IT@COOKBOOK

쉽게 배우는 소프트웨어 공학

2판

Chapter 09 테스트

목차

01. 테스트의 이해

■ 테스트와 소프트웨어 테스트

- 소프트웨어 개발에서의 테스트
  - 사후약방문 (死後藥方文): 사람이 죽은 뒤에 약을 짓는다 -> 일을 그르친 뒤에는 아무리 뉘우쳐봐야 이미 늦었다
  - 높은 곳에서 사람이 떨어질 때를 대비해 구급차를 준비하는 것보다 사전에 울타리를 쳐 놓는 것이 훨씬 안전
  - 소프트웨어 개발 과정에서 테스트는 울타리에 비유

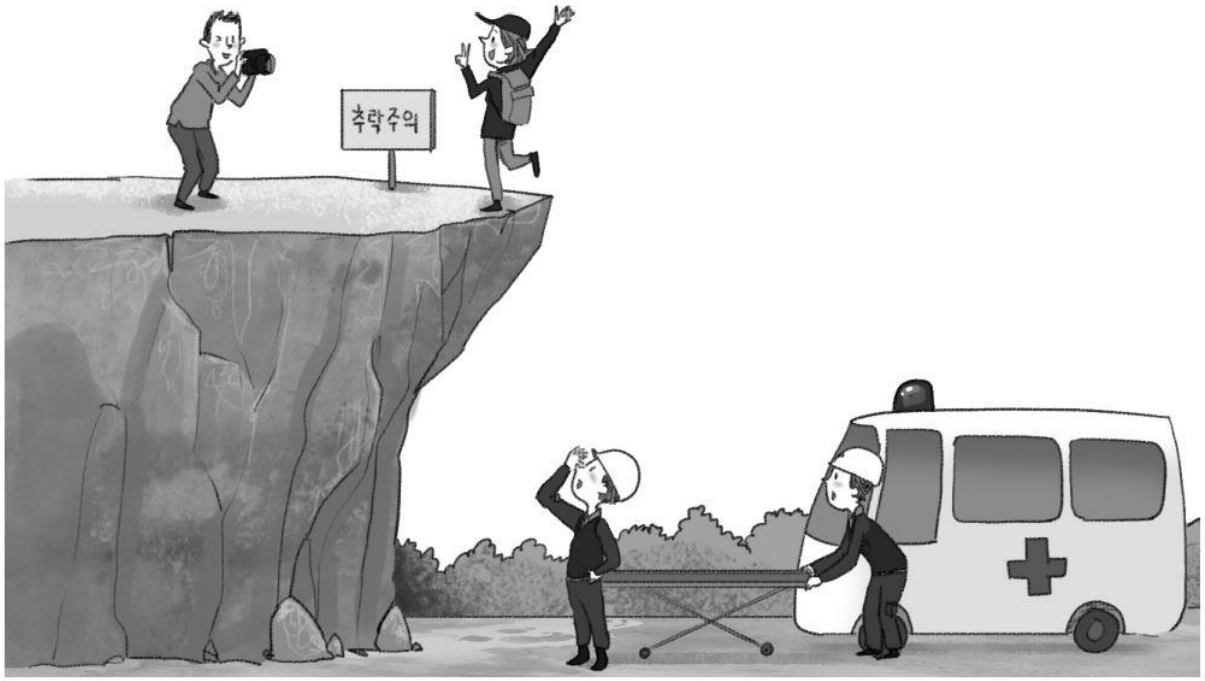


그림 9-1 **테스트가 주는 교훈:** 절벽 밑에 구급차를 마련해 놓는 것보다 절벽 위에 울타리를 쳐 놓는 것이 안전하다.

01. 테스트의 이해

■ 테스트와 소프트웨어 테스트

▪ 소프트웨어 오류로 인한 사고 사례

- 걸프전 당시 페트리어트 미사일 실패
  - 미사일이 1km를 날아갈 때마다 0.5~1초 정도의 아주 미세한 오차 발생 때문

- 방사선 치료기 테락-25의 사고
  - 테락-20을 테락-25로 버전 업할 때, 사고가 없던 테락-20만 믿고 테락-25의 전체 코드에 대한 테스트를 생략했기 때문

- 상업용 우주선 아리안 5호의 공중 폭발
  - 64비트 값을 16비트 정수로 변환하는 과정에서 오버플로가 발생
  - 아리안 4호 프로젝트에서 내버려둔 불필요한 데드 코드를 간과한 결과

01. 테스트의 이해

■ 테스트의 필요성과 특징

▪ 전문가들이 정의한 소프트웨어 테스트

- IEEE
  - 시스템이 명시된 요구를 잘 만족하는지, 즉 예상된 결과와 실제 결과가 어떤 차이를 보이는지 수동이나 자동으로 검사하고 평가하는 작업을 의미
- 그 밖의 정의
  - 달, 다익스트라, 호어: 테스트는 결함이 있음을 보여줄 뿐, 결함이 없음을 증명할 수는 없음

