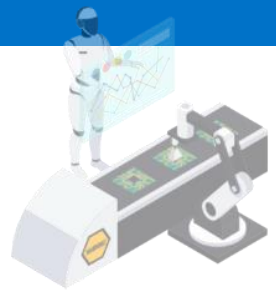
5강. 소프트웨어 테스트

컴퓨터과학과 김희천 교수

목차



Chapter. 1

소프트웨어 테스트 개요

1. 소프트웨어 테스트

- **소프트웨어 품질 보증을 위한 활동**
  - V&V 활동의 하나
  - 품질 확보를 위해 개발과정에서 테스트 비중이 큼
- **프로그램을 실행시켜 요구 사항의 만족을 보이거나 결함을 찾기 위한 활동**
  - 입력 데이터를 가지고 실행시키며 동적 테스트라고 함
  - 오류를 찾기 위한 것이며, 오류가 없음을 증명하는 것이 아님
- **성공적인 테스트란 발견되지 못했던 결함을 찾는 테스트임**

2. 결함 테스트와 검증 테스트

결함 테스트

- 소규모 코드에서 결함을 찾고자 하는 것
- 부정확한 계산이나 데이터 오류 등이 발생하는지 확인
- 좁은 의미에서 테스트라고 할 때, 결함 테스트를 의미

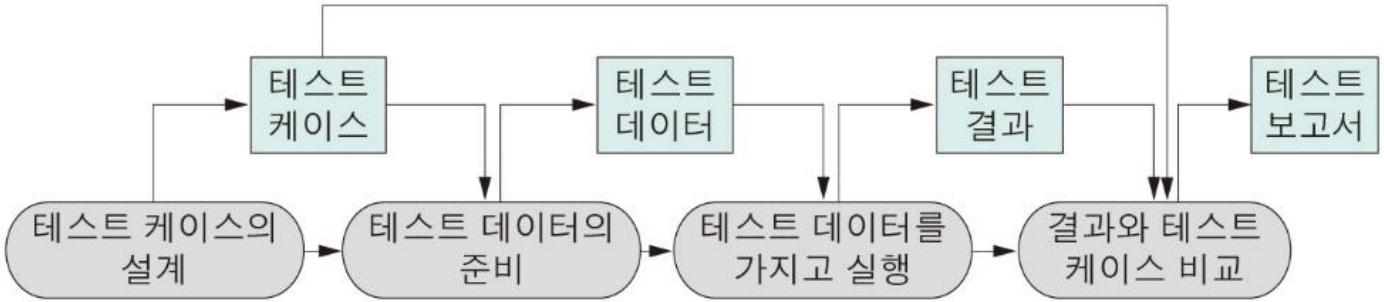
검증 테스트

- 주요 시스템의 기능을 검증하기 위한 것
- 주어진 요구 명세의 만족을 보이는 인수 테스트와 같은 고수준 테스트

3. 테스트 작업의 원칙

- 테스트 케이스는 입력 값 외에 결과 값을 포함해야 함
- 자신이 작성한 프로그램을 스스로 테스트하지 말 것
- 프로그래밍 조직이 자체적으로 테스트하지 말 것
- 테스트 작업의 후반부로 가서도 소홀히 하지 말 것
- 올바르지 못한 입력 값이나 예상하기 힘든 입력 값을 고려해야 함
- 정상적 동작의 확인은 물론 절대로 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

4. 소프트웨어 테스트 프로세스



- **테스트 케이스**
  - 테스트를 위한 입력과 기대되는 출력, 무엇을 검사할 지에 관한 설명을 포함
- **테스트 데이터**
  - 테스트에 사용되는 입력 데이터

5. 테스트 작업의 고려사항과 우선 순위

+ 고려 사항

- × 모든 가능한 실행 경로나 모든 가능한 테스트 데이터를 테스트하는 것은 현실적으로 불가능
- × 가능한 테스트 케이스들 중 일부만을 실행
- × 오류의 발견 확률이 높은 입력 값을 구해서 테스트해야 함
- × 어느 수준까지 테스트할 것인지 정함

