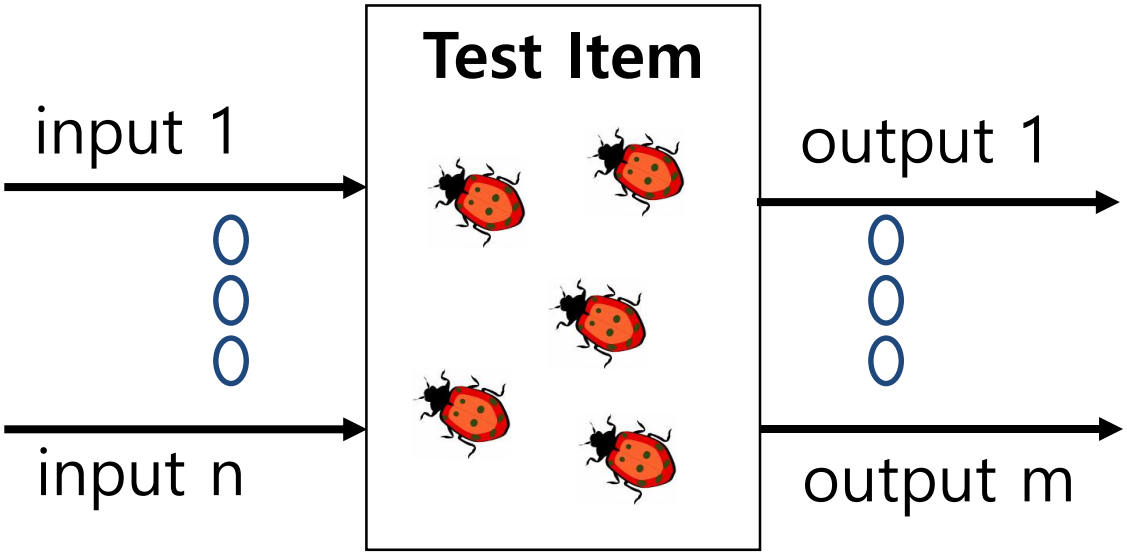
블랙박스 테스팅 기본

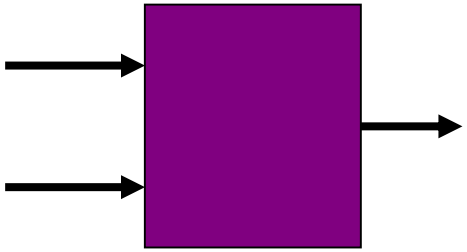
Black Box Test Design

- We don't know where defects are
- Thus, we have to check every part of the test item
- Various input values need to be examined.



Problem of Exhaustive Testing

- A module has two input INTEGER parameters
  - Word size : 32 bits
  - The number of input conditions : 2^{64}



- If we assume that
  - Testing completely automated
  - Execution time for each test case : 10^{-7} second (100 nanoseconds)

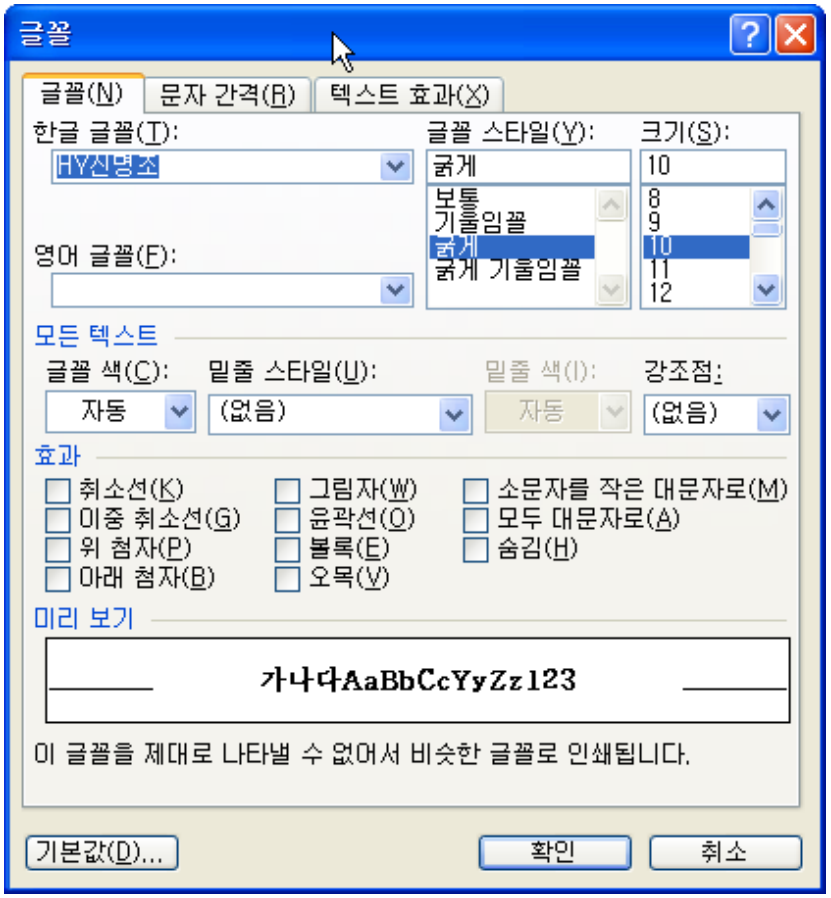
- Total time required :

$$\frac{2^{64} \cdot 100 \cdot 10^{-9}}{3600 \cdot 24 \cdot 365} = 58{,}494 \text{ years}$$

Problem of Exhaustive Testing

❖ How many cases do we have to check ?

