



# 11장. 블랙박스 테스트



# Contents

1. 블랙박스 테스트 개요
2. 시나리오 기반 테스트
3. 테스트 단계
4. 테스트 도구

# 1. 블랙박스 테스트 개요 (1/3)

## ● 명세기반 테스트

- 원시 코드는 보지 않고, 목적코드를 실행시켜 결함을 발견하는 방법
- 소스 코드에 대한 정보 없이 요구사항 명세만 이용하여 테스트 케이스를 생성하는 기법
- 기능 테스트(Functional Testing)

● SW의 특징, 요구사항, 공개된 설계도 등 필요

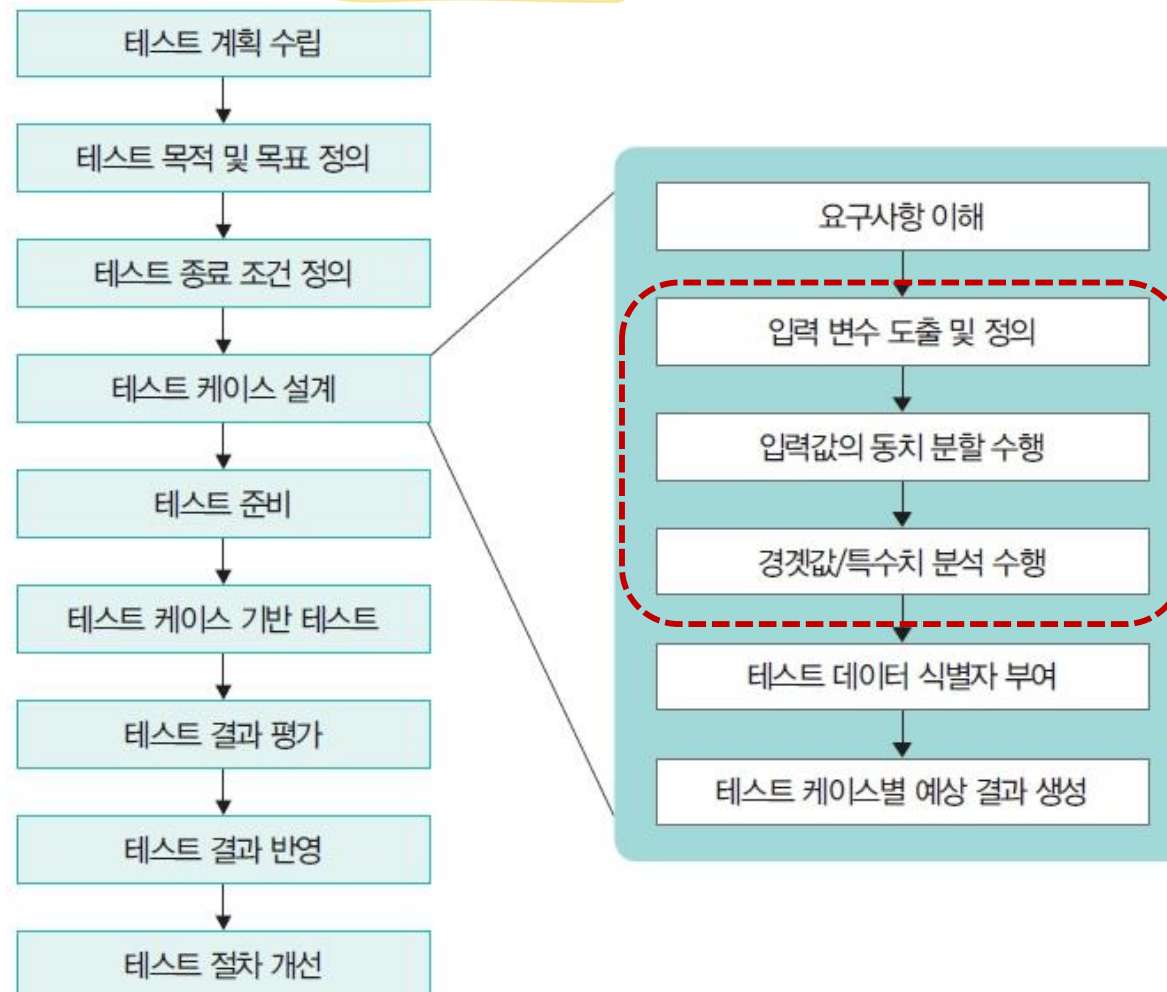
● 단위 테스트, 통합 테스트, 시스템 테스트, 인수 테스트 등 SW 검사의 모든 단계에 적용 가능

# 1. 블랙박스 테스트 개요 (2/3)

- 검출하려는 결함 대상
  - 부정확성, 인터페이스 결함, 자료구조, 성능, 시작과 종료에 대한 결함
- 블랙박스 테스트 기법의 종류
  - 동등분할(Equivalence Partitioning)
  - 경계값 분석(Boundary Value Analysis)
  - 원인-결과 그래프(Cause-Effect Graph)

# 1. 블랙박스 테스트 개요 (3/3)

## ● 블랙박스 테스트 수행 절차



## 1.1 동등분할 기법 (1/4)

- 동등분할(동치분할)
  - 입력 데이터의 영역을 동치 클래스라 불리는 영역으로 분할하여 각 영역으로부터 하나 또는 그 이상의 대표값을 선택하여 테스트케이스로 사용
  - 동치(equivalent)의 의미
    - 각 동치 클래스로부터 선정된 입력 값에 의해 오류가 발견되면 클래스에 속한 다른 값들에 의해서도 동일한 오류가 발견
    - 만약 각 동치 클래스로부터 선정된 입력 값에 의해 오류가 발견되지 못한다면 클래스에 속한 다른 값들에 의해서도 오류가 발견 되지 않아야 함

