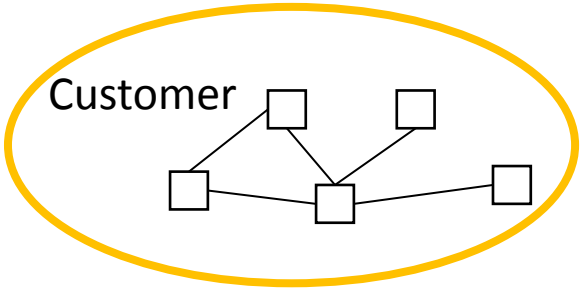
[Integration testing]
(통합 테스트)



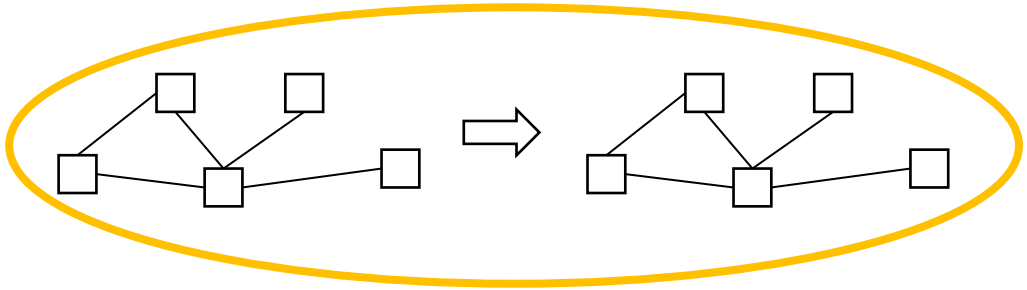
[System testing]
(시스템 테스트)



[Acceptance testing]
(인수 테스트)



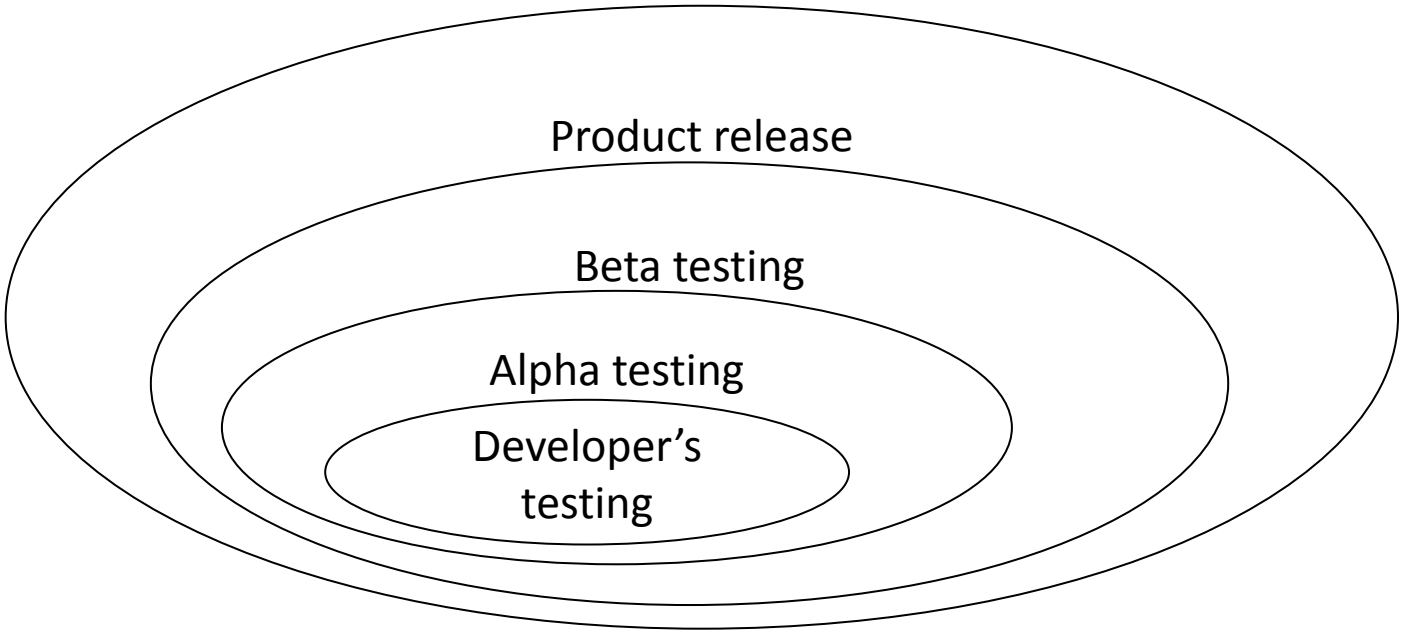
[Regression testing]
(회귀 테스트)



[참고: Udacity]

Testing Approaches

- The level of tolerance for problems
  - *Alpha testing* by a set of users internal to the development organization
  - *Beta testing* by a set of users outside the development organization



[참고: Udacity]

Testing Approaches

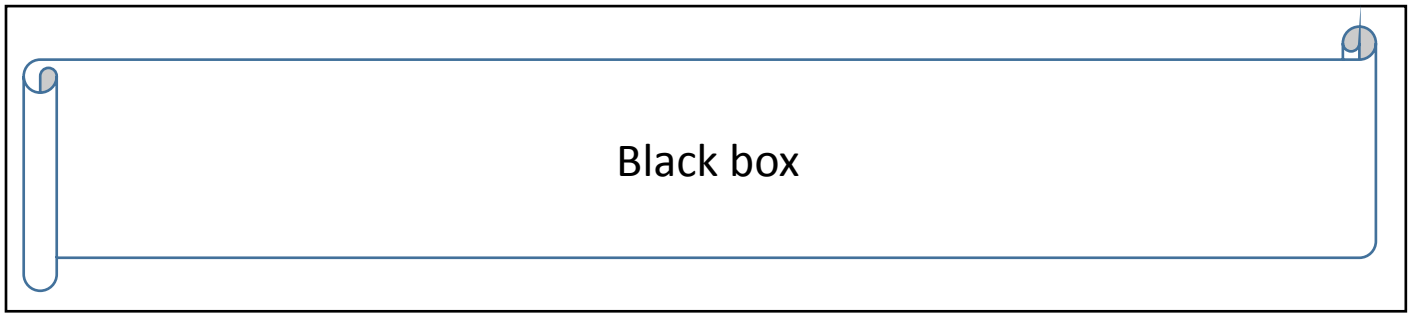
- Black box / white box testing
  - Black box testing
    - Based on a specification of the software
    - Cover as much specified behavior as possible
    - Cannot reveal errors due to implementation details
  - White box testing
    - Based on the code
    - Cover as much coded behavior as possible
    - Cannot reveal errors due to missing paths
- These two approaches are complementary to each other.

