2과목-소프트웨어 개발

(Part 4. 애플리케이션 테스트 관리)

소프트웨어 개발 총 파트

소프트웨어 개발 2과목은 총 5Part로 이루어져있다.

1장 데이터 입, 출력 구현(29.49%)

2장 통합 구현(1.92%)

3장 제품 소프트웨어 패키징(19.87%)

4장 애플리케이션 테스트 관리(35.26%)

5장 인터페이스 구현(13.46%)

애플리케이션 테스트 관리

애플리케이션 테스트 관리 Part는 12개의 섹션으로 구성되어 있다.

01 애플리케이션 테스트

02 애플리케이션 테스트의 분류

03 테스트 기법에 따른 애플리케이션 테스트

04 개발 단계에 따른 애플리케이션 테스트

05 통합 테스트

06 애플리케이션 테스트 프로세스

07 테스트 케이스 / 테스트 시나리오/ 테스트 오라클

08 테스트 자동화 도구

09 결함 관리

10 애플리케이션 성능 분석

11 복잡도

12 애플리케이션 성능 개선

4. 애플리케이션 테스트 관리-SEC_01(애플리케이션 테스트)

- 이 장을 공부하면서 반드시 알아두어야 할 키워드

; 화이트박스 테스트, 블랙박스테스트, 단위 테스트, 통합 테스트, 하향식 통합 테스트, 상향식 통합 테스트, 테스트 케이스, 테스트 오라클, 빅오 표기법, 순한 복잡도

1) 애플리케이션 테스트의 개념

; 애플리케이션 테스트는 애플리케이션에 잠재되어 있는 결함을 찾아내는 일련의 행위 또는 절차이다.

● 애플리케이션 테스트는 개발된 소프트웨어가 고객의 요구사항을 만족시키는지 확인(Validation)하고 소프트웨어가 기능을 정확히 수행하는지 검증(Verification)한다.

● 애플리케이션 테스트를 실행하기 전에 개발한 소프트웨어의 유형을 분류하고 특성을 정리해서 중점적으로 테스트할 사항을 정리해야 한다.

예) 소프트웨어 유형별 특성

소프트웨어명	제공 유형	기능 유형	사용 환경	개발 유형	중점 사항
A. xx오픈 DB 구축	서비스 제공 소프트웨어	산업 특화	Web	신규 개발	기능 구현 시 사용자 요구사항의 누락 여부
B. Xx 통합 서비스 구현	서비스 제공 소프트웨어	산업 특화	Web	시스템 통합	기존 시스템과 신규 시스템의 데이터 손실 및 정합성 여부
C. xx오피스	상용 소프트웨어	산업 범용	C/S	신규 개발	다양한 OS환경 지원 여부

확인(Validation)은 사용자의 입장에서 개발한 소프트웨어가 고객의 요구 사항에 맞게 구현되었는지를 확인하는 것이다.
검증(Verification)은 개발자의 입장에서 개발한 소프트웨어가 명세서에 맞게 만들어졌는지를 점검하는 것이다.

4. 애플리케이션 테스트 관리-SEC_01(애플리케이션 테스트)

2) 소프트웨어의 분류

; 소프트웨어(Software)는 하드웨어를 동작시켜 사용자가 작업을 편리하게 수행하도록 하는 프로그램과 자료 구조 등을 총칭하는 것으로 상용 소프트웨어와 서비스 제공 소프트웨어로 구분된다.

● 상용 소프트웨어 : 보통의 사용자들이 공통적으로 필요로 하는 기능을 제공하는 소프트웨어로 산업의 특성에 따라 산업 범용 소프트웨어와 산업 특화 소프트웨어로 구분된다.

산업 범용 소프트웨어	•시스템 소프트웨어 : 하드웨어 전체를 제어하고 운영하는 소프트웨어로 운영체제 데이터 관리, 스토리지 소프트웨어, 소프트웨어 공학 도구, 가상화 소프트웨어, 시스템 보안 소프트웨어로 구분된다. •미들웨어 : 운영체제와 해당 운영체제에 의해 실행되는 응용 프로그램 사이에서 운영체제가 제공하는 서비스 이외에 추가적인 서비스를 제공하는 소프트웨어로 분산 시스템 소프트웨어, IT 자원 관리, 서비스 플랫폼, 네트워크 보안 소프트웨어로 구분된다. •응용 소프트웨어 : 특정 업무를 처리하기 위한 소프트웨어로 영상 처리, CG/NR, 콘텐츠 배포, 자연어 처리, 음성 처리, 기업용 소프트웨어로 구분된다.
산업 특화 소프트웨어	특정 분야에서 요구하는 기능만을 구현한 소프트웨어로, 자동차, 항공, 조선, 건설, 패션 의류, 농업, 의료, 국방, 공공 분야 등을 지원하는 소프트웨어가 있다.

4. 애플리케이션 테스트 관리-SEC_01(애플리케이션 테스트)

2) 소프트웨어의 분류

● 서비스 제공 소프트웨어 : 소프트웨어를 개발하여 판매하려는 것이 아니라 특정 사용자가 필요로 하는 기능만을 구현해서 제공하는 소프트웨어이다.

신규 개발 소프트웨어	새로운 서비스를 제공하기 위해 개발된 소프트웨어
기능 개선 소프트웨어	사용자 편의성, 응답 속도, 화면 UI(User Interface), 업무 프로세스 등 기존 서비스 기능을 개선하기 위해 개발된 소프트웨어
추가 개발 소프트웨어	업무나 산업 환경의 변화, 법이나 제도의 개정 등으로 인해 기존 시스템에 새로운 기능을 추가하기 위해 개발된 소프트웨어
시스템 통합 소프트웨어	시스템 별로 서비스되던 것을 원스톱(One-Stop) 서비스로 제공하기 위해 업무 기능이나 데이터 등을 통합하여 개발한 소프트웨어

3) 애플리케이션 테스트의 필요성

● 애플리케이션 테스트를 통해 프로그램 실행 전에 오류를 발견하여 예방할 수 있다.

● 애플리케이션 테스트는 프로그램이 사용자의 요구사항이나 기대 수준 등을 만족시키는지 반복적으로 테스트하므로 제품의 신뢰도를 향상시킨다.

● 애플리케이션의 개발 초기부터 애플리케이션 테스트를 계획하고 시작하면 단순한 오류 발견뿐만 아니라 새로운 오류의 유입도 예방할 수 있다.

● 애플리케이션 테스트를 효과적으로 수행하면 최소한의 시간과 노력으로 많은 결함을 찾을 수 있다.

4. 애플리케이션 테스트 관리-SEC_01(애플리케이션 테스트)

4) 애플리케이션 테스트의 기본 원리

- 애플리케이션 테스트는 소프트웨어의 잠재적인 결함을 줄일 수 있지만 소프트웨어에 결함이 없다고 증명할 수는 없다. 즉 완벽한 소프트웨어 테스팅은 불가능하다.
- 애플리케이션의 결함은 대부분 개발자의 특성이나 애플리케이션의 기능적 특징 때문에 특정 모듈에 집중되어 있다. 애플리케이션의 20%에 해당하는 코드에서 전체 80%의 결함이 발견된다고 하여 파레토 법칙을 적용하기도 한다.
- 애플리케이션 테스트에서는 동일한 테스트 케이스로 동일한 테스트를 반복하면 더 이상 결함이 발견되지 않는 '살충제 패러독스(Pesticide Paradox)' 현상이 발생한다. 살충제 패러독스를 방지하기 위해서 테스트 케이스를 지속적으로 보완 및 개선해야 한다.
- 애플리케이션 테스트는 소프트웨어 특징, 테스트 환경, 테스터 역량 등 정황(Context)에 따라 테스트 결과가 달라질 수 있으므로, 정황에 따라 테스트를 다르게 수행해야 한다.
- 소프트웨어의 결함을 모두 제거해도 사용자의 요구사항을 만족시키지 못하면 해당 소프트웨어는 품질이 높다고 말할 수 없다. 이것을 오류-부재의 궤변(Absence of Errors Fallacy)이라고 한다.