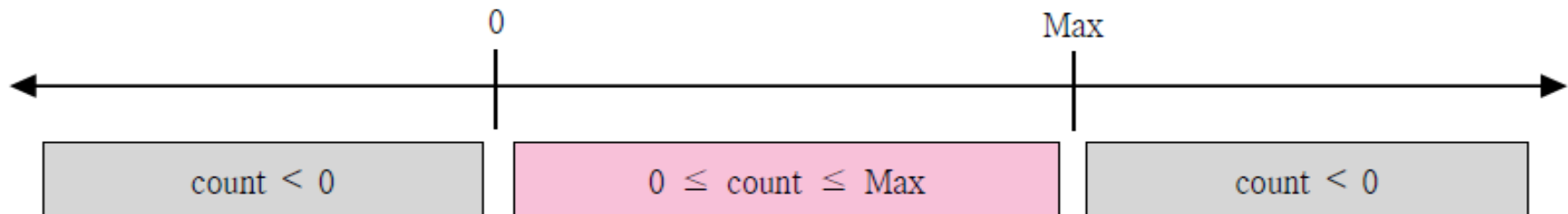
# 1.1 동등분할 기법 (2/4)

## ● 동치 클래스

- 시스템에 의해 동일하게 처리되고 같은 출력 결과를 생산하는 입력 조건 또는 입력 데이터 값들의 모임
- 정상적인 범위의 동치 클래스
- 비정상적인 범위의 동치 클래스

(예) 입력 범위 조건  $0 \leq \text{count} \leq \text{Max}$  의 동치 클래스

- 정상범위  $0 \leq \text{count} \leq \text{Max}$
- 비정상범위  $\text{count} < 0, \text{count} > \text{Max}$



# 1.1 동등분할 기법 (3/4)

- 동등분할기법 수행 절차
  - 명세로부터 입력 변수들을 식별
  - 입력 변수를 동치 클래스로 분할
  - 각 입력 변수의 동치 클래스들을 적절한 조합 연산자를 사용하여 조합
  - 가능하지 못한 조합이 있는지 점검하고 제거
  - 각 조합된 클래스로부터 최소한 하나의 대표 값을 선정하여 테스트 케이스에 반영하여 테스트케이스 테이블을 작성



# 1.1 동등분할 기법 (4/4)

$$(s, n) \rightarrow \begin{matrix} (n, \text{비}n) & (\text{비}n, n) \\ (n, n) & (\text{비}n, \text{비}n) \end{matrix}$$

- (예제) 길이 N인 스트링 s와 정수 n을 입력으로 받아서 s에서 가장 많이 출현하는 문자 n개 찾기

입력	정상적인 동치 클래스	비정상적 동치 클래스
s	EQ1: 숫자를 가진 스트링 EQ2: 소문자를 가진 스트링 EQ3: 대문자를 가진 스트링 EQ4: 특수문자를 가진 스트링 EQ5: 0에서 N사이의 길이를 가진 스트링	IEQ1: ASCII가 아닌 문자 IEQ2: 길이가 N보다 큰 스트링
n	EQ6: 정상 범위의 정수	IEQ3: 정수형 범위를 벗어난 수

- 각 동치클래스에서, s와 n의 쌍으로 테스트케이스를 선정

## 1.2 경계치 분석 (1/3)

동등분할과 같이!

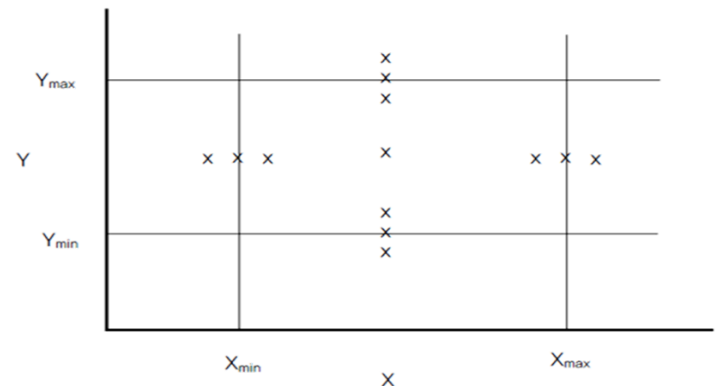
- BVA(Boundary Value Analysis)
  - 오류는 보통 입력 영역의 경계에서 발생하는 경향이 있음
  - 동치 클래스의 경계에 있는 값들을 테스트 케이스로 선정
  - (예) 하나의 입력 값  $x$ 의 범위가  $\min \leq X \leq \max$ 로 주어졌다면, 경계값은  $\min-1$ ,  $\min$ ,  $\min+1$ ,  $\max-1$ ,  $\max$ ,  $\max+1$ 이 됨



# 1.2 경계치 분석 (2/3)

$x_1, y_1, z_1$  일 때  
 ①  $x_1$ 의 경계치  $\rightarrow y_1, z_1$ 는 정상값  
 $\min < x < \max$   
 ②  $y_1$ 의 경계치  $\rightarrow x_1, z_1$ 는 정상값  
 ③  $z_1$ 의 경계치  $\rightarrow x_1, y_1$ 는 정상값  
 +  $x_1, y_1, z_1$  모두 정상값

- (예) 입력 변수가 순서를 갖는 집합, Ordered Set={A, B, C, D, F}로 정의되었다면, 첫 번째 요소와 마지막 요소를 테스트 데이터로 선정
- (예) 변수가 여러 개 일 때
  - 한 변수에 대해 여러 경계값을 선택하면서, 다른 변수는 정상적인 값으로 고정
  - 모든 변수에 대해 범위 안의 정상적인 값을 가진 테스트 케이스를 추가
  - 총  $6n+1$ 개의 테스트케이스 존재



