Chap 10　테스트

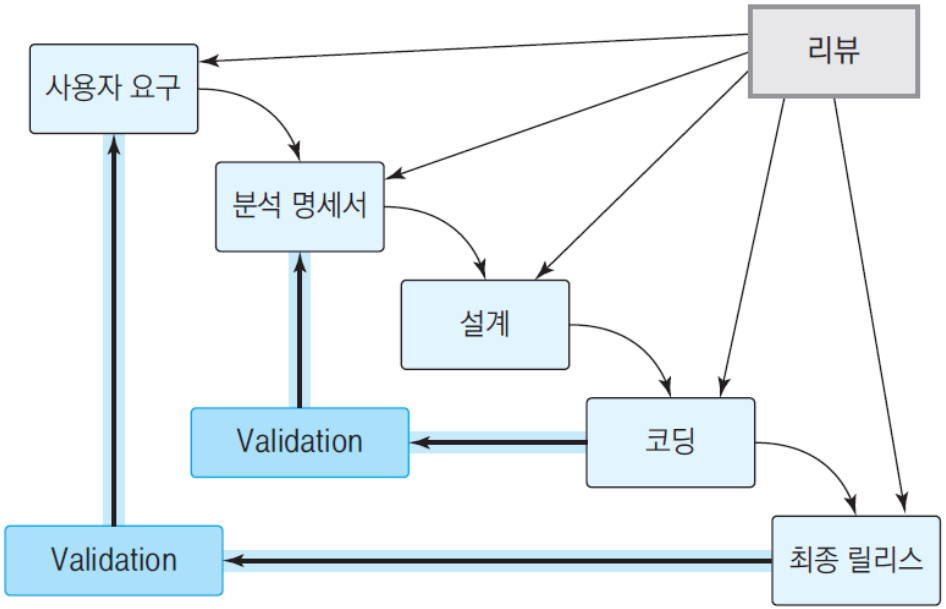
목 차

테스트

- 소프트웨어 개발은 인간 중심의 활동이며 지적 활동
  - 오류가 발생하기 쉬운 활동
- 결함을 낮추는 방법
  - 방지: 인스펙션, 정적 분석
  - 식별하고 제거: 테스트, 디버깅
- 테스트
  - 시험할 소프트웨어에 테스트 케이스를 주어 실행시킨 후 시스템의 동작이 예상한 대로 실행되는지 확인하는 것

검증과 확인

- 검증(verification)
  - "제품을 올바르게 구축하고 있는가?"
- 확인(validation)
  - "올바른 제품을 만들고 있는가?"



10.1 테스트 기초

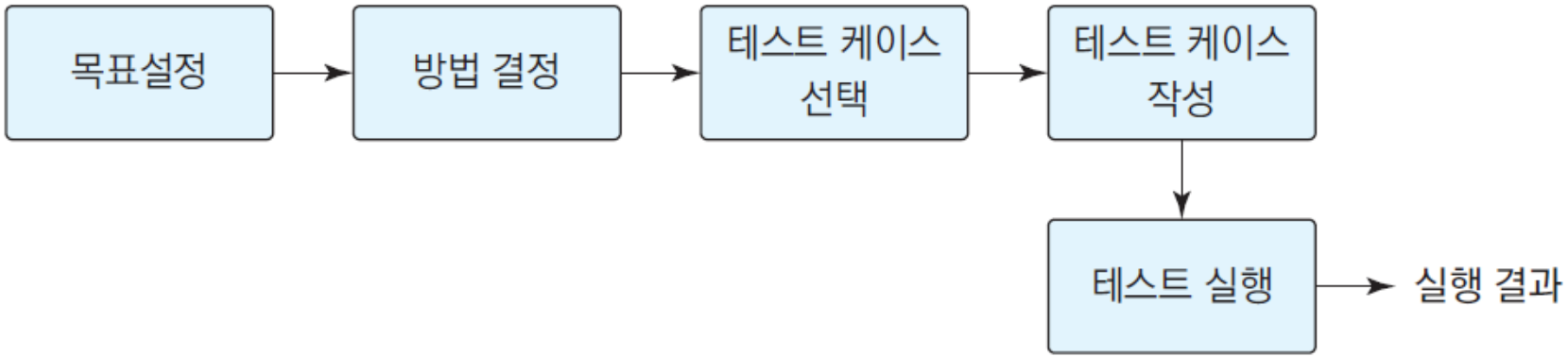
- 버그
  - 문제, 결함 또는 난이도를 나타내는 데 일반적으로 사용되는 용어
- 오류
  - 개발자가 잘못하여 설계나 코딩에 실수한 것
- 결함
  - 시스템이 고장을 일으키게 하는 오류의 결과
  - 코드 또는 문서에 오류가 있다고 선언된 것
- 고장
  - 시스템이 원하는 작업을 수행할 수 없는 상황, 현상