특정 모듈 집중 : 대부분의 결함이 소수의 특정 모듈에 집중해서 발생하는 것을 결함 집중(Defect Clustering)이라고 한다.
파레토 법칙(Pareto Principle) : 파레토의 법칙은 상위 20% 사람들이 전체 부의 80%를 가지고 있다거나, 상위 20% 고객이 매출의 80%를 창출한다는 의미로 이 법칙이 애플리케이션 테스트에도 적용된다는 것이다. 즉 테스트로 발견된 80%의 오류는 20%의 모듈에서 발견되므로 20%의 모듈을 집중적으로 테스트하여 효율적으로 오류를 찾자는 것이다.
살충제 패러독스 : 살충제 패러독스는 살충제를 지속적으로 뿌리면 벌레가 내성이 생겨서 죽지 않는 현상을 의미한다.

4. 애플리케이션 테스트 관리-SEC_01(애플리케이션 테스트)

4) 애플리케이션 테스트의 기본 원리

- 테스트와 위험은 반비례한다. 테스트를 많이 하면 할수록 미래에 발생할 위험을 줄일 수 있다.
- 테스트는 작은 부분에서 시작하여 점점 확대하며 진행해야 한다.
- 테스트는 개발자와 관계없는 별도의 팀에서 수행해야 한다.

4. 애플리케이션 테스트 관리-SEC_01(애플리케이션 테스트) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(애플리케이션 테스트)

1. 소프트웨어 테스트에서 검증(Verification)과 확인(Validation) 에 대한 설명으로 틀린 것은?

① 소프트웨어 테스트에서 검증과 확인을 구별하면 찾고자 하는 결함 유형을 명확하게 하는 데 도움이 된다.

② 검증은 소프트웨어 개발 과정을 테스트하는 것이고, 확인은 소프트웨어 결과를 테스트 것이다.

③ 검증은 작업 제품이 요구 명세의 기능, 비기능 요구사항을 얼마나 잘 준수하는지 측정하는 작업이다.

④ 검증은 작업 제품이 사용자의 요구에 적합한지 측정하며, 확인 은 작업 제품이 개발자의 기대를 충족시키는지를 측정한다.

애플리케이션 테스트는 개발된 소프트웨어가 고객의 요구사항을 만족시키는지 확인(Validation)하고 소프트웨어가 기능을 정확히 수행하는지 검증(Verification)한다.

확인(Validation) : 사용자의 입장에서 개발한 소프트웨어가 고객의 요구사항에 맞게 구현되었는지를 확인하는 것이다.

검증(Verification) : 개발자의 입장에서 개발한 소프트웨어가 명세서에 맞게 만들어졌는지를 점검하는 것이다.

3. 소프트웨어 테스트에서 오류의 80%는 전체 모듈의 20% 내에서 발견된다는 법칙은?

① Brooks의 법칙 ② Boehm의 법칙

③ Pareto의 법칙 ④ Jackson의 법칙

Brooks의 법칙 : "지체되는 소프트웨어 개발 프로젝트에 인력을 더하는 것은 오히려 개발을 늦출 뿐이다"라고 주장한 법칙이다.

Boehm의 법칙 : Boehm은 나선형 모형을 제안한 사람

Jackson의 법칙 : IT용어에는 없고, 의학적인 용어이다.

4. 다음 중 애플리케이션 테스트에 대한 설명으로 틀린 것은?

① 애플리케이션 테스트는 프로그램 실행 전에 코드 리뷰, 인스펙션 등을 통해 사전에 오류를 발견하여 예방할 수 있다.

② 애플리케이션 테스트를 반복적으로 실행하여 제품의 신뢰도를 향상시킬 수 있다.

③ 테스팅은 프로그램 개발이 완료된 후 체계적으로 계획하여 실행해야 한다.

④ 성공적인 테스트는 아직 발견되지 않은 오류를 찾아내는 것이다.

애플리케이션 테스트를 통해 프로그램 실행 전에 오류를 발견하여 예방할 수 있다.

4. 애플리케이션 테스트 관리-SEC_01(애플리케이션 테스트) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(애플리케이션 테스트)

5. 다음 중 애플리케이션 테스트에 대한 설명으로 틀린 것은?

① 애플리케이션 테스트는 소프트웨어에 잠재되어 있는 결함을 찾아내는 일련의 행위이다.

② 테스트는 고객의 요구사항을 만족했는지 Verification 해야 한다.

③ 테스트는 오류 검출뿐만 아니라 새로운 오류의 유입도 방지 할 수 있다.

④ 테스트를 효과적으로 실행하면 최소한의 시간과 노력으로 많은 결함을 찾을 수 있다.

6. 개발된 소프트웨어가 사용자의 요구사항 및 기대 수준 등을 만족하는지 점검하기 위해 애플리케이션 테스트를 진행하려고 할 때, 이에 대한 설명으로 틀린 것은?

① 완벽한 테스팅은 불가능하다.

② 테스팅은 정황(Context)에 의존한다.

③ 파레토 법칙을 적용할 수 있다.

④ 테스트와 위험은 정비례한다.

테스트와 위험은 반비례한다. 테스트를 많이 하면 할수록 미래에 발생할 위험을 줄일 수 있다.

4. 애플리케이션 테스트 관리-SEC_02(애플리케이션 테스트의 분류)

1) 프로그램 실행 여부에 따른 테스트

; 애플리케이션을 테스트 할 때 프로그램의 실행 여부에 따라 정적 테스트와 동적 테스트로 나뉜다.

정적 테스트	•프로그램을 실행하지 않고 명세서나 소스 코드를 대상으로 분석하는 테스트이다. •소프트웨어 개발 초기에 결함을 발견할 수 있어 소프트웨어의 개발 비용을 낮추는데 도움이 된다. •종류: 워크스루, 인스펙션 코드 검사 등
동적 테스트	•프로그램을 실행하여 오류를 찾는 테스트로, 소프트웨어 개발의 모든 단계에서 테스트를 수행할 수 있다. •종류 : 블랙박스 테스트, 화이트박스 테스트

인스펙션(Inspection) : 인스펙션은 워크스루를 발전시킨 형태로 소프트웨어 개발 단계에서 산출된 결과물의 품질을 평가하며 이를 개선 하기 위한 방법 등을 제시한다.

워크스루(Walkthrough, 검토 회의)

•워크스루는 소프트웨어 개발자의 작업 내역을 개발자가 모집한 전문가들이 검토하는 것을 말한다.

•소프트웨어 검토를 위해 미리 준비된 자료를 바탕으로 정해진 절차에 따라 평가한다.

•오류의 조기 검출을 목적으로 하며 발견된 오류는 문서화 한다.

블랙박스 테스트 : 소프트웨어의 내부 구조나 작동 원리를 모르는 상태에서 소프트웨어의 동작을 검사하는 방법

화이트 박스 테스트 : 소프트웨어 혹은 제품의 내부 구조, 동작을 세밀하게 검사하는 테스트 방식으로, 외부에서 요구사항에 따른 예상 결과값을 테스트 하는 것과는 다르게 내부 소스 코드를 테스트하는 기법으로 사용자가 들여다 볼 수 없는 구간의 코드 단위를 테스트 한다. 개발자가 소프트웨어 또는 컴포넌트 등의 로직에 대한 테스트를 수행하기 위해 설계 단계에서 요구된 사항을 확인하는 개발자 관점의 단위테스팅 기법이다.

4. 애플리케이션 테스트 관리-SEC_02(애플리케이션 테스트의 분류)

2) 테스트 기반(Test Bases)에 따른 테스트

; 애플리케이션을 테스트 할 때 무엇을 기반으로 수행하느냐에 따라 명세 기반, 구조 기반, 경험 기반 테스트로 나뉜다.