## 1.2 경계치 분석 (3/3)

- (예) 입력 값  $D = \{ \text{정수 } n \mid 0 < n < 100 \}$  인 프로그램의 경우,
  - 정상범위 동치클래스 =  $\{1, 2, \dots, 99\}$
  - 비정상 범위 동치클래스 =  $\{0, -1, -2, \dots, 100, 101, 102\}$

	테스트케이스	값
1	정상범위 동치클래스의 경계	$1 \Rightarrow 0, 1, 2$
2	정상범위 동치클래스의 경계	$99 \Rightarrow 98, 99, 100$
3	비정상범위 동치클래스의 경계	$0 \Rightarrow -1, 0, 1$
4	비정상범위 동치클래스의 경계	$100 \Rightarrow 99, 100, 101$

## 1.3 원인-결과 그래프 (1/6)

- 동치 클래스 분할과 경계값 분석의 단점
  - 각각의 입력을 별도로 생각하므로, 서로 다른  $n$ 개 입력 조건에 대해  $2^n$ 개 조합에 대한 테스트 케이스 필요
- 원인-결과 그래프(Cause-Effect Graph)
  - 입력 조건의 조합을 체계적으로 선택할 수 있음
  - 프로그램의 외부요건을 원인(입력조건)과 결과(처리 또는 출력)의 논리적 관계로 표현
  - 원인-결과 그래프를 기반으로 의사결정표를 작성하여 테스트 케이스 생성

# 1.3 원인-결과 그래프 (2/6)

## ● 의사결정표(Decision Table)

- 처리가 산출하는 출력이 복잡한 의사결정에 의해 좌우 되거나, 광범위한 입력 자료에 대한 의사결정이 필요한 경우 등에 사용되는 의사결정 로직을 나타낸 표

Yes/No, Male/Female

		1	2	3	4	5	6	7	8
입력 변수 조건	조건	연령 > 21	Y	Y	Y	Y	N	N	N
		성별	M	M	F	F	M	M	F
		체중 > 80(kg)	Y	N	Y	N	Y	N	Y
조건에 의해 수행될 수 있는 동작	처리	치료 1	V				V		V
		치료 2		V			V		
		치료 3			V			V	V
		치료 불필요				V		V	

# 1.3 원인-결과 그래프 (3/6)

- 원인-결과 그래프 작성 방법
  - 노드 : 원인(입력조건), 결과(출력조건)
  - 에지 : 노드들 간의 처리 관계를 나타냄

기호	표기법	사용 예	설명
동치(Equal, $-$ )	$\textcircled{A} - \textcircled{B}$	if A then B	A=1이면 B=1, A=0이면 B=0
부정(NOT, $\sim$ )	$\textcircled{A} \sim \textcircled{B}$	if not A then B	A=1이면 B=0, A=0이면 B=1
합(OR, $\vee$ )	$\textcircled{A} \vee \textcircled{B}$	if A or B then C	(A, B, C)에서 (1, 1)=1, (1, 0)=1, (0, 1)=1, (0, 0)=0
곱(AND, $\wedge$ )	$\textcircled{A} \wedge \textcircled{B}$	if A and B then C	(A, B, C)에서 (1, 1)=1, (1, 0)=0, (0, 1)=0, (0, 0)=0

# 1.3 원인-결과 그래프 (4/6)

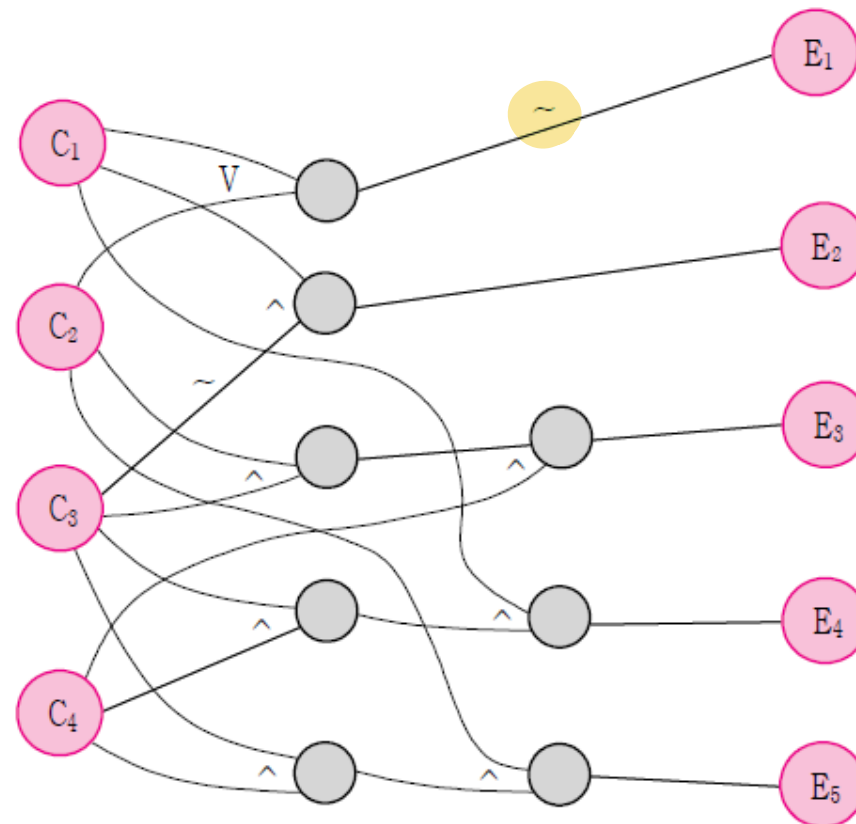
## ● 원인-결과 그래프 예

원인 :

