5강. 소프트웨어 테스트

■ 학습개요

소프트웨어 테스트는 품질 보증을 위한 활동으로 프로그램의 실행을 통해 존재하는 결함을 발견하기 위한 목적을 가진다. 테스트 작업은 단위 테스트, 통합 테스트 및 시스템 테스트로 구분할 수 있다. 시스템이 완전히 통합되면 기능적 요구사항 외에 비기능적 요구사항이 만족되는지 확인하고 검증해야 한다. 화이트박스 테스트는 프로그램의 논리 구조에 바탕을 두고 있어 구조 테스트라 하고 단위 테스트에 주로 사용된다. 블랙박스 테스트는 명세서에 기초한 입력과 그것의 출력 결과를 검사하여 기능적 요구사항을 확인하기 위한 것이다.

■ 학습목표

1	소프트웨어 테스트 작업의 목적을 이해한다.
2	소프트웨어 테스트 작업의 기본 원칙들을 열거할 수 있다.
3	시스템 통합 방식을 열거하고 장단점을 설명할 수 있다.
4	화이트박스 테스트에서 테스트 케이스 선정 기준을 구별할 수 있다.
5	블랙 박스 테스트에서 테스트 데이터를 구하는 방법을 나열하고 설명할 수 있다.
6	성능 테스트의 중요성과 방법을 기술할 수 있다.

■ 주요용어

용어	해설
소프트웨어 테스트	프로그램을 실행시켜 결함을 찾고자 하는 활동
결함 테스트/검증 테스트	결함 테스트는 소규모 코드에서 결함을 찾고자 하는 것이며 검증 테스트는 고객이 원하는 것인지를 보이기 위한 고수준 테스트
상향식/하향식 통합	제어 계층 구조상의 최상위 모듈부터 시작하여 아래의 하위 모듈들을 통합시켜 가는 것을 하향식이라 하며, 최하위 수준을 먼저 만들고 위 수준의 모듈들을 차례로 통합하는 것을 상향식 통합이라 함
화이트박스 테스트	프로그램의 논리 구조에 바탕을 두어 테스트하는 것으로 구조 테스트라고도 함
테스트 케이스 선정 기준	화이트박스 테스트 방식에서 테스트 케이스들이 얼마나 적정한지를 판단하는 기준으로 문장, 분기, 조건 검증 기준 등
블랙 박스 테스트	명세서에 기초하여 입력 데이터를 선정한 후 출력 결과를 조사하여 프로그램의 기능을 테스트하는 것으로 기능 테스라고도 함
회귀 테스트	프로그램의 수정으로 생길 수 있는 새로운 오류의 발생여부를 밝히기 위한 테스트 방법
원인-결과 그래프	프로그램의 명세를 분석하여 원인에 해당하는 입력 조건과 그 원인으로 발생되는 출력 결과를 논리적으로 연결하여 표현한 그래프
성능 테스트	평균 응답 시간이나 처리율 등의 성능 요인들을 파악하기 위한 테스트 방법

메뉴	학습하기 (PPT 내용)

5.1 소프트웨어 테스트 개요

소프트웨어 테스트
- 결함을 찾기 위한 의도로 프로그램을 실행시키는 것
  - 소프트웨어 품질 보증을 위한 활동
  - 검토를 통해 결함을 찾는 것은 정적 테스트라고 함
- 성공적인 테스트란 발견되지 못했던 결함을 찾는 작업