테스트 원리

- 테스트는 오류를 발견하려고 프로그램을 실행시키는 것이다.
- 완벽한 테스트는 불가능하다.
- 테스트는 창조적인 일이며 힘든 일이다.
- 테스트는 오류의 유입을 방지할 수 있다.
- 테스트는 구현과 관계없는 독립된 팀에 의하여 수행되어야 한다

테스트 작업 과정

1. 테스트에 의하여 무엇을 점검할 것인지 정한다
2. 테스트 방법을 결정한다.
3. 테스트 케이스를 개발한다.
4. 테스트의 예상되는 올바른 결과를 작성한다.
5. 테스트 케이스로 실행시킨다.



테스트 단계

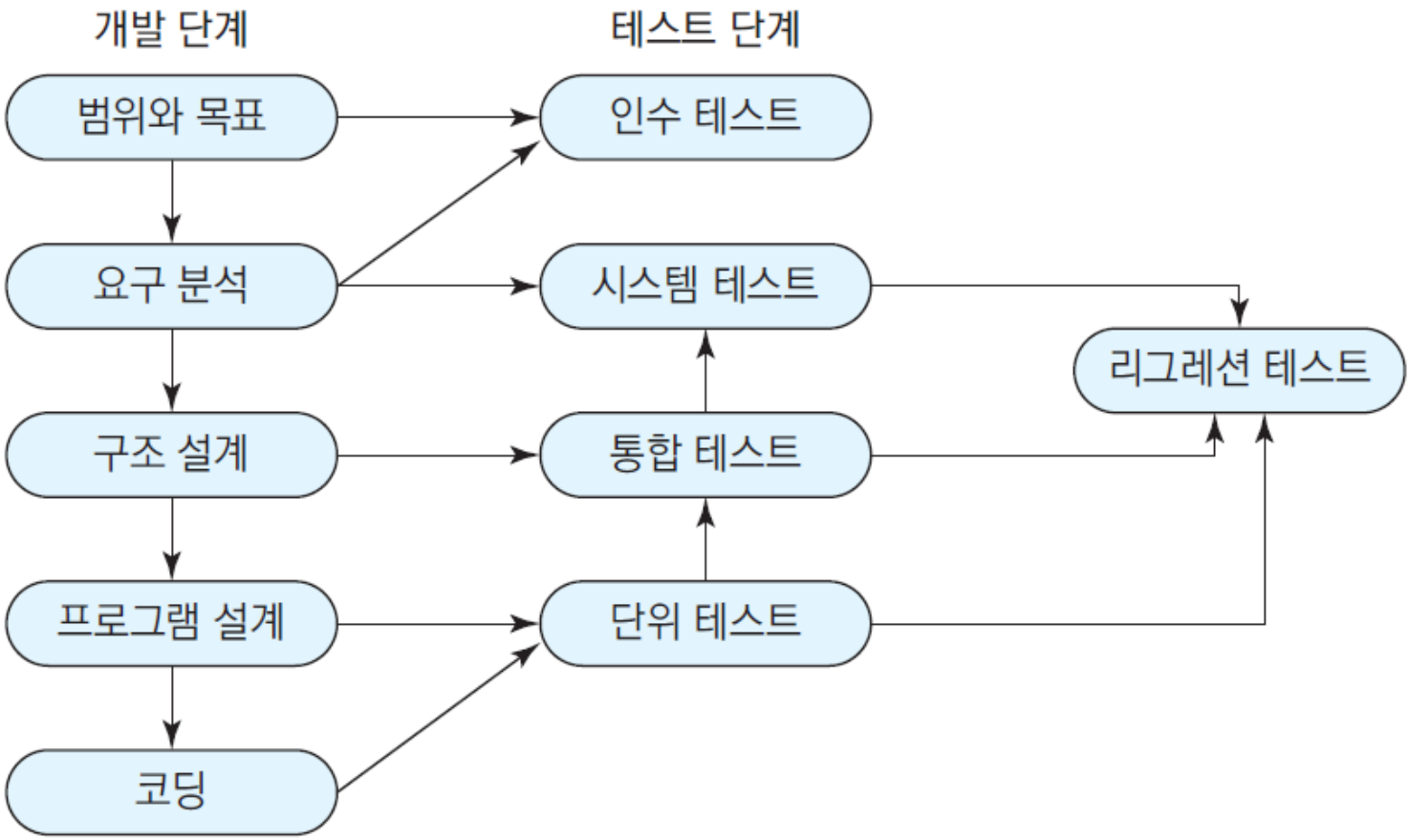


그림 10.5 테스트 단계와 소프트웨어 개발 단계의 관계

테스트 케이스

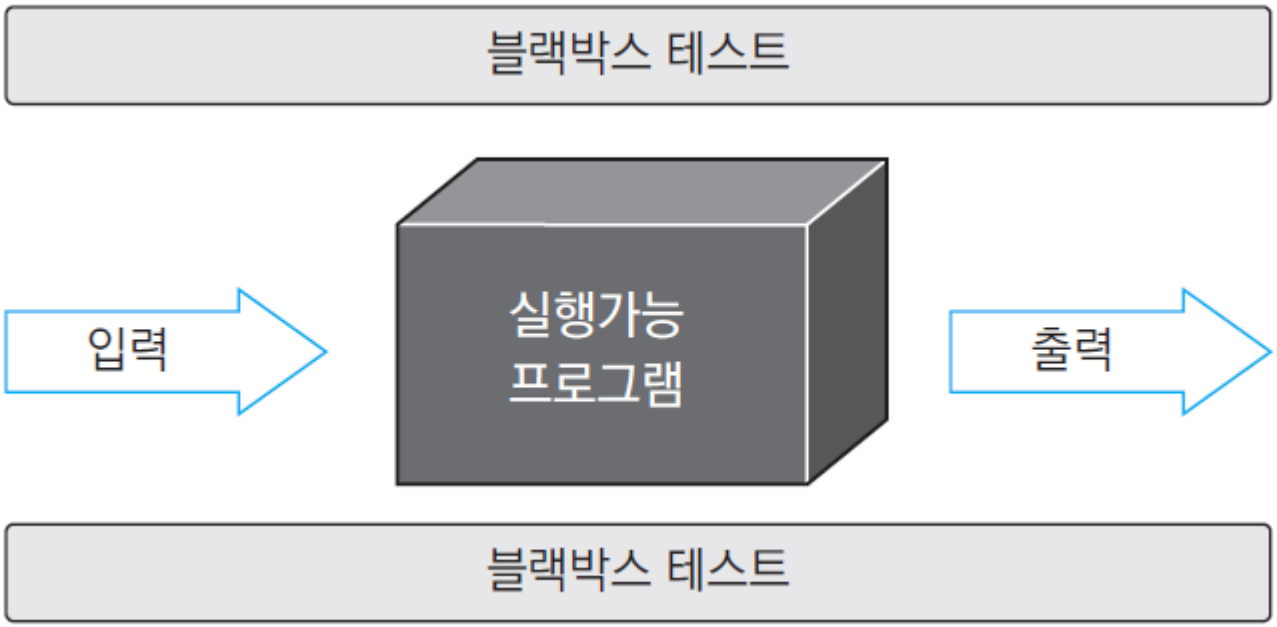
- 결함을 검사할 수 있는 입력
- 시험조건, 테스트 데이터, 예상 결과
- 테스트 케이스 명세서

