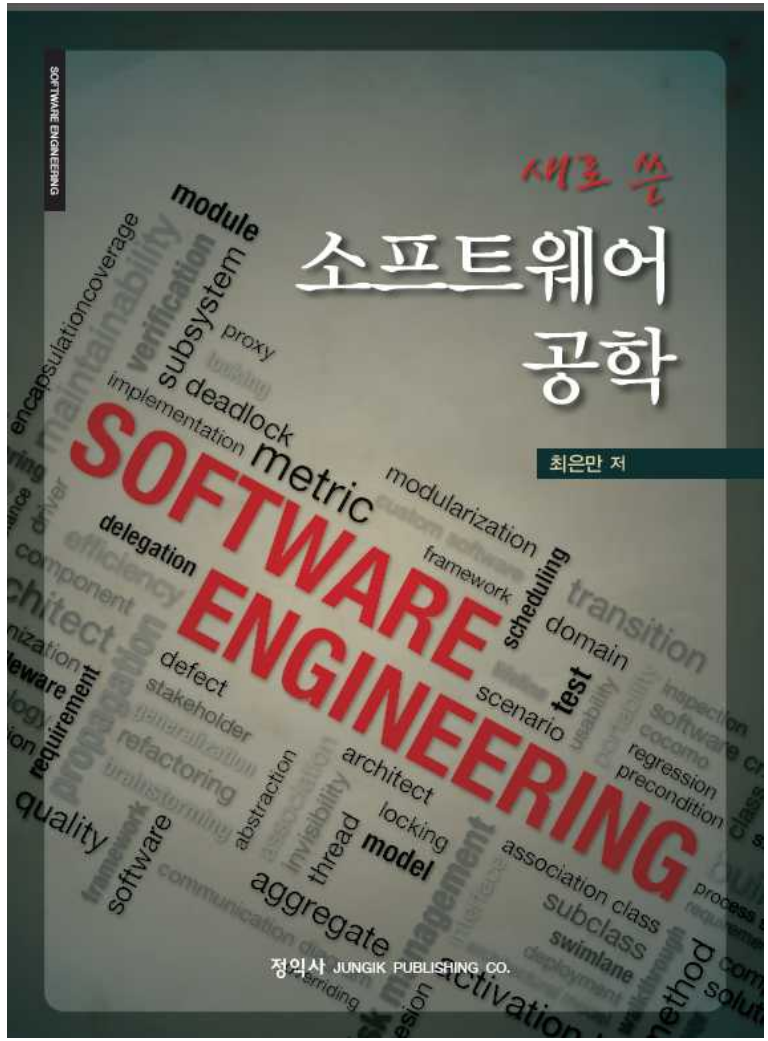
소프트웨어 공학 개론



강의 13: 테스팅

최은만
동국대학교 컴퓨터공학과

Questions

- Q1: 왜 테스트 작업이 중요한가?
- Q2: 테스팅은 어떻게 하나?
- Q3: 테스트에 필요한 도구는?

소 개

- 테스트
  - 테스트에 필요한 시간과 노력 ---> 매우 크다
  - 그러나 테스트는 대부분 초보자나 개인의 역량에 맡기는 경우가 많음

- 정 의
  - *시스템이 정해진 요구를 만족하는지, 예상과 실제 결과가 어떤 차이를 보이는지 수동 또는 자동 방법을 동원하여 검사하고 평가하는 일련의 과정*[IEEE, 1993]
  - *숨어있는 결함을 찾기 위해 소프트웨어를 작동 시키는 일련의 행위와 절차*

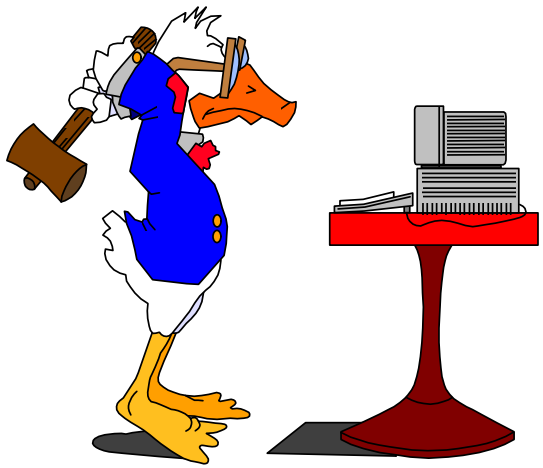
    ---> 결함이 없음을 증명하는 것이 아니고 결함이 존재함을 보여주는 작업
  - 테스트 --> 분석, 설계 도중에 일어나는 검증, 검토 등 품질보증을 위한 모든 행위

소프트웨어 심리

- 프로그래머의 심리
  - 자신의 프로그램이 맞다고 생각, 그러나 프로그램은 테스트에 의하여 확임됨
  - 프로그램 안의 오류를 인신 공격, 또는 개인 평가로 해석하여 테스트를 기피
  - egoless programming, democratic programming
- 심리적 불안
  - 테스트는 파괴적인 과정(destructive process)으로 생각
  - 이를 극복하기 위하여 테스트하기 전에 예상 결과를 준비
  - independent test, test automation

테스트에 대한 올바른 이해

- 테스트는 오류를 발견하려고 프로그램을 수행 시키는 것
- 따라서 테스트에 의하여 오류가 발견되지 않았다고 하여 프로그램에 오류가 없는 것은 아님
  - 완벽한 테스트는 불가능하다.
  - 테스트는 창조적인 일이며 힘든 일이다.
  - 테스트는 오류의 유입을 방지할 수 있다.
  - 테스트는 구현에 관계없는 독립된 팀에 의하여 수행되어야 함



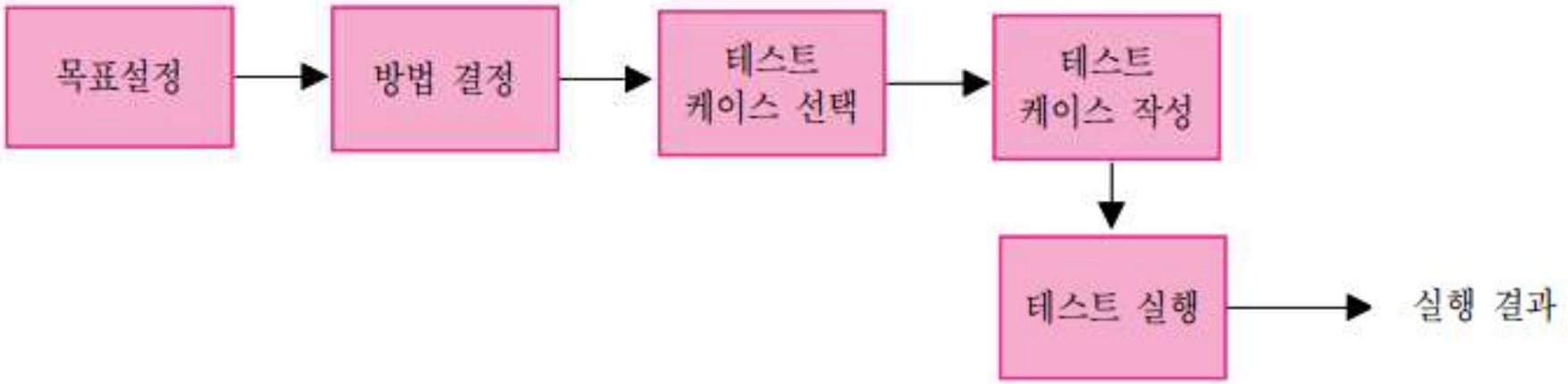
테스팅 용어

- 오류(error)
  - 프로그램 실행 결과가 예상결과와 다른 경우
  - 결함 및 고장을 일으키게 한 인간의 실수
- 결함(fault)
  - 버그(bug)
  - 소프트웨어 오작동의 원인
- 고장(failure)
  - 명세로 작성된 요구와 기능을 제대로 수행할 수 없는 경우
  - 모든 결함이 고장을 발생하는 것은 아님