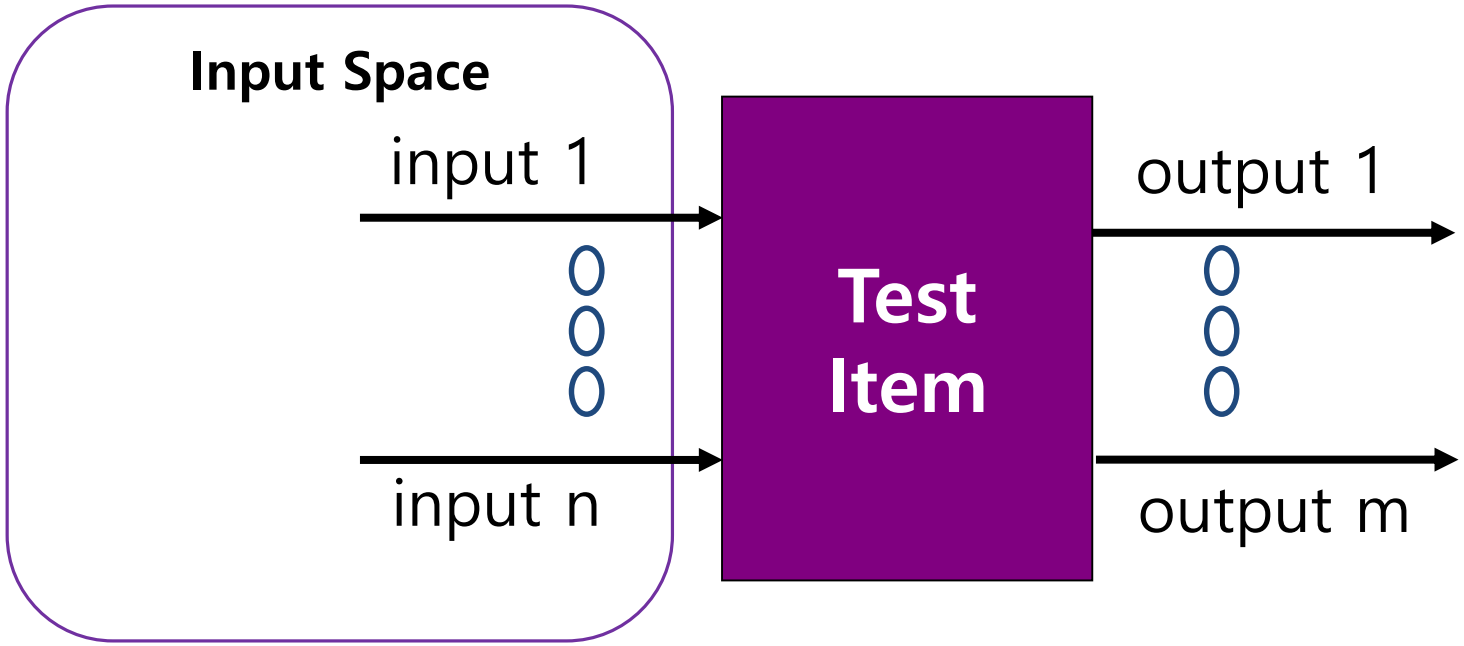
Input	Cases
한글 글꼴	N1
글꼴 스타일	N2
크기	N3
글꼴 색	N4
밑줄 스타일	N5
…	…
효과	Nn



❖ Total cases: $\prod_1^n Ni$

Reality of Test Design

- Realistic Test Design tries
  - Not to miss meaningful inputs
  - To reveal defects which possibly exist in the system



Test Design Techniques

- Equivalence Partitioning Testing
- Boundary-Value Analysis Testing

Single Input

- Combinatorial Testing
- Decision Table/Cause-Effect Graphing
- Pair-wise Testing

Multiple Inputs

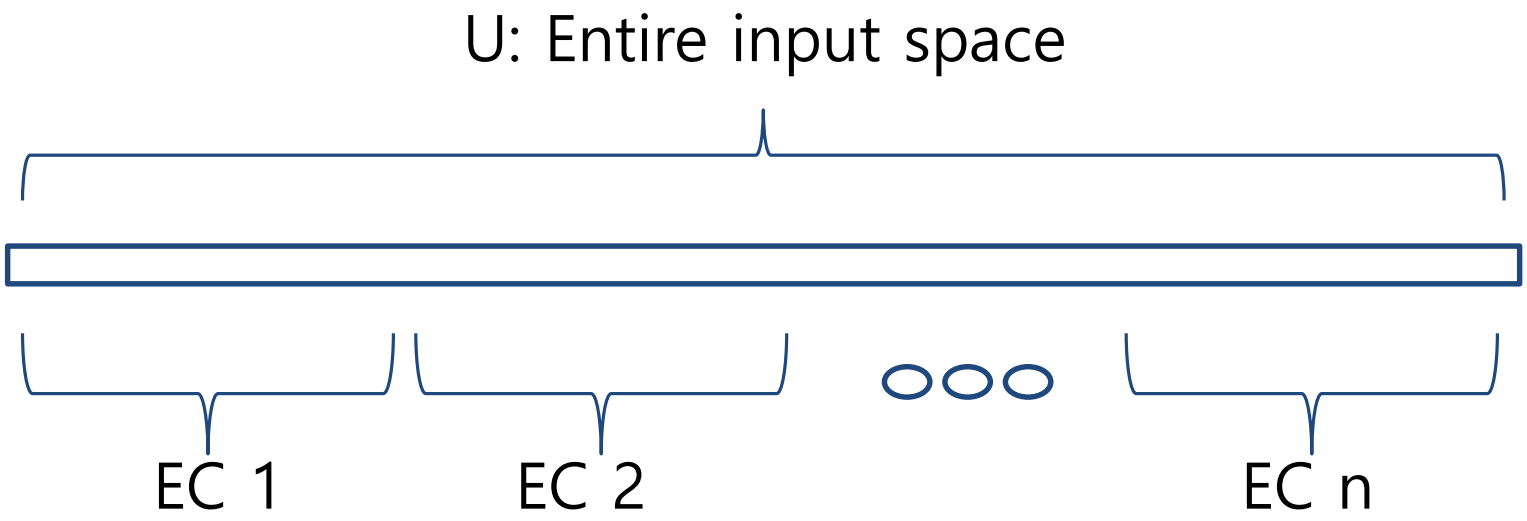
Single Function

- State-based Testing
- Use Case Testing

Sequence of Functions

Equivalence Partitioning

- Partition input space into a small number of equivalence classes



- n << |U|

Equivalence Partitioning: An Example

❖ Given the following specification,

Test Item

- **입장방식 determine입장방식(int age)**
- **입장방식 = {입장불가, 부모동반요구, 단독입장가능}**

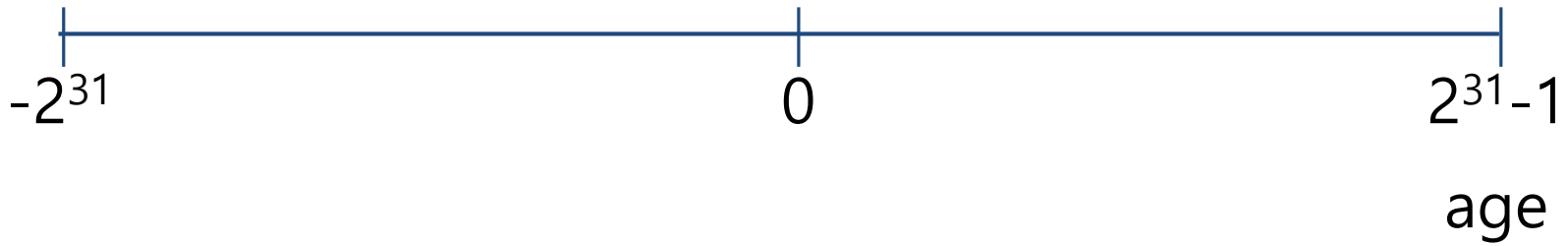
Specification

- 나이가 10세 이하이면 입장 불가
- 나이가 11-17이면 부모 동반 요구
- 나이가 18이상 이면 단독 입장 가능

Equivalence Partitioning

- Step 1: equivalence class identification
  - Divide the input space into **equivalence class**