표 9.1 테스트 케이스

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
⋮	⋮	⋮	⋮	⋮

10.2 블랙박스 테스트

- 내부 경로에 대한 지식을 보지 않고 테스트 대상의 기능이나 성능을 테스트
  - 요구 사항 및 사양을 기반



동등 분할 기법

- 동등 클래스(equivalence class)
  - 시스템의 동작이 같을 것으로 예상되는 입력

AGE Enter Age *18에서 60 사이의 숫자만 허용

동등 클래스 분할		
Invalid	Valid	Invalid
〈=17	18-60	〉=61

그림 10.7 동등 클래스 분할

a) AGE <= 17

b) AGE >= 61

c) 18 <= AGE <= 60

경계값 분석

- 동치 클래스의 경계에서 문제를 발생하는 특수한 값이 존재
- 동치 클래스 경계에 있는 값을 가진 테스트 케이스는 높은 효율을 가짐
- 동치클래스의 경계에 있는 값을 테스트 입력으로 선택

<예> 하나의 입력 값 X에 대한 테스트 케이스

입력값 조건: $min \leq X \leq max$

케이스: min-1, min, min+1, max-1, max, max+1

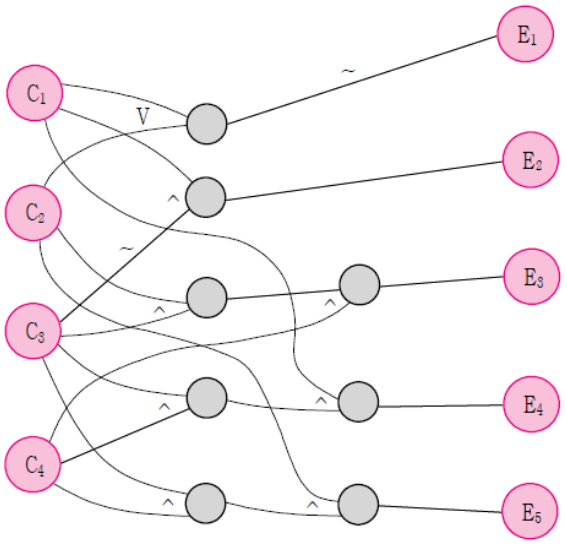
원인과 결과 그래프

- 입력 조건의 조합을 체계적으로 선택하는 테스트 기법
- 노드와 기호로 표시
  - 노드: 원인(입력조건), 결과(출력 조건)
  - 기호: ∧(and), ∨(or), ~(not)