▪ 소프트웨어 테스트의 정의

- 소프트웨어에 내에 존재하지만 드러나지 않고 숨어 있는 오류를 발견할 목적으로, 개발 과정에서 생성되는 문서나 프로그램에 있는 오류를 여러 기술을 이용해 검출하는 작업
- 오류를 찾아내 정상적으로 실행될 수 있도록 하는 정도이지, 소프트웨어에 오류가 없음을 확인시켜주지는 못함
- 테스트는 오류를 찾고 올바르게 수정하여 프로그램을 작동시킬 수는 있지만, 그 프로그램이 완전하고 정확하다고 증명할 수는 없음

01. 테스트의 이해

■ 테스트의 필요성과 특징

▪ 소프트웨어 테스트의 목표

- 작은 의미
  - 원시 코드 속에 남아 있는 오류를 발견하는 것
  - 결함이 생기지 않도록 예방하는 것
- 큰 의미
  - 개발된 소프트웨어가 고객의 요구를 만족시키는지 확인시켜주는 것
  - 개발자와 고객에게 사용하기에 충분한 소프트웨어임을 보여주는 것
  - 결과적으로 테스트의 목표는 '개발된 소프트웨어에 신뢰성을 높여주는 것

01. 테스트의 이해

■ 테스트의 필요성과 특징

▪ 소프트웨어 테스트 수행의 어려움

- 테스트 케이스가 적어 효과에 한계가 있음
- 완벽한 테스트 케이스를 도출하기 어려움
- 테스트를 위한 실제 사용 환경을 구축하기 어려움
- 작은 실수를 발견하기 어려움
- 테스트 중요성에 대한 인식이 부족함
- 고객의 요구 사항을 충족시켜야 함
- 테스트 단계에서만 수행되는 단순한 활동이 아니라 개발 단계와 함께함
- 파레토 원리를 적용할 수 있음
- 모듈 단위를 점점 확대해나가며 진행함
- 완벽한 테스트는 불가능
- 개발자와 다른 별도의 팀에서 수행
- 살충제 패러독스(테스트 내성) 문제 해결을 위해 테스트 케이스 업데이트가 필요함

01. 테스트의 이해

■ 테스트 절차

▪ 테스트 절차

- 측정 도구나 장비를 이용해 구현된 소프트웨어가 사용자의 요구를 만족하는지를 테스트하는 절차
- 소프트웨어 개발 단계와 연계해 구성

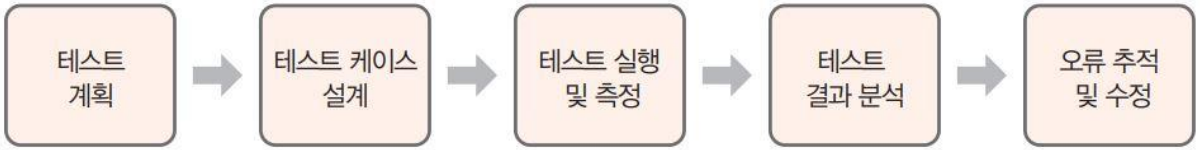


그림 9-2 테스트 절차

01. 테스트의 이해

■ 테스트 절차

▪ 테스트 계획

- 테스트 목표를 정의하고, 테스트 대상 및 범위를 결정하며, 테스트 계획서를 작성하고 검토

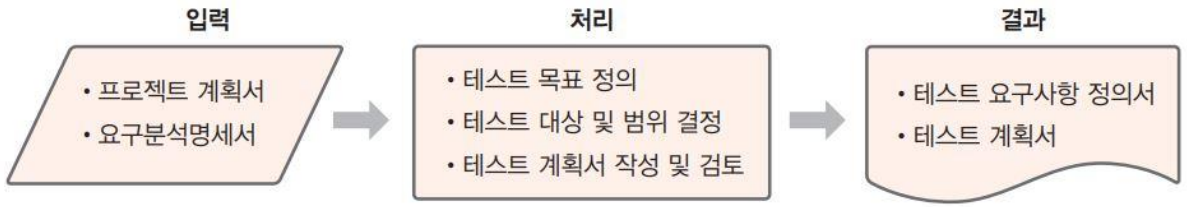


그림 9-3 소프트웨어 테스트 계획 단계

- 테스트 목표 정의
  - 요구분석명세서를 기반으로 테스트할 목표를 정의
  - 테스트 항목 중에서 어떤 항목을 중점적으로 테스트할 것인지 명확히 나타냄 (예: 보안성과 안정성 중 무엇을 중점적으로)
- 테스트 대상 및 범위 결정
  - 테스트할 대상과 범위를 결정하는데, 이때는 업무 시스템별로 구분할 수 있음
- 테스트 계획서 작성 및 검토
  - 테스트의 목적, 담당 인원, 테스트 전략과 접근 방법 수립, 필요한 자원 및 자원 확보 일정, 실시할 테스트의 종류, 적용할 테스트 기법, 일정 등에 관한 정보를 기록
  - 작성된 테스트 계획서를 가다듬고, 이를 확정