- The entire input space: 2^32

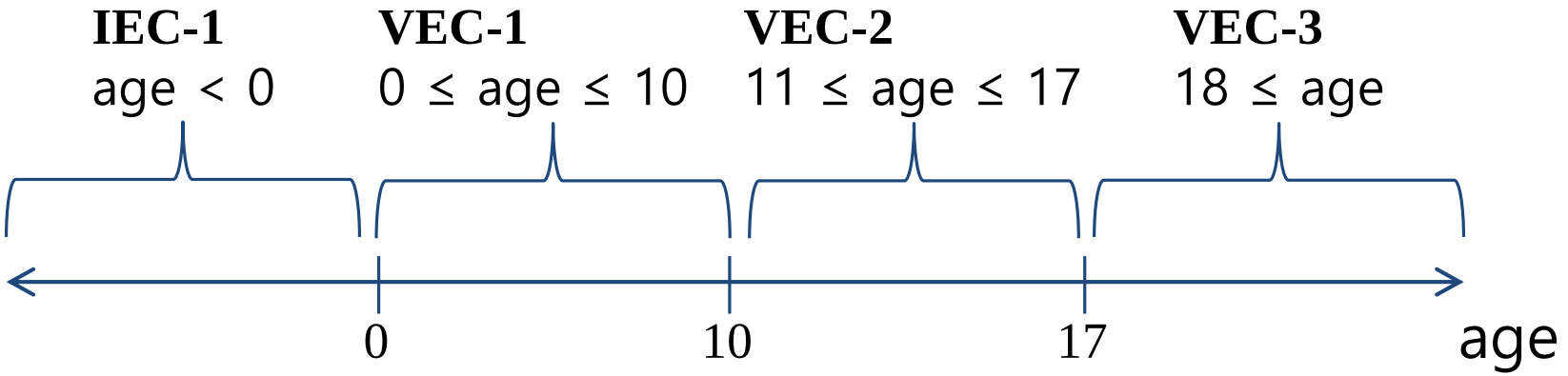


Equivalence Partitioning

Specification(Refined)

- 나이가 0 미만이면 — InvalidAgeException
- 나이가 10세 이하이면 — 입장 불가
- 나이가 11-17이면 — 부모 동반 요구
- 나이가 18 이상이면 — 단독 입장 가능

❖ Identify three valid and one invalid equivalence classes



Equivalence Partitioning

❖ Items in an equivalence class are considered the same in terms of defect detection capability

Specification

- 나이가 10세 이하이면
  - 입장 불가
- 나이가 11-17이면
  - 부모 동반 요구
- 나이가 18 이상이면
  - 단독 입장가능

```
입장방식 determine입장방식(int age) {
    입장방식 result ;
    if ( age <= 10 )
        result = 입장불가 ;
    else if ( age <= 17 )
        result = 부모동반입장 ;
    else
        result = 단독입장가능 ;
    return result ;
}
```


Equivalence Partitioning

❖ Step2: Test Case Selection

- Pick at least one element for each class which can reduce the total number of test cases

동등클래스	유형	조건	테스트 입력	예상 결과
VEC-1	Valid	0 ≤ age ≤ 10	5	입장불가
VEC-2	Valid	11 ≤ age ≤ 17	14	부모동반입장
VEC-3	Valid	age >= 18	30	단독입장가능
IEC-1	Invalid	age < 0	-5	InvalidAgeException

Implementation

```
public enum 입장방식 { 입장불가, 부모동반입장, 단독입장가능 }
```

```
public interface IEntrance {
    public abstract 입장방식 determine입장방식(int age);
}
```

```
public class InvalidAgeException extends RuntimeException {
    private int age;
    public InvalidAgeException(int age) { this.age = age ; }
}
```

```
public class Entrance implements IEntrance {
  public 입장방식 determine입장방식(int age) {
      [redacted]
  }
}
```


JUnit Test Script

```
public class EntranceTest_EquivalencePartitioning {
  private IEntrance entrance ;
  @Before
  public void setUp() throws Exception {
    entrance = EntranceFactory.getInstance(EntranceCodeKind.ORIGINAL) ;
  }
  @After
  public void tearDown() throws Exception { entrance = null ; }
  @Test
  public void test_입장불가() {
    입장방식 expected = 입장방식.입장불가 ;
    int age = 5 ;
    입장방식 actual = entrance.determine입장방식(age) ;

    assertEquals(expected, actual) ;
  }

  …
}
```


Practice: 2차 방정식

❖2차 방정식

Test Item

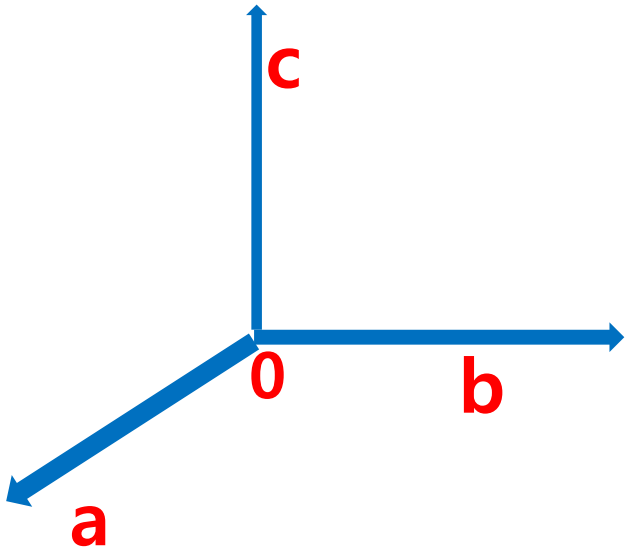
- void quadratic(int a, int b, int c, Solution* s1, Solution* s2)

Specification

- 2차 방정식이 아니면 InvalidEquation 예외 발생
- 2차 방정식이면 2개의 실근, 1개의 중근, 또는 2개의 허근 계산

Practice: 2차 방정식

❖ Step 1: Equivalence Class Identification



동등클래스 ID	유형	조건
VEC-1	Valid	b^2 – 4ac > 0
VEC-2	Valid	b^2 – 4ac = 0
VEC-3	Valid	b^2 – 4ac < 0
IEC-1	Invalid	a = 0

Practice: 2차 방정식

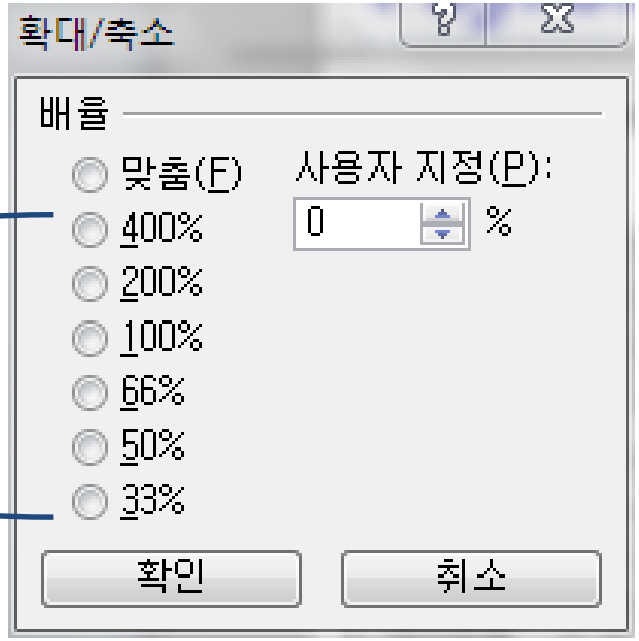
❖Step 2: Test Case Selection

동등클래스 ID	유형	조건	테스트 입력			예상 결과	
			a	b	c	s1	s2
VEC-1	Valid	b^2 – 4ac > 0	1	3	2	-1	-2
VEC-2	Valid	b^2 – 4ac = 0	1	-2	1	1	1
VEC-3	Valid	b^2 – 4ac < 0	1	2	2	-1-i	-1+i
IEC-1	Invalid	a = 0	0	DC	DC	InvalidEquation 예외 발생	