<예>

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



결정 테이블

- 각각의 결과들에 대하여 조건의 조합을 나열

  <예>

  x: don't care

  0

  1

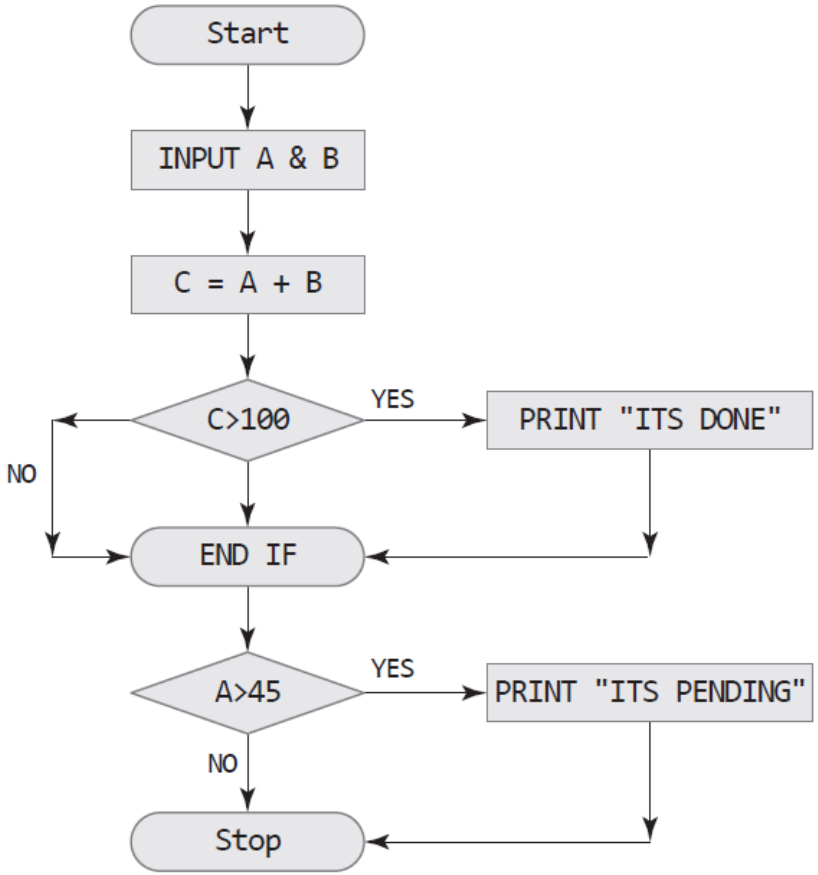
No.	1	2	3	4	5
c1	0	1	x	x	1
c2	0	x	1	1	x
c3	x	0	1	1	1
c4	x	x	0	1	1
e1	1				
e2		1			
e3			1		
e4				1	
e5					1

10.3 화이트박스 테스트

- 모듈의 논리적인 구조를 체계적으로 점검 – 구조적 테스트
- 테스트 과정
  1. 원시 코드를 통해 애플리케이션의 구조를 이해 – 논리흐름도
  2. 검증 기준 (커버리지)을 정한다.
  3. 각 경로를 구동시키는 테스트 데이터를 준비