테스팅 원리

- 테스팅은 오류를 발견하려고 프로그램을 실행시키는 것
- 완벽한 테스팅을 불가능
- 테스팅은 창조적이면서 어려운 작업
- 테스팅은 오류의 유입을 방지
- 테스팅은 구현과 관계없는 독립된 팀에 의해 수행되야함

테스팅 과정

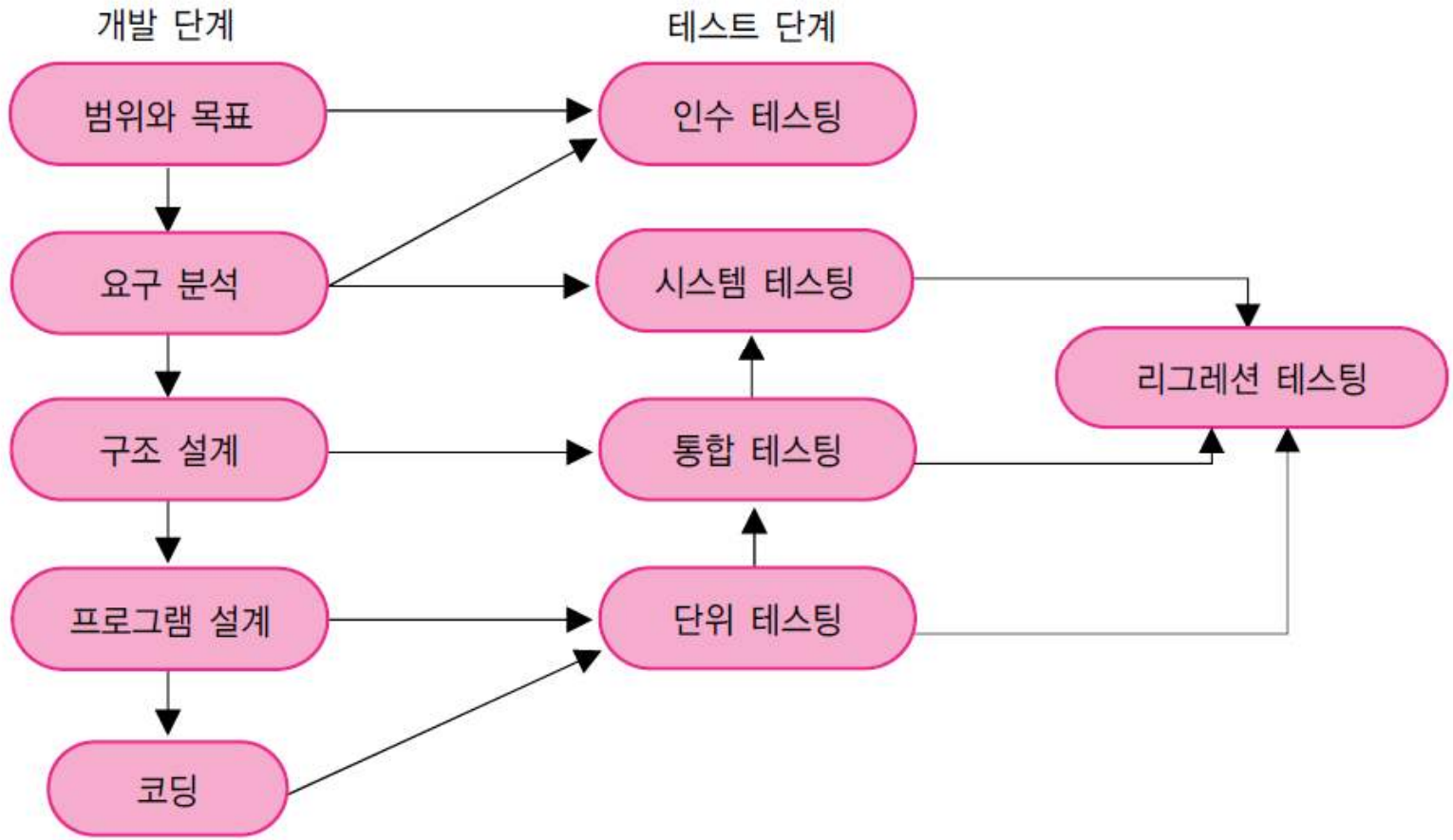


테스트의 원리

- 테스트의 단계
  1) 테스트에 의하여 무엇을 점검할 것인지 정한다.
     <예> 테스트의 목표 - 기능의 완벽성, 신뢰도
  2) 테스트 방법을 결정한다.
     <예> 검사, 증명, 블랙박스 테스트, 화이트 박스 테스트, 자동화 도구
  3) 테스트 케이스를 개발한다.
     - 테스트 자료, 시행 조건
  4) 테스트의 예상되는 올바른 결과를 작성한다.
     - 테스트 오라클(test oracle)
  5) 테스트 케이스로 실행시킨다.
     - 테스트 하니스(test harness)가 필요

테스트와 개발 단계 (V모형)

- 테스트 단계와 소프트웨어 개발 단계의 관계



테스트의 유형

- Validation
  - Are we building the right product?
- Verification
  - Are we building the product right?
- 인증(Certification)
  - A written guarantee
- 정적 분석(Static Analysis)
- 동적 분석(Dynamic Analysis)
- 단계별
  - 단위 테스트(unit test)
  - 통합 테스트(integration test)
  - 인수 테스트(acceptance test)

테스트의 유형

- 시험 방법
  - 화이트 박스 테스트
  - 블랙 박스 테스트
- 기능 시험
- 성능 시험
- 스트레스 시험
- Benchmark 시험
- Field 시험
- 리그레션 테스트
- 품질 보증