Practice: PowerPoint – 보기 – 확대/축소

- 확대/축소
  - 배율: [10..400]
- A total of 391 cases for Exhaustive approach

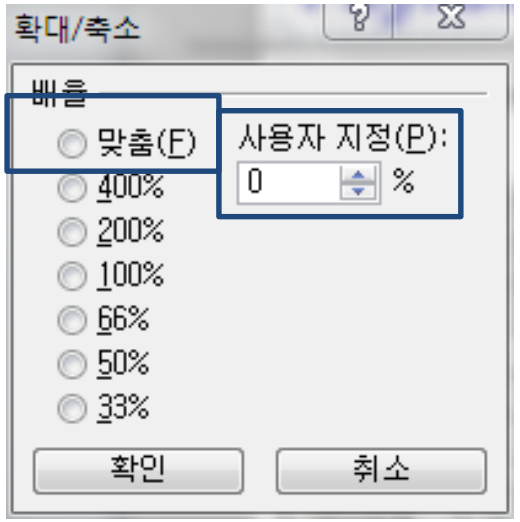
Input-based Partitioning

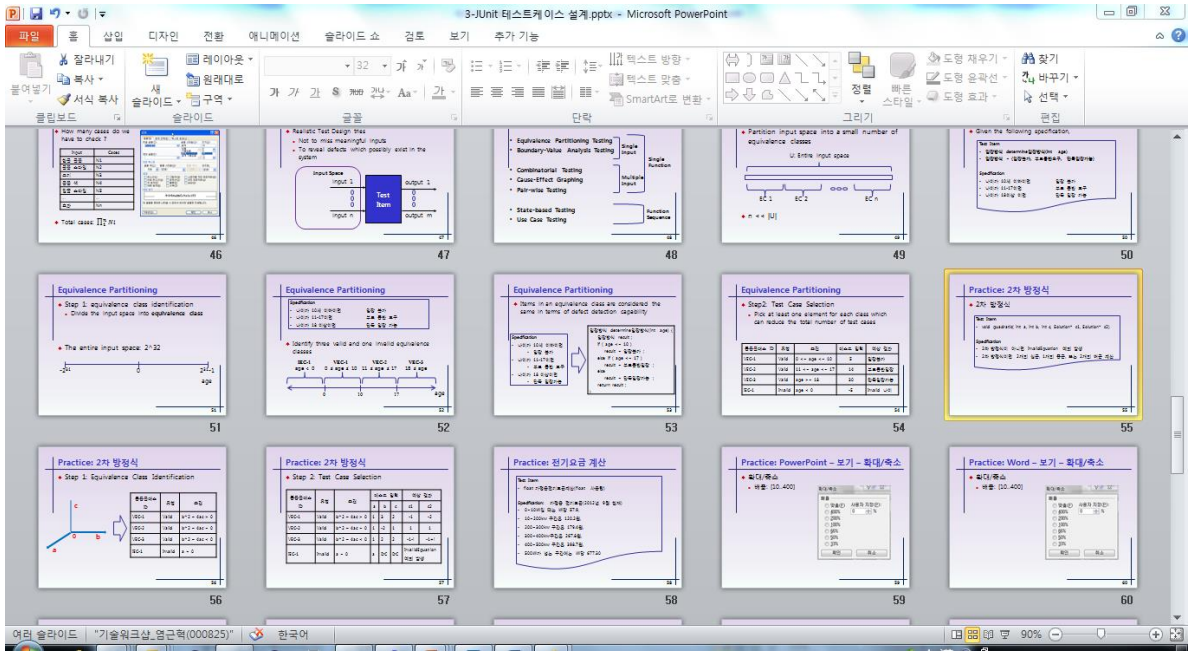


Practice: PowerPoint – 보기 – 확대/축소

❖Output-based Partitioning

출력	EC1	EC2	EC3	EC4	ECn
가로	1	2	1	2	..
세로	1	1	2	2	..



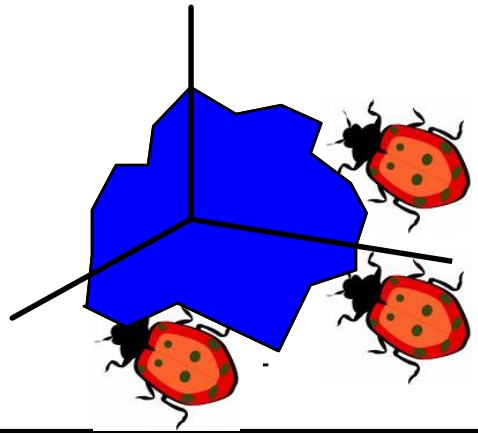


Boundary Value Analysis

- Select elements
  - near the boundary of an equivalence class
  - rather than selecting arbitrary element in the equivalence class
- Rationale
  - Programmers often make mistakes on the boundaries of the equivalence classes/input domain
  - A simple method with a high pay off in finding defects