- C1. 명령어가 입금
- C2. 명령어가 출금
- C3. 계좌번호가 정상
- C4. 트랜잭션 금액이 정상

결과 :

- E1. '명령어 오류'를 인쇄
- E2. '계정번호 오류'를 인쇄
- E3. '출금액 오류'를 인쇄
- E4. 트랜잭션 금액 입금
- E5. 트랜잭션 금액 출금



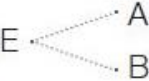
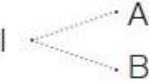
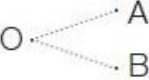




## 1.3 원인-결과 그래프 (5/6)

- 원인-결과 그래프 기법 절차
  1. 프로그램을 적합한 크기로 분할
  2. 원인과 결과 찾기
    - 명세서에서 원인과 결과를 찾아 일련번호와 같은 식별자를 각각 부여
    - 원인은 하나의 입력 조건이고, 결과는 출력 조건이 됨
  3. 원인-결과 그래프를 작성
    - 프로그램 명세가 의미하는 내용을 분석하여, 이에 알맞은 원인과 결과를 연결하는 논리 그래프를 작성

# 1.3 원인-결과 그래프 (6/6)

## 4. 그래프에 제한조건을 표시

명칭	기호	설명	
배타적 <sup>Exclusive</sup> 관계	E 	두 개가 동시에 존재할 수 없다.	A=1이면 B=0, A=0이면 B=1 (A=B=0은 가능, A=B=1은 불가능)
포함 <sup>Inclusive</sup> 관계	I 	적어도 둘 중 한쪽은 성립한다.	A=1일 때 B=1, B=0 B=1일 때 A=1, A=0
선택 <sup>Only one</sup> 관계	O 	항상 한쪽만 성립한다.	A=1이며 B=0, 또는 A=0이며 B=1(A=B=0은 불가능)
필요 <sup>Require</sup> 관계	R 	한쪽이 성립하면 다른 쪽도 성립한다(B가 성립하기 위해서는 A가 반드시 필요하다).	A=1이면 B=1 또는 B=0이고 A=1이면 B=0(B=1은 불가능)
강요 <sup>Mask</sup> 관계	M 	한쪽이 성립하면 다른 쪽은 성립하지 않는다.	A=1이면 B=0

## 5. 의사결정표로 변환

## 6. 테스트 케이스 작성

- 의사결정표의 각 열을 테스트 케이스로 도출

# 1.4 원인-결과 그래프 예제 (1/3)

## ● 문제 : 도어락 비밀번호

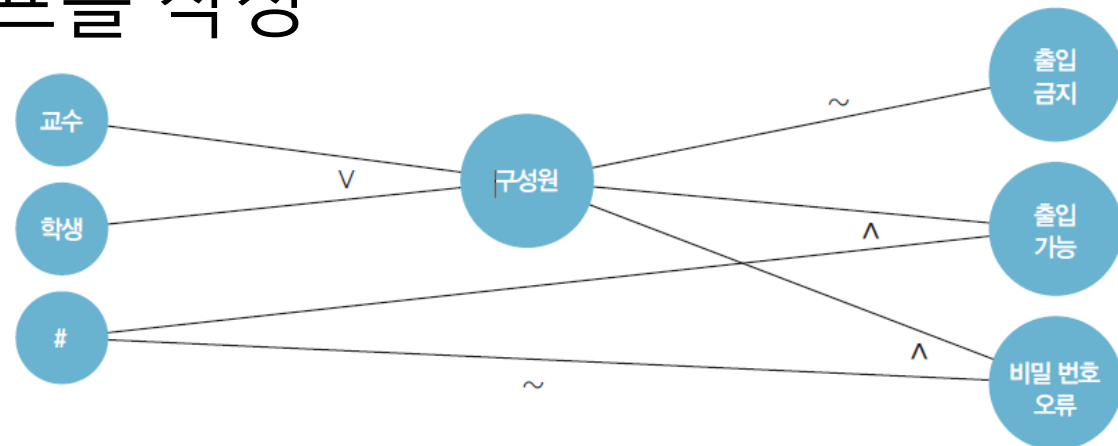
- 첫 번째 열이 P 또는 S로 시작하고, 두 번째 열이 #이면 '출입 가능'을 출력한다.
- 첫 번째 열이 P 또는 S로 시작하지 않으면 '출입 금지'를 출력한다.
- 첫 번째 열이 P 또는 S로 시작하고, 두 번째 열이 #이 아니면 '비밀번호 오류'를 출력한다.
- P는 교수(Professor), S는 학생(Student)을 의미한다.