명세 기반 테스트	•사용자의 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 만들어 구현하고 있는지 확인하는 테스트이다. •종류 : 동등 분할, 경계 값 분석 등
구조 기반 테스트	•소프트웨어 내부의 논리 흐름에 따라 테스트 케이스를 작성하고 확인하는 테스트이다. •종류 : 구문 기반, 결정 기반, 조건 기반 등
경험 기반 테스트	•유사 소프트웨어나 기술 등에 대한 테스터의 경험을 기반으로 수행하는 테스트이다. •경험 기반 테스트는 사용자의 요구사항에 대한 명세가 불충분하거나 테스트 시간에 제약이 있는 경우 수행하면 효과적이다. •종류 : 에러 추정, 체크 리스트, 탐색적 테스팅

동등 분할 테스팅은 적절한 커버리지를 유지하면서 테스트케이스의 수를 관리 가능한 크기로 줄이는 데 사용되는 기법이다. 이 간단한 기술은 우리가 공식적인 테스트 설계 방법으로 인식하지 못할지라도 거의 모든 테스터가 직관적으로 사용하는 방법이다. 동등 분할 테스팅이 그 자체로 유용한 기법이지만 이것의 가장 큰 기여는 우리를 경계 값 테스팅으로 이끄는 데에 있다. 일반적으로 경계에 많은 결함이 숨어 있기 때문에 경계 값 테스팅은 경계 분석에 집중하는 기술이다. 숙련된 테스터라면 이러한 상황을 여러 번 경험했을 것이며, 경험이 없는 테스터라도 경계에서 실수가 가장 자주 발생할 것이라는 직감을 가진다.

4. 애플리케이션 테스트 관리-SEC_02(애플리케이션 테스트의 분류)

3) 시각에 따른 테스트

; 애플리케이션을 테스트 할 때 누구를 기준으로 하느냐에 따라 검증(Verification) 테스트와 확인(Validation) 테스트로 나뉜다.

검증(Verification) 테스트	개발자의 시각에서 제품의 생산 과정을 테스트하는 것으로, 제품이 명세서대로 완성됐는지를 테스트 한다.
확인(Validation) 테스트	사용자의 시각에서 생산된 제품의 결과를 테스트하는 것으로 사용자가 요구한대로 제품이 완성됐는지, 제품이 정상적으로 동작하는지를 테스트한다.

4. 애플리케이션 테스트 관리-SEC_02(애플리케이션 테스트의 분류)

4) 목적에 따른 테스트

; 애플리케이션을 테스트 할 때 무엇을 목적으로 테스트를 진행하느냐에 따라 회복(Recovery), 안전(Security), 강도(Stress), 성능(Performance), 구조(Structure), 회귀(Regression), 병행(Parallel) 테스트로 나뉜다.

회복(Recovery) 테스트	시스템에 여러 가지 결함을 주어 실패하도록 한 후 올바르게 복구되는지를 확인하는 테스트이다.
안전(Security) 테스트	시스템에 설치된 시스템 보호 도구가 불법적인 침입으로부터 시스템을 보호할 수 있는지를 확인하는 테스트이다.
강도(Stress)테스트	시스템에 과도한 정보량이나 빈도 등을 부과하여 과부하 시에도 소프트웨어가 정상적으로 실행되는지를 확인하는 테스트이다.
성능(Performance) 테스트	소프트웨어의 실시간 성능이나 전체적인 효율성을 진단하는 테스트로, 소프트웨어의 응답 시간, 처리량 등을 테스트한다.
구조(Structure) 테스트	소프트웨어 내부의 논리적인 경로, 소스 코드의 복잡도 등을 평가하는 테스트이다.
회귀(Regression) 테스트	소프트웨어의 변경 또는 수정된 코드에 새로운 결함이 없음을 확인하는 테스트이다.
병행(Parallel) 테스트	변경된 소프트웨어와 기존 소프트웨어에 동일한 데이터를 입력하여 결과를 비교하는 테스트이다.

4. 애플리케이션 테스트 관리-SEC_02(애플리케이션 테스트의 분류) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(애플리케이션 테스트의 분류)

1. 테스트를 목적에 따라 분류했을 때, 강도(Stress) 테스트에 대한 설명으로 옳은 것은?

① 시스템에 고의로 실패를 유도하고 시스템이 정상적으로 복귀하는지 테스트한다.

② 시스템에 과다 정보량을 부과하여 과부하 시에도 시스템이 정상적으로 작동되는지를 테스트한다.

③ 사용자의 이벤트에 시스템이 응답하는 시간, 특정 시간 내에 처리하는 업무량, 사용자 요구에 시스템이 반응하는 속도 등을 테스트한다.

④ 부당하고 불법적인 침입을 시도하여 보안 시스템이 불법적인 침투를 잘 막아내는지 테스트한다.

회복(Recovery)테스트 : 시스템에 여러 가지 결함을 주어 실패하도록 한 후 올바르게 복구가 되는지를 확인하는 테스트이다.

안전(Security)테스트 : 시스템에 설치된 시스템 보호 도구가 불법적인 침입으로부터 시스템을 보호할 수 있는지 확인하는 테스트이다.

강도(Stress) 테스트 : 시스템에 과도한 정보량이나 빈도 등을 부과하여 과부하 시에도 시스템이 정상적으로 작동되는지를

3. 다음 중 확인(Validation) 테스트에 대한 설명으로 옳은 것은?

① 개발자의 시각에서 테스트를 진행한다.

② 제품이 올바르게 생산되고 있는가를 확인한다.

③ 소프트웨어가 명세서대로 만들어졌는지를 중점을 두고 테스트 한다.

④ 소프트웨어가 사용자의 요구사항을 충족시키는가에 중점을 두고 테스트한다.

검증(Verification) 테스트

개발자의 시각에서 제품의 생산 과정을 테스트하는 것으로, 제품이 명세서대로 완성됐는지를 테스트 한다.

확인(Validation) 테스트

사용자 시각에서 생산된 제품의 결과를 테스트 하는 것으로 사용자가 요구한대로 제품이 완성됐는지, 제품이 정상적으로 동작하는지를 테스트를 한다.

4. 다음 중 애플리케이션 테스트에 대한 설명으로 틀린 것은?

① 사용자의 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 만들어 구현하고 있는지 확인하는 테스트는 명세 기반 테스트이다.

② 테스터의 이전 경험과 기술을 기반으로 수행하는 테스트는 경험 기반 테스트이다.

4. 애플리케이션 테스트 관리-SEC_02(애플리케이션 테스트의 분류) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(애플리케이션 테스트의 분류)

5. 다음 중 워크스루(Walkthrough)와 인스펙션(Inspection)에 대한 설명으로 가장 옳지 않은 것은?

① 워크스루는 전문가들에 의해 개발자의 작업 내역이 검토 된다.

② 워크스루는 제품 개발자가 주체가 된다.

③ 워크스루는 오류 발견과 발견된 오류의 문제 해결에 중점을 둔다.

④ 인스펙션은 워크스루를 발전시킨 형태이다.

인스펙션(Inspection) : 인스펙션은 워크스루를 발전시킨 형태로 소프트웨어 개발 단계에서 산출된 결과물의 품질을 평가하며 이를 개선하기 위한 방법 등을 제시한다.

워크스루(Walkthrough, 검토 회의)

워크스루는 소프트웨어 개발자의 작업 내역을 개발자가 모집한 전문가들이 검토하는 것을 말한다.

소프트웨어 검토를 위해 미리 준비된 자료를 바탕으로 정해진 절차에 따라 평가한다.

오류의 조기 검출을 목적으로 하며, 발견된 오류는 문서화한다.