테스트 작업의 우선 순위

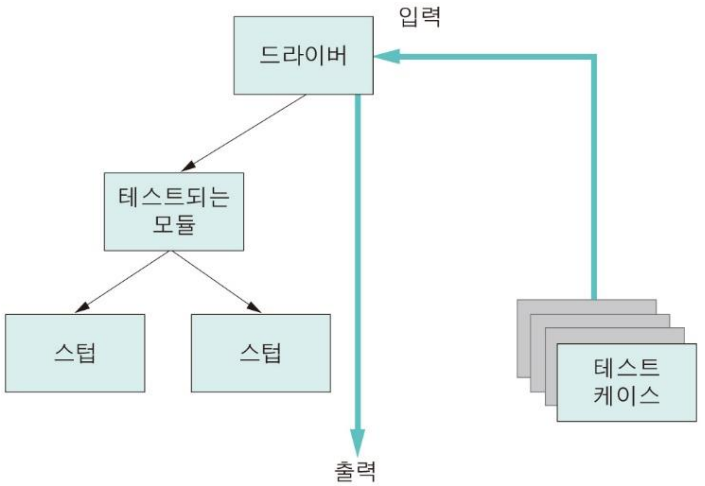
- 전체 시스템을 테스트하는 것이 모듈 하나보다 중요함
- 기존 기능을 테스트하는 것이 새로운 기능보다 중요함
- 일반적 상황을 테스트하는 것이 예외적인 경우보다 중요함

Chapter. 2

단계별 테스트

1. 단위 테스트

- 시스템을 구성하는 기본 단위를 테스트
  - 결함 발견을 위해 개별적 모듈을 독립적으로 확인하는 작업
- 다른 모듈과 통합되기 전에 수행함
- 드라이버(driver)와 스텁(stub)을 사용
  - 드라이버는 테스트되는 모듈을 호출하며 결과를 출력해 주는 프로그램
  - 스텁은 테스트되는 모듈에 의해 호출되는 모듈



2. 통합 테스트

- **테스트된 개별 모듈들을 통합하여 상호작용으로 인한 문제가 있는지 테스트**
  - 주로 상호작용에서 생기는 결함을 발견하기 위함
  - 모듈들이 정확히 호출되며, 함께 동작하고, 올바른 데이터가 인터페이스를 통해 적시에 전달되는지 검사
  - 테스트 케이스는 모듈 간의 인터페이스를 검사할 목적으로 개발됨
- **통합 테스트는 프로그램을 구축해 가는 기술이며 최종적으로 시스템이 구축됨**
- **주로 블랙박스 테스트 기법을 사용**

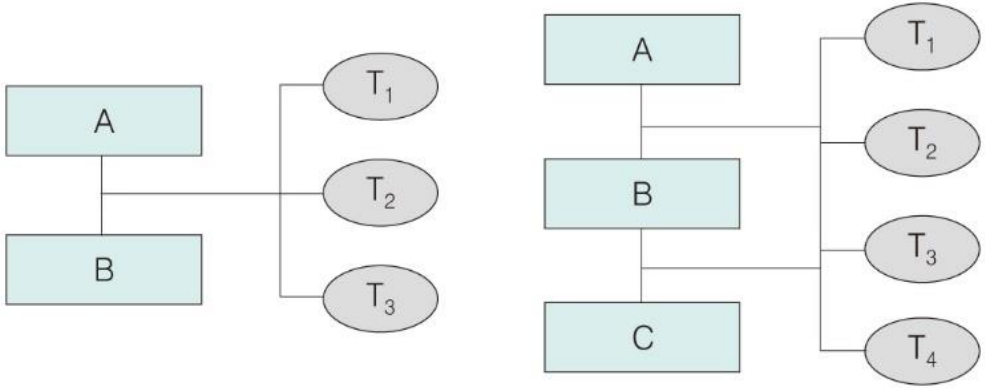
3. 시스템 통합 방식 (1/5)

+ 빅뱅 통합

× 모듈들을 모두 개발한 후 한꺼번에 통합

+ 점증적 통합

× 모듈을 하나씩 추가하여 통합한 후 테스트함



3. 시스템 통합 방식 (2/5)