테스트 케이스

- 테스트 케이스 선택이 무엇보다 중요
- 좋은 테스트 케이스란?
  - 효과적으로 결함을 드러낼 수 있는 테스트 케이스

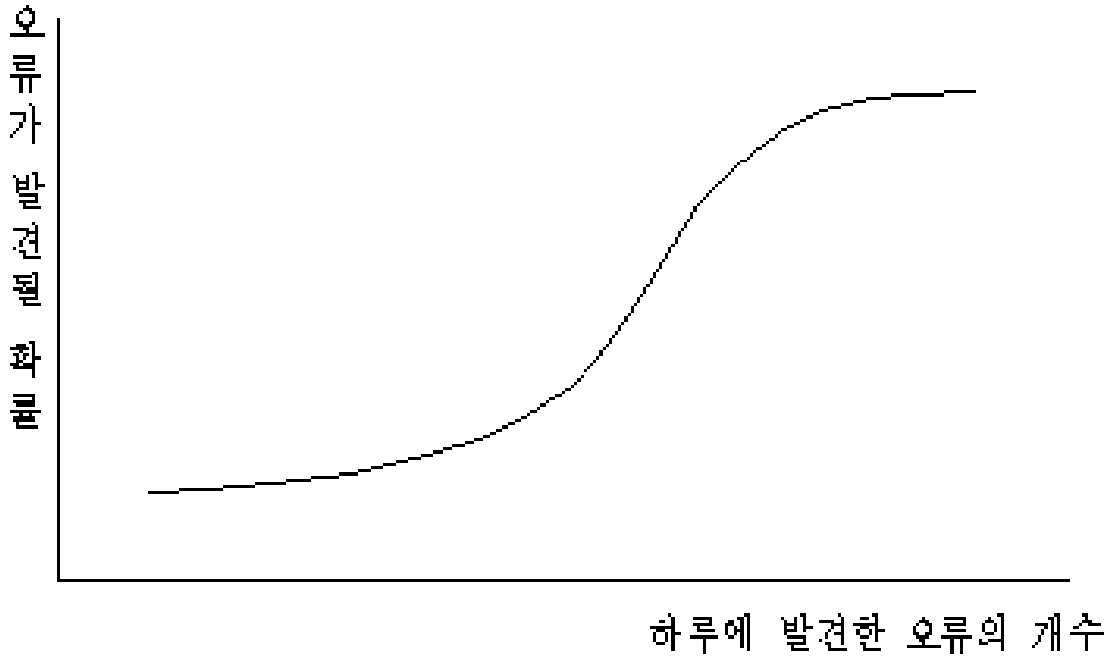
- 전수 테스팅(exhaust testing)
  - 가능한 입력을 모두 테스팅

- 테스트 케이스
  - 테스트 데이터와 예상 결과

고유번호	테스트 대상	테스트 조건	테스트 데이터	예상 결과
FT-1-1	로그인 기능	시스템 초기 화면	정상적인 사용자 ID('gdhong')와 패스워드('1234')	시스템 입장
FT-1-2	"	"	비정상적 사용자 ID('%$##')와 패스워드(' ')	로그인 오류 메시지

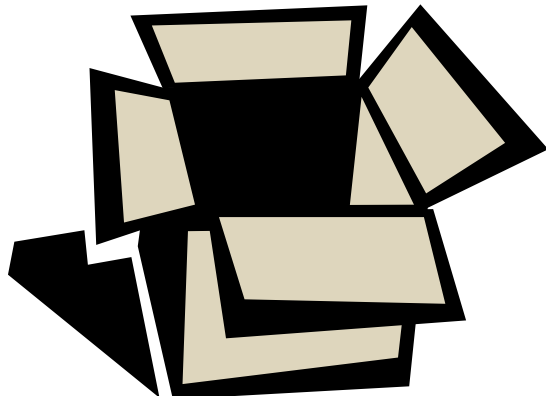
오류 패턴

- 테스트는 시스템의 오류 패턴과 매우 밀접한 관련
- 모듈에 오류가 너무 많다면 오류 없는 모듈이 될 수 있도록 테스트하여 고치는 것은 한계



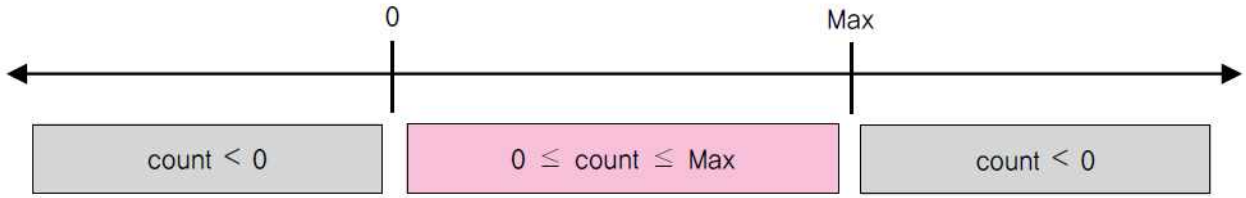
블랙 박스 테스트

- 블랙 박스 테스팅 vs. 화이트 박스 테스팅
  - 프로그램의 구조를 고려하느냐 안하느냐에 따라 구별
- 모듈이 요구에 맞게 잘 작동하는가에 초점
- 기능 테스팅(functional testing)
  - 모듈의 외형(입력, 출력)
  - 모듈의 기능 위주의 검사
- 전수 테스팅
  - 모든 기능에 대하여 전부 테스팅



동치 클래스 분해

- 전수 테스팅이 불가능하여 입력값의 영역을 동치 클래스로 나누고 대표값을 실행
- 내부 구조를 보지 않고 이상적인 최적의 동치클래스를 결정하기가 어려움
  - 예) 절대값을 구하는 모듈
    - 양의 정수와 음의 정수가 동치 클래스
- 입력이 일정한 범위 안의 값을 가져야 한다면 최소한 세 개의 동치 클래스가 존재한다. 범위보다 작은 값, 범위 내의 값, 범위를 넘어서는 큰 값



  - <예> 현금자동 지급기의 총 지급액 범위:1000원-30만원
    ① 1000원에서 30만원 사이의 값(정상)
    ② 1000원 미만의 값(비정상)
    ③ 30만원보다 큰 값(비정상)

동치 분해

- 또 다른 방법
  - 동치 클래스와는 다른 동작을 보일만한 특수한 값을 찾아내기
  - 예: 0, 비정상적인 입력

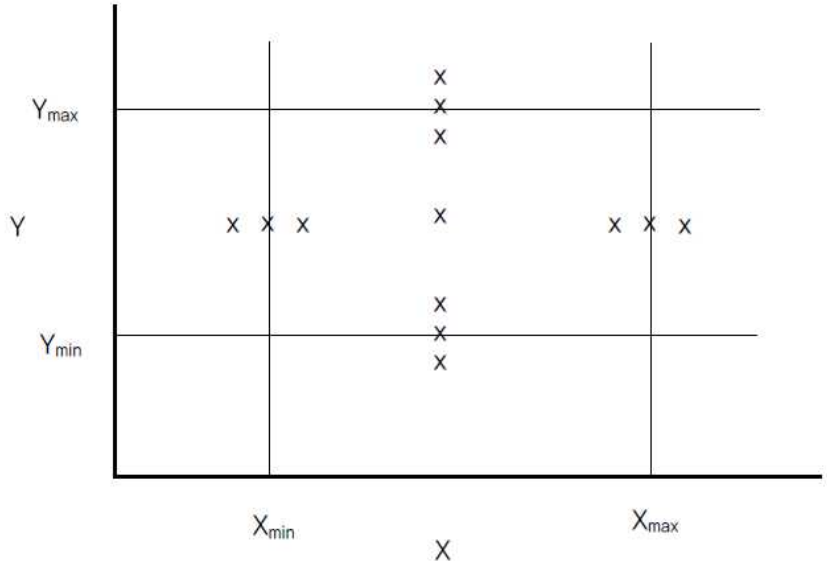
- 출력에 대한 통치 클래스
  - 예: 흑자, 적자, 본전
  - 예: 길이가 N인 스트링 s, 정수 n