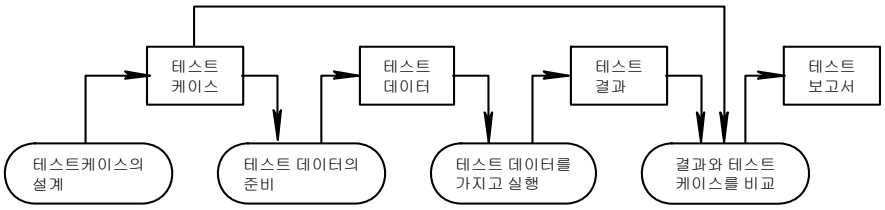
결함 테스트와 검증 테스트
- 결함 테스트
  - 소규모 코드에서 결함을 찾고자 하는 것
  - 소규모 코드를 확인(verify)하기 위함
  - 좁은 의미에서 테스트라고 할 때 결함 테스트를 의미
- 검증 테스트
  - 주요 시스템의 기능을 검증(validate)하기 위한 것
  - 인수 테스트와 같은 고수준 테스트

5.2 소프트웨어 테스트 원칙

테스트 작업의 원칙
- 테스트 케이스는 입력값 외에 출력값을 포함해야 함
- 자신이 작성한 프로그램을 테스트하지 말 것
- 프로그래밍 개발 조직이 자체적으로 테스트하지 말 것
- 테스트 작업의 후반부로 가서도 소홀히 하지 말 것
- 올바르지 못한 입력값이나 예상하기 힘든 입력값을 고려해야 함
- 정상적 동작의 확인은 물론 절대 해서는 안되는 행위를 하지 않는지 확인할 것
- 테스트 케이스를 버리지 말고 재사용할 것
- 오류가 없을 것이라는 가정을 하지 말 것
- 오류가 발견된 곳에서 추가적인 오류가 발생할 가능성이 높음

5.3 테스트 프로세스

테스트 프로세스



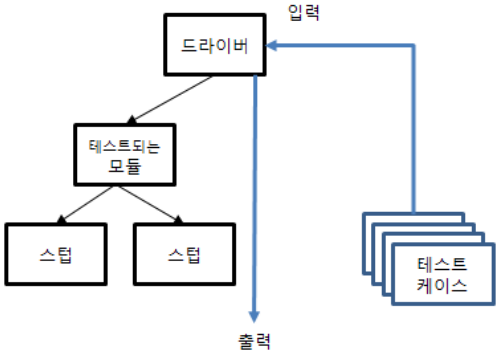
[그림 5-1] 테스트 프로세스

- 테스트 케이스: 테스트를 위한 입력과 기대되는 출력, 무엇을 검사할지에 관한 설명을 포함
- 테스트 데이터: 테스트에 사용되는 입력
- 고려 사항
  - 모든 가능한 실행 경로를 테스트하는 것은 현실적으로 불가능
  - 가능한 테스트 케이스들 중 일부만을 실행
  - 오류의 발견 확률이 높은 입력값을 구해서 테스트해야 함

테스트 작업의 우선순위
- 전체 시스템을 테스트하는 것이 부품 하나하나를 테스트하는 것보다 중요함
- 오래된 기능을 테스트하는 것이 새로운 기능을 테스트하는 것보다 중요함
- 일반적인 상황을 테스트하는 것이 예외적인 경우를 테스트하는 것보다 중요함

5.4 단계별 테스트

단위 테스트
- 시스템을 구성하는 기본 빌딩 블록을 테스트
- 개별적 모듈을 독립적으로 확인하는 작업
- 통합 테스트 전에 수행
- 드라이버(driver)와 스텁(stub)을 사용함
  - 드라이버는 테스트되는 모듈을 호출하며 결과를 출력해 주는 프로그램
  - 스텁은 테스트되는 모듈에 의해 호출되는 모듈



[그림 5-2] 단위 테스트

통합 테스트

- 단위 테스트된 개별 모듈들을 통합하여 상호작용으로 인한 문제가 있는지 테스트
  존재하는 결함을 발견하기 위함
  모듈들이 제 기능을 수행하고 모듈들이 정확히 호출되며 올바른 데이터가 인터페이스를 통해 적시에 전달되는지 검사
  테스트 케이스는 모듈 간의 인터페이스를 검사할 목적으로 개발됨
- 통합 테스트는 프로그램을 구축해 가는 기술이며 최종적으로는 시스템이 구축됨
- 주로 블랙박스 테스트 기법을 사용