01. 테스트의 이해

■ 테스트 절차

▪ 테스트 계획

- 테스트 설계 기법을 정의하고, 도출된 테스트 케이스에 입력 값으로 사용 할 원시 데이터를 작성

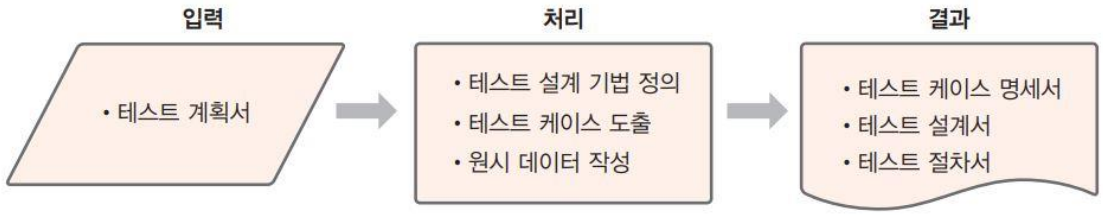


그림 9-4 소프트웨어 테스트 케이스 설계 단계

- 테스트 케이스 설계 기법 정의
  - 테스트할 프로젝트 문제의 성격을 파악하고 이 문제에 적합한 테스트 기법을 선정
- 테스트 케이스 도출
  - 테스트 기법이 결정되면 이 기법을 이용해 테스트 케이스를 도출
- 원시 데이터 작성
  - 결정된 테스트 케이스를 수행하기 위해 필요한 원시 데이터를 작성

01. 테스트의 이해

■ 테스트 절차

▪ 테스트 실행 및 측정

- 테스트 환경을 구축하고 도출된 테스트 케이스를 이용해 테스트를 실시하고 테스트 실행 결과를 문서화

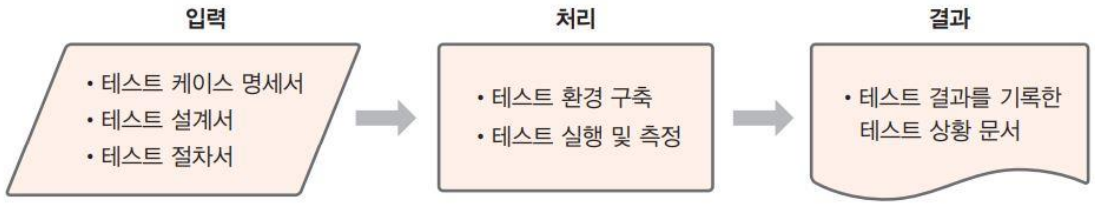


그림 9-5 소프트웨어 테스트 실행 및 측정 단계

- 테스트 환경 구축
  - 테스트 계획서에 정의된 환경 및 자원을 설정해 테스트를 실행할 준비를 함

- 테스트 실행 및 측정
  - 정의된 테스트 케이스를 실행하고 실행 결과를 측정

01. 테스트의 이해

■ 테스트 절차

▪ 테스트 실행 및 측정

- 테스트가 끝나면 계획 대비 결과를 비교·분석하고 테스트 결과에 대한 보고서를 작성

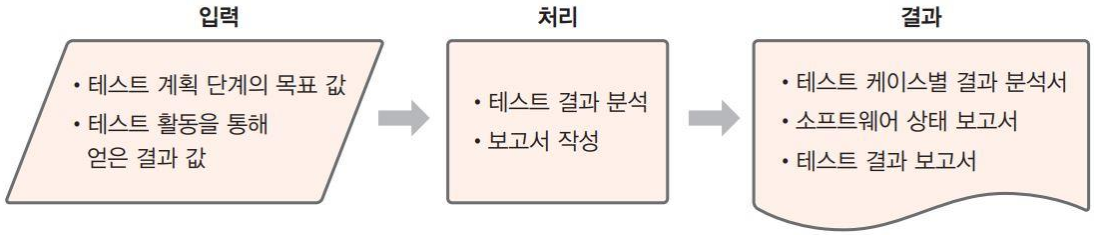


그림 9-6 소프트웨어 테스트 결과 분석 및 보고서 작성 단계

- 테스트 결과 분석
  - 테스트 활동을 통해 얻은 결과 값과 테스트 계획 단계에서 목표한 값을 비교
  - 예정된 테스트 품질 목표가 달성되었는지를 비교·분석

- 보고서 작성 테스트 결과
  - 분석을 기반으로 테스트 결과 보고서를 작성
  - 테스트 보고서에는 테스트를 수행한 결과와 테스트를 수행하는 데 사용된 방법 등을 기술
  - 결과에 따른 평가와 권고 사항도 기술

01. 테스트의 이해

■ 테스트 절차

▪ 오류 추적 및 수정

- 테스트 결과 어디에서, 어떤 종류의 오류가 발생했는지 확인하고 수정

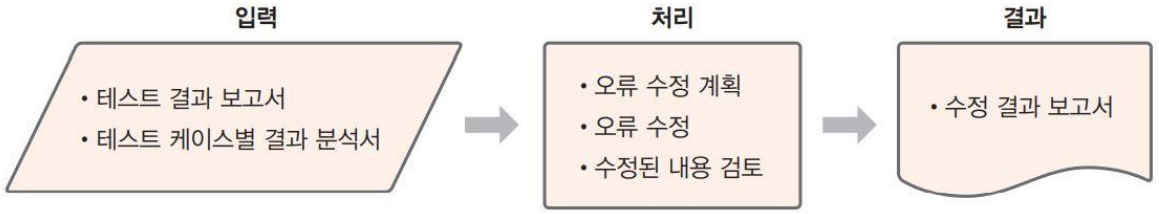


그림 9-7 오류 추적 및 수정 단계

- 오류 수정 계획
  - 테스트 결과 보고서를 기반으로 오류가 발생한 위치를 찾아냄
  - 오류 수정 우선순위를 결 정해 오류 제거 계획을 설립
- 오류 수정
  - 오류 제거 계획을 기초로 디버깅 도구 등을 이용해 오류를 수정
- 수정된 내용 검토
  - 수정된 코드를 검토한 후 오류 수정 결과 보고서를 작성