## ● 원인-결과 찾기

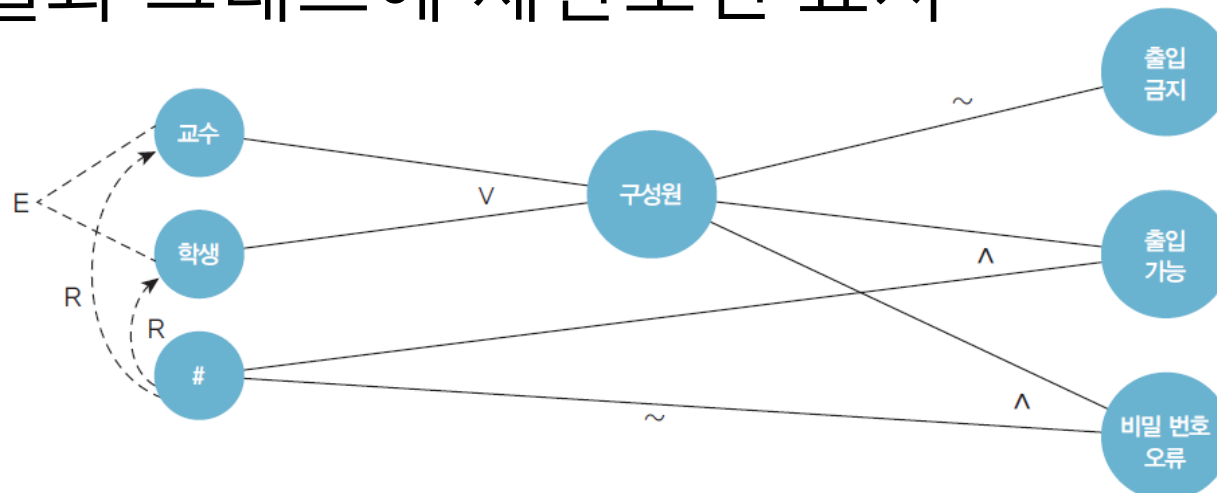
원인	결과
원인 1 : 첫 번째 열이 P	결과 1 : '출입 가능' 출력
원인 2 : 첫 번째 열이 S	결과 2 : '출입 금지' 출력
원인 3 : 두 번째 열이 #	결과 3 : '비밀번호 오류' 출력

# 1.4 원인-결과 그래프 예제 (2/3)

## 2. 원인-결과 그래프를 작성



## 3. 원인-결과 그래프에 제한조건 표시



# 1.4 원인-결과 그래프 예제 (3/3)

## 4. 의사결정 테이블로 변환

테스트 케이스		1	2	3	4	5
입력 항목	원인 1(교수)	F	T	F	T	F
	원인 2(학생)	T	F	F	F	T
	원인 3(#)	T	T	X	F	F
중간 노드	구성원	T	T	F	T	T
출력 값	출입 금지	F	F	T	F	F
	출입 가능	T	T	F	F	F
	비밀번호 오류	F	F	F	T	T

P와 S는 배타적  
관계이므로 동시에  
T가 될 수 없음

원인 3에 대해  
원인 1과 원인2가  
필요관계이므로

## 5. 테스트 케이스 작성

	테스트케이스	Ex
1	결과 '출입금지'에 관련된 원인의 조합	3열로부터 : "B4w2X"
2	결과 '출입가능'에 관련된 원인의 조합	1열로부터 : "S#808"
3	결과 '출입가능'에 관련된 원인의 조합	2열로부터 : "P#12A"
...	...	

## 2. 시나리오 기반 테스트 (1/4)

- 객체지향 방식의 프로그램에 적용
  - 유스케이스 명세로부터 시스템의 액터를 중심으로 시나리오 생성하고, 테스트 케이스 추출

### 1. 액터의 입력과 액션을 파악

- <예> 사용자 등록 사용 사례로부터 입력요소 추출

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 패스워드를 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

사용자 ID, 패스워드, 재입력 패스워드

## 2. 시나리오 기반 테스트 (2/4)

### 2. 입력 값을 결정 : 정상/비정상/예외 값 분류

- <예> 파악된 입력 요소의 값 결정

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	

## 2. 시나리오 기반 테스트 (3/4)

- 테스트 케이스 생성을 위한 입력 값 조합 규칙
  - 테스트 조합이 프로그램 기능과 동작의 정확성을 가진다면 선택
  - 테스트 조합이 오류를 발견할 가능성이 있다면 선택
  - 테스트 조합이 선택된 다른 테스트 조합에 의해 포함될 수 있다면 삭제(중복제거), 유지할지 삭제할지 불분명한 것은 선택