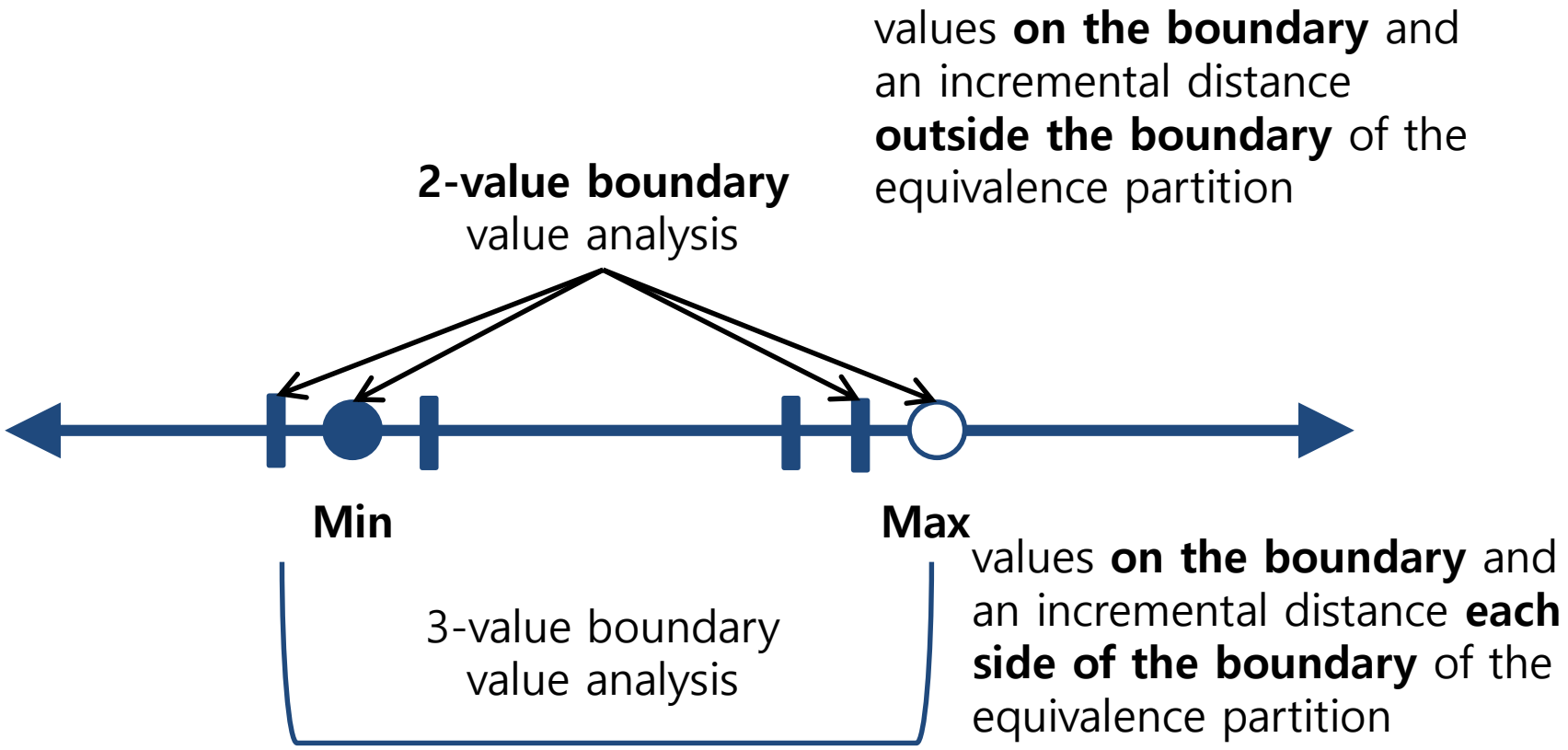
"Bugs lurk in corners and congregate at boundaries."

Boris Beizer



Boundary Value Analysis

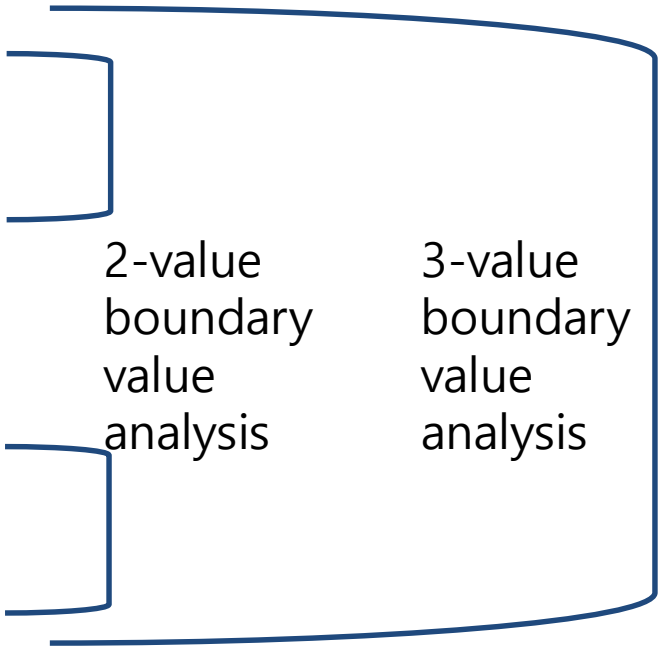
- Equivalence class: [Min .. Max)
- 예) 100 <= X < 300



Boundary Value Analysis: 기본 개념

- ❖ Equivalence class: [Min .. Max)
- ❖ 예) 100 <= X < 300

경계	입력 값	예시	유형
Min	**Min보다 바로 작은 값**	**99**	**Invalid**
	Min 값	**100**	**Valid**
	Min보다 바로 큰 값	101	Valid
Max	Max보다 바로 작은 값	298	Valid
	Max 값	**299**	**Valid**
	Max보다 바로 큰 값	**300**	**Invalid**



Boundary Value Analysis: Example

❖ 속도가 70 이상 100 이하이면 합법적 속도

- Specification: 70≤ 속도(S) ≤ 100
- 테스트 데이터 = {69, 70, 71, 85, 99, 100, 101}

❖ Test cases

Boundary	테스트케이스	
	입력	Expected Output
70	69	Illegal
	70	Legal
	71	Legal
	85	Legal
100	99	Legal
	100	Legal
	101	Illegal

Boundary Value Analysis: Example

- Specification: 70≤ 속도(S) ≤ 100

Test Input Data	69	70	71	85	99	100	101
Expected	I	L	L	L	L	L	I
Common Faulty code	Actual Output						
70 < S ≤ 100	I	I	L	L	L	L	I
69 ≤ S ≤ 100	L	L	L	L	L	L	I
70 ≤ S ≤ 99	I	L	L	L	L	I	I
S ≤ 100	L	L	L	L	L	L	I
70 ≤ S	I	L	L	L	L	L	L

Boundary Value Analysis - 입장방식

❖ Step 1: Equivalence Class Identification

Equivalence Classes
- **VEC-1** : 0 ≤ age ≤ 10
- **VEC-2** : 11 ≤ age ≤ 17
- **VEC-3** : 18 ≤ age
- **IEC-1** : age < 0

Equivalence Class ID	Type	Condition
VEC-1	Valid	0 <= age <= 10
VEC-2	Valid	11 <= age <= 17
VEC-3	Valid	age >= 18
IEC-1	Invalid	age < 0

Boundary Value Analysis - 입장방식

❖ Step 2: select boundary values for each class

Equivalence Classes
- **VEC-1** : 0 ≤ age ≤ 10
- **VEC-2** : 11 ≤ age ≤ 17
- **VEC-3** : 18 ≤ age
- **IEC-1** : age < 0