02. 테스트의 분류

■ 사용 목적에 따른 테스트

▪ 목적에 따른 분류

- 성능 테스트
  - 소프트웨어의 효율성을 진단하는 테스트
  - 사용자의 요구사항 중에서 성능과 관련된 요구사항을 시스템이 얼마나 준수하는지 테스트
  - 시스템의 전 후 관계에서 소프트웨어 실행 시간을 테스트
  - 일반적으로 예상된 부하에 대한 실행 시간 , 응답 시간, 처리 능력 등을 체크하고, 자원 사용량 등을 테스트

- 스트레스 테스트
  - 평소보다 많은 비정상적인 값, 양, 빈도, 부피 등으로 부하를 발생시켜 부하가 최고치인 상황에서 시스템의 반응을 살피고 이때 발생하는 오류를 찾는 것
  - 비정상적이고 과도한 부하 상황에서 시스템이 잘 견디는지, 발생하는 오류에는 어떤 것들이 있는지를 확인

- 보안 테스트
  - 부당하고 불법적인 침입을 시도해, 시스템을 보호하기 위해 구축된 보안 시스템이 불법적인 침투를 잘 막아내는지 테스트
  - 외부 해커의 입장에서 비밀번호를 알려고 시도하거나, 구축해 놓은 방어 체계를 뚫거나 파괴하는 등 여러 가지 방법을 사용

02. 테스트의 분류

■ 사용 목적에 따른 테스트

▪ 프로그램 실행 여부에 따른 테스트

• 정적 테스트

- 자동차 수리시 보닛을 열어 부품 상태는 괜찮은지, 나사가 잘 조여 있는지 등을 면밀히 살펴보고 고장 난 곳을 찾음
- 정적 테스트: [그림 9-8]처럼 프로그램을 실행하지 않고 코드를 검토하며 오류를 찾는 방법

그림 9-8 자동차 내부 상태를 체크해 고장을 찾는 방법

02. 테스트의 분류

■ 사용 목적에 따른 테스트

▪ 프로그램 실행 여부에 따른 테스트

• 동적 테스트

- 직접 주행하면서 엔진 소리가 필요 이상으로 크지는 않은지, RPM 수가 정상 수치보다 높게 올라가는지 확인
- 핸들의 떨림 현상이나 브레이크를 밟았을 때 밀리는 정도를 체크해 고장의 원인을 찾음
- 동적 테스트: [그림 9-9]처럼 프로그램을 실행하면서 찾는 경우

그림 9-9 자동차 주행을 통해 고장을 찾는 방법

03. 정적 테스트

■ 정적 테스트

- 정적 테스트의 개요
  - 프로그램 코드를 실행하지 않고 여러 참여자가 모여 소프트웨어 개발 중에 생성되는 모든 명세나 코드를 검토해서 실패보다는 결함을 찾아내는 방법
  - 정적 테스트를 위해 도구를 사용하면 프로그램 코드 분석뿐만 아니라 HTML, XML과 같은 산출물도 검토할 수 있음

그림 9-10 정적 테스트 방법

04. 동적 테스트

■ 명세 기반 테스트

▪ 명세 기반 테스트(블랙박스 테스트)

- 의사는 몸속을 직접 열어 확인하는 것이 아니라 다양한 의료 장비를 통해 병의 원인을 찾음
- 입력 값에 대한 예상 출력 값을 정해놓고 그대로 결과가 나오는지를 체크
- 프로그램 내부의 구조나 알고리즘을 보지 않고, 요구 분석 명세서나 설계 사양서에서 테스트 케이스를 추출하여 테스트
- 기능을 어떻게 수행하는가 보다는 사용자가 원하는 기능을 수행하는가 테스트

그림 9-14 블랙박스 테스트 비유: 청진기로 환자 진찰하기

04. 동적 테스트

■ 명세 기반 테스트

▪ 신텍스 기법

- 가장 단순한 방법으로, 문법에 기반을 둔 테스트
- 문법을 정해놓고 적합/부적합 입력 값에 따른 예상 결과가 제대로 나오는지 테스트

표 9-3 ID와 비밀번호의 적합 기준

	적합	부적합
ID	알파벳과 숫자(4~8자)	특수 기호 사용, 4자 미만, 9자 이상
비밀번호	알파벳·숫자·특수문자로 구성, 7글자 이상, 특수문자는 반드시 포함	7자 미만, 특수문자 미사용