8.2 Black-box Testing

- Black box testing example
  - Specification:

    - inputs an integer and prints it

  - Code:



[참고: Udacity]

8.2 Black-box Testing

- Advantages
  - Focus on the domain
  - No need for the code (=> early test design)
  - Catches logic defects
  - Applicable at all granularity levels

8.2 Black-box Testing

- From specifications to test cases
  - by a systematic functional testing approach

  - From functional specification
  - Step1. identify *independently testable features*
  - Step2. identify *relevant inputs*
  - Step3. derive *test cases specification*
  - Step4. generate *test cases*

8.2 Black-box Testing

- Example) Usecase diagram specification

- Step1: Identify independent testable features

UC1: 새 고객 등록

액터: 새 고객	시스템: 웹 애플리케이션
	0. 시스템이 사용자 등록 링크를 가진 홈페이지를 디스플레이 한다.
1. 사용자가 고객 등록 링크를 클릭한다.	2. 시스템이 새 고객의 등록 양식을 디스플레이한다.
3. 사용자가 *사용자 ID, 패스워드, 재입력 패스워드*를 넣고 제출 버튼을 누른다.	4. 시스템이 로그인 ID와 패스워드를 검증하고 4.1 등록이 성공되었음을 디스플레이하거나 4.2 오류 메시지를 디스플레이하고 사용자에게 다시 시도할 것을 요구한다.
5. 사용자가 등록 성공 페이지를 본다.	

사용자 입력 요소

8.2 Black-box Testing

- Step2: Identify relevant inputs

입력 요소	타입	값의 명세	정상	비정상	예외
로그인 ID	스트링	문자 길이가 8에서 20 사이	로그인 ID가 명세를 만족하여야 하며 다른 사용자와 중복되지 않아야	값의 명세를 만족하지 않거나 다른 사용자와 중복	스트링의 길이가 0, 1 또는 매우 큰 값 하나 이상의 빈칸이나 특수문자가 존재
패스워드	패스워드	길이가 8에서 12개의 문자, 적어도 하나의 문자, 숫자, 특수문자를 포함	패스워드 규칙에 맞는 패스워드	패스워드 규칙에 맞지 않는 패스워드	길이가 0, 1, 매우 큰 길이를 가진 패스워드 하나 또는 그 이상의 빈칸, 제어문자를 가진 패스워드
재입력한 패스워드	패스워드	패스워드와 같음	패스워드와 매치됨	재입력된 패스워드가 패스워드와 매치되지 않거나 복사-붙여넣기로 입력됨	

[최은만 교재]

8.2 Black-box Testing

- Step3: Derive test case specification

테스트 케이스	로그인 ID	패스워드	재입력된 패스워드	예상 결과
1	정상	정상	정상	등록이 성공되었다는 페이지가 보임
2	정상	정상	비정상	오류 메시지가 보임
3	정상	비정상	정상	오류 메시지가 보임
~~4~~	~~정상~~	~~비정상~~	~~비정상~~	<테스트 케이스 3에 포함>
5	정상	예외	정상	오류 메시지가 보임
~~6~~	~~정상~~	~~예외~~	~~비정상~~	<테스트 케이스 2, 3에 포함>
7	비정상	정상	정상	오류 메시지가 보임
~~8~~	~~비정상~~	~~정상~~	~~비정상~~	<테스트 케이스 2, 7에 포함>
~~9~~	~~비정상~~	~~비정상~~	~~정상~~	<테스트 케이스 3, 7에 포함>
~~10~~	~~비정상~~	~~비정상~~	~~비정상~~	<테스트 케이스 2, 3, 7에 포함>
~~11~~	~~비정상~~	~~예외~~	~~정상~~	<테스트 케이스 5, 7에 포함>
~~12~~	~~비정상~~	~~예외~~	~~비정상~~	<테스트 케이스 2, 5, 7에 포함>
13	예외	정상	정상	오류 메시지가 보임
~~14~~	~~예외~~	~~정상~~	~~비정상~~	<테스트 케이스 7, 13에 포함>
~~15~~	~~예외~~	~~비정상~~	~~정상~~	<테스트 케이스 3, 13에 포함>
~~16~~	~~예외~~	~~비정상~~	~~비정상~~	<테스트 케이스 3, 7, 13에 포함>
~~17~~	~~예외~~	~~예외~~	~~정상~~	<테스트 케이스 5, 13에 포함>
~~18~~	~~예외~~	~~예외~~	~~비정상~~	<테스트 케이스 2, 5, 13에 포함>

8.2 Black-box Testing

- Step4: Generate test cases

테스트 케이스	로그인 ID	패스워드	재입력된 패스워드	예상 결과
1	"newuser@naver.com"	"gls123%KSL"	"gls123%KSL"	등록이 성공되었다는 페이지가 보임
2	"newuser@naver.com"	"gls123%KSL"	"xxxxxxxxx"	오류 메시지가 보임
3	"newuser@naver.com"	"vvvvv"	"gls123%KSL"	오류 메시지가 보임
5	"newuser@naver.com"	"z"	"gls123%KSL"	〈테스트 케이스 3에 포함〉
7	"olduser@naver.com"	"gls123%KSL"	"gls123%KSL"	오류 메시지가 보임
13	"new user@naver.com"	"gls123%KSL"	"gls123%KSL"	〈테스트 케이스 2, 3에 포함〉