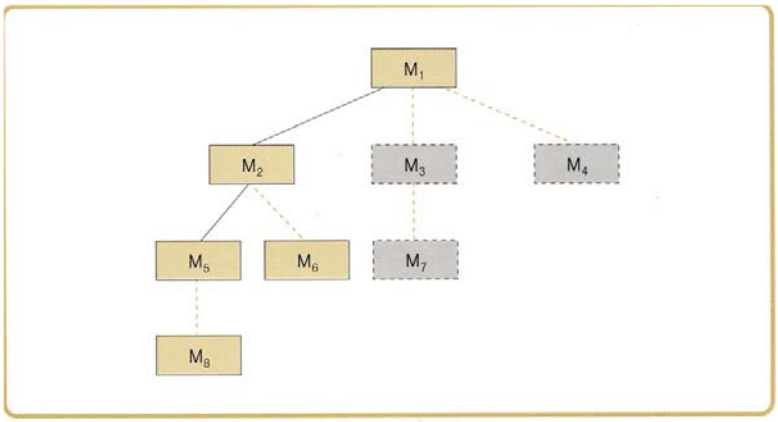
시스템 통합 방식

- 빅뱅 통합
  모듈들을 모두 개발한 후 한꺼번에 통합
- 점증적 통합
  모듈을 하나씩 추가하여 통합한 후 테스트함

하향식 통합

- 점증적 통합 방식으로 제어 계층 구조상에서 최상위 모듈부터 시작하여 아래 모듈들을 차례로 통합시킴
  통합되어 테스트되는 모듈들의 하위 모듈에 대해 스텁을 작성해야 함
  깊이 우선 방식과 너비 우선 방식

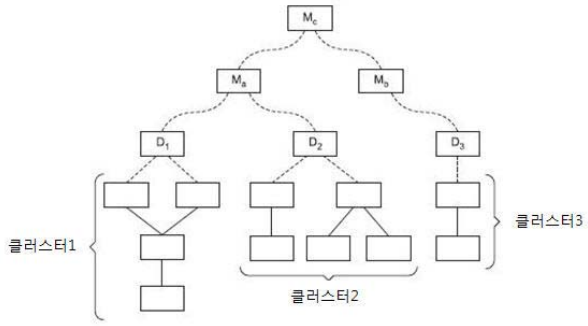
-장단점
  초기에 소프트웨어 구조가 갖추어지고 개발자에게 심리적 안정감을 줌
  병행 작업이 어렵고 입출력 모듈이 하위에 위치하므로 테스트 작업이 어려움



[그림 5-4] 하향식 통합(깊이 우선)

상향식 통합

- 프로그램 구조에서 최하위 모듈들을 먼저 만들어 테스트하고 위 수준으로 올라가며 통합
- 하위 모듈들을 통합하다 보면 클러스터를 형성하게 됨



[그림 5-4] 상향식 통합

- 장단점
  - 초기 단계에서 병행 작업이 가능하며 대규모 시스템을 통합할 때 적당
  - 골격을 갖추는 데 오랜 시간이 걸리고 같은 수준의 모듈들이 준비되어야 테스트할 수 있음

회귀 테스트
- 프로그램을 수정할 때, 수정으로 인한 오류의 발생 여부를 밝히기 위한 테스트 방법
- 이전 단계에서 사용한 테스트 케이스 집합을 재사용할 수 있음
  - 수정된 부분과 수정에 의한 파급 효과를 분석하여 선택적으로 재사용해야 함

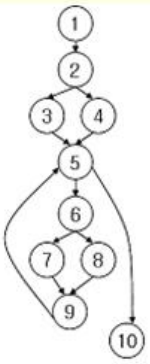
샌드위치 테스트
- 상향식과 하향식을 조합한 방식

시스템 테스트
- 완전한 시스템이 구축된 후 기능적/비기능적 요구사항이 만족되는지 테스트
  - 블랙박스 테스트 작업을 수행하며 성능이나 신뢰도를 테스트할 수 있음
- 릴리스 테스트라고도 하며 테스트 작업에 고객이 포함되면 인수 테스트가 됨