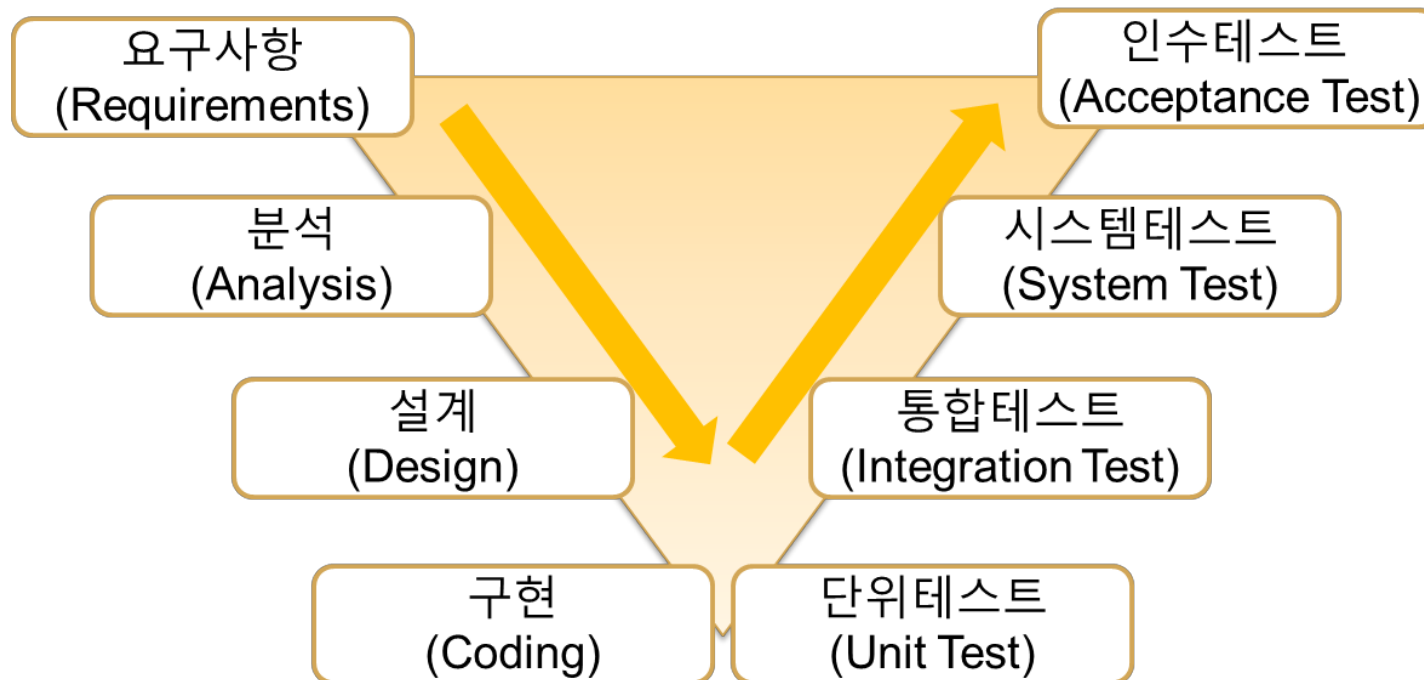


## 2. 시나리오 기반 테스트 (4/4)

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>

### 3. 테스트 단계

- 테스트의 단계



## 3.1 통합 테스트 (1/7)

- 단위 테스트를 통과한 모듈들이 통합되어 실행될 때 발생할 수 있는 문제를 검사하기 위한 활동
  - 여러 개발 팀에서 개발한 각각의 단위 모듈을 대상
  - 모듈-모듈 간의 결합을 테스트
- 모듈의 결합 순서에 따라 방법이 다름
  - 빅뱅(big-bang) 비점진적 : 모든걸 동시에
  - 하향식(top-down)
  - 상향식(bottom-up)
  - 연쇄식(threads)

## 3.1 통합 테스트 (2/7)

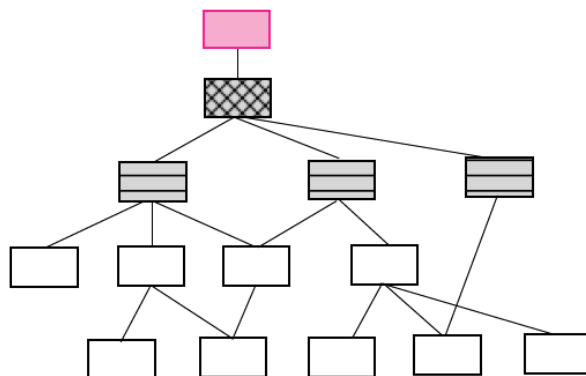
- 빅뱅(비점증적) 통합

- 모든 모듈을 한 번에 통합하여 테스트를 수행하는 방식
- 시스템의 중요 부분과 부수적인 부분을 구별하지 않음
- 장점 : 테스트 실행을 한번에 진행할 수 있음
- 문제점 : 테스트 과정에서 결함이 발견된 경우 결함의 원인이 무엇인지, 즉 어떤 모듈 간의 인터페이스에서 문제가 발생하였는지 식별하기 어려움

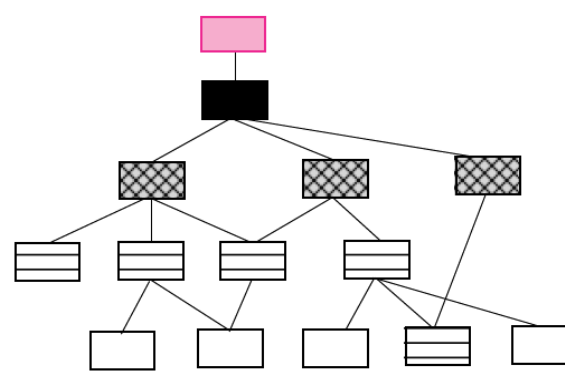
## 3.1 통합 테스트 (3/7)

### 하향식 통합

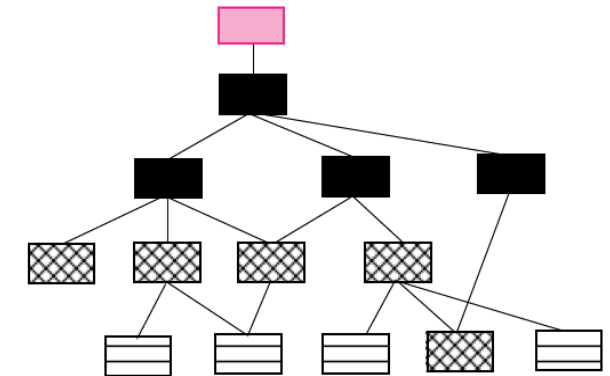
- 시스템 구조상 최상위에 있는 단위 모듈부터 한 번에 하나씩 통합하여 테스트를 수행하는 방식



(a) 1 단계 통합



(b) 2 단계 통합



(c) 3 단계 통합

## 3.1 통합 테스트 (4/7)

### ● 하향식 통합의 장점

- 인터페이스 간에 발생하는 오류의 검출이 매우 쉬움
- 상위층의 중요한 모듈의 인터페이스를 조기에 테스트
- 스텝을 이용하여 시스템 모습을 일찍 구현가능