표 9.2 ▶ 정상 및 비정상 동치 클래스

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

경계값 분석

- 동치 클래스의 경계에 있는 값을 선택
  - 경계에 있는 값을 가진 테스트 케이스는 높은 효율을 보임
- 범위 경계선 상의 값과 하나 작은 값, 하나 큰 값
- 변수가 여러 개일 때
  - 하나는 고정, 하나는 변동(범위 밖, 안, 경계)



원인 결과 그래프

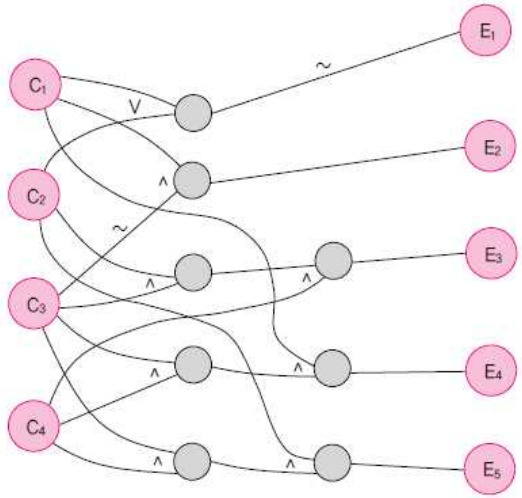
- 동치 클래스, 경계값 분석의 단점
  - 각각의 입력을 별도로 생각
- 원인 결과 그래프
  - 입력 조건의 조합을 체계적으로 선택하여 개수를 조절
- 예) 은행 데이터 베이스
  - 입금 계정 번호 트랜젝션_금액
  - 출금 계정 번호 트랜젝션_금액

원인:

c1. 명령어가 입금
c2. 명령어가 출금
c3. 계정 번호가 정상
c4. 트랜젝션 금액이 정상

결과:

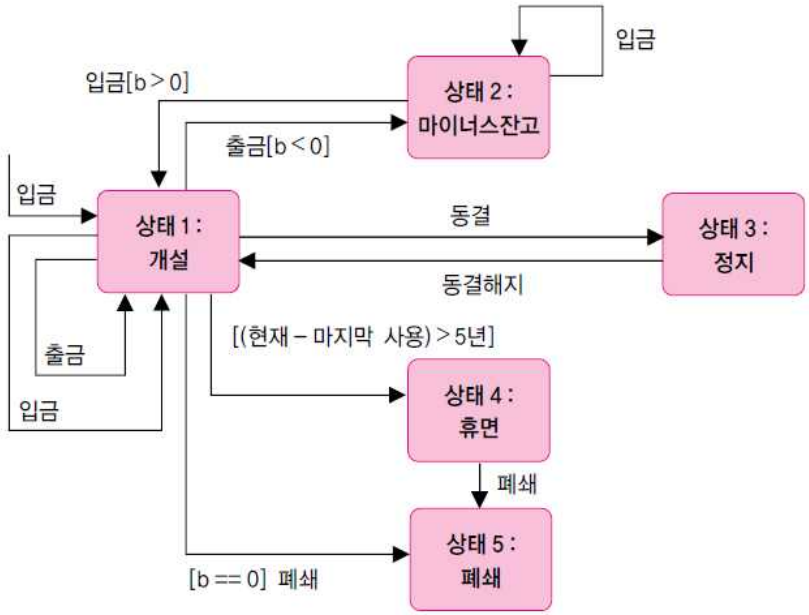
e1. '명령어 오류'라고 인쇄
e2. '계정 번호 오류'라고 인쇄
e3. '출금액 오류'라고 인쇄
e4. 트랜젝션 금액 출금
e5. 트랜젝션 금액 입금



상태 기반 테스팅

- 상태 기반 시스템
  - 시스템의 동작과 출력은 제공되는 입력만이 아니라 시스템의 상태에 의하여 좌우됨

- 상태 모델
  - 상태를 저장하는 시스템을 모델링
  - 상태 공간은 변수의 제곱에 비례



상태 기반 테스팅

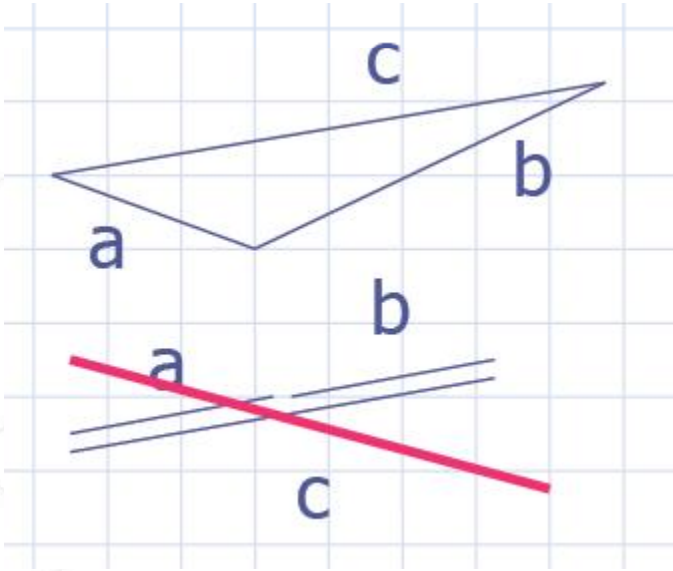
- **테스트 케이스**

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−마지막사용일)>5년
10	1→5	폐쇄(잔고=0)
11	4→5	폐쇄

퀴즈

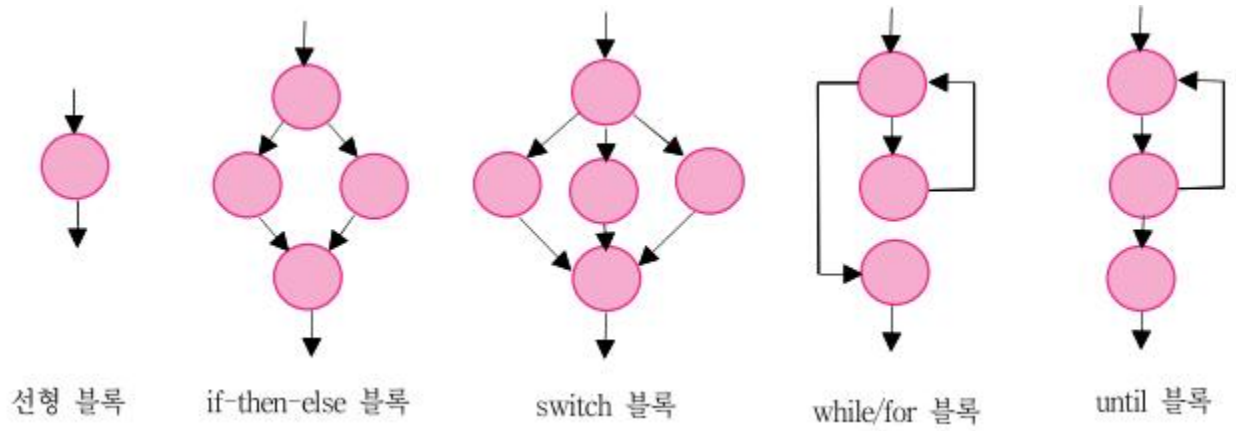
- 다음 프로그램을 테스팅하기 위한 테스트 케이스는 몇 개가 필요한가?(From Glen Myers, The Art of Software Testing)