[최은만 교재]

8.2 Black-box Testing

- 조합 방법
  - 각 선택 조합 (Each choice): 각 입력 인자의 분할된 영역에서 최소한 하나의 입력 값을 선택

  - 모든 조합 (All combinations): 모든 가능한 영역 입력 값들의 조합

  - 페어와이즈 조합(Pairwise): 각 영역의 값을 다른 영역의 값과 최소한 한번은 짝을 짓는 조합

    => 모든 조합에 비해 테스트 케이스 수를 줄이면서도 오류 검출 능력은 사실상 비슷한 경향

    (이유: 오류는 모든 입력의 조합에 의해 발생하기 보다 기껏해야 두 입력의 조합으로 발생하기 때문)

pict (Pairwise independent combinatorial tool): https://github.com/microsoft/pict

8.2 Black-box Testing

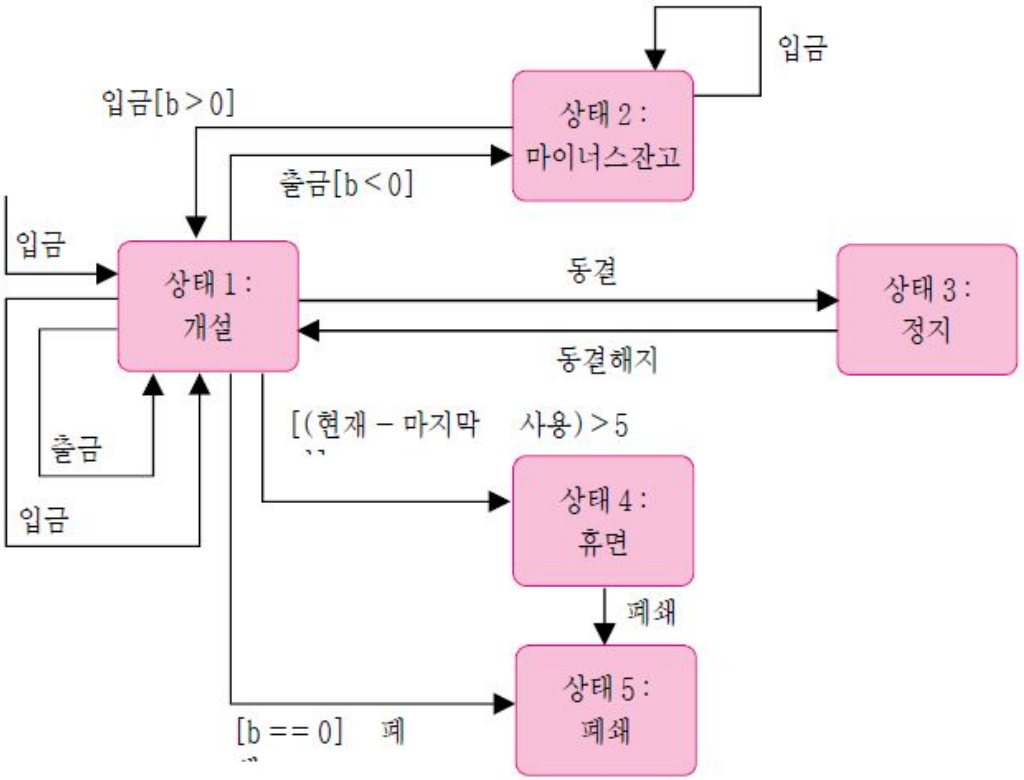
- A: A1,A2,A3, B: B1,B2,B3, C: C1,C2라고 가정
  - 페어와이즈 방식의 가능한 조합

Test Case	A	B	C
1	A1	B1	C1
2	A1	B2	C2
3	A1	B3	C2
4	A2	B1	C2
5	A2	B2	C1
6	A2	B3	C1
7	A3	B1	C2
8	A3	B2	C1
9	A3	B3	C2

Q. (B,C)의 모든 쌍을 9개의 TC에 포함? (B1,C1), (B1,C2), (B2,C1), (B2,C2), (B3,C1), (B3,C2)

8.2 Black-box Testing

- Example) State diagram specification
  - *Generate test cases to cover all state transitions (edges)*



8.2 Black-box Testing

- Example) State diagram specification

[최은만 교재]

No.	트랜지션	테스트 케이스
1	start →1	입금
2	1→1	출금
3	1→1	입금
4	1→2	입금(잔고>0)
5	2→2	입금
6	2→1	출금(잔고<0)
7	1→3	동결
8	3→1	동결해지
9	1→4	(현재–마지막 사용일)>5년
10	1→5	폐쇄(잔고=0)
11	4→5	폐쇄

8.2 Black-box Testing

- Functional testing based on program specification (cf. specification-based testing)

- Creating/selecting test cases
  - Equivalence class partitioning
  - Boundary value analysis
  - Cause-effect graph
  - Model-based testing (e.g., state diagram)

- Combinatorial test case generation
  - A catalogue of tools www.pairwise.org
  - PICT (https://github.com/microsoft/pict)
  - TSL generator (https://github.com/alexorso/tslgenerator)

8.3 White-box Testing

- White box testing example
  - Specification:

Not available

  - Code:

White box

```
int fun (int param) {
    int result;
    result = param / 2;
    return result;
}
```

[참고: Udacity]

8.3 White-box Testing

- Structural testing based on program code (cf. program-based testing)
- Advantages
  - Allows for covering the coded behavior
  - Can be used to compare test suites
  - Based on the code
    - => can be measured objectively
    - => can be measured automatically
- Different kinds
  - Control-flow based
  - Data-flow based
  - Fault-based