4. 애플리케이션 테스트 관리-SEC_03(테스트 기법에 따른 애플리케이션 테스트)

1) 화이트박스 테스트(White Box Test)

; 화이트박스 테스트는 모듈의 원시 코드를 오픈 시킨 상태에서 원시 코드의 논리적인 모든 경로를 테스트 하여 테스트 케이스를 설계하는 방법이다.

- 화이트박스 테스트는 설계된 절차에 초점을 둔 구조적 테스트로 프로시저 설계의 제어 구조를 사용하여 테스트 케이스를 설계하며, 테스트 과정의 초기에 적용된다.
- 모듈 안의 작동을 직접 관찰한다.
- 원시 코드(모듈)의 모든 문장을 한 번 이상 실행함으로써 수행된다.
- 프로그램의 제어 구조에 따라 선택, 반복 등의 분기점 부분들을 수행함으로써 논리적 경로를 제어한다.

4. 애플리케이션 테스트 관리-SEC_03(테스트 기법에 따른 애플리케이션 테스트)

2) 화이트박스 테스트의 종류

; 화이트 박스 테스트의 종류에는 기초 경로 검사, 제어 구조 검사 등이 있다.

기초 경로 검사 (Base Path Testing)	•대표적인 화이트박스 테스트 기법이다. •테스트 케이스 설계자가 절차적 설계의 논리적 복잡성을 측정할 수 있게 해주는 테스트 기법으로 테스트 측정 결과는 실행 경로의 기초를 정의하는 데 지침으로 사용된다.
제어 구조 검사 (Control Structure Testing)	•조건 검사(Condition Testing) : 프로그램 모듈 내에 있는 논리적 조건을 테스트하는 테스트 케이스 설계 기법 •루프 검사(Loop Testing) : 프로그램의 반복(Loop) 구조에 초점을 맞춰 실시하는 테스트 케이스 설계 기법 •데이터 흐름 검사(Data Flow Testing) : 프로그램에서 변수의 정의와 변수 사용의 위치에 초점을 맞춰 실시하는 테스트 케이스 설계 기법

기초경로(Base Path) : 수행 가능한 모든 경로를 의미한다.

4. 애플리케이션 테스트 관리-SEC_03(테스트 기법에 따른 애플리케이션 테스트)

3) 화이트박스 테스트의 검증 기준

; 화이트박스 테스트의 검증 기준은 테스트 케이스들이 테스트에 얼마나 적정한지를 판단하는 기준으로, 문장 검증 기준, 분기 검증 기준, 조건 검증 기준, 분기/조건 기준이 있다.

문장 검증 기준 (Statement Coverage)	소스 코드의 모든 구문이 한 번 이상 수행되도록 테스트 케이스 설계
분기 검증 기준 (Branch Coverage)	결정 검증 기준(Decision Coverage)이라고도 불리며, 소스 코드의 모든 조건문에 대해 조건이 True인 경우와 False인 경우가 한 번 이상 수행되도록 테스트 케이스 설계
조건 검증 기준 (Condition Coverage)	소스 코드의 조건문에 포함된 개별 조건식의 결과가True인 경우와False인 경우가 한 번 이상 수행되도록 테스트 케이스 설계
분기/조건 기준 (Branch/Condition Coverage)	분기 검증 기준과 조건 검증 기준을 모두 만족하는 설계로, 조건문이 True인 경우와 False인 경우에 따라 조건 검증 기준의 입력 데이터를 구분하는 테스트 케이스 설계

; 검증 기준의 종류에는 크게 기능 기반 커버리지, 라인 커버리지, 코드 커버리지가 있으며, 화이트박스 테스트에서 사용되는 문장 검증 기준, 분기 검증 기준 등은 모두 코드 커버리지에 해당한다.

- 기능 기반 커버리지 : 실제 테스트가 수행된 기능의 수 / 전체 기능의 수
- 라인 커버리지(Line Coverage) : 테스트 시나리오가 수행한 소스 코드의 라인 수/ 전체 소스 코드의 라인 수
- 코드 커버리지(Code Coverage) : 소스 코드의 구문, 분기, 조건 등의 구조 코드 자체가 얼마나 되었는지를 측정하는 방법

4. 애플리케이션 테스트 관리-SEC_03(테스트 기법에 따른 애플리케이션 테스트)

4) 블랙박스 테스트(Black Box Test)

; 블랙박스 테스트는 소프트웨어가 수행할 특정 기능을 알기 위해서 각 기능이 완전히 작동되는 것을 입증하는 테스트로, 기능 테스트라고도 한다.

- 프로그램의 구조를 고려하지 않기 때문에 테스트 케이스는 프로그램 또는 모듈의 요구나 명세를 기초로 결정한다.
- 소프트웨어 인터페이스에서 실시되는 테스트이다.
- 부정확하거나 누락된 기능, 인터페이스 오류, 자료 구조나 외부 데이터베이스 접근에 따른 오류, 행위나 성능 오류, 초기화와 종료 오류 등을 발견하기 위해 사용되며, 테스트 과정의 후반부에 적용된다.
- 블랙박스 테스트의 종류에는 동치 분할 검사, 경계 값 분석, 원인-효과 그래프 검사, 오류 예측 검사, 비교 검사 등이 있다.

4. 애플리케이션 테스트 관리-SEC_03(테스트 기법에 따른 애플리케이션 테스트)

5) 블랙박스 테스트의 종류

동치 분할 검사 (Equivalence Partitioning Testing, 동치 클래스 분해)	•입력 자료에 초점을 맞춰 테스트 케이스(동치 클래스)를 만들고 검사하는 방법으로 동등 분할 기법이라고도 한다. •프로그램의 입력 조건에 타당한 입력 자료와 타당하지 않은 입력 자료의 개수를 균등하게 하여 테스트 케이스를 정하고, 해당 입력 자료에 맞는 결과가 출력되는지 확인하는 기법이다.
경계 값 분석 (Boundary Value Analysis)	•입력 자료에만 치중한 동치 분할 기법을 보완하기 위한 기법이다. •입력 조건의 중간 값보다 경계 값에서 오류가 발생될 확률이 높다는 점을 이용하여 입력 조건의 경계 값을 테스트 케이스로 선정하여 검사하는 기법이다.
원인-효과그래프 검사 (Cause-Effect Graphing Testing)	입력 데이터 간의 관계와 출력에 영향을 미치는 상황을 체계적으로 분석한 다음 효용성이 높은 테스트 케이스를 선정하여 검사하는 기법이다.
오류 예측 검사 (Error Guessing)	•과거의 경험이나 확인자의 감각으로 테스트하는 기법이다. •다른 블랙 박스 테스트 기법으로는 찾아낼 수 없는 오류를 찾아내는 일련의 보충적 검사 기법이며, 데이터 확인 검사라고도 한다.
비교 검사 (Comparison Testing)	여러 버전의 프로그램에 동일한 테스트 자료를 제공하여 동일한 결과가 출력되는지 테스트하는 기법이다.

4. 애플리케이션 테스트 관리-SEC_03(테스트 기법에 따른 애플리케이션 테스트)

5) 블랙박스 테스트의 종류

예제) A 애플리케이션에서 평가점수에 따른 성적부여 기준이 다음과 같을 때, 동치 분할 검사와 경계 값 분석의 테스트 케이스를 확인하시오.

평가 점수	성적
90~100	A
80~89	B
70~79	C
0~69	D

<동치 분할 검사>

테스트 케이스	1	2	3	4
입력 값	60	75	82	96
예상 결과값	D	C	B	A
실제 결과값	D	C	B	A

동치 분할 검사는 입력 자료에 초점을 맞춰 테스트 케이스를 만들어 검사하므로 평가점수를 입력한 후 점수에 맞는 성적이 출력되는지 확인한다.