- 입력창에서 세 개의 정수값을 읽어 그것이 삼각형의 세 변의 길이로 해석하여 정삼각형, 이등변 삼각형, 부등변 삼각형인지 체크하는 그로그램



화이트박스 테스팅

- 모듈의 논리적인 구조를 체계적으로 점검하는 구조적 테스팅
- 여러가지 프로그램 구조를 기반으로 테스트
- 논리 흐름도(logic-flow diagram)을 이용
  - 노드: 모듈내의 모든 세그먼트
  - 간선: 제어 흐름



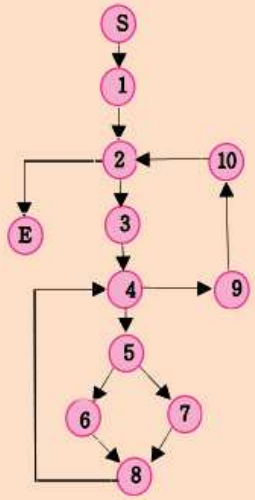
기본 경로 테스팅

- 기본경로(basis path)
  - 독립적인 논리 흐름을 검사하는 테스트 케이스를 생성

- 시작 노드에서 종료 노드까지의 서로 다른 경로로써 싸이클은 최대 한번만 지나야 함

  <예> Remove 함수에 대한 논리흐름 그래프와 테스트 케이스

```
public void  Remove(LinkedList list) {
      Item p, q;
①   p = IItem(list.getFirst();
②   while(p''=null) {
③       q=(Item)p.getNext();
④       while (q!=null) {
⑤           if(p.compareTo(q)==0)
⑥               list.remove(q);
⑦           else q=(Item)p.getNext();
⑧       }
⑨       p=(Item)p.getNext();
⑩   }
  }
```



1. S-1-2-E: 빈 리스트
2. S-1-2-3-4-9-10-2-E: 한 개의 요소를 가진 리스트
3. S-1-2-3-4-5-6-8-4-9-10-2-E: 중복 요소를 가진 리스트
4. S-1-2-3-4-5-7-8-4-9-10-2-E: 중복 요소가 없는 리스트

싸이클로매틱 복잡도

- 기본 경로의 수를 결정하는 이론

- 싸이클로매틱 복잡도 계산 3가지 방법
  - 폐쇄 영역의 수 +1 : 논리 흐름 그래프는 이차원 평면을 여러 영역으로 나누며, 이 중 폐쇄된 영역의 수 에 1을 더한 값
  - 노드와 간선의 수 : 간선의 수에서 노드의 수를 빼고 2를 더한 값
  - 단일 조건의 수 +1 : 참과 거짓으로 판별되는 원자적 조건의 수에 1을 더한 값

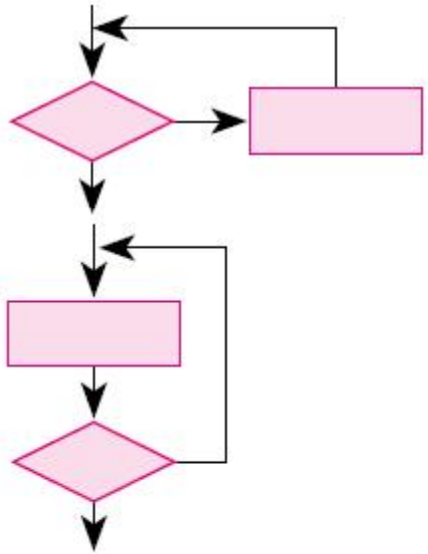
- 3가지 방법의 값이 같아야 함

테스트 커버리지

- 테스트를 어느 정도 완벽히 수행할 것인가의 기준
  - 노드 커버리지
    - 논리 흐름 그래프의 각 노드가 테스트 케이스에 의하여 적어도 한 번씩 방문되어야 하는 검증기준
    - 프로그램 문장 100% 커버
  - 간선 커버리지
    - 논리 흐름 그래프의 각 간선이 테스트 케이스에 의하여 적어도 한 번씩 방문되어야 하는 검증기준
    - 모든 분기점 테스트(Branch coverage)
  - 기본 경로 커버리지
    - 모든 기본 경로가 적어도 한 번씩 방문되어야 하는 검증기준
    - 간선 커버리지의 50%
  - 모든 경로 커버리지
    - 모든 가능한 경로를 적어도 한 번씩 테스트하는 검증기준
    - 현실적으로 불가능

반복문의 테스팅(1)

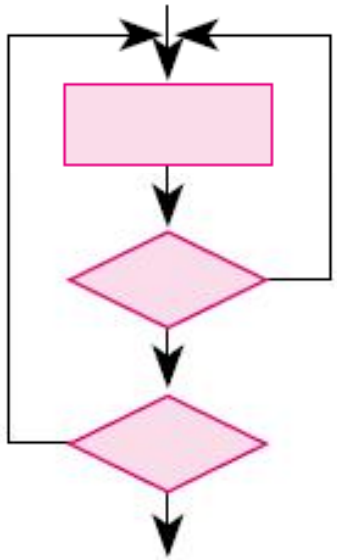
- Beizer 반복구조 분류
- 단순 반복
- 경계값 분석 방법 이용
  - 반복 구조를 들어가지 않고 생략
  - 반복 구조 안에서 한 번 반복
  - 반복 구조 안에서 두 번 반복
  - 일정한 횟수의 반복
  - 반복 최대 횟수 – 1 만큼 반복
  - 반복 최대 횟수만큼 반복
  - 반복 최대 횟수 +1 만큼 반복



반복문의 테스팅(2)

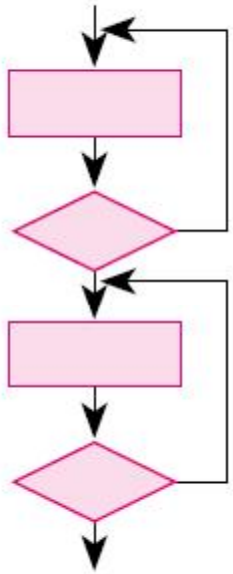
- 중첩된 반복

- 가장 내부에 있는 반복 구조부터 테스트 (단, 외부 반복 구조는 최소 반복횟수로 지정)
  - 최소 횟수의 반복
  - 최소 횟수보다 하나 많은 반복
  - 범위 내 임의의 횟수 반복
  - 최대 횟수보다 하나 적은 반복
  - 최대 횟수의 반복
  - 외부로 향하여 다음 반복구조를 테스트