8.3 White-box Testing

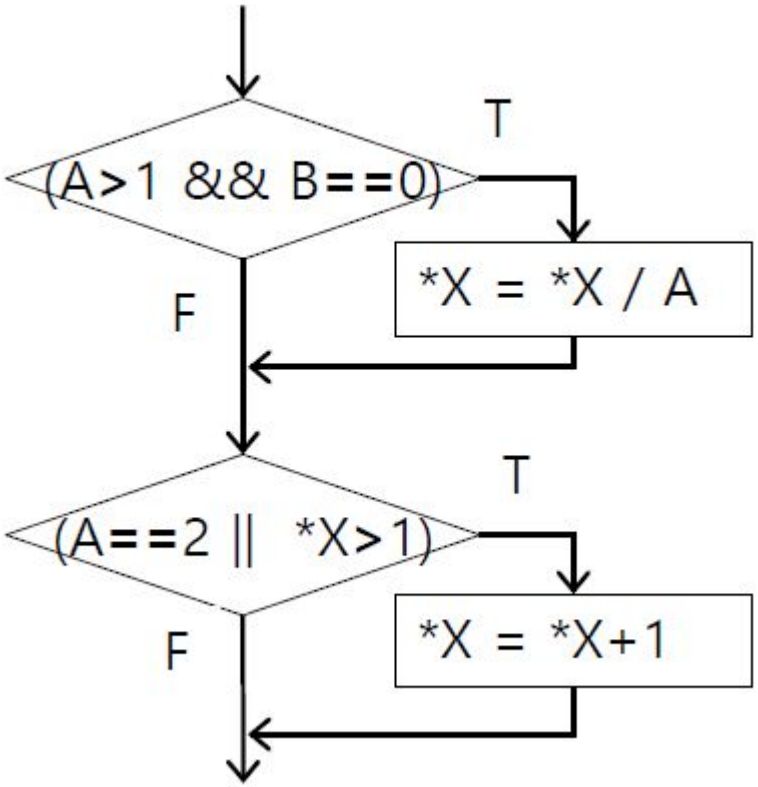
- Control-flow graph (e.g., see the next slide)
- Basic path
  - A straight path from a start node to an end node traversing only once for each cycle with no in- or out- edge into or from a middle node.
  - Cyclomatic complexity (T. J. McCabe)
    - # of independent basic paths
      - (independent path: a path that has at least one edge that has not been traversed)
- A test covering all basic paths
  - => White-box testing

cf. 교재 기본 경로(basic paths) 예제 및 Cyclomatic number 구하기 예제

Control-flow Graph

•

```
   void F(int A, int B, int* X) {
1     if ( A > 1 && B == 0 )
2         *X = *X / A;
3     if ( A == 2 || *X > 1 )
4         *X = *X + 1;
   }
```



Q. How many basic paths are in the example?

[참고: 채흥석 교수 강의]

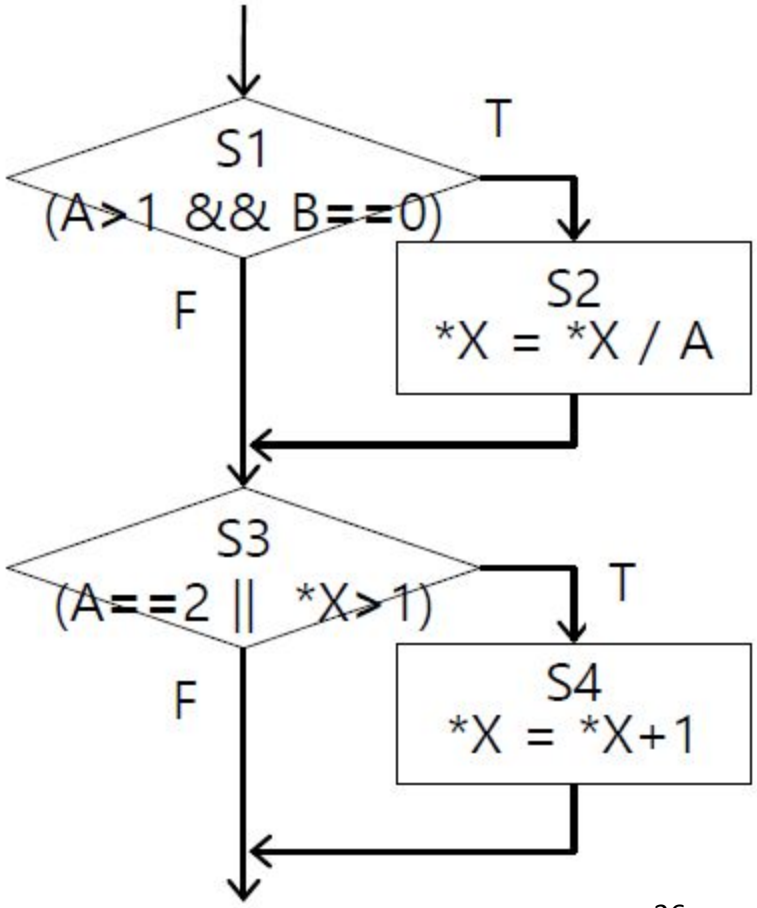
Test Coverage

- Basic path coverage
  - A test suite {TC*1*, …, TC*k*} is basic-path coverage if it exercises all the basic paths.
- All-path coverage
  - A test suite {TC*1*, …, TC*k*} is all-path coverage if it exercises all path combinations
- Statement coverage
- Branch coverage
- Condition coverage
- Multiple condition coverage
- MC/DC coverage