4. 애플리케이션 테스트 관리-SEC_03(테스트 기법에 따른 애플리케이션 테스트)

5) 블랙박스 테스트의 종류

예제) A 애플리케이션에서 평가점수에 따른 성적부여 기준이 다음과 같을 때, 동치 분할 검사와 경계 값 분석의 테스트 케이스를 확인하시오.

평가 점수	성적
90~100	A
80~89	B
70~79	C
0~69	D

<경계 값 분석>

테스트 케이스	1	2	3	4	5	6	7	8	9	10
입력 값	-1	0	69	70	79	80	89	90	100	101
예상 결과값	오류	D	D	C	C	B	B	A	A	오류
실제 결과값	오류	D	D	C	C	B	B	A	A	오류

경계 값 분석은 입력 조건의 경계 값을 테스트 케이스로 선정하여 검사하므로 평가 점수의 경계 값에 해당하는 점수를 입력한 후 올바른 성적이 출력되는지 확인한다.

4. 애플리케이션 테스트 관리-SEC_03(테스트 기법에 따른 애플리케이션 테스트) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(테스트 기법에 따른 애플리케이션 테스트)

1. 소프트웨어 테스트와 관련한 설명으로 틀린 것은?

① 화이트박스 테스트는 모듈의 논리적인 구조를 체계적으로 점검 할 수 있다.

② 블랙박스 테스트는 프로그램의 구조를 고려하지 않는다.

③ 테스트 케이스에는 일반적으로 시험 조건, 테스트 데이터, 예상 결과가 포함되어야 한다.

④ 화이트박스 테스트에서 기본 경로(Basis Path)란 흐름 그래프 의 시작 노드에서 종료 노드까지의 서로 독립된 경로로 싸이클을 허용하지 않는 경로를 말한다.

화이트박스 테스트는 모듈의 원시 코드를 오픈 시킨 상태에서 원시 코드의 논리적인 모든 경로를 테스트 하여 테스트 케이스를 설계하는 방법이다.

화이트박스 테스트는 설계된 절차에 초점을 둔 구조적 테스트로 프로시저 설계의 제어 구조를 사용하여 테스트 케이스를 설계하며 테스트 과정의 초기에 적용된다.

모듈 안의 작동을 직접 관찰한다.

원시 코드(모듈)은 모든 문장을 한 번 이상 실행함으로써 수행된다.

3. 블랙박스 테스트를 이용하여 발견할 수 있는 오류가 아닌 것은?

① 비 정상적인 자료를 입력해도 오류 처리를 수행하지 않는 경우

② 정상적인 자료를 입력해도 요구된 기능이 제대로 수행되지 않는 경우

③ 반복 조건을 만족하는데도 루프 내의 문장이 수행되지 않는 경우

④ 경계 값을 입력할 경우 요구된 출력 결과가 나오지 않는 경우

화이트박스 테스트 종류

기초 경로 검사(Base Path Testing) : 대표적인 화이트박스 테스트 기법이다. 테스트 케이스 설계자가 절차적 설계의 논리적 복잡성을 측정할 수 있게 해주는 테스트 기법으로 테스트 측정 결과는 실행 경로의 기초를 정의하는 데 지침으로 주로 사용된다.

제어 구조 검사(Control Structure Testing)

조건 검사 : 프로그램 모듈 내에 있는 논리적 조건을 테스트 하는 테스트케이스 설계 기법

루프 검사 : 프로그램의 반복 구조에 초점을 맞춰 실시하는 테스트 케이스 설계 기법

데이터 흐름 검사 : 프로그램에서 변수의 정의와 변수 사용의 위치에 초점을 맞춰 실시하는 테스트케이스 설계 기법

4. 블랙박스 테스트 기법으로 거리가 먼 것은?

4. 애플리케이션 테스트 관리-SEC_03(테스트 기법에 따른 애플리케이션 테스트) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(테스트 기법에 따른 애플리케이션 테스트)

5. 평가점수에 따른 성적부여는 다음 표와 같다. 이를 구현한 소프트웨어를 경계 값 분석 기법으로 테스트 하고자 할 때 다음 중 테스트 케이스의 입력 값으로 옳지 않은 것은?

평가 점수	성적
80~100	A
60~79	B
0~59	C

① 59 ② 80

③ 90 ④ 101

경계 값 분석 기법은 경계 값을 입력하여 검사하는 기법이다.

6. 화이트박스 검사 기법에 해당하는 것으로만 짝지어진 것은?

㉠ 데이터 흐름 검사	㉡ 루프 검사
㉢ 동등 분할 검사	㉣ 경계값 분석
㉤ 원인 결과 그래프 기법	㉥ 오류 예측 기법

① ㉠, ㉡ ② ㉠, ㉣

③ ㉡, ㉤ ④ ㉢, ㉥

7. 화이트박스 테스트와 관련한 설명으로 틀린 것은?

① 화이트박스 테스트의 이해를 위해 논리 흐름도(Logic Flow Diagram) 를 이용할 수 있다.

② 테스트 데이터를 이용해 실제 프로그램을 실행함으로써 오류를 찾는 동적 테스트(Dynamic Test)에 해당한다.

③ 프로그램의 구조를 고려하지 않기 때문에 테스트 케이스는 프로그램 또는 모듈의 요구나 명세를 기초로 결정한다.

④ 테스트 데이터를 선택하기 위하여 검증 기준(Test Coverage)을 정한다.

8. 모듈의 논리적 구조를 체계적으로 점검하는 구조 테스트로, 이 방식의 종류에는 기초 경로 검사, 조건 검사, 데이터 흐름 검사, 루프 검사 등이 있는 것은?

① 화이트 박스 테스트

② 블랙 박스 테스트

③ 레드 박스 테스트

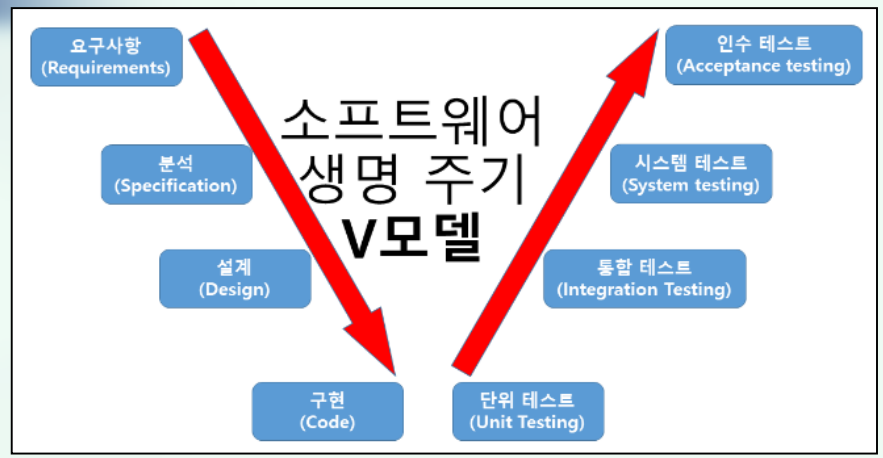
① 블루 박스 테스트

4. 애플리케이션 테스트 관리-SEC_04(개발 단계에 따른 애플리케이션 테스트)

1) 개발 단계에 따른 애플리케이션 테스트

; 애플리케이션 테스트는 소프트웨어의 개발 단계에 따라 단위 테스트, 통합 테스트, 시스템 테스트, 인수 테스트로 분류된다. 이렇게 분류된 것을 테스트 레벨이라고 한다.

- 애플리케이션 테스트는 소프트웨어의 개발 단계에서부터 테스트를 수행하므로 단순히 소프트웨어에 포함된 코드 상의 오류뿐만 아니라 요구 분석의 오류, 설계 인터페이스 오류 등도 발견할 수 있다.
- 애플리케이션 테스트와 소프트웨어 개발 단계를 연결하여 표현한 것을 V-모델이라 한다.