논리 흐름도

- 모듈 내의 제어 흐름을 간선으로 표시한 그래프

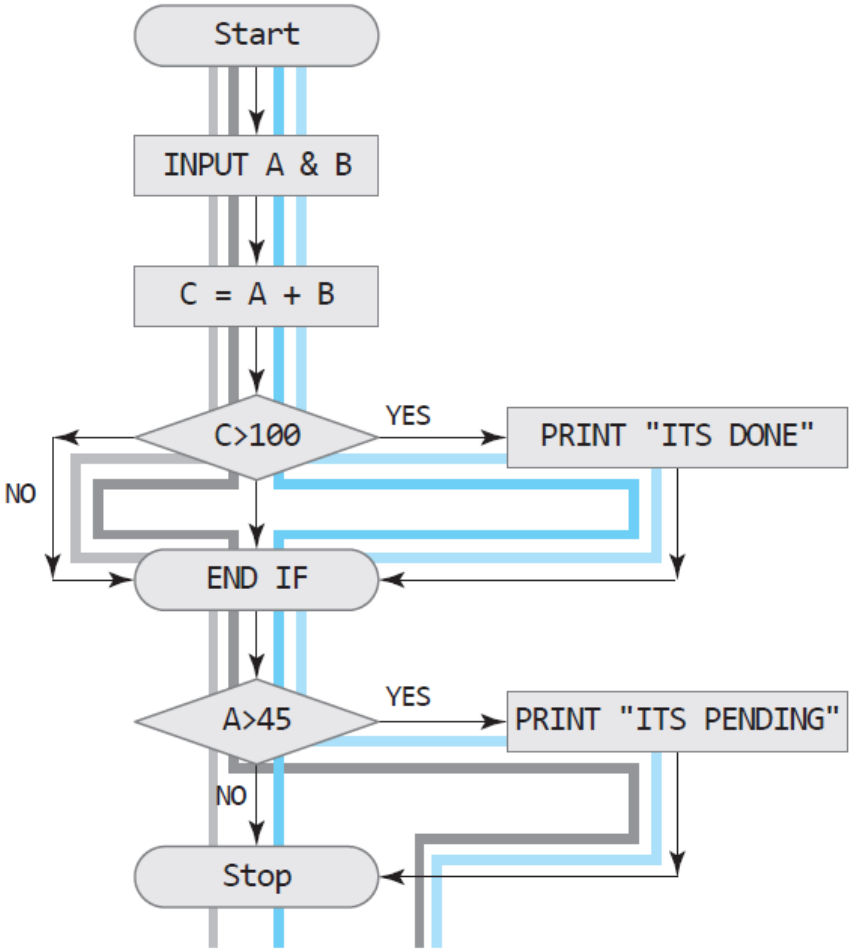


검증 기준

- 문장 커버리지
  - 각 라인이 적어도 한 번 실행되는지를 검증
  - Test Case_01: A=50, B=60
- 분기 커버리지
  - IF 문에는 True와 False의 두 가지
  - Test Case_01: A=50, B=60
  - Test Case_02: A=30, B=30
- 경로 커버리지
  - 모든 실행 경로를 테스트하는 기준

경로 커버리지

- TestCase_01: A=50, B=60(C>100: true, A>45: true)
- TestCase_02: A=55, B=40(C>100: false, A>45: true)
- TestCase_03: A=40, B=65(C>100: true, A>45: false)
- TestCase_04: A=30, B=30(C>100: false, A>45: false)