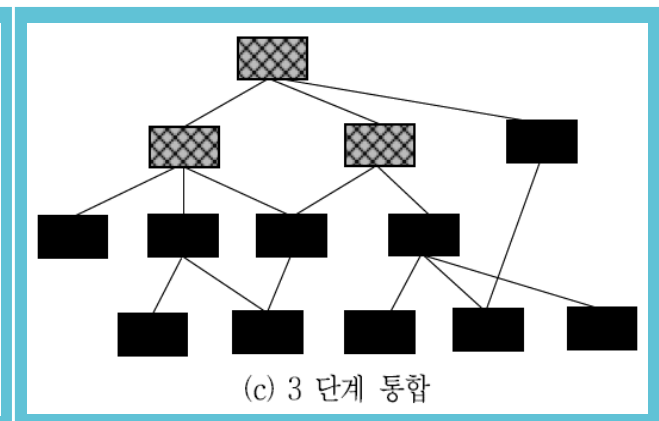
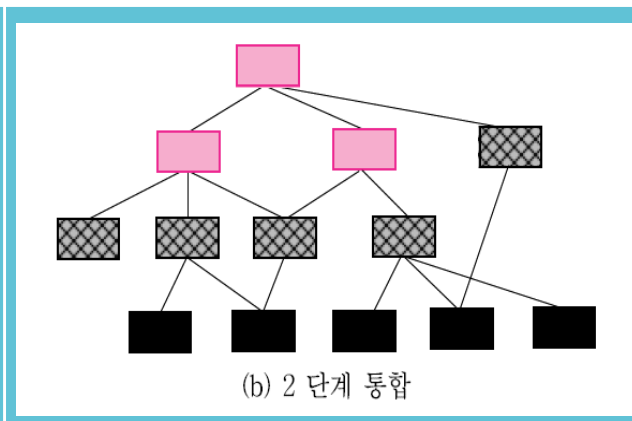
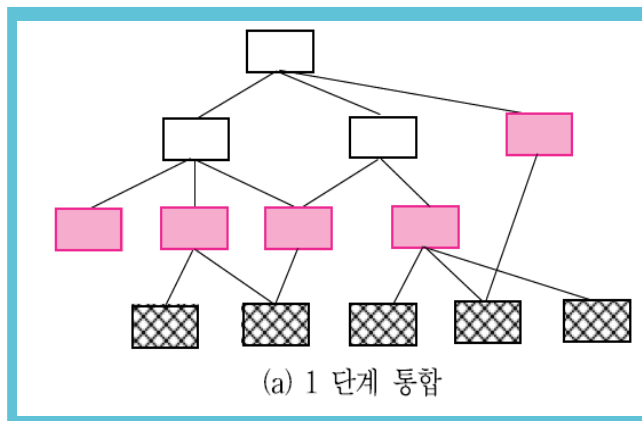
### ● 하향식 통합의 단점

- 통합되지 않은 모듈에 대한 스텝 개발에 대한 노력 필요
- 입출력이나 처리 기능이 마지막에 구현되어 있다면, 중요 모듈에 대한 테스트가 늦어짐

## 3.1 통합 테스트 (5/7)

### ● 상향식 통합

- 시스템 구조상 최하위에 있는 모듈부터 통합



## 3.1 통합 테스트 (6/7)

- 상향식 통합의 장점
  - 최하위의 단위 함수를 기준으로 상위 호출 모듈을 단계적으로 통합하면서 진행하므로 하위층 모듈을 많이 테스트할 수 있음
- 상향식 통합의 단점
  - 통합을 수행하는 후반부에 결함이 발생하는 경우, 결함 수정으로 인해 하위 모듈에 영향을 미칠 수 있기 때문에 테스트를 다시 수행해야 함
  - 초기에 시스템의 골격을 확인하기 어려움



## 3.1 통합 테스트 (7/7)

- 연쇄식 통합 : 스레드 기반 통합
  - 전체 모듈에서 제어의 중심에 있는 스레드(thread)를 기반으로 주변에 있는 모듈을 하나씩 통합하면서 테스트를 수행
- 연쇄식 통합의 장점
  - 시스템을 나누어 개발 하기 용이 : 초반에 시스템의 주요 기능만 수행하도록 만들고, 추후 추가적인 기능을 첨부

## 3.2 시스템 테스트 (1/2)

- 개발된 시스템을 사용될 하드웨어 플랫폼에 설치한 뒤 수행하는 활동
  - 소프트웨어가 제공하는 기능에만 관심이 있는 게 아니라, 성능 등의 품질 측면에도 관심 있음
- 테스트 종류
  - 기능 테스트
  - 성능 테스트
  - 사용성 테스트

## 3.2 시스템 테스트 (2/2)

- 기능 테스트

- 기능적 요구와 구현 상 차이를 발견하기 위한 테스트
- 유스케이스 모델에서 오류를 일으킬만한 유스케이스 인스턴스를 찾아내서 테스트 데이터 선정

- 성능 테스트

- 작업부하, 처리량, 반응시간, 효율성 등을 테스트
- 스트레스 테스트, 성능 테스트, 보안 테스트

- UI 테스트

- UI에 대한 결함이나 오류처리, 데이터 입출력, 디스플레이, 문서와 도움말 등에 대한 결함

11/29

로그인 UC 001 - 로그인 성공 ...  
 002 - ID 입력 확인  
 ⋮

## 3.3 인수 테스트 (1/2)

( expected result : 테스트 케이스대로 했을 때 예상 결과  
 actual result : 지능작성불가 → 뒤의 두 칸은 바꿔주기

- 개발 소프트웨어 시스템을 사용자에게 전달하기 전에 수행하는 활동, 구축된 시스템에 각 요구사항이 올바르게 구현되었는지 점검하는 **데모 형식으로 테스트** 진행 : 원래는 사용자가 해야 함
- 시스템을 당장 사용할 수 있도록 모든 준비가 되어 있는지 확인

- 개발자가 데모해줌 → 개발자를 제외한 의뢰자 또는 대리인이 테스트 수행
- 실제 업무 절차를 따라 테스트 수행

## 3.3 인수 테스트 (2/2)

### ● 테스트 유형

#### • 알파 테스트

사용자 환경처럼  
만들어서 개발자가?