•검증(Verification) 테스트 : 개발자 기준의 테스트로, 단위 테스트, 통합 테스트, 시스템 테스트가 해당됨
•확인(Validation) 테스트 : 사용자 기준의 테스트로, 인수 테스트가 해당됨

4. 애플리케이션 테스트 관리-SEC_04(개발 단계에 따른 애플리케이션 테스트)

2) 단위 테스트(Unit Test)

; 단위 테스트는 코딩 직후 소프트웨어 설계의 최소 단위인 모듈이나 컴포넌트에 초점을 맞춰 테스트하는 것이다.

- 단위 테스트에서는 인터페이스, 외부적 I/O, 자료 구조, 독립적 기초 경로, 오류 처리 경로, 경계 조건 등을 검사한다.
- 단위 테스트는 사용자의 요구사항을 기반으로 한 기능성 테스트를 최우선으로 수행한다.
- 단위 테스트는 구조 기반 테스트와 명세 기반 테스트로 나뉘지만 주로 구조 기반 테스트를 시행한다.
- **단위 테스트로 발견 가능한 오류** : 알고리즘 오류에 따른 원치 않는 결과, 탈출구가 없는 반복문의 사용, 틀린 계산 수식에 의한 잘못된 결과

테스트 방법	테스트 내용	테스트 목적
구조 기반 테스트	프로그램 내부 구조 및 복잡도를 검증하는 화이트 박스(White Box) 테스트 시행	제어 흐름, 조건 결정
명세 기반 테스트	목적 및 실행 코드 기반의 블랙박스(Black Box)테스트 시행	동등 분할, 경계 값 분석

4. 애플리케이션 테스트 관리-SEC_04(개발 단계에 따른 애플리케이션 테스트)

3) 통합 테스트(Integration Test)

; 통합 테스트는 단위 테스트가 완료된 모듈들을 결합하여 하나의 시스템으로 완성시키는 과정에서의 테스트를 의미한다.

● 통합 테스트는 모듈 간 또는 통합된 컴포넌트 간의 상호 작용 오류를 검사한다.

4) 시스템 테스트(System Test)

; 시스템 테스트는 개발된 소프트웨어가 해당 컴퓨터 시스템에서 완벽하게 수행되는가를 점검하는 테스트이다.

● 환경적인 장애 리스크를 최소화하기 위해서는 실제 사용 환경과 유사하게 만든 테스트 환경에서 테스트를 수행해야 한다.

● 시스템 테스트는 기능적 요구사항과 비기능적 요구사항으로 구분하여 각각을 만족하는지 테스트한다.

테스트 방법	테스트 내용
기능적 요구사항	요구사항 명세서, 비즈니스 절차, 유스케이스 등 명세서 기반의 블랙박스(Black Box) 테스트 시행
비기능적 요구사항	성능 테스트, 회복 테스트, 보안 테스트, 내부 시스템의 메뉴 구조, 웹 페이지의 네비게이션 등 구조적 요소에 대한 화이트박스(White Box) 테스트 시행

환경적인 장애 리스크는 OS, DBMS, 시스템 운영 장비 등 테스트 시 사용할 물리적 논리적 테스트 환경과 실제 소프트웨어를 사용할 환경이 달라서 발생할 수 있는 바람직하지 못한 결과를 의미한다.

4. 애플리케이션 테스트 관리-SEC_04(개발 단계에 따른 애플리케이션 테스트)

5) 인수 테스트(Acceptance Test)

; 인수 테스트는 개발한 소프트웨어가사용자의 요구사항을 충족하는지에 중점을 두고 테스트하는 방법이다.

- 인수 테스트는 개발한 소프트웨어를 사용자가 직접 테스트한다.
- 인수 테스트에 문제가 없으면 사용자는 소프트웨어를 인수하게 되고, 프로젝트는 종료된다.
- 인수 테스트는 다음과 같이 6가지의 종류로 구분해서 테스트한다.

테스트 종류	설명
사용자 인수 테스트	사용자가 시스템 사용의 적절성 여부를 확인한다.
운영상의 인수 테스트	시스템 관리자가 시스템 인수 시 수행하는 테스트 기법으로 백업/복원 시스템, 재난 복구 사용자 관리, 정기 점검 등을 확인한다.
계약 인수 테스트	계약상의 인수/검수 조건을 준수하는지 여부를 확인한다.
규정 인수 테스트	소프트웨어가 정부 지침, 법규, 규정 등 규정에 맞게 개발되었는지 확인한다.
알파 테스트	•개발자의 장소에서 사용자가 개발자 앞에서 행하는 테스트 기법이다. •테스트는 통제된 환경에서 행해지며, 오류와 사용상의 문제점을 사용자와 개발자가 함께 확인하면서 기록한다.
베타 테스트	•선정된 최종 사용자가 여러 명의 사용자 앞에서 행하는 테스트 기법으로, 필드 테스팅(Field Testing)이라고도 불린다. •실 업무를 가지고 사용자가 직접 테스트하는 것으로 개발자에 의해 제어되지 않은 상태에서 테스트가 행해지며, 발견된 오류와 사용상의 문제점을 기록하고 개발자에게 주기적으로 보고한다.

4. 애플리케이션 테스트 관리-SEC_04(개발 단계에 따른 애플리케이션 테스트) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(개발 단계에 따른 애플리케이션 테스트)

1. 개별 모듈을 시험하는 것으로, 모듈이 정확하게 구현되었는지, 예정한 기능이 제대로 수행되는지를 점검하는 것이 주 목적인 테스트는?

① 통합 테스트(Integration Test)

② 단위 테스트(Unit Test)

③ 시스템 테스트(System Test)

④ 인수 테스트(Acceptance Test)

단위 테스트는 코딩 직후 소프트웨어 설계의 최소 단위인 모듈이나 컴포넌트에 초점을 맞춰 테스트하는 것이다.

단위 테스트에서는 인터페이스, 외부적 I/O, 자료구조, 독립적 기초 경로, 오류 처리 경로, 경계 조건 등을 검사한다.

단위 테스트는 사용자의 요구사항을 기반으로 한 기능성 테스트를 최우선으로 한다.

단위 테스트 구조 기반 테스트와 명세 기반 테스트로 나뉘지만 주로 구조 기반 테스트를 시행한다.

단위 테스트로 발견 가능한 오류 : 알고리즘 오류에 따른 원치 않은 결과, 탈출구가 없는 반복문의 사용, 틀린 계산 수식에 의한

3. 검증(Validation) 검사 기법 중 개발자의 장소에서 사용자가 개발자 앞에서 행해지며, 오류와 사용상의 문제점을 사용자와 개발자가 함께 확인하면서 검사하는 기법은?

① 디버깅 검사 ② 형상 검사

③ 자료구조 검사 ④ 알파 검사

알파 테스트

개발자의 장소에서 사용자가 개발자 앞에서 행하는 테스트 기법이다.

테스트는 통제된 환경에서 행해지며, 오류와 사용상의 문제점을 사용자와 개발자와 함께 확인하면서 기록한다.

4. 알파, 베타 테스트와 가장 밀접한 연관이 있는 테스트 단계는?

① 단위 테스트 ② 인수 테스트

③ 통합 테스트 ④ 시스템 테스트

인수 테스트는 개발한 소프트웨어가 사용자의 요구사항을 충족하는지에 중점을 두고 테스트하는 방법이다.

인수 테스트는 개발 소프트웨어를 사용자가 직접 테스트를 한다.

인수 테스트에 문자가 없다면 사용자는 소프웨어를 인수하게 되고, 프로젝트는 종료된다.

사용자 인수 테스트

4. 애플리케이션 테스트 관리-SEC_04(개발 단계에 따른 애플리케이션 테스트) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(개발 단계에 따른 애플리케이션 테스트)

5. 필드 테스팅(Field Testing)이라고도 불리며, 개발자 없이 고객의 사용환경에 소프트웨어를 설치하여 검사를 수행하는 인수 검사 기법은?