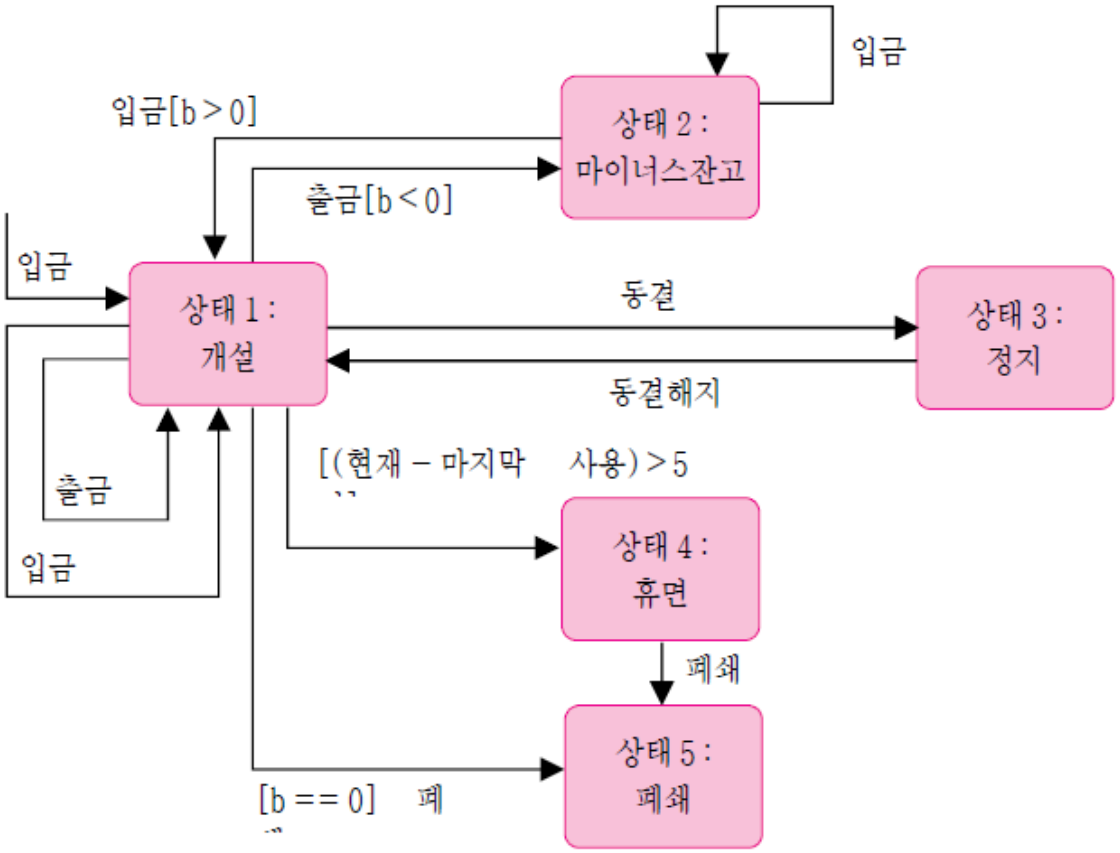
Path Coverage

10. 4 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 계정의 상태 모델 예시



상태 기반 테스트 케이스

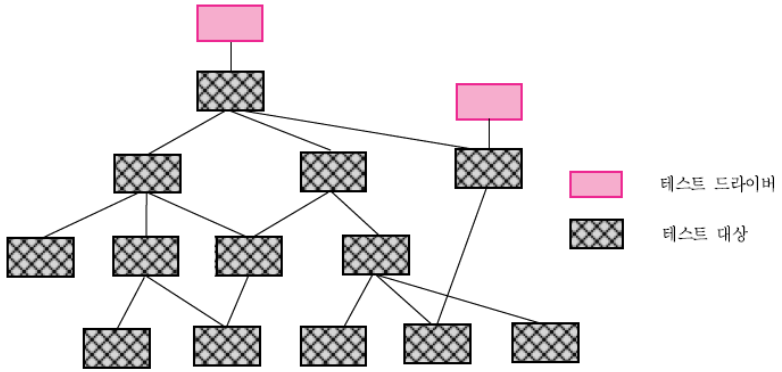
No.	트랜지션	테스트 케이스
1	start →1	입금
2	1→1	출금
3	1→1	입금
4	1→2	입금(잔고>0)
5	2→2	입금
6	2→1	출금(잔고<0)
7	1→3	동결
8	3→1	동결해지
9	1→4	(현재-마지막사용일)>5년
10	1→5	폐쇄(잔고=0)
11	4→5	폐쇄

10.5 통합 테스트

- 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

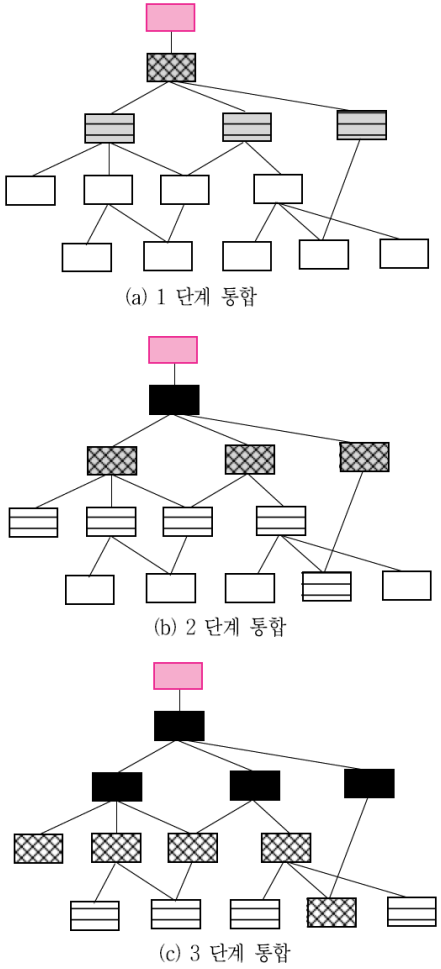
빅뱅 통합

- 한 번에 모든 모듈을 모아 통합
- 장점
  - 일정에 대한 관리가 편함
  - 통합을 위하여 스텁을 구성할 필요가 없음
- 단점
  - 오류의 위치와 원인을 찾기 어려움
  - 단위 테스트에 많은 시간과 노력이 듦
    - 준비해야 할 드라이버/스텁 수가 많음
  - 개발 진도를 예측하기 어려움