Statement Coverage

- The percentage of the statements exercised by the test suite

	TC1			TC2		
	A	B	*X	A	B	*X
	3	0	1	2	0	3
S1	√			√		
S2	√			√		
S3	√			√		
S4				√		
	3 / 4			4 / 4		
	4 / 4					

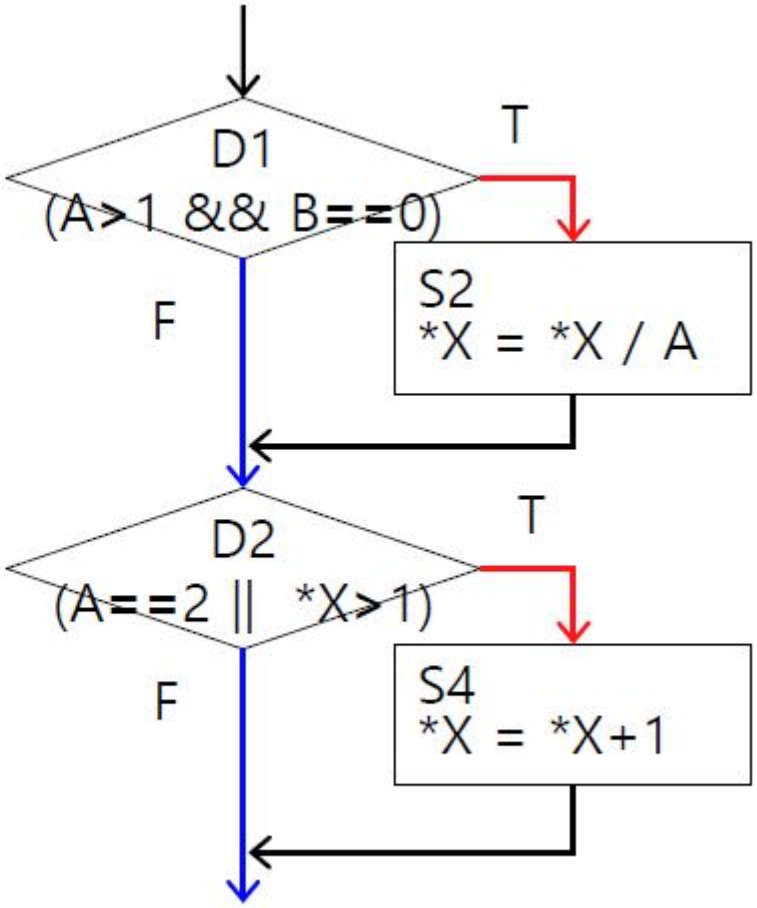


[참고: 채흥석 교수 강의]

Branch Coverage (Decision Cov.)

- The percentage of the branches exercised by the test suite

	TC1	TC1
	A=2 B=0 *X=4	A=1 B=1 *X=1
A > 1	T	F
B == 0	T	F
A == 2	T	F
*X > 1	T	F
	4 / 8	4 / 8
	8 / 8	



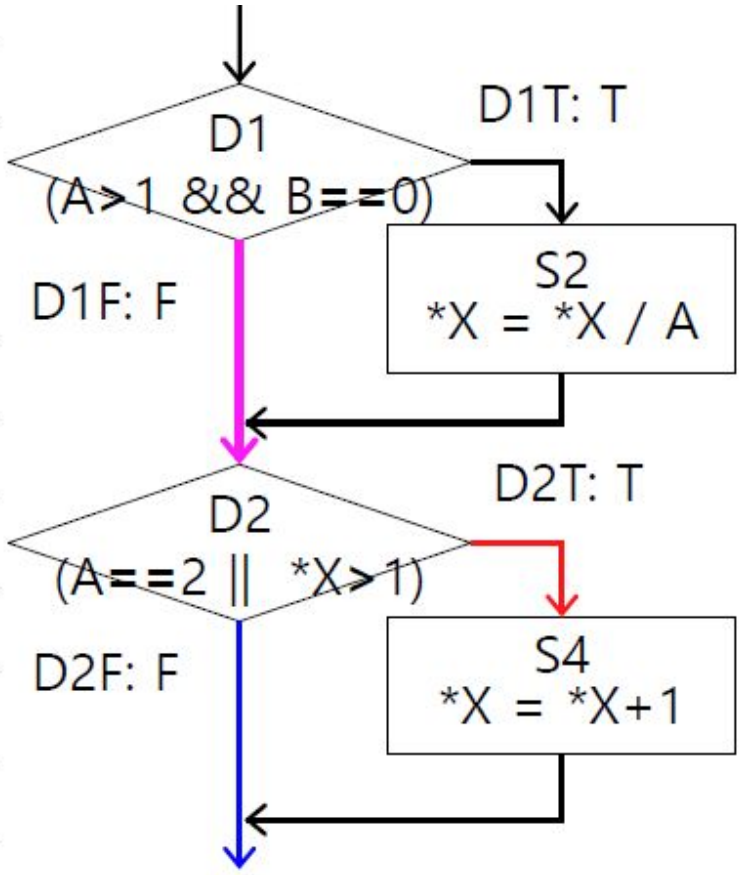
(*) TC1은 Statement Cov.이지만 Branch Cov.는 아니다.

[참고: 채흥석 교수강의]

Condition Coverage

- The test suite exercised all (atomic) conditions both as true and false.

	TC1	TC1
	A=1 B=0 *X=3	A=2 B=1 *X=1
A > 1	F	T
B == 0	T	F
A == 2	F	T
*X > 1	T	F
	4 / 8	4 / 8
	8 / 8	



[참고: 채흥석 교수 강의]

Branch Cov. vs. Condition Cov.