5.5 화이트박스 테스트(구조 테스트)

화이트박스 테스트
- 프로그램의 논리 구조에 바탕을 둔 구조 테스트
  - 프로그램 구현 사항을 알아야 함
  - 프로그램의 제어 흐름 그래프에서 경로를 분석하여 테스트 케이스 개발
  - 소규모의 프로그램에 적용
  - 자동화된 테스트 도구를 사용할 수 있음

```
1:  read x;
2:  if (x > 0)
3:        print x+1;
4:  else
        print x-1;
5:  while (x > 5)
6:    { if (x equals 10)
7:          print "oh";
8:      else
          print "yes";
9:      subtract 4 from x;
    }
10: print "End";
```



[그림 5-6] 프로그램과 제어 흐름 그래프

테스트 케이스 선정 기준

- 모든 가능한 실행 경로를 테스트할 수 없으므로 적정 수의 테스트 경로를 실행해야 함
  효과적인 테스트 케이스의 집합을 구했는지 또는 테스트 작업이 적정한지를 판단하는 기준
- 문장 검증 기준(SC)
  프로그램의 모든 문장을 한 번 이상 실행
- 분기 검증 기준(DC)
  모든 분기점에서 참과 거짓에 해당하는 경로를 한 번 이상 실행
- 조건 검증 기준(CC)
  모든 분기점에서 조건식을 구성하는 단일 조건의 참과 거짓을 한번 이상 실행
  예) if (x>0 && y<=-3)에서 테스트 데이터는 (x=1, y=1), (x=-1, y=-4)

	(x>0 && y<=-3)	x>0	y<=-3
(x=1, y=1)	F	T	F
(x=-1, y=-4)	F	F	T

  <- 조건 검증 기준은 만족하나 분기 검증 조건을 만족하지 못함
- 조건/분기 검증 기준(CDC)
  조건 검증 기준과 분기 검증 기준을 모두 만족해야 함
  예) if (x>0 && y<=-3)에서 테스트 데이터는 (x=1, y=-4), (x=-1, y=4)

	(x>0 && y<=-3)	x>0	y<=-3
(x=1, y=-4)	T	T	T
(x=-1, y=4)	F	F	F

  <- 'short-circuiting'의 경우 문제가 됨
- 수정된 조건/분기 검증 기준(MCDC)
  예) if (x>0 && y<=-3)에서 테스트 데이터는 (x=4, y=-4), (x=-1, y=-4), (x=4, y=-2)

	(x>0 && y<=-3)	x>0	y<=-3
(x=4, y=-4)	T	T	T
(x=-1, y=-4)	F	F	T
(x=4, y=-2)	F	T	F

- 복수 조건 검증 기준(MCC)

조건식을 구성하는 단일 조건식들의 모든 가능한 참/거짓 조합을 한 번 이상 실행

예)

	(x>0 && y<=-3)	x>0	y<=-3
(x=1, y=1)	F	T	F
(x=-1, y=-4)	F	F	T
(x=1, y=-4)	T	T	T
(x=-1, y=4)	F	F	F

- 경로 검증 기준(PC)