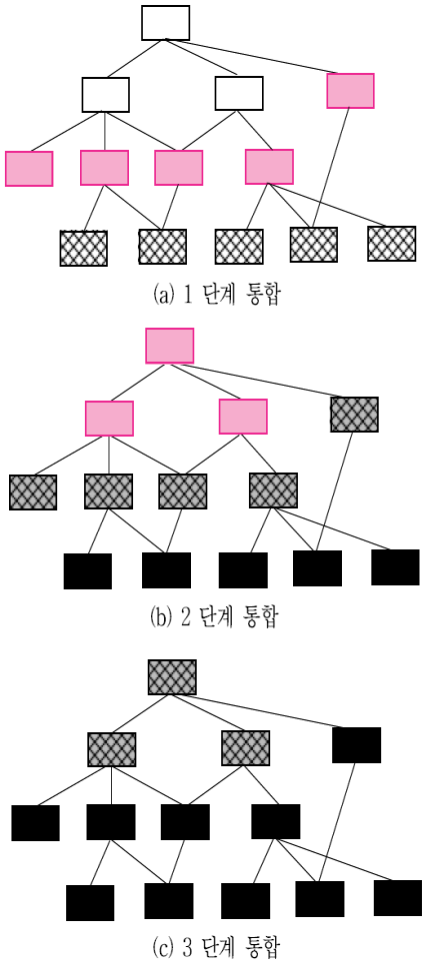
하향식 통합

- 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



상향식 통합

- 시스템 구조상 최하위에 있는 모듈부터 통합
- 장점
  - 점증적 통합 방식
    - 오류 발견이 쉬움
    - 하드웨어 사용 분산
  - 하위층 모듈을 더 많이 테스트
- 단점
  - 초기에 시스템의 뼈대가 갖추어지지 않음
  - 상위층의 중요한 인터페이스가 마지막에 가서야 확인 가능
  - 의뢰자에게 시스템을 시험해 볼 가능을 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

10.6 시스템 및 인수 테스팅

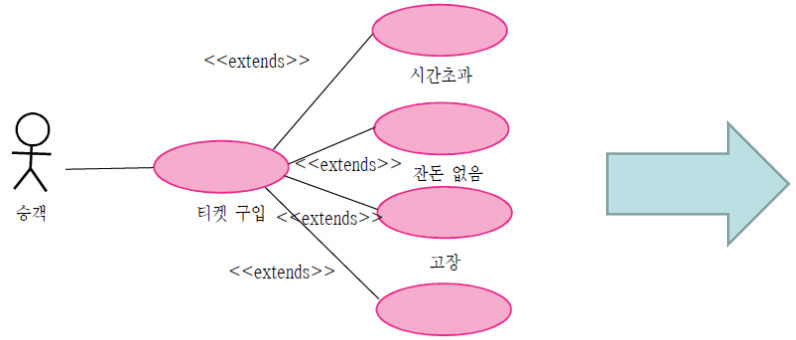
- 컴포넌트 통합 후 수행하는 테스트 기법
- 테스트 종류
  - 기능 테스트
  - 성능 테스트
  - 보안 테스트
  - 사용성 테스트
  - 인수 테스트
  - 설치 테스트

기능 테스트

- 기능적 요구와 시스템의 차이를 발견하기 위한 테스트
- 사용자와 관련되어 있으며 오류를 유발할 가능성이 많은 테스트를 선정
- 사용사례 모델을 검토하고 오류를 일으킬만한 유스케이스 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 테스트
    - 시스템 처리능력의 몇 배의 작업부하를 처리하고 견딜 수 있는지 측정
  - 성능 테스트
    - 정상적인 사용 환경에서 시스템의 성능을 측정하는데 사용
    - 시뮬레이션을 이용한 테스팅 가능
  - 보안 테스트
    - 시스템의 보안 취약점을 찾아내려는 목적

UI 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 테스트)