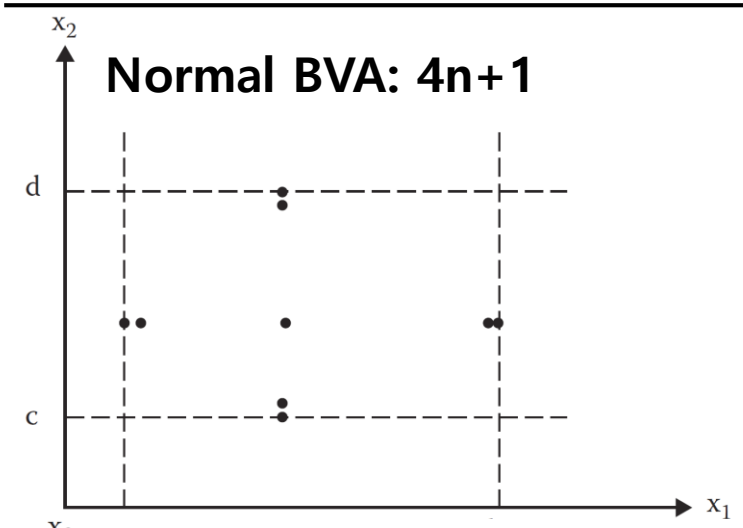
3-value boundary value analysis

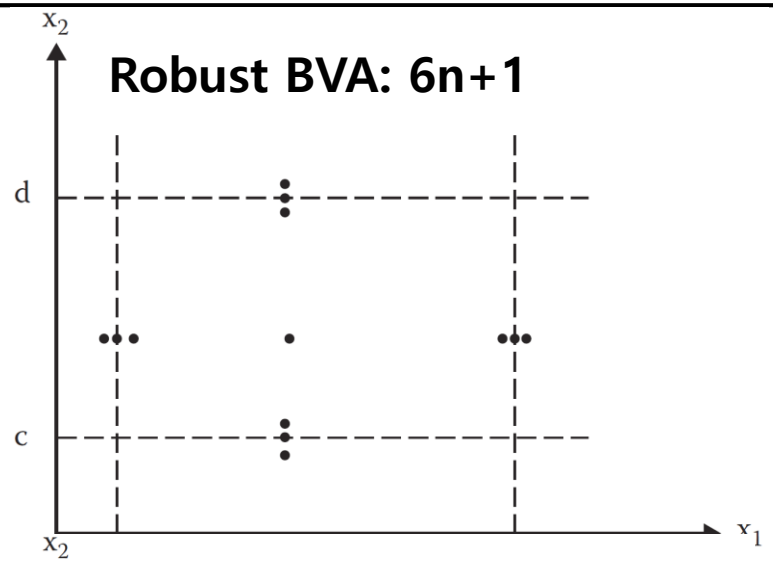
Equivalence Class ID	Inputs		Expected Result
	BVA	EP	
VEC-1	**0, 1, 9, 10**	5	입장불가
VEC-2	**11, 12, 16, 17**	14	부모동반요구
VEC-3	**18, 19**	30	단독입장가능
IEC-1	**-1, -2**	-5	Invalid 나이

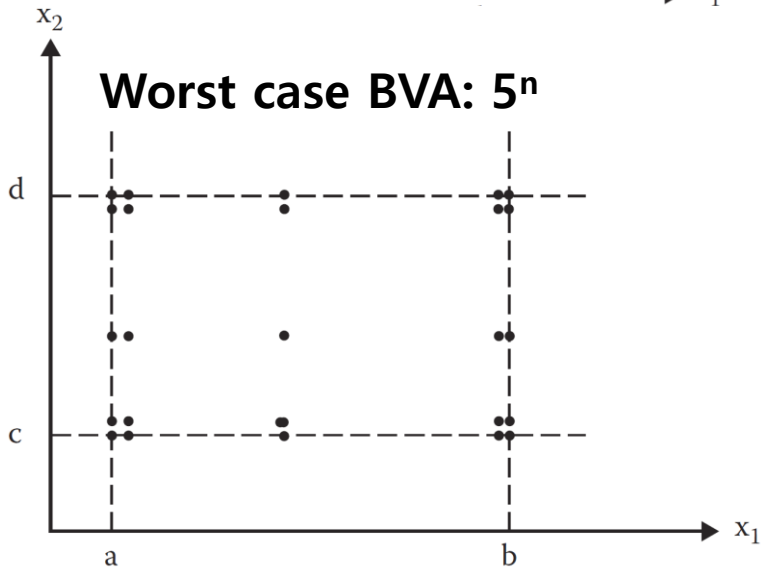
JUnit Test Script

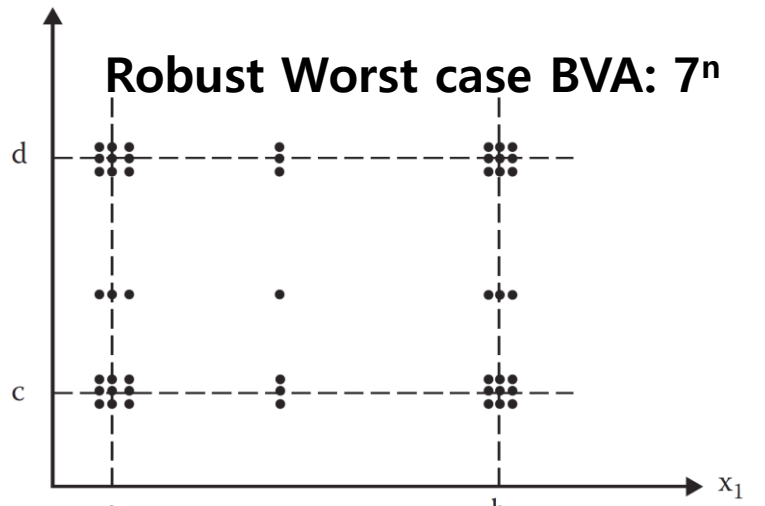
```
public class EntranceTest_BVA {
  private IEntrance entrance ;
  @Before
  public void setUp() throws Exception {
    entrance = EntranceFactory.getInstance
      (EntranceCodeKind.MUTATED_FOR_BOUNDARY_VALUE) ;
  }
  @After
  public void tearDown() throws Exception { entrance = null ; }
  @Test
  public void test_입장불가() {
    입장방식 expected = 입장방식.입장불가 ;
    int[] ages = {0, 1, 9, 10} ;
    for ( int i = 0 ; i < ages.length ; i ++ ) {
      int age = ages[i] ;
      입장방식 actual = entrance.determine입장방식(age) ;
      assertEquals("Error found with age " + age, expected, actual) ;
    }
  }
  @Test public void test_부모동반입장() { }
  @Test public void test_단독입장가능() { }
  @Test(expected=InvalidAgeException.class) public void test_Invalid나이() { }
}
```


BVA Extensions









, Software Testing- A Craftsman's Approach

Combinatorial Testing

❖ 예제: payroll

함수: float payroll(String id, String workhours)	
인자	id: 직원의 사번, 5자리의 수 workhours: 근로시간
기능 설명	40시간까지는 $6/시간 이후에는 $9/시간

Combinatorial Testing

- Step 1: EC Identification for each input

인자	설명
Id(ID)	직원의 사번, 5자리의 수
Workhours(H)	근로시간

동등 클래스 ID	유형	설명
V1	Valid	5 digit, good id
I1	Invalid	5 digit, bad id
I2		More than 5 digit
I3		Fewer than 5 digit
I4		Non digit

동등 클래스 H	유형	설명
V2	Valid	[0, 40]
V3		(40, MAX)
I5	Invalid	less than 0
I6		greater than MAX
I7		Non-float

Combinatorial Testing

❖ Step 2: Test case selection

Each test case covers

1. as **many** uncovered **valid equivalence classes** as possible and ➔ **minimized EP**

Valid Test case

2. **only one** uncovered **invalid equivalence class ➔ one-to-one EP**

Invalid Test case

Cases / EQ Classes \ Test			TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8	TC9
Valid Class	ID	V1	X	X					X	X	X
	H	V2	X		X	X	X	X			
		V3		X							
Invalid Class	ID	I1			X						
		I2				X					
		I3					X				
		I4						X			
	H	I5							X		
		I6								X	
		I7									X