✚ 하향식 통합

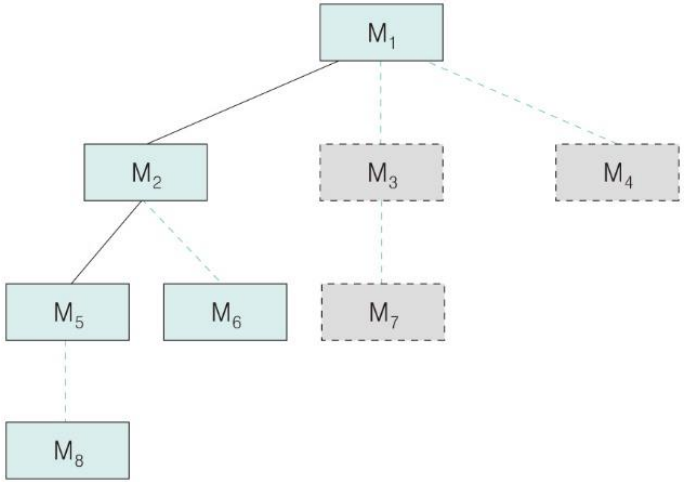
- 점증적 통합 방식으로 제어 계층 구조상에서 최상위 모듈부터 시작하여 아래 모듈들을 차례로 통합시킴
- 통합되어 테스트되는 모듈들의 하위 모듈에 대해 스텁이 필요함
- 깊이 우선 방식과 너비 우선 방식

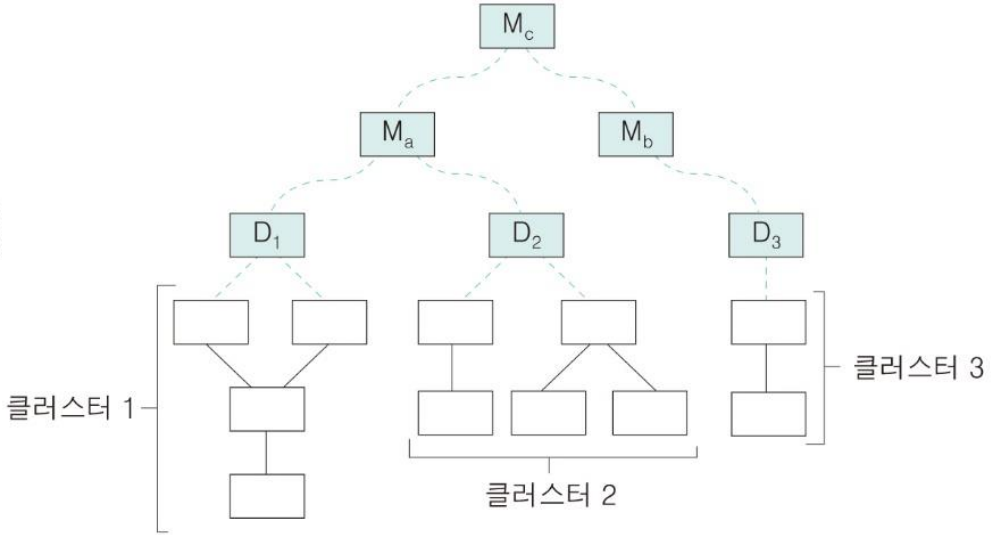
✕ 장단점

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

3. 시스템 통합 방식 (3/5)

- 하향식 통합(깊이 우선)과 상향식 통합(오른쪽)





3. 시스템 통합 방식 (4/5)

✚ 상향식 통합

- 프로그램 구조에서 최하위 모듈들을 먼저 만들어 테스트하고 상위 수준으로 올라가며 통합과 테스트를 함
- 하위 모듈들을 통합하다 보면 클러스터를 형성하게 됨
- 통합되는 모듈을 제어하기 위해 드라이버가 필요함

✕ 장단점

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

3. 시스템 통합 방식 (5/5)

✚ 샌드위치 테스트

✕ 상향식과 하향식을 조합한 방식

회귀 테스트

- 프로그램을 수정할 때, 수정으로 인한 오류의 발생 여부를 밝히기 위한 테스트 방법
- 이전 단계에서 사용한 테스트케이스 집합을 재사용할 수 있음
- 수정된 부분과 수정에 의한 파급 효과를 분석하여 선택적으로 재사용하는 것이 필요함

4. 시스템 테스트

- 완전한 시스템이 구축된 후, 기능적 요구사항과 비기능적 요구사항이 만족되는지 확인하고 검증하기 위해 테스트하는 것
  - 사용자에게 전달되는 시스템 버전을 테스트함
  - 요구사항 명세서에 기초하여 블랙박스 테스트 작업을 수행
  - 성능이나 신뢰도를 테스트할 수 있음
- 릴리스 테스트라고도 하며
  테스트 작업에 고객이 포함되면 인수 테스트가 됨