프로그램에 존재하는 모든 실행 가능한 경로를 한 번 이상 테스트

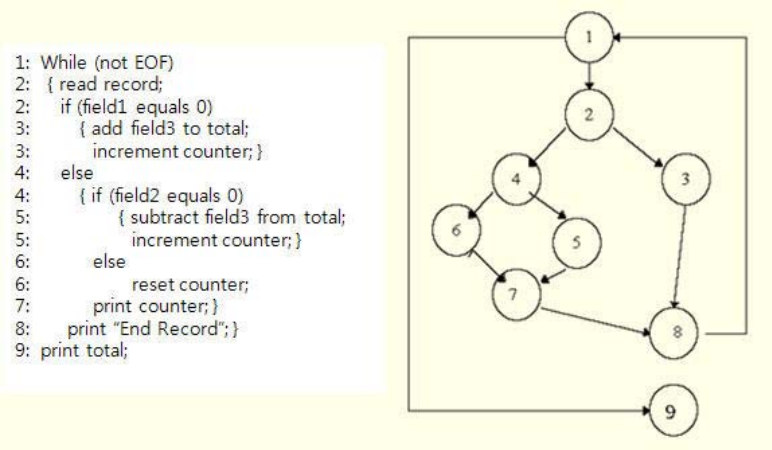
반복 문장이 있다면 실행 가능한 경로의 수는 무한대이므로 사실상 불가능

- 기본 경로 테스트

시작 노드에서 종료 노드까지의 선형 독립적인 경로(기본 경로)를 모두 테스트

메케이브의 사이클로매틱 수는 기본 경로의 개수와 일치함

예) 그림에서 (1, 9), (1, 2, 3, 8, 1, 9), (1, 2, 4, 5, 7, 8, 1, 9), (1, 2, 4, 6, 7, 8, 1, 9)를 만족하는 테스트 집합을 구함



[그림 5-7] 프로그램과 제어 흐름 그래프

5.6 블랙박스 테스트

블랙박스 테스트

- 명세서에 기초하여 기능을 검사하기 위한 테스트 데이터를 개발

프로그램의 구조를 고려하지 않음

기능 테스트 또는 행위 테스트라고 함

- 기능적 요구사항을 검사할 수 있으며 오류를 일으킬 가능성이 높은 입력 조건을 파악해야 함

- 장점

코드를 분석하지 않으므로 테스터는 개발자로부터 자유로움
특정 프로그래밍 언어에 대한 지식이 없어도 됨
사용자 관점에서 테스트를 수행
요구 명세서만 작성되면 테스트 케이스를 설계할 수 있음

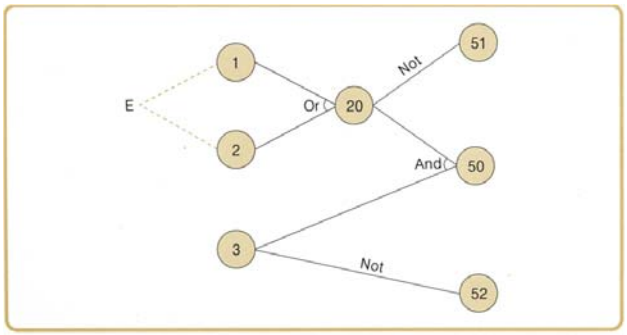
블랙박스 테스트를 위한 테스트 케이스 개발 방법
- 동치 분할
입력 집합을 몇 개의 동치 클래스들로 나누어 테스트하는 것
요구사항에 기초하여 분할을 만든 후 각 분할에서 대표 값을 선정함

예)

테스트 케이스#	테스트 데이터	예상 세액	동치 클래스	세율
1	950	57	₩1~₩1200	6%
2	3300	528	₩1201~₩4600	16%
3	5500	1375	₩4601~₩8800	25%
4	9200	3220	₩8801~	35%

- 경계값 분석
동치 분할 방법의 변형으로 동치 클래스를 정의한 후 경계값과 경계값의 직전/직후 값을 테스트하는 것
경계값 주변에서 오류의 가능성이 높다는 점을 가정한 방법
예) 0~100 사이의 정수 입력 : -1, 0, 100, 101을 테스트 데이터로 함

- 원인-결과 그래프
명세서를 분석하여 원인에 해당하는 입력 조건과 그것의 출력 결과를 논리적으로 연결한 그래프를 작성하고 의사결정 테이블로 바꾼 후 테스트 케이스를 개발함