Combinatorial Testing

EQ Classes \ Test Cases			TC1	TC2	TC3	TC4	TC5	TC6	TC7	TC8	TC9
Valid Classes	ID	V1	X	X					X	X	X
	H	V2	X		X	X	X	X			
		V3		X							
Invalid Classes	ID	I1			X						
		I2				X					
		I3					X				
		I4						X			
	H	I5							X		
		I6								X	
		I7									X

ID-number		
Valid	5 digit, good id	V1
Invalid	5 digit, bad id	I1
	More than 5 digit	I2
	Fewer than 5 digit	I3
	other	I4

Number of hours		
Valid	[0, 40]	V2
	(40, MAX)	V3
Invalid	less than 0	I5
	greater than MAX	I6
	other	I7

ID	Input		Expected Output	
	ID	H	name	pay
TC1	15632	35.0	Sam	210
TC2	42567	42.3	Sue	260.70
TC3	99999	31.5	Id error	
TC4	135774	37.3	Id error	
TC5	4465	28.5	Id error	
TC6	15632	35.0	Id error	
TC7	15632	-2	Hour error	
TC8	15632	300	Hour error	
TC9	15632	ten	Hour error	

Practice: FTP 클라이언트 프로그램

- –p 옵션
  - 서버 접속을 위한 포트를 지정
  - 지정할 수 있는 포트 번호를 1024 – 65535로 제한
- –r 옵션
  - 서버로의 접속 시도 횟수 지정
  - 지정할 수 있는 시도 횟수를 0 – 65535 로 제한
    - ✓0: 무한 시도
    - ✓그 외의 수: 지정된 횟수만큼만 시도

Practice: FTP 클라이언트 프로그램

❖ Step 1: EC Identification

변수	동등 클래스	동등 클래스 유형
p (포트)	p<= 1023	Invalid
	1024 <= p <= 65535	Valid
	p >= 65536	Invalid
r (재시도횟수)	r < 0	Invalid
	r = 0	Valid
	1 <= r <= 65535	Valid
	r >= 65536	Invalid

Practice: FTP 클라이언트 프로그램

❖ Step 2: Test case selection

동등 클래스	변수	동등 클래스	T1	T2	T3	T4	T5	T6
Valid	p	1024 <= p <= 65535	X	X			X	X
	r	r = 0	X		X			
		1 <= r <= 65535		X		X		
Invalid	p	p<= 1023			X			
		p >= 65536				X		
	r	r < 0					X	
		r >= 65536						X

Pair-wise 방식

❖ Example

함수: void format(type, size, method, filesystem)	
Type	Primary, Logical, Single, Span
Size	10, 100, 500, 1000
Method	Quick, Slow
Filesystem	FAT, FAT32, NTFS

- All the possible combinations = 4 * 4 * 2 * 3 = 96

Pair-wise 방식

- Every pair of two parameters are considered

Number	Type	Size	Method	File system
1	Logical	500	Slow	FAT
2	Single	10	quick	FAT
3	Primary	100	quick	FAT
4	Span	10	slow	FAT
5	Logical	10	slow	FAT
6	Primary	500	quick	NTFS
7	Span	1000	quick	FAT32
8	Single	100	slow	FAT
9	Span	100	quick	FAT
10	Single	1000	slow	NTFS
11	Logical	500	slow	FAT32
12	Primary	1000	slow	FAT32
13	Logical	100	Slow	FAT
14	Logical	1000	quick	NTFS
15	Span	500	quick	NTFS
16	Primary	1000	Slow	FAT
17	Single	500	quick	FAT32

Pict: Pairwise Testing Tool

- **P**airwise **I**ndependent **C**ombinatorial **T**esting tool
- http://download.microsoft.com/download/f/5/5/f55484df-8494-48fa-8dbd-8c6f76cc014b/pict33.msi

```
C:₩Program Files₩PICT>pict
Pairwise Independent Combinatorial Testing

Usage: pict model [options]

Options:
 /o:N     - Order of combinations (default: 2)
 /d:C     - Separator for values  (default: ,)
 /a:C     - Separator for aliases (default: |)
 /n:C     - Negative value prefix (default: ~)
 /e:file - File with seeding rows
 /r[:N]   - Randomize generation, N - seed
 /c       - Case-sensitive model evaluation
 /s       - Show model statistics
```


Pairwise Testing: Practice

- 일반적인 워드 프로세서에서 텍스트의 글꼴을 설정하기 위한 기능이 제공된다.
- 선택된 문자열에 대하여 글꼴 이름, 글꼴 크기, 글꼴 스타일, 글꼴 효과를 지정할 수가 있다.
- 각 요소 별로 선택 가능한 값들은 다음과 같다.

요소 유형	선택 가능한 값들
이름	고딕, 맑은고딕, 궁서, 바탕
크기	10, 14, 18, 22, 28
스타일	보통, 굵게, 기울임
효과	위첨자, 아래첨자

- 조건 1: 글꼴의 이름이 "궁서" 일 때는 위 첨자는 테스트하지 않는다.
- 조건 2: 글꼴의 크기가 18 이상 일 때는 스타일 중에서 굵게와 기울임은 반드시 테스트 한다.

Model File: Font.txt

```
이름:          고딕, 맑은고딕, 궁서, 바탕
크기:          10, 14, 18, 22, 28
스타일:        보통, 굵게, 기울임
효과:          위첨자, 아래첨자

# 글꼴의 이름이 "궁서" 일 때는 위 첨자는 테스트하지 않는다.
IF [이름] = "궁서" THEN [효과] <> "위첨자";

# 글꼴의 크기가 18 이상일 때는 스타일 중에서
# 굵게와 기울임을 반드시 테스트 한다.
IF [크기] > 18 THEN [스타일] IN {"굵게", "기울임"} ;
```


Pict 실행 결과

% pict Font.txt

이름	크기	스타일	효과
궁서	28	굵게	아래첨자
바탕	22	굵게	위첨자
바탕	28	기울임	위첨자
궁서	10	기울임	아래첨자
맑은고딕	14	보통	위첨자
바탕	14	보통	아래첨자
맑은고딕	10	굵게	위첨자
바탕	10	보통	위첨자
고딕	10	보통	위첨자
맑은고딕	28	기울임	아래첨자
고딕	14	기울임	아래첨자
고딕	22	굵게	아래첨자
궁서	18	보통	아래첨자
고딕	28	굵게	위첨자
고딕	18	굵게	위첨자
바탕	18	기울임	아래첨자
맑은고딕	22	기울임	위첨자
궁서	14	굵게	아래첨자
궁서	22	굵게	아래첨자
맑은고딕	18	굵게	아래첨자