① 베타 검사 ② 알파 검사

③ 형상 검사 ④ 복구 검사

6. 소프트웨어 생명주기 모델 중 V 모델과 관련한 설명으로 틀린 것은?

① 요구 분석 및 설계 단계를 거치지 않으며 항상 통합 테스트를 중심으로 V 형태를 이룬다.

② Perry에 의해 제안되었으며 세부적인 테스트 과정으로 구성 되어 신뢰도 높은 시스템을 개발하는데 효과적이다.

③ 개발 작업과 검증 작업 사이의 관계를 명확히 들어내 놓은 폭포수 모델의 변형이라고 볼 수 있다.

④ 폭포수 모델이 산출물 중심이라면 V 모델은 작업과 결과의 검증에 초점을 둔다.

소프트웨어 생명 주기의 V-모델은 요구사항 분석 -> 설계 -> 구현 단계로 수행되며 각 단계를 테스트와 연결하여 표현한다.

7. 다음 중 개발 단계에 따른 소프트웨어 테스트 종류가 아닌 것은?

① 단위 테스트

② 시스템 테스트

③ 화이트박스 테스트

④ 통합 테스트

개발 단계에 따른 테스트의 종류 : 단위 테스트, 통합 테스트, 시스템 테스트, 인수 테스트가 있다.

화이트박스 테스트는 모듈의 원시 코드를 오픈 시킨 상태에서 원시 코드의 논리적인 모든 경로를 테스트하는 것으로, 테스트 기법에 따른 소프트웨어 테스트의 한 종류이다.

4. 애플리케이션 테스트 관리-SEC_05(통합 테스트)

1) 통합 테스트(Integration Test)

; 통합 테스트는 단위 테스트가 끝난 모듈을 통합하는 과정에서 발생하는 오류 및 결함을 찾는 테스트 기법이다.

● 통합 테스트 방법에는 비점진적 통합 방식과 점진적 통합 방식이 있다.

비점진적 통합 방식	•단계적으로 통합하는 절차 없이 모든 모듈이 미리 결합되어 있는 프로그램 전체를 테스트하는 방법으로 빅뱅 통합 테스트 방식이 있다. •규모가 작은 소프트웨어에 유리하며 단시간 내에 테스트가 가능하다. •전체 프로그램을 대상으로 하기 때문에 오류 발견 및 장애 위치 파악 및 수정이 어렵다.
점진적 통합 방식	•모듈 단위로 단계적으로 통합하면서 테스트하는 방법으로 하향식, 상향식, 혼합식 통합 방식이 있다. •오류 수정이 용이하고, 인터페이스와 연관된 오류를 완전히 테스트 할 가능성이 높다.

빅뱅 통합 테스트 : 모듈 간의 상호 인터페이스를 고려하지 않고 단위 테스트가 끝난 모듈을 한꺼번에 결합시켜 테스트 하는 방법이다. 주로 소규모 프로그램이나 프로그램의 일부만을 대상으로 테스트 할 때 사용된다.

4. 애플리케이션 테스트 관리-SEC_05(통합 테스트)

2) 하향식 통합 테스트(Top Down Integration Test)

; 하향식 통합 테스트는 프로그램의 상위 모듈에서 하위 모듈 방향으로 통합하면서 테스트하는 기법이다.

- 주요 제어 모듈을 기준으로 하여 아래 단계로 이동하면서 통합하는데, 이때 깊이 우선 통합법이나 넓이 우선 통합법을 사용한다.
- 테스트 초기부터 사용자에게 시스템 구조를 보여줄 수 있다.
- 상위 모듈에서는 테스트 케이스를 사용하기 어렵다.
- 하향식 통합 방법은 다음과 같은 절차로 수행된다.

① 주요 제어 모듈은 작성된 프로그램을 사용하고, 주요 제어 모듈의 종속 모듈들은 스텁(Stub)으로 대체한다.

② 깊이 우선 또는 넓이 우선 등의 통합 방식에 따라 하위 모듈인 스텁들이 한 번에 하나씩 실제 모듈로 교체된다.

③ 모듈이 통합될 때마다 테스트를 실시한다.

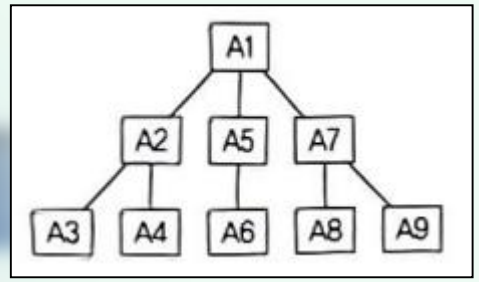
④ 새로운 오류가 발생하지 않음을 보증하기 위해 회귀 테스트를 실시한다.

stub : 토막, 꽁초, 남은 부분.. 이라는 뜻으로 dummy객체가 마치 실제로 동작하는 것처럼 보이도록 만들어 놓은 것이다.
회귀 테스트 : 이미 테스트된 프로그램의 테스팅을 반복하는 것으로 통합 테스트로 인해 변경된 모듈이나 컴포넌트에 새로운 오류가 있는지 확인하는 테스트이다.

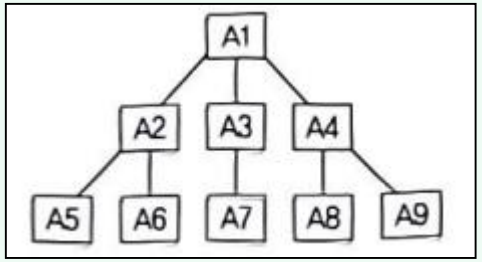
4. 애플리케이션 테스트 관리-SEC_05(통합 테스트)

2) 하향식 통합 테스트(Top Down Integration Test)

- 깊이 우선 통합법은 주요 제어 모듈을 중심으로 해당 모듈에 종속된 모든 모듈을 통합하는 것으로 다음 그림에 대한 통합 순서는 A1, A2, A3, A4, A5, A6, A7, A8, A9 순이다.



- 넓이 우선 통합법은 구조의 수평을 중심으로 해당하는 모듈을 통합하는 것으로 다음 그림에 대한 통합 순서는 A1, A2, A3, A4, A5, A6, A7, A8, A9 순입니다.



4. 애플리케이션 테스트 관리-SEC_05(통합 테스트)

3) 상향식 통합 테스트(Bottom Up Integration Test)

; 상향식 통합 테스트는 프로그램의 하위 모듈에서 상위 모듈 방향으로 통합하면서 테스트하는 기법이다.

- 가장 하위 단계의 모듈부터 통합 및 테스트가 수행되므로 스텁(Stub)은 필요하지 않지만, 하나의 주요 제어 모듈과 관련된 종속 모듈의 그룹인 클러스터(Cluster)가 필요하다.
- 상향식 통합 방법은 다음과 같은 절차로 수행된다.
  ① 하위 모듈들을 클러스터(Cluster)로 결합한다.
  ② 상위 모듈에서 데이터의 입·출력을 확인하기 위해 더미 모듈인 드라이버(Driver)를 작성한다.
  ③ 통합된 클러스터 단위로 테스트한다.
  ④ 테스트가 완료되면 클러스터는 프로그램 구조의 상위로 이동하여 결합하고 드라이버는 실제 모듈로 대체된다.

클러스터 : 똑같은 구성의 여러 대의 서버를 병렬로 연결한 상태를 말한다. 쉽게 말하자면 여러 대의 서버 컴퓨터를 마치 하나의 가상 컴퓨터 처럼 업무를 수행하도록 하는 것을 의미한다.