[그림 5-10] 원인-결과 그래프

<표 5-1> [그림 5-10]을 **의사결정 테이블로 바꾼 결과**

	테스트 케이스			
	1	2	3	4
원인				
1	1	0	0	
2	0	1	0	
3	1	1		0
결과				
50	1	1	0	0
51	0	0	1	0
52	0	0	0	1

5.7 시스템 테스트와 비기능성 테스트

비기능성 테스트
- 기능적 요구사항 이외의 것을 테스트하는 것으로 시스템의 동작 방식과 시스템의 제약 조건을 확인하기 위한 목적을 가짐

성능 테스트

- 실제 운영 중에 성능 수준을 보장할 수 있는지 테스트하는 것
  - 평균 응답 시간, 부하율, 피크 시간의 성능 검사
  - 부하를 계속적으로 증가시키면서 테스트 작업을 연속적으로 수행함
  - 트랙잭션의 유형별 비중을 고려하여 테스트 케이스를 작성함
- 부하 테스트
  - 여러 사용자가 동시 작업하는 것을 가정하고 성능 요인을 변화시키면서 관찰함
  - 비정상적인 높은 부하를 주고 관찰하여 잘못된 점을 발견(스트레스 테스트)
- 스트레스 테스트
  - 시스템의 설계 한도를 벗어난 요구를 시험해 보는 것
  - 네트워크 과부하로 인한 성능 저하가 우려되는 분산 시스템의 테스트
  - 워드 프로세서가 문서를 읽을 때 데이터 볼륨을 비정상적으로 높이는 것(볼륨 테스트)
- 고려 사항
  - 시스템에 부하가 걸릴 때도 심각한 고장으로 연결되지 않아야 함
  - 스트레스 테스트를 통해서 정상적인 상황에서 드러나지 않았던 결함을 발견할 수 있음

보안 테스트

- 시스템을 보호하기 위해 보안성을 테스트하는 작업
  - 기밀 유지, 무결성, 가용성을 보장하기 위함
  - 보안 오류를 야기하는 고의적 침입 시도, 지속적인 서비스 요청을 통해 다른 사용자의 서비스를 방해하는 행위, 불법적인 로그인 등을 시도하여 테스트함

메뉴	**연습문제**

1. 소프트웨어 테스트 작업에 관한 설명으로 올바른 것은?

① 테스트 케이스는 테스트를 위한 입력 데이터만을 의미한다.

② 작성자 스스로 프로그램을 테스트할 때 보다 공격적인 테스트 작업을 수행할 수 있다.

③ 테스트 케이스를 설계할 때는 비정상적인 입력값도 고려해야 한다.

④ 테스트 작업의 목적은 프로그램에 오류가 없음을 보이기 위한 것이다.

<정답> ③

<해설> 테스트 작업의 목적은 오류를 발견하기 위한 것이다. 테스트 케이스는 입력값외에 기대되는 출력결과와 무엇을 테스트하는 것인지에 관한 설명까지 포함한다.

2. 통합 테스트에 관한 설명으로 틀린 것은?

① 상향식은 하위 모듈에서 위로 올라가면서 통합하는 것이다.

② 하향식은 초기에 사용자에게 전체 구조를 보여줄 수 있다.

③ 상향식에서는 드라이버 모듈이 하향식에서는 스텁이 필요하다.

④ 연쇄식 통합은 한꺼번에 모든 모듈들을 통합하는 것이다.

<정답> ④

<해설> 연쇄식 통합은 기능별로 모듈들을 통합하는 것이다. 이것을 스레드라고 한다. 스레드별로 독립적으로 통합을 한 후 최종적으로 스레드들을 통합한다.

3. 화이트 박스 테스트에 관한 설명으로 맞는 것은?

① 명세서에 기초해 입력에 대한 모듈의 출력 결과를 검사하여 기능을 테스트한다.

② 모듈의 제어 구조에 기초하여 테스트 케이스 선정 기준을 정한다.

③ 모든 가능한 입력 조합을 주어 테스트한다.