표 9-4 적합/부적합 입력 값에 대한 예상 처리 결과

번호	적합/부적합	입력 값(ID/비번)	예상 처리 결과
1	적합	sogong/abcd#78	정상 등록
2	부적합	so*gong/abcd#78	에러 메시지: 등록 불가(ID-특수문자 사용)
3	부적합	sogong/abcdefg	에러 메시지: 등록 불가(비밀번호-특수문자 미사용)
4	부적합	gong/abcd	에러 메시지: 등록 불가(ID, 비밀번호-길이 부적합)

04. 동적 테스트

■ 명세 기반 테스트

▪ 동등 분할 기법

- 각 영역에 해당하는 입력 값을 넣고 예상되는 출력 값이 나오는지 실제 값과 비교
- 단순하고 이해하기 쉬우며 사용자가 작성 가능하다는 장점

표 9-5 평가 점수에 따른 상여금

평가 점수	상여금
90~100점	600만 원
80~89점	400만 원
70~79점	200만 원
0~69점	0원

표 9-6 동등 분할 기법의 예

테스트 케이스	1	2	3	4
점수 범위	0~69점	70~79점	80~89점	90~100점
입력 값	60점	73점	86점	94점
예상 결과 값	0원	200만 원	400만 원	600만 원
실제 결과 값	0원	200만 원	400만 원	600만 원

▪ 경계 값 분석 기법

- 경계에 있는 값을 테스트 데이터로 생성하여 테스트하는 방법
- 경계 값과 경계 이전 값, 경계 이후 값을 가지고 테스트

표 9-7 경계 값 분석 기법의 테스트 예

테스트 케이스	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
입력 값	-1	0	1	69	70	71	79	80	81	89	90	91	99	100	101
예상 결과	오류	0	0	0	200	200	200	400	400	400	600	600	600	600	오류
실제 결과	오류	0	0	0	200	200	200	400	400	400	600	600	600	600	오류

04. 동적 테스트

■ 구현 기반 테스트

▪ 구현 기반 테스트의 개요

- 위장 내시경 검사, 조직 배양 검사, 개복해서 어느 부위에 어떤 이상이 생겼는지 확인하며 병을 진단
- 입력 데이터를 가지고 실행 상태를 추적함으로써 오류를 찾아내기 때문에 동적 테스트 부류에 속함
- 프로그램 내부에서 사용되는 변수나 서브루틴 등의 오류를 찾기 위해 프로그램 코드의 내부 구조를 테스트 설계의 기반으로 사용

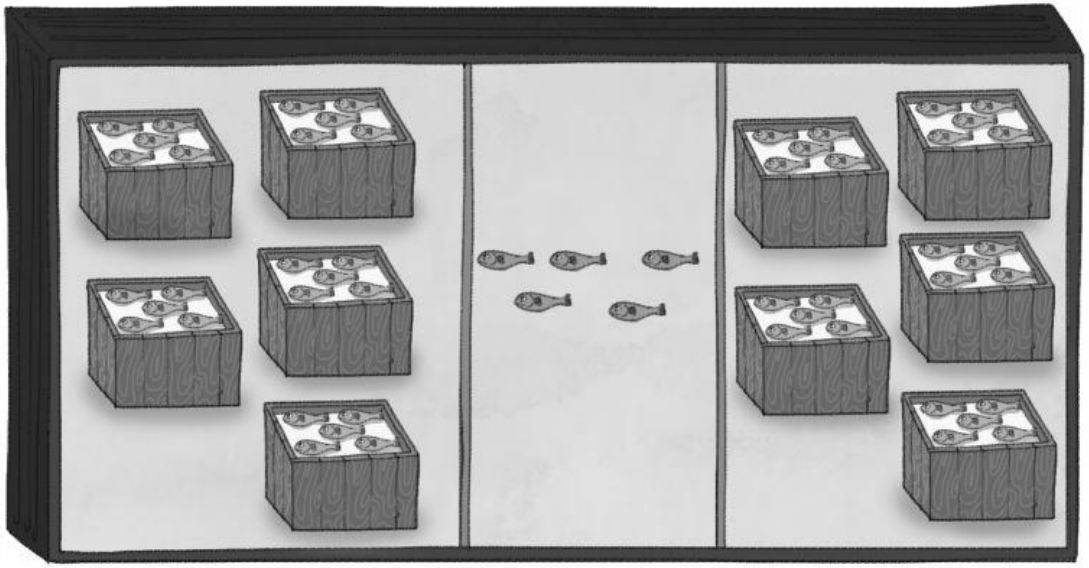
그림 9-15 구현 기반 테스트 비유: 위 내시경 검사

04. 동적 테스트

■ 구현 기반 테스트

- 화이트박스 테스트 절차

① 테스트 데이터 적합성 기준을 선정한다.



- 컨테이너 차량 1대 = 대형 박스 10박스
- 대형 박스 1개 = 중형 박스 5개
- 중형 박스 1개 = 소형 박스 3개
- 소형 박스 1개 = 조기 10마리

그림 9-16 컨테이너 안의 조기 수량

04. 동적 테스트

■ 구현 기반 테스트

▪ 화이트박스 테스트 절차

② 테스트 데이터를 생성한다.