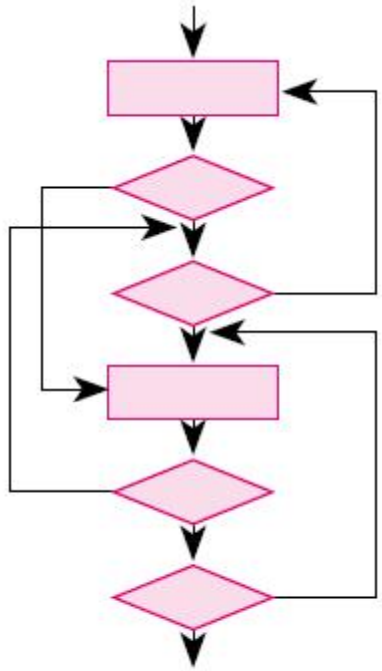


반복문의 테스팅(3)

- 연속된 반복
  - 반복구조가 서로 독립적이면 '단순반복'
  - 반복구조가 어느 한쪽이 포함된 관계라면 '중첩된 반복'
- 비구조화 반복
  - 구조적 반복 형태로 변경하여 테스트



(c) 연속된 반복



(d) 비구조화 반복

사용 사례 기반 테스팅

- **사용 사례 명세로부터 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 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

사용 사례 기반 테스팅

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	

사용 사례 기반 테스팅

3. 테스트 케이스 생성

- 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(중복 제거), 유지할지 삭제할지 불분명한 것은 선택

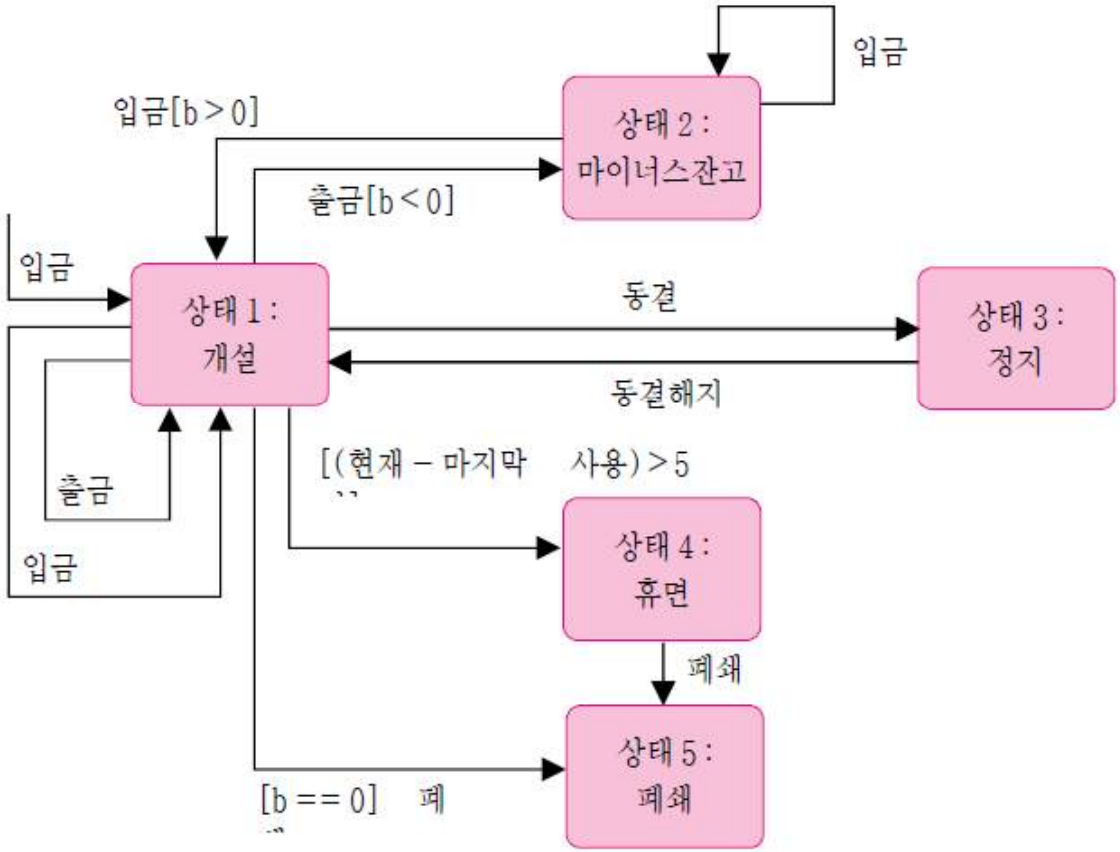
테스트 케이스	로그인 ID	패스워드	재입력된 패스워드	예상 결과
1	정상	정상	정상	등록이 성공되었다는 페이지가 보임
2	정상	정상	비정상	오류 메시지가 보임
3	정상	비정상	정상	오류 메시지가 보임
~~4~~	~~정상~~	~~비정상~~	~~비정상~~	<테스트 케이스 3에 포함>
5	정상	예외	정상	오류 메시지가 보임
~~6~~	~~정상~~	~~예외~~	~~비정상~~	<테스트 케이스 2, 3에 포함>
7	비정상	정상	정상	오류 메시지가 보임
~~8~~	~~비정상~~	~~정상~~	~~비정상~~	<테스트 케이스 2, 7에 포함>
~~9~~	~~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>

상태 기반 테스팅

- 같은 입력에 대해 같은 동작을 보이며 동일한 결과를 생성하는 시스템(state-less system)을 대상
  - 배치 처리 시스템
  - 계산 중심 시스템
  - 하드웨어로 구성된 회로
- 시스템의 동작은 시스템의 상태에 의해 좌우됨
- 상태 모델 구성요소
  - 상태 – 시스템의 과거 입력에 대한 영향을 표시
  - 트랜지션 – 이벤트에 대한 반응으로 시스템이 하나의 상태에서 다른 상태로 어떻게 변해가는지를 나타냄
  - 이벤트 – 시스템에 대한 입력
  - 액션 – 이벤트에 대한 출력

상태 기반 테스팅

<예> 예금 계좌의 상태 모델 예시



상태 기반 테스팅

- 검증 기준(coverage)
  - 모든 트랜지션
    - 테스트 케이스 집합이 상태 그래프의 모든 트랜지션을 점검
  - 모든 트랜지션 쌍
    - 테스트 케이스 집합이 모든 이웃 트랜지션의 쌍을 점검
    - 유입(incoming)과 방출(outgoing) 트랜지션 쌍을 의미
  - 트랜지션 트리
    - 테스트 케이스 집합이 모든 단순 경로를 만족시키는 기준