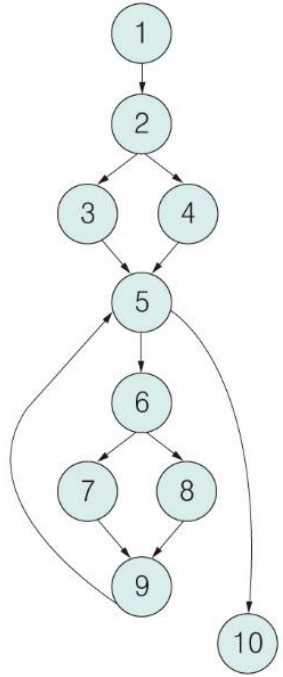
Chapter. 3

화이트박스 테스트

1. 화이트박스 테스트

- 프로그램의 논리 구조에 바탕을 둔 테스트
  - 구조 테스트라고 할 수 있음
  - 프로그램 구현 사항을 알아야 함
  - 프로그램의 제어 흐름 그래프에서 경로를 분석하여 테스트케이스를 개발
  - 소규모의 프로그램에 적용함
  - 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:       els
             print “yes”;
9:       subtract 4 from x;
   }
10: print “End”
```



2. 테스트케이스 선정 기준 (1/5)

코드 커버리지

- 화이트박스 테스트에서 효과적인 테스트 케이스의 집합을 구하는 기준
- 모든 가능한 실행 경로를 테스트할 수 없으므로 적정 수의 테스트 경로를 실행해야 함

+ 문장 검증 기준(SC)

× 프로그램의 모든 문장을 한 번 이상 실행

+ 분기 검증 기준(DC)

× 모든 제어 분기점에서 참과 거짓에 해당하는 경로를 각각 한 번 이상 실행

2. 테스트케이스 선정 기준 (2/5)

+ 조건 검증 기준(CC)

- × 모든 분기점에서 조건식을 구성하는 단일 조건의 참과 거짓을 각각 한번 이상 실행
- × 예) if (x>0 && y<=-3)에서 테스트 데이터는 (x=1, y=1), (x=-1, y=-4)
- × 분기 검증 조건의 불만족 문제

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

+ 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)
- × ‘short-circuiting’의 문제

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

2. 테스트케이스 선정 기준 (3/5)

✚ 수정된 조건/분기 검증 기준(MCDC)

- ✕예) if (x>0 && y<=-3)

	(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 각각 한 번 이상 실행

	(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

2. 테스트케이스 선정 기준 (4/5)

+ 경로 검증 기준

- × 프로그램에 존재하는 모든 실행 가능한 경로를 한 번 이상 테스트
- × 반복 문장이 있다면 실행 가능한 경로의 수는 무한대이므로 사실상 불가능

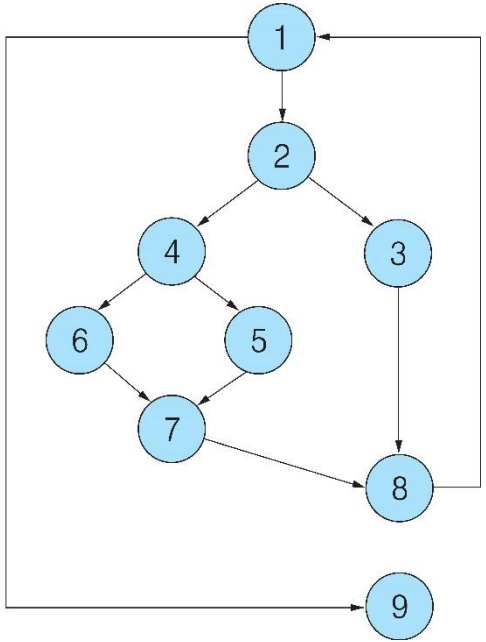
+ 기본 경로 테스트

- × 시작 노드에서 종료 노드까지의 선형 독립적인 경로(기본 경로)를 모두 테스트
- × 메케이브의 사이클로매틱 수는 기본 경로의 개수와 일치함
- × 예) 그림에서 기본 경로는 (1, 9), (1, 2, 3, 8, 1, 9), (1, 2, 4, 5, 7, 8, 1, 9), (1, 2, 4, 6, 7, 8, 1, 9). 이 경로를 실행시키는 테스트 집합을 구함

2. 테스트케이스 선정 기준 (5/5)

✚ 프로그램과 제어 흐름 그래프