- 테스트 데이터 적합성 기준을 근거로 한 가지 방법이 결정되면 명세나 원시 코드를 분석해 선정된 기준을 만족하는 입력 데이터를 만듬

③ 테스트를 실행한다. (테스트 방법)

- 문장 검증 기준
- 분기 검증 기준
- 조건 검증 기준
- 분기/조건 검증 기준
- 다중 조건 검증 기준
- 기본 경로 테스트

04. 동적 테스트

■ 구현 기반 테스트

- 문장 검증 기준

① 원시 코드를 제어 흐름 그래프 형태로 표현

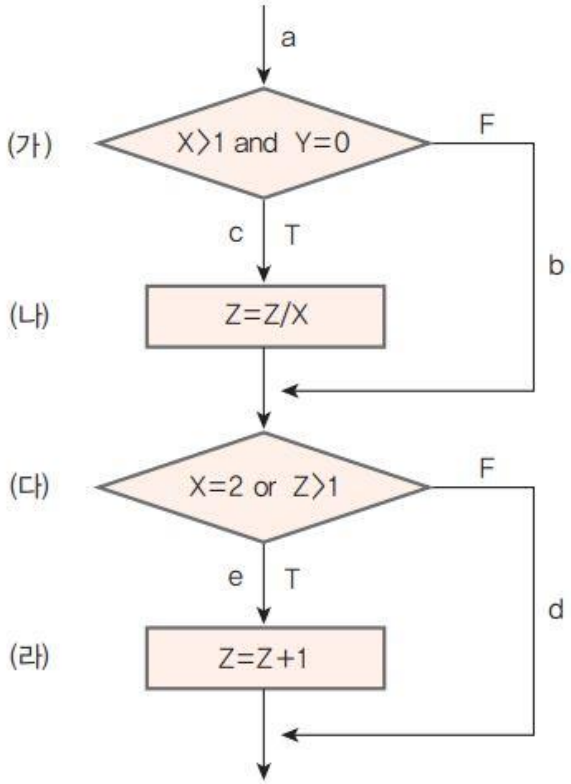


그림 9-17 제어 흐름으로 표현한 그래프

04. 동적 테스트

■ 구현 기반 테스트

▪ 문장 검증 기준

② 원시 코드를 제어 흐름 그래프 형태로 표현

표 9-12 가능한 모든 경로

번호	경로(가, 다)	가능 경로	만족 여부	이유
1	경로 1(T, T)	a-c-e	만족	(가), (나), (다), (라) 문장을 모두 지나감.
2	경로 2(T, F)	a-c-d	불만족	(라) 문장을 안 지나감.
3	경로 3(F, T)	a-b-e	불만족	(나) 문장을 안 지나감.
4	경로 4(F, F)	a-b-d	불만족	(나), (라) 문장을 안 지나감.

③ 모든 경로 중 문장 검증 기준을 만족하는 경로를 선택한다.

표 9-13 문장 검증 기준을 만족하는 경로

번호	경로(가, 다)	가능 경로	만족 여부	이유
1	경로 1(T, T)	a-c-e	만족	(가), (나), (다), (라) 문장을 모두 지나감.

④ 선택한 경로에 해당하는 테스트 데이터를 가지고 실행한다.

표 9-14 테스트 데이터로 실행

번호	경로(가, 다)	테스트 데이터	가능 경로	출력 값	만족 여부
1	경로 1(T, T)	X=2, Y=0, Z=3	a-c-e	2.5	만족

04. 동적 테스트

■ 구현 기반 테스트

▪ 기본 경로 테스트

- 기본 경로 테스트는 매케이브가 만듦
- 원시 코드의 독립적인 경로가 최소한 한 번은 실행되는 테스트 케이스를 찾아 테스트를 수행
- 원시 코드의 독립적인 경로를 모두 수행하는 것을 목적으로 함

① 순서도를 작성

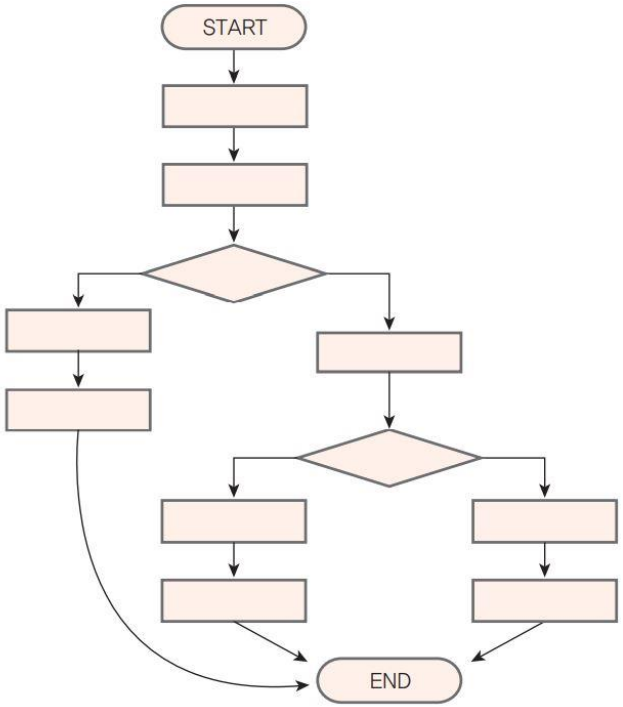


그림 9-19 순서도의 표현

05. 소프트웨어 개발 단계에 따른 테스트

■ V 모델

- V 모델의 개요
  - 요구사항 정의, 분석, 설계, 구현 단계와 테스트 과정의 단위 테스트, 통합 테스트, 시스템 테스트, 인수 테스트가 V자로 연결
- V 모델 과정에 따른 테스트
  - 단위 테스트, 통합 테스트, 시스템 테스트, 인수 테스트, 회귀 테스트