통합 테스팅

- 모듈의 인터페이스 결합을 테스트
  - 여러 개발 팀에서 개발한 각각의 단위 모듈을 대상
  - 모듈-모듈 간의 결합을 테스트

- 모듈의 결합 순서에 따라 방법이 다름
  - 빅뱅(big-bang)
  - 하향식(top-down)
  - 상향식(bottom-up)
  - 연쇄식(threads)

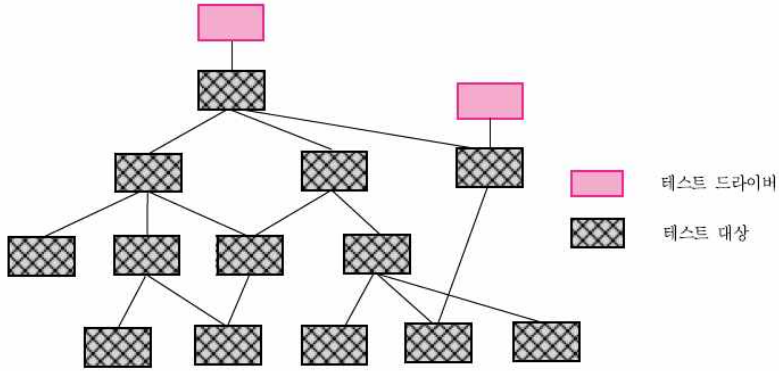
- 용어
  - 드라이버
    - 시험 대상 모듈을 호출하는 간이 소프트웨어
  - 스텁
    - 시험 대상 모듈이 호출하는 또 다른 모듈

빅뱅 통합

- 한 번에 모든 모듈을 모아 통합

- 장점
  - 고도의 신뢰도가 요구되는 시스템의 경우 중요 부분을 먼저 구현하기 때문에 의뢰자에게 신뢰감을 줄 수 있음.
  - 중요 부분을 먼저 구현함으로써 여러 번 테스트가 반복되어 완고한 개발이 가능함

- 단점
  - 오류의 위치와 원인을 찾기 어려움
  - 단위 테스트에 많은 시간과 노력이 듬
    - 준비해야 할 드라이버/스텁 수가 많음
  - 개발 진도를 예측하기 어려움

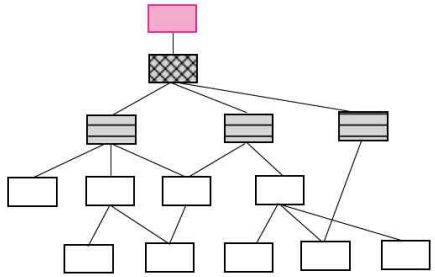


하향식 통합

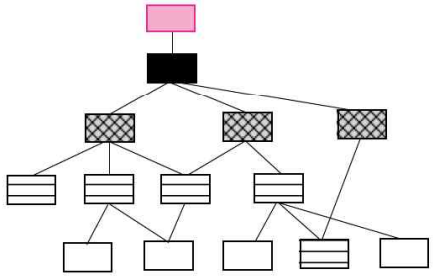
- 시스템 구조상 최상위에 있는 모듈부터 통합

- 장점
  - 중요한 모듈의 인터페이스를 조기에 테스트
  - 스텁을 이용하여 시스템 모습을 일찍 구현가능
  - 개발자 입장에서 용이함

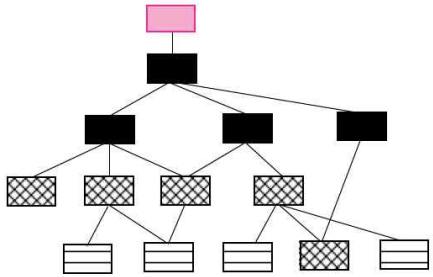
- 단점
  - 입출력 모듈이 상대적으로 하위에 있음
    - 테스트 케이스 작성 및 실행이 어려움
  - 중요 기능이 마지막에 구현됨



(a) 1 단계 통합



(b) 2 단계 통합

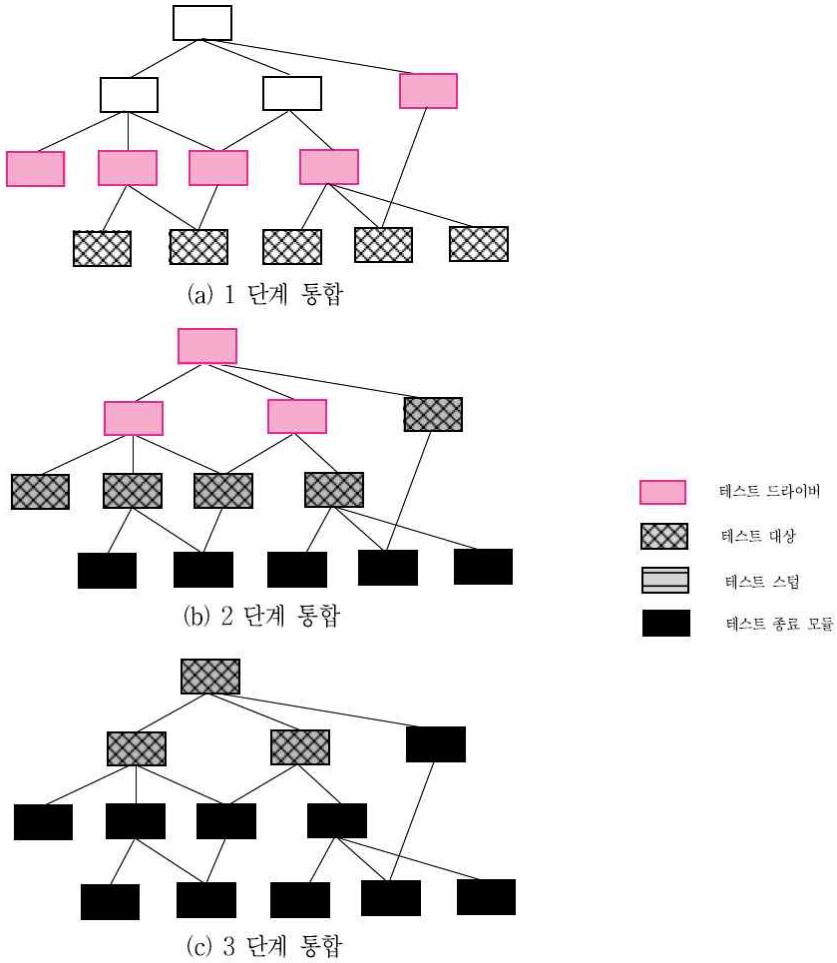


(c) 3 단계 통합

상향식 통합

- 시스템 구조상 최하위에 있는 모듈부터 통합

- 장점
  - 점증적 통합 방식
    - 오류 발견이 쉬움
    - 하드웨어 사용 분산
  - 하위층 모듈을 상위층보다 더 많이 테스트
- 단점
  - 초기에 시스템의 뼈대가 갖추어지지 않음
  - 상위층의 중요한 인터페이스가 마지막에 가서야 확인 가능
  - 의뢰자에게 시스템을 시험해 볼 기회를 충분히 제공하지 못함