```
1: while(not EOF)
2: { read record;
2:    if(field1 equals 0)
3:       { add field3 to total;
3:         increment counter; }
4:    else
4:       { if(field2 equals 0)
5:            { subtract field3 from total;
5:              increment counter; }
6:         else
6:              reset counter;
7:         print counter; }
8:     print  "End Record"  ; }
9: print total;
```



Chapter. 4

블랙박스 테스트

1. 블랙박스 테스트

✚ 명세서에 기초하여 기능을 검사하기 위한 테스트 데이터를 개발
- 프로그램의 구조를 고려하지 않음
- 주어진 입력에 대한 출력 결과를 조사함
- 기능 테스트 또는 행위 테스트라고 함

✚ 기능적 요구사항을 검사할 수 있고, 오류를 일으킬 가능성이 높은 입력 조건을 파악해야 함

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

2. 블랙박스 테스트 방법 (1/3)

- 완전 테스트
- 랜덤 테스트
- 동치 분할
  - 입력 집합을 몇 개의 동치 클래스들로 나누어 테스트하는 것
  - 요구사항에 기초하여 분할을 만든 후, 각 분할에서 대표 값을 선정함

테스트 케이스	테스트 데이터	예상 세액	동치 클래스	세율
1	950	57	1~1200	6%
2	3300	495	1201~4600	15%
3	5500	1320	4601~8800	24%
4	9200	3220	8801~	35%

2. 블랙박스 테스트 방법 (2/3)

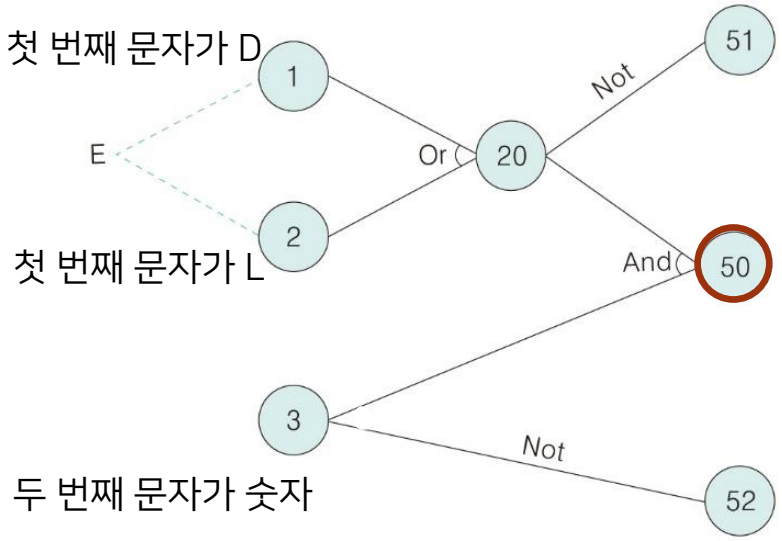
+ 경계값 분석

- × 동치 분할 방법의 변형으로 동치 클래스를 정의한 후
  경계값과 경계값의 직전/직후 값을 가지고 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

2. 블랙박스 테스트 방법 (3/3)



	테스트 케이스			
	1	2	3	4
원인				
1	1	0	0	
2	0	1	0	
3	1	1		0
결과				
50	1	1		
51			1	
52				1
예	D5	L4	B2	DA

Chapter. 5

비기능성 테스트

1. 비기능성 테스트와 성능 테스트

✚ 비기능성 테스트

- 기능적 요구사항 이외의 것을 테스트하는 것으로 시스템의 동작 방식과 시스템의 제약 조건을 확인하기 위한 목적을 가짐
- 신뢰성, 성능, 안전성, 빠른 복구, 사용편의성, 확장성 등의 확인을 위함

✕ 성능 테스트

- 실제 운영 중에 성능 수준을 보장할 수 있는지 테스트하는 것
- 평균 응답 시간, 시간당 처리율, 피크 시간의 성능 검사
- 테스트 작업을 장기적 연속적으로 수행함
- 트랙잭션의 유형별 비중을 고려하여 테스트 케이스를 작성함

2. 부하 테스트

- ✚ 성능 테스트의 일종으로 여러 사용자가 동시 프로그램을 사용하는 것을 가정하고 성능 요인을 변화시키면서 관찰함
- ✚ 비정상적인 높은 부하를 주고 관찰하여 잘못된 점을 발견(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
  - 스트레스 테스트를 통해서 정상적인 상황에서 드러나지 않았던 결함을 발견할 수 있음
  - 시스템에 부하가 걸릴 때도 심각한 고장으로 연결되지 않아야 함

3. 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

다음강의

6강. 사용자 요구분석