- Condition coverage does not always imply Branch coverage

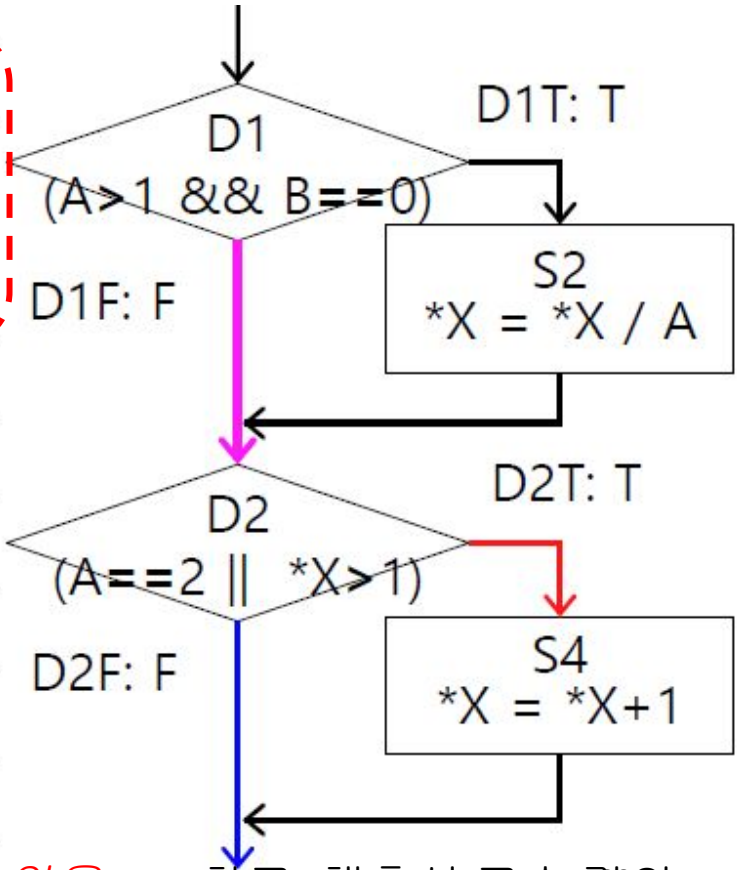
동일한 테스트케이스

	TC1	TC1
	A=1 B=0 *X=3	A=2 B=1 *X=1
A > 1	F	T
B == 0	T	F
A == 2	F	T
*X > 1	T	F
	4 / 8	4 / 8
	8 / 8	

Condition cov. 만족.

	TC1	TC2
	A=1 B=0 *X=3	A=2 B=1 *X=1
D1T		
D1F	√	√
D2T	√	
D2F		√
	2 / 4	2 / 4
	3 / 4	

Branch cov. 만족하지 않음.



[참고: 채흥석 교수 강의]

Condition/Branch Coverage

- Goal: To design test cases (TCs) to satisfy both condition cov. and branch cov.

  => Multiple-condition coverage (all combinations of atomic conditions)

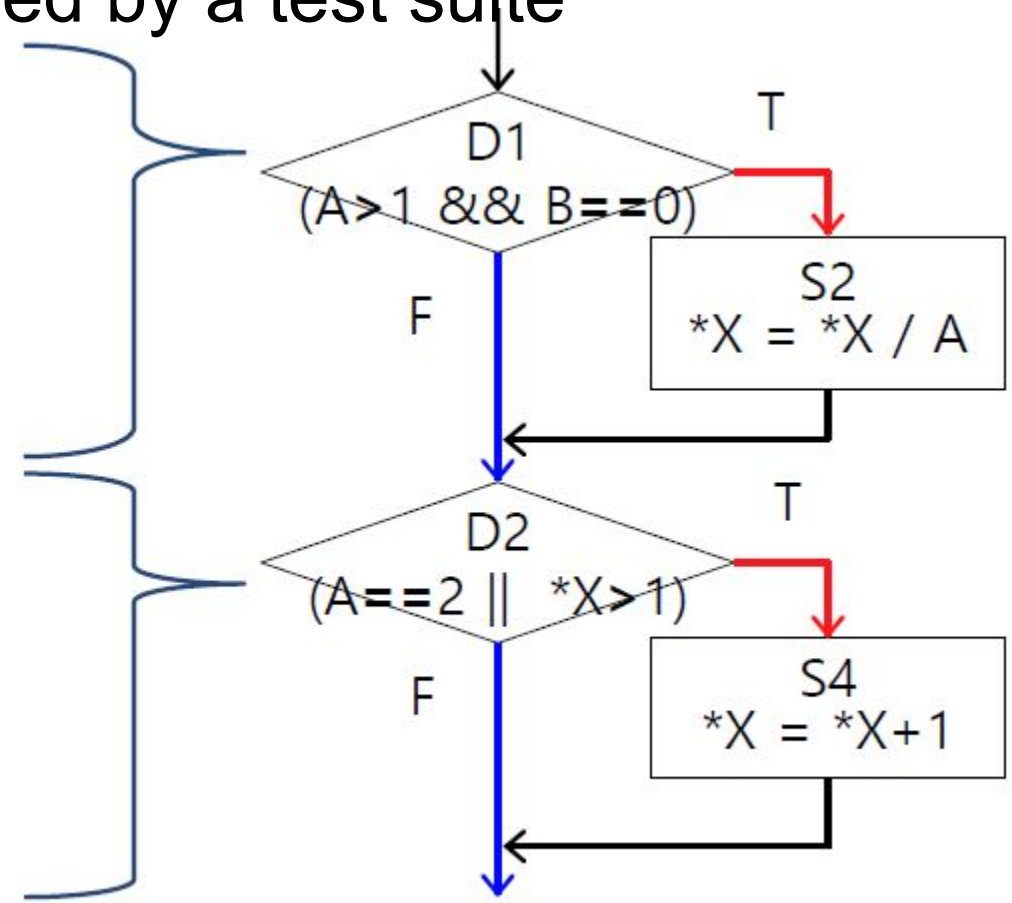
  => Modified condition, decision coverage (MC/DC)

(용어) Branch coverage == Decision coverage

Multiple-Condition Coverage

- The percentage of all combinations of each condition exercised by a test suite

1	A > 1, B = 0
2	A > 1, B ≠ 0
3	A ≤ 1, B = 0
4	A ≤ 1, B ≠ 0
5	A = 2, *X > 1
6	A = 2, *X ≤ 1
7	A ≠ 2, *X > 1
8	A ≠ 2, *X ≤ 1



Multiple condition cov.는 모든 조합을 테스트하므로 Condition/Decision cov.이다.
그런데, TC의 수가 2^{# of (atomic) conditions}로 너무 많다.