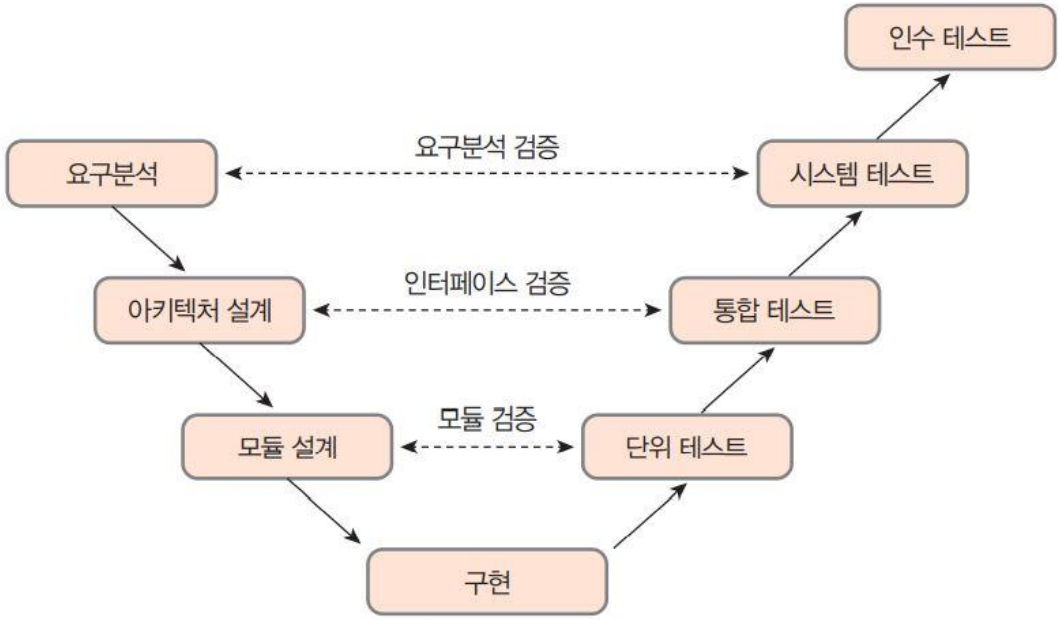


그림 9-22 V 모델

05. 소프트웨어 개발 단계에 따른 테스트

■ 단위 테스트

- 단위 테스트(모듈 테스트)
  - 프로그램의 기본 단위인 모듈을 테스트
  - 구현 단계에서 각 모듈의 개발을 완료한 후 개발자가 요구분석명세서대로 정확히 구현되었는지 테스트
  - 개별 모듈이 제대로 구현되어 정해진 기능을 정확히 수행하는지를 테스트
  - 한 모듈을 테스트하려면 그 모듈과 직접 관련된 상위 모듈과 하위 모듈까지 모두 존재해야 함

- 단위 테스트 수행 후 발견되는 오류
  - 잘못 사용한 자료형
  - 잘못된 논리 연산자
  - 알고리즘 오류에 따른 원치 않는 결과
  - 틀린 계산 수식에 의한 잘못된 결과
  - 탈출구가 없는 반복문의 사용

05. 소프트웨어 개발 단계에 따른 테스트

■ 통합 테스트

- 통합 테스트의 개요
  - 단위 테스트가 끝난 모듈을 통합하는 과정에서 발생할 수 있는 오류를 찾는 테스트
  - '모듈 간의 상호작용이 정상적으로 수행되는가' 테스트
  - 모듈 사이의 인터페이스 오류는 없는지, 모듈이 올바르게 연계되어 동작하고 있는지 체크

05. 소프트웨어 개발 단계에 따른 테스트

■ 통합 테스트

▪ 점진적 모듈 통합 방법: 하향식 기법

- 시스템을 구성하는 모듈의 계층 구조에서 맨 위의 모듈부터 시작해 점차 하위 모듈 방향으로 통합하는 방법
- 최상위 모듈 A를 모듈 B와 통합해 테스트를 실시
- 그 다음으로 모듈 C를 먼저 선택할지(넓이 우선 방식), 아니면 B 아래에 있는 모듈 E를 먼저 선택할지(깊이 우선 방식) 결정
  - 넓이 우선 방식: (A, B)→(A, C)→(A, D)→(A, B, E)→(A, B, F)→(A, C, G)와 같이 같은 행에서는 옆으로 가며 통합 테스트
  - 깊이 우선 방식: (A, B)→(A, B, E)→(A, B, F)→(A, C)→(A, C, G)→(A, D) 순으로 통합하며 테스트