④ 동치 분할에 의해 정상적인 입력값 가운데 중간값을 테스트한다.

<정답> ②

<해설> 화이트 박스 테스트를 구조 테스트라 하며 블랙박스 테스트를 기능 테스트 또는 행위 테스트라고 한다.

4. 화이트박스 테스트 선정 기준들 간의 관계를 설명한 것으로 잘못된 것은?

① 분기 검증 기준을 만족하는 테스트 케이스 집합은 문장 검증 기준을 만족한다.

② 조건 검증 기준을 만족하는 테스트 케이스 집합은 분기 검증 기준을 만족한다.

③ 조건/분기 검증 기준의 테스트 범위는 분기와 조건 검증 기준의 테스트 범위를 포함한다.

④ 모든 에지를 한 번 이상 수행한다면 분기 검증 기준의 테스트 범위를 포함하는 것이다.

<정답> ②

<해설> 조건 검증 기준은 모든 분기점에서 조건식을 구성하는 단일 조건들의 참과 거짓을 한 번 이상 수행해야 한다는 것이다. 그러나 이것이 전체 조건식의 참과 거짓을 한번 이상 수행하도록 보장하지는 못한다.

5. 블랙박스 테스트 방법에 해당하지 않는 것은?

① 동치 분할

② 제어 구조 분석

③ 경계값 분석

④ 원인-결과 그래프

<정답> ②
<해설> 소스 코드의 제어 구조나 데이터 흐름을 분석하여 테스트 케이스를 개발하는 것은 화이트박스 테스트 방법이다.

메뉴	정리하기

1. 소프트웨어 테스트의 목적은 무엇인가?

□ 소프트웨어에 숨어 있는 결함을 발견하기 위한 것이다. 테스트 작업이 정확성을 보이기 위한 것은 아니다.

2. 프로그램의 작성자나 개발 조직이 스스로 프로그램을 테스트한다고 할 때의 문제점은 무엇인가?

□ 자신의 실수를 스스로 발견하기 어려운 점, 자신의 코드에 대해 공격적인 테스트를 하기 어렵다는 점이다. 또한 개발 조직은 마감일이나 비용에 맞추기 위해 테스트를 소홀히 할 수 있기 때문이다.

3. 하향식 통합 방법을 설명하라.

□ 프로그램 구조를 구축해 가는 점증적 방법의 하나이다. 제어 계층 구조의 최상위 모듈부터 시작하여 아래 방향으로 진행하면서 모듈들을 차례로 통합시킨다. 상위 수준의 모듈들이 먼저 테스트되고 반복 테스트되며 초기에 소프트웨어 구조가 갖추어진다. 병행 작업이 어려우며 입출력 모듈이 하위 수준에 위치하므로 테스트 작업 쉽지 않고 많은 스텁들이 필요하게 된다.

4. 화이트박스 테스트 케이스의 선정 기준에서 경로 검증 기준을 설명하라.

□ 프로그램에 존재하는 모든 실행 가능한 경로를 한 번 이상 실행한다는 것이다. 그러나 단순한 반복 구조만 있어도 가능한 실행 경로는 매우 커지므로 현실적으로 불가능하다. 따라서 반복 횟수를 2회로 제한하여 실행 경로는 고려하거나, 제어 그래프상에서 모든 에지를 한 번 이상 수행하는 것으로 가정하기도 한다.

5. 블랙박스 테스트에서 경계값 분석 방법을 설명하라?

□ 동치 분할 방법의 변형으로 동치 클래스의 범위를 정의한 후, 각 범위의 중간값보다 경

계에서 오류 발생 확률이 높다는 점을 이용한 테스트 방법이다.

6. 스트레스 테스트란 무엇인가?

□ 성능 테스트 방법의 하나로 시스템의 설계 한도를 벗어난 비정상적인 높은 부하를 주어 테스트하는 것이다. 이러한 상황에서도 심각한 고장으로 연결되지 않도록 해야 한다.