[참고: 채흥석 교수 강의]

Modified Cond. Decision Cov. (MC/DC)

- MC/DC TCs pose the same condition/decision coverage as multiple condition coverage TCs
  - But # of MC/DC TCs is known to be strictly smaller than # of multiple condition coverage TCs.

- MC/DC pairs
  - 예제 코드) if (A and B and C) { … }
  - TC1: A,B,C = (T,T,T) : outcome=T
  - TC2: A,B,C = (T,T,F) : outcome=F
  - Only the values of C are different in TC1 and TC2.
    - => TC1(A) == TC2(A), TC1(B) ==TC2(B), TC1(C)!=TC2(C)
    - => TC1-outcome != TC2-outcome
  - TC1 and TC2 is called an MC/DC pair.

Modified Condition Decision Cov. (MC/DC)

Decision 결과가 다르고 단 하나의 조건의 값만 다른 것을 MC/DC 쌍이라 부른다.

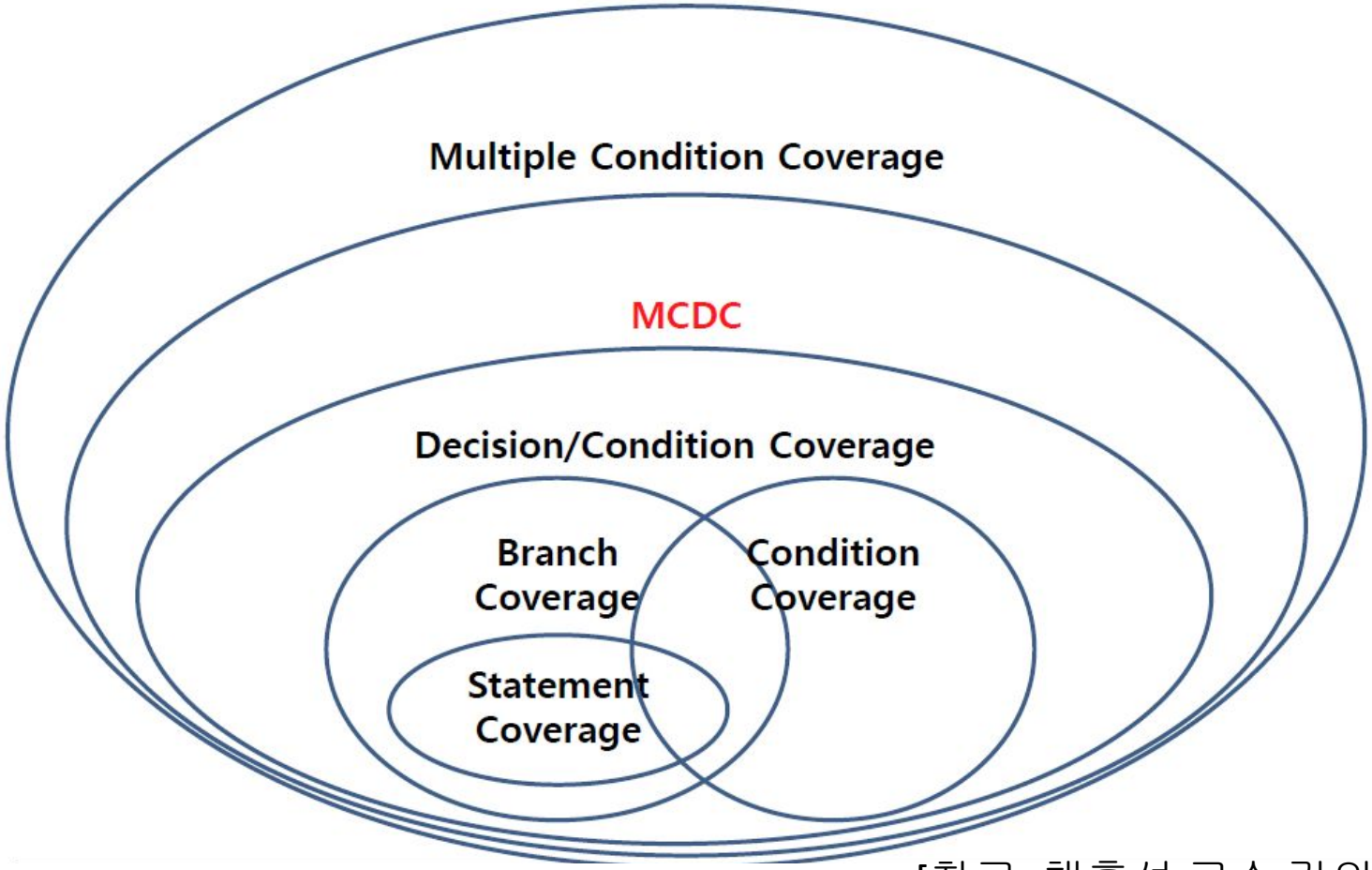
	A	B	C	A and B and C				A or B or C			
				Outcome	A	B	C	Outcome	A	B	C
TC1	T	T	T	T	√	√	√	T			
TC2	T	T	F	F			√	T			
TC3	T	F	T	F		√		T			
TC4	T	F	F	F				T	√		
TC5	F	T	T	F	√			T			
TC6	F	T	F	F				T		√	
TC7	F	F	T	F				T			√
TC8	F	F	F	F				F	√	√	√

** A and B and C 예제에서 MC/DC coverage의 TCs = {TC1,TC2,TC3,TC5}*
하지만, Multiple-condition coverage의
TCs={TC1,TC2,TC3,TC4,TC5,TC6,TC7,TC8}

[참고: 채흥석 교수 강의]

Comparison

* MCDC (Multiple condition decision coverage)
Condition coverage와 branch coverage를 달성하면서도 multiple condition coverage보다 TC수가 적은 방법