사용자 환경에서 X

- 선택된 사용자가 개발 환경에서 시험하는 것
- 개발자가 소프트웨어를 테스트하는 사용자를 모니터하면서 오류와 문제점을 기록

#### • 베타 테스트 : 실제인사테스트 (베타버전 배포)

개발자가 설치해줌 →

- 고객의 사용 환경에서 시험하는 것(필드 테스트)
- 개발자 없이 사용 환경에 소프트웨어를 설치하여 시험
- 고객이 문제점을 개발자에게 보고

## 3.4 회귀 테스트 (1/3)

- 개발된 소프트웨어 혹은 운영 중인 소프트웨어에 대해 기능을 추가하거나 결함을 제거한 후 해당 소프트웨어 테스트하는 활동
- 회귀 테스트 유형
  - 전수 회귀(Reset All) 테스트
  - 선택적 회귀 테스트
  - 우선순위 기반 회귀 테스트

10개 모듈 중 1개를 고치면

· 원칙적으로는 10개 모두 테스트

· 비용문제 → 테스트 범위 설정 필요

## 3.4 회귀 테스트 (2/3)

- ① ● 전수 회귀(Reset All) 테스트 *전부다*
  - 본래 SW 테스트를 수행했던 모든 테스트 케이스와 추가된 기능에 대한 테스트 케이스로 테스트 수행
  - 수정 변경으로 발생 가능성이 있는 모든 경우에 대해 테스트 *단* 테스트 비용이 더 들어갈 수 있으나, *장* 테스트 커버리지가 향상되고 완전성 향상됨 *→ 안정성!*
  - 고위험 시스템에 활용
- ② ● 선택적 회귀 테스트 *영향범위안에 있는것만*
  - 변경 대상 위주로 영향범위를 결정해서 테스트
  - *장* 테스트 수행범위 최소화 및 투자대비 효과적이지만 *단* 테스트 완전성 부족

## 3.4 회귀 테스트 (3/3)

- ⑦ 우선순위 기반 회귀 테스트 변경된 것 + 중요한 것 (우선순위↑)
  - 변경·추가된 부분을 포함하여 소프트웨어를 구성하는 핵심 기능 위주로 우선순위를 정하여 테스트 수행
- ⑧ 중요도와 결함 위험 가능성을 기반으로 테스트를 수행하기 때문에 비용 효과적임
- ⑨ 낮은 우선순위에 대한 변경 영향을 고려하지 못하는 상황이 발생 가능
  - 위험도가 낮은 시스템 일반적인 시스템



## 4. 테스트 도구

- 테스트 작업을 자동화
- 도구 종류
  - 코드 분석 도구
  - 테스트 케이스 생성 도구
  - 테스트 케이스 실행 도구
  - 단위 테스트 도구

## 4.1 코드 분석 도구 (1/2)

### ● 정적 분석 도구

- 프로그램을 실행하지 않고 분석
- 코드 분석 도구 : 원시 코드의 문법 검사
- 구조 검사 도구 : 원시코드의 구조적인 결함 확인
- 데이터 분석 도구
  - 원시코드를 검사하여 잘못된 링크나 데이터 정의의 충돌, 잘못된 데이터의 사용을 발견
- 순서 검사 도구
  - 이벤트 순서가 올바른지 체크

### ● SonaQube, (Sparrow, CodePrism)

비산태그

↳ 우리나라가 개발

## 4.1 코드 분석 도구 (2/2)

- 동적 분석 도구 *블랙박스테스팅*
  - 프로그램을 실행하면서 분석
  - 프로그램이 수행되는 동안 이벤트의 상태 파악을 위한 특정한 변수나 조건의 스냅샷(snapshot)을 생성
    - 시스템의 성능 또는 기능에 영향을 주는 분기점(breakpoint) 파악에 도움
  - 모듈의 호출 횟수나 수행된 문장 번호를 리스트로 만들어 줌
- Rational AppScan(IBM), Security Suite(HP) 등

## 4.2 단위 테스트 도구 (1/2)

- XUnit : 테스트 프레임워크 통칭
  - Java : JUnit
  - C++ : CppUnit
  - Python : PyUnit
  - Java Script : JSUnit
  - 다양한 언어로 작성된 프로그램의 단위 테스트와 리그레션 테스트를 지원
- 테스트 케이스와 테스트 결과의 체크를 한데 묶은 패키지
- 테스트 케이스의 반복 실행가능

## 4.2 단위 테스트 도구 (2/2)

### • Junit

- 자바 프로그래밍언어용 단위 테스트 프레임워크
- @Test 어노테이션을 제공하여 매우 쉽고 간결하게  
여기부터  
테스트코드 테스트 코드를 작성 및 실행
- 단정(assert) 메서드로 테스트 케이스의 수행 결과를  
판별

```

package SimpleCalculator;
public class SimpleCalculator {
    private int res = 0;

    public void add(int x, int y) {      res = x + y;      }
    public void sub(int x, int y) {      res = x - y;      }
    public void inc(int d)      {      res += d;          }
    public void dec(int d)      {      res -= d;          }
    public int getResult()      {      return res;        }
}

```

```

package SimpleCalculator;
import static org.junit.Assert.*;
import org.junit.Test;

public class SimpleCalculatorTest {
    @Test
    public void testAdd() throws Exception {
        SimpleCalculator calc = new SimpleCalculator();
        calc.add(10, 20);
        assertEquals(30, calc.getResult());
    }
}

```

12/4 사진

12/4