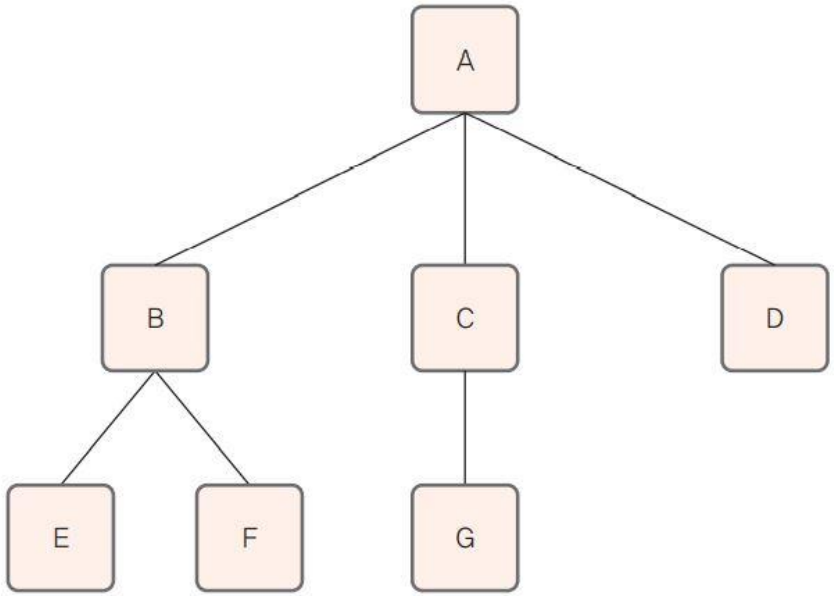


그림 9-24 점진적 모듈 통합 방법을 설명하기 위한 모듈 구성도

05. 소프트웨어 개발 단계에 따른 테스트

■ 통합 테스트

▪ 점진적 모듈 통합 방법: 상향식 기법

- 하향식 기법과는 반대로 가장 아래에 있는 모듈부터 테스트를 시작
- 하향식에서 스텁이 필요했다면, 상향식에서는 상위 모듈의 역할을 하는 테스트 드라이버가 필요
  - 드라이버는 하위 모듈을 순서에 맞게 호출하고, 호출할 때 필요한 매개변수를 제공하며, 반환 값을 전달하는 역할

• 상향식 기법의 순서

① 모듈 E와 F를 모듈 B에 통합해 테스트

② 모듈 G를 모듈 C에 통합해 테스트

③ 모듈 B, C, D를 모듈 A 에 통합해 테스트

• 상향식 기법의 장점과 단점

- 최하위 모듈들을 개별적으로 병행해 테스트할 수 있기 때문에 하위에 있는 모듈들을 충분히 테스트할 수 있다는 점
- 정밀한 계산이나 데이터 처리가 요구되는 시스템 같은 경우에 사용하면 좋음
- 상위 모듈에 오류가 발견되면 그 모듈과 관련된 하위의 모듈을 다시 테스트해야 하는 번거로움이 생김

05. 소프트웨어 개발 단계에 따른 테스트

■ 통합 테스트

▪ 시스템 테스트

• 시스템 테스트의 개요

- 시스템 전체가 정상적으로사용자의 요구 사항들을 만족하는지 테스트
- 사용자에게 개발된 시스템을 작동하는지를 체크
- 모듈이 모두 통합된 후 전달하기 전에 개발자가 진행하는 마지막 테스트
- V & V에서 확인에 해당
- 실제 사용 환경과 유사하게 테스트 환경을 만들고 요구 분석 명세서에 명시한 기능적, 비기능적 요구 사항을 충족하는지 테스트
- 주로 부하를 주는 상황에서 수행하고, 비기능적 테스트를 중심으로 수행

▪ 인수 테스트

• 인수 테스트의 개요

- 시스템이 예상대로 동작하는지 확인하고, 요구사항에 맞는지 확신하기 위해 하는 테스트
- 시스템을 인수하기 전 요구분석명세서에 명시된 대로 모두 충족시키는지 사용자가 테스트
- 인수 테스트가 끝나면 사용자는 인수를 승낙한 것
- 시스템이 사용자에게 정상적으로 인수되면 프로젝트는 종료
- 인수 테스트 시 인수 테스트 계획서는 사용자가 직접 작성하거나, 개발자가 작성한 것을 사용자가 승인

05. 소프트웨어 개발 단계에 따른 테스트

■ 통합 테스트

▪ 인수 테스트

- 알파 테스트(내부 필드 테스트)
  - 개발 완료된 소프트웨어를 베타 테스트 전에 회사 내의 다른 직원에게 개발자 환경에서 테스트
  - 개발자가 그들이 사용하는 것을 보면서 오류와 사용상의 문제점을 파악

- 베타 테스트
  - 개발 완료되어 알파 테스트를 거친 소프트웨어를 시장에 상품으로 내놓기 전에 시장의 피드백을 얻기 위한 목적으로 수행
  - 특정 사용자에게 미리 사용해 보도록 배포
  - 베타 버전을 받은 사용자들이 사용자 입장에서 사용해보고 문제점이나 오류를 개발자에게 알려주는 방식으로 테스트
  - 그런 다음 개발자가 베타 테스트를 통해 보고된 문제점을 수정해 제품을 출시