Pairwise 테스트: Example

- **Deployment(or configuration) testing**: to ensure that software can execute correctly on a variety of platforms

client server

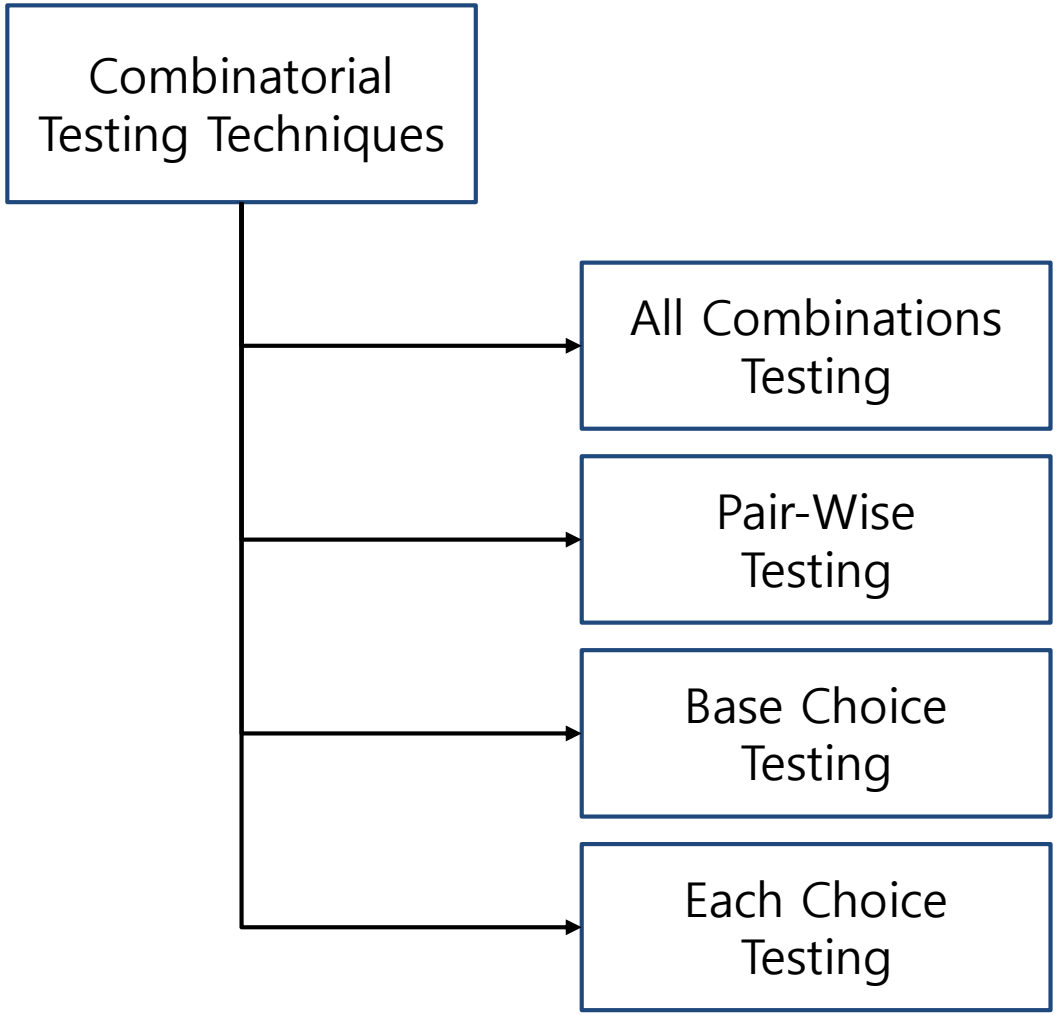
Client		Server	
OS	Browser	OS	Web Server
Windows 98 Windows ME Windows XP	IE Firefox Chrome	Windows NT Windows 2000 Linux	IIS Apache WebLogic

Pairwise 테스트: Example

- All the possible combinations = 3 * 3 * 3 * 3 = 81
- Based on pairwise testing, 9 configurations could be considered.

TC	Server OS	Web Server	Client OS	Browser
1	WinNT	IIS	Win98	IE
2	WinNT	Apache	WinME	Firefox
3	WinNT	WebLogic	WinXP	Chrome
4	Win2K	IIS	WinME	Chrome
5	Win2K	Apache	WinXP	IE
6	Win2K	WebLogic	Win98	Firefox
7	Linux	IIS	WinXP	Firefox
8	Linux	Apache	Win98	Chrome
9	Linux	WebLogic	WinME	IE

Combinatorial Testing in IEEE 29119



Combinatorial Testing: Example

Travel preferences is chosen through three sets of radio buttons

Destination = Paris, London, Sydney
Class = First, Business, Economy
Seat = Aisle, Window

Test Conditions: P(Parameter)-V(Value) Pair

TCOND1: Destination	– Paris
TCOND2: Destination	– London
TCOND3: Destination	– Sydney
TCOND4: Class	– First
TCOND5: Class	– Business
TCOND6: Class	– Economy
TCOND7: Seat	– Aisle
TCOND8: Seat	– Window

All Combinations

❖ Unique combinations of P-V pairs

TC1: Destination – Paris,	Class – First,	Seat – Aisle
TC2: Destination – Paris,	Class – First,	Seat – Window
TC3: Destination – Paris,	Class – Business,	Seat – Aisle
TC4: Destination – Paris,	Class – Business,	Seat – Window
TC5: Destination – Paris,	Class – Economy,	Seat – Aisle
TC6: Destination – Paris,	Class – Economy,	Seat – Window
TC7: Destination – London,	Class – First,	Seat – Aisle
TC8: Destination – London,	Class – First,	Seat – Window
TC9: Destination – London,	Class – Business,	Seat – Aisle
TC10: Destination – London,	Class – Business,	Seat – Window
TC11: Destination – London,	Class – Economy,	Seat – Aisle
TC12: Destination – London,	Class – Economy,	Seat – Window
TC13: Destination – Sydney,	Class – First,	Seat – Aisle
TC14: Destination – Sydney,	Class – First,	Seat – Window
TC15: Destination – Sydney,	Class – Business,	Seat – Aisle
TC16: Destination – Sydney,	Class – Business,	Seat – Window
TC17: Destination – Sydney,	Class – Economy,	Seat – Aisle
TC18: Destination – Sydney,	Class – Economy,	Seat – Window

Pair-Wise

❖ the unique pairs of P-V pairs for different parameters

TC1: Destination – Paris,	Class – First,	Seat – Aisle
TC2: Destination – Paris,	Class – Business,	Seat – Window
TC3: Destination – Paris,	Class – Economy,	Seat – Aisle
TC4: Destination – London,	Class – First,	Seat – Aisle
TC5: Destination – London,	Class – Business,	Seat – Window
TC6: Destination – London,	Class – Economy,	Seat – Aisle
TC7: Destination – Sydney,	Class – First,	Seat – Window
TC8: Destination – Sydney,	Class – Business,	Seat – Aisle
TC9: Destination – Sydney,	Class – Economy,	Seat – Window

Each Choice and Base Choice

- Each choice(or 1-wise) testing: the set of P-V Pairs

TC1: Destination – Paris,	Class – First,	Seat – Aisle
TC2: Destination – London,	Class – Business,	Seat – Window
TC3: Destination – Sydney,	Class – Economy,	Seat – Aisle

- Base choice testing: select "base-choice" value and substitute one P-V pair until all P-V pairs covered

TC1: Destination – London,	**Class – Economy,**	**Seat – Window**
TC2: Destination – **Paris**,	Class – Economy,	Seat – Window
TC4: Destination – **Sydney**,	Class – Economy,	Seat – Window
TC4: Destination – London,	Class – **First**,	Seat – Window
TC5: Destination – London,	Class – **Business**,	Seat – Window
TC6: Destination – London,	Class – Economy,	Seat – **Aisle**

Q&A