[참고: 채흥석 교수 강의]

Comparison

Coverage	Covered Element
Statement Coverage	Every statement
Decision Coverage	Every decision
Condition Coverage	Every condition
Condition/Decision Coverage	Every condition and decision
MC/DC	Every condition with independent effect on decision
Multiple Condition Coverage	Every combination of condition

Exercise

https://github.com/kwanghoon/TestingExamples

8.4 Object-oriented Testing

- Types of Testing on Test Items
  - Class/OO testing
  - Component testing
  - Database testing
  - Web testing

- OO Testing
  - Use Case based testing
  - State based testing

8.5 Integration Testing

- Types of Testing on Test Levels (when to test)
  - Unit testing
  - Integration testing
  - System testing
  - Acceptance testing
  - Regression testing

- Integration
  - Big-bang integration
  - Top-down integration
  - Bottom-up integration
  - Threads integration

8.6 System & Acceptance Testing

- Requirement Testing
- Performance Testing
- Security Testing
- User Interface/Usability Testing
- Acceptance Testing

8.7 Testing Tools

- Testing Automation Tools

- Code Analysis Tools
  - Static analysis
  - Dynamic analysis

- Test case generators and executors

- Unit testing tools
  - Junit, CUnit, CPPUnit, …
  - Mockito, Cmock, … (Mocking tools, Stub)
  - JaCoCo, Clover (Coverage measurement tools)

Trends in Testing

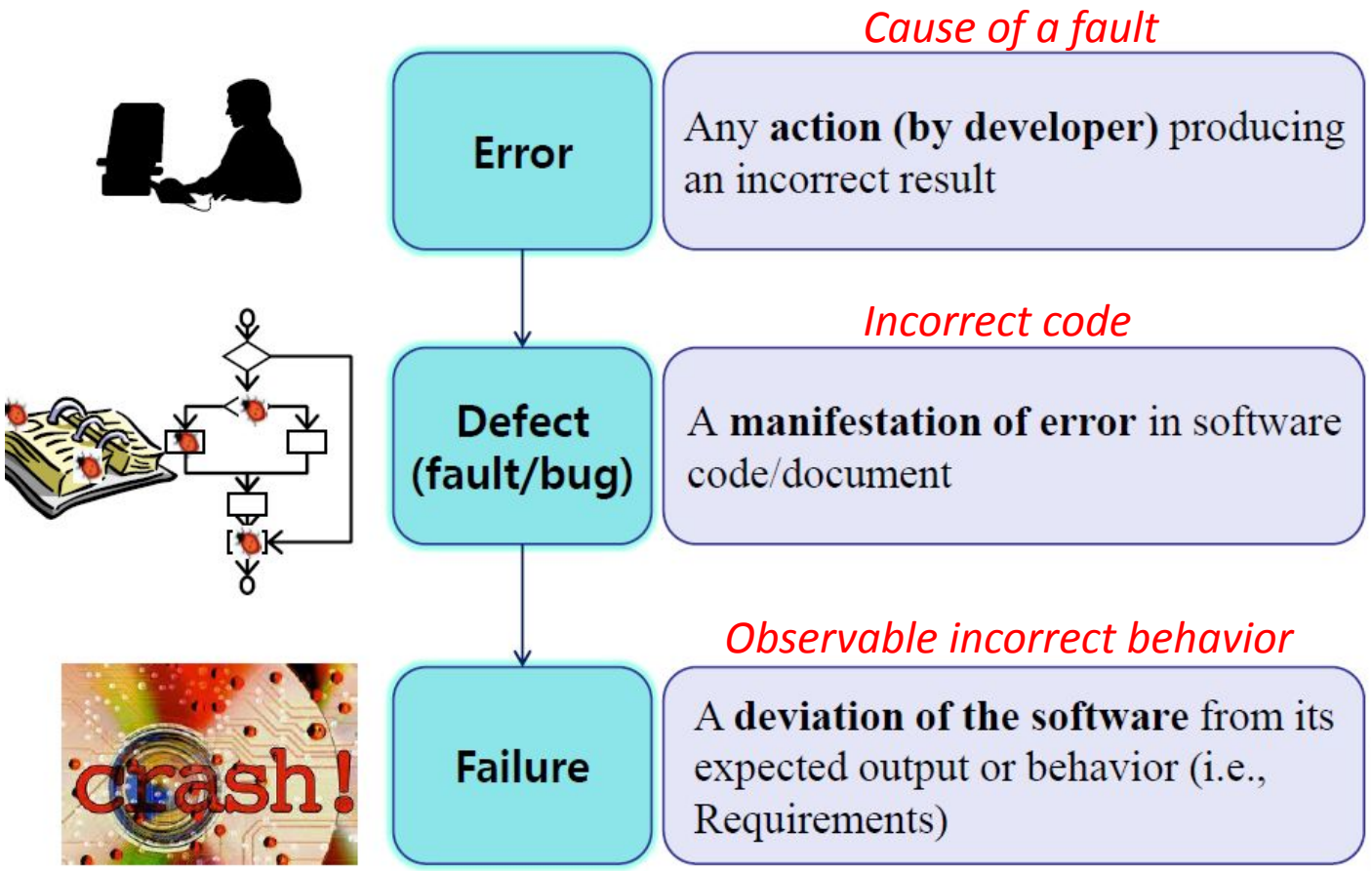
- Traditional approach
  - Developers finish code, some ad-hoc testing
  - QA staffs test the code
  - E.g., Bill Gates, Microsoft
    - “50% of my company employees are testers, and the rest spends 50% of their time testing”

- Today/Agile development
  - Developers test their own code (not by testers)
  - Testing tools & processes highly automated
  - QA/testing group improves testability & tools

Summary

- Error, Defect, and Failure
- Black box testing
- White box testing (Test coverage)
- Unit / Integration / System / Acceptance Testing
- Testing Tools

Characteristics of Error, Fault, Failure



[참고: 채흥석 교수 강의]