연쇄식 통합

- 특정 기능을 수행하는 모듈의 최소 단위(thread)로 부터 시작
  - 입력, 출력
  - 어느 정도의 기본 기능을 수행하는 모듈
- 상대적으로 중요한 모듈부터 개발
- 장점
  - 초기에 시스템의 골격이 형성
    - 사용자 의견을 빨리 확인 가능
  - 시스템을 나누어 개발 하기 쉽다

시스템 및 인수 테스팅

- 컴포넌트 통합 후 수행하는 테스트 기법

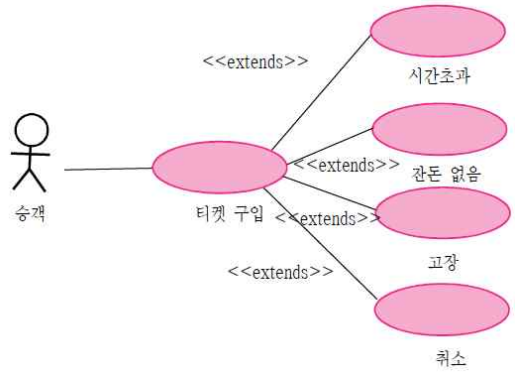
- 테스트 종류
  - 기능 테스트
  - 성능 테스트
  - 보안 테스트
  - 사용성 테스트
  - 인수 테스트
  - 설치 테스트

기능 테스트

- 기능적 요구와 시스템의 차이를 발견하기 위한 테스트
- 사용자와 관련되어 있으며 오류를 유발할 가능성이 많은 테스트를 선정
- 사용사례 모델을 검토하고 오류를 일으킬만한 사용사례 인스턴스를 찾아낸다.
- 테스트 케이스
  - 일반적인 사례
  - 예외적인 사례

기능 테스트

- **기능 테스트 케이스 작성 과정**



사용사례 이름	티켓구입
시작 조건	승객이 티켓 판매기 앞에 선다. 승객이 티켓을 살만한 충분한 돈이 있다.
사건의 흐름	1. 승객이 여행할 목적지의 구간을 선택한다. 승객이 여러 구간을 입력하였다면 마지막 누른 버튼만을 티켓판매기가 인식한다. 2. 티켓판매기가 총 금액을 표시한다. 3. 승객이 돈을 넣는다.
	4. 승객이 충분한 돈을 넣기 전에 새로운 구간을 선택하였다면 티켓판매기는 승객이 넣은 모든 동전과 지폐를 돌려준다. 5. 승객이 총액보다 많은 동전을 넣었다면 티켓판매기 초과분을 돌려준다. 6. 티켓판매기가 표를 발행한다. 7. 승객이 거스름돈과 표를 받는다.
종료조건	승객이 선택한 표를 받는다.

테스트 케이스 이름	티켓 정상 구입
시작 조건	승객이 티켓 판매기 앞에 선다. 승객이 1000원을 가지고 있다.
사건의 흐름	1. 승객이 여행할 목적지의 구간 2, 3, 1, 2를 선택한다. 2. 티켓판매기가 800, 1000, 600, 800원을 차례로 표시한다. 3. 승객이 1000원 지폐를 넣는다. 4. 승객이 100원 동전 두 개를 받고 2구간의 표를 받는다. 5. 승객이 또 한 번 1000원 지폐를 넣고 1-4까지의 과정을 반복한다. 6. 승객이 500원 동전 2개를 넣고 1-4를 반복한다. 7. 승객이 500원 동전과 100원 동전 세 개를 넣고 1-2를 반복한다. 8. 승객이 1구간을 누르고 1000원을 넣는다. 티켓 판매기는 1구간 표를 발행하고 400원의 거스름돈을 배출한다. 9. 승객이 3구간을 누르고 1000원을 넣는다. 티켓 판매기는 3구간 표를 발행한다.
종료조건	승객이 선택한 표를 받는다.

성능 테스트

- 시스템의 여러 측면 테스트
  - 작업 부하(workload)
    - 시스템이 처리하고 생성하는 작업의 양
  - 처리량(throughput)
    - 트랜잭션 의 수
    - 시간 당 처리하는 메일 수
  - 반응 시간(response time)
    - 시스템 요구를 처리하는 데 걸리는 총 시간
  - 효율성
    - 주어진 작업 처리를 위한 CPU시간과 메모리 같은 자원의 량의 비율
  - 자원 효율성

성능 테스트

- 테스트 방법
  - 스트레스 테스팅
    - 시스템 처리능력의 몇 배의 작업부하를 처리하고 견딜 수 있는지 측정
  - 성능 테스팅
    - 정상적인 사용 환경에서 시스템의 성능을 측정하는데 사용
    - 시뮬레이션을 이용한 테스팅 가능
  - 보안 테스팅
    - 시스템의 보안 취약점을 찾아내려는 목적

사용자 인터페이스 테스트

- 기능, 성능, 보안 테스트와 목적이 다름
  - 인간 공학적인 목적

- 테스트 목적
  - 보고 느끼는 UI에 대한 결함
  - 데이터 입력과 출력 디스플레이에 대한 결함
  - 액터-시스템 사이의 동작 결함
  - 오류 처리에 대한 결함
  - 문서와 도움말에 대한 결함

인수 테스트

- 시스템을 당장 사용할 수 있도록 모든 준비가 되어 있는지 확인
- 개발자를 제외한 의뢰자 또는 대리인이 테스트 수행
- 시스템 요구 분석서를 기반으로 한 테스트 수행
- 실제 업무 절차를 따라 테스트 수행
- 테스트 유형
  - 알파 테스트
    - 선택된 사용자가 개발 환경에서 시험하는 것
  - 베타 테스트
    - 선택된 사용자가 외부 환경에서 시험하는 것(필드 테스팅)

테스트 도구

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

코드 분석 도구

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

코드 분석 도구

- 동적 분석 도구
  - 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
    - 테스트 만족도를 평가하는 지표

테스트 케이스 생성 도구

- 테스트 케이스를 자동으로 생성
- 도구 유형
  - 자료 흐름도
    - 자료 흐름도를 이용하여 define-use 관계를 찾음
  - 기능 테스트 방법
    - 주어진 기능을 구동시키는 모든 가능한 상태를 파악하여 이에 대한 입력을 작성
  - 입력 도메인 분석
    - 원시코드의 내부를 참조하지 않고 입력 변수가 가질 수 있는 값의 도메인을 분석
  - 랜덤 테스트
    - 입력 값을 무작위로 추출하여 테스트

테스트 케이스 실행 도구

- 테스트 작업 수행이나 계획을 도와줌

- 도구 유형
  - 캡처 및 리플레이
    - 미리 입력된 입력 값을 이용하여 실행하고 결과를 캡처하여 비교
  - 스텁과 드라이버
    - 자동으로 스텁, 드라이버를 생성
  - 자동 테스트 환경
    - 통합된 테스트 환경

단위 테스트 도구

- XUnit 등 다양한 언어로 작성된 프로그램의 단위 테스트와 리그레션 테스트를 지원
- 테스트 케이스와 테스트 결과의 체크를 한데 묶은 패키지
- 테스트 케이스의 반복 실행가능(리그레션 테스트)

Questions?