4. 애플리케이션 테스트 관리-SEC_05(통합 테스트)

4) 테스트 드라이버와 테스트 스텁의 차이점

구분	드라이버(Driver)	스텁(Stub)
개념	테스트 대상의 하위 모듈을 호출하는 도구로, 매개 변수(Parameter)를 전달하고 모듈 테스트 수행 후의 결과를 도출한다.	제어 모듈이 호출하는 타 모듈의 기능을 단순히 수행하는 도구로, 일시적으로 필요한 조건만을 가지고 있는 시험용 모듈이다.
필요 시기	상위 모듈 없이 하위 모듈이 있는 경우 하위 모듈 구동	상위 모듈은 있지만 하위 모듈이 없는 경우 하위 모듈 대체
테스트 방식	상향식(Bottom Up) 테스트	하향식(Top-Down) 테스트
공통점	소프트웨어 개발과 테스트를 병행할 경우 이용	
차이점	•이미 존재하는 하위 모듈과 존재하지 않는 상위 모듈 간의 인터페이스 역할을 한다. •소프트웨어 개발이 완료되면 드라이버는 본래의 모듈로 교체된다.	•일시적으로 필요한 조건만을 가지고 임시로 제공되는 가짜 모듈의 역할을 한다. •시험용 모듈이기 때문에 일반적으로 드라이버보다 작성하기 쉽다.

4. 애플리케이션 테스트 관리-SEC_05(통합 테스트)

5) 혼합식 통합 테스트

; 혼합식 통합 테스트는 하위 수준에서는 상향식 통합, 상위 수준에서는 하향식 통합을 사용하여 최적의 테스트를 지원하는 방식으로, 샌드위치(Sandwich)식 통합 테스트 방법이라고도 한다.

6) 회귀 테스팅(Regression Testing)

; 회귀 테스트는 이미 테스트된 프로그램의 테스팅을 반복하는 것으로, 통합 테스트로 인해 변경된 모듈이나 컴포넌트에 새로운 오류가 있는지 확인하는 테스트이다.

- 회귀 테스트는 수정한 모듈이나 컴포넌트가 다른 부분에 영향을 미치는지, 오류가 생기지 않았는지 테스트하여 새로운 오류가 발생하지 않음을 보증하기 위해 반복 테스트 한다.
- 회귀 테스트는 모든 테스트 케이스를 이용해 테스팅 하는 것이 가장 좋지만 시간과 비용이 많이 필요하므로 기존 테스트 케이스 중 변경된 부분을 테스트할 수 있는 테스트 케이스만을 선정하여 수행한다.
- 회귀 테스트의 테스트 케이스 선정 방법
  - 모든 애플리케이션의 기능을 수행할 수 있는 대표적인 테스트 케이스를 선정한다.
  - 애플리케이션 기능 변경에 의한 파급 효과를 분석하여 파급 효과가 높은 부분이 포함된 테스트 케이스를 선정한다.
  - 실제 수정이 발생한 모듈 또는 컴포넌트에서 시행하는 테스트 케이스를 선정한다.

4. 애플리케이션 테스트 관리-SEC_05(통합 테스트) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(통합 테스트)

1. 테스트 드라이버(Test Driver)에 대한 설명으로 틀린 것은?

① 시험 대상 모듈을 호출하는 간이 소프트웨어이다.

② 필요에 따라 매개 변수를 전달하고 모듈을 수행한 후의 결과를 보여줄 수 있다.

③ 상향식 통합 테스트에서 사용된다.

④ 테스트 대상 모듈이 호출하는 하위 모듈의 역할을 한다.

테스트 드라이버(Driver)
테스트 대상의 하위 모듈을 호출하는 도구로, 매개변수(Parameter)를 전달하고 모듈 테스트 수행 후의 결과를 도출한다.
상위 모듈 없이 하위 모듈이 있는 경우 하위 모듈 구동을 하고 상향식(Bottom Up)테스트에서 사용된다.
이미 존재하는 하위 모듈과 존재하지 않는 상위 모듈 간의 인터페이스 역할을 한다.
소프트웨어 개발이 완료되면 드라이버는 본래의 모듈로 교체된다.

스텁(Stub)
제어 모듈이 호출하는 타 모듈의 기능을 단순히 수행하는 도구로 일시적으로 필요한 조건만 가지고 있는 시험용 모듈이다.
상위 모듈은 있지만 하위 모듈이 없는 경우 하위 모듈을 대체하고, 하향식(Top-Down)테스트에 사용된다.
시험용 모듈이기 때문에 일반적으로 드라이버보다는 작성하기가 수월하다.

2. 통합 테스트(Integration Test)와 관련한 설명으로 틀린 것은?

3. 하향식 통합에 있어서 모듈 간의 통합 시험을 위해 일시적으로 필요한 조건만을 가지고 임시로 제공되는 시험용 모듈을 무엇이라고 하는가?

① Stub ② Driver

③ Procedure ④ Function

4. 다음이 설명하는 애플리케이션 통합 테스트 유형은?

깊이 우선 방식 또는 너비 우선 방식이 있다.
상위 컴포넌트를 테스트 하고 점증적으로 하위 컴포넌트를 테스트 한다.
하위 컴포넌트 개발이 완료되지 않은 경우 스텁(Stub)을 사용하기도 한다.

① 하향식 통합 테스트 ② 상향식 통합 테스트

③ 회귀 테스트 ④ 빅뱅 테스트

상위 컴포넌트를 먼저 테스트 하고 하위 컴포넌트를 테스트 한다는 것은 하향식으로 테스트를 수행한다는 의미이다.

4. 애플리케이션 테스트 관리-SEC_05(통합 테스트) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(통합 테스트)

5. 상향식 통합 테스트(Bottom-up Integration Test)의 과정이 옳게 나열된 것은?

㉠ 드라이버라는 제어 프로그램의 작성
㉡ 낮은 수준의 모듈들을 클러스터로 결합
㉢ 클러스터의 검사
㉣ 클러스터를 상위로 결합

① ㉠ -> ㉡ -> ㉢ -> ㉣
② ㉡ -> ㉢ -> ㉠ -> ㉣
② ㉡ -> ㉠ -> ㉢ -> ㉣
④ ㉠ -> ㉡ -> ㉣ -> ㉢

6. 하향식 통합에 있어서 모듈 간의 통합 시험을 하기 위해 일시적으로 필요한 조건만을 가지고 임시로 제공되는 시험용 모듈을 무엇이라고 하는가?

① Driver
② Stub
③ Sub-Program
④ Dummy-Program

7. 하향식 통합에 대한 설명으로 가장 적합하지 않은 것은?

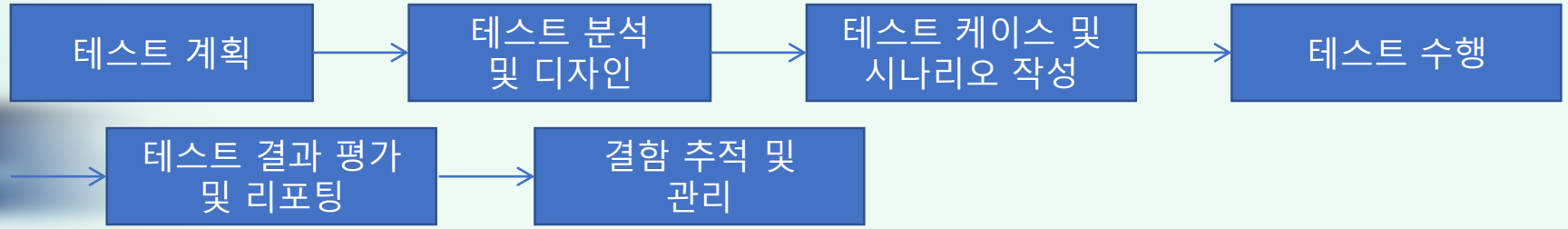
① 프로그램의 상위 모듈에서 하위 모듈 방향으로 통합하면서 테스트하는 기법이다.
② 마지막까지 독립된 프로그램 형태를 갖지 못한다.
③ 주요 제어 모듈의 종속 모듈들을 스텁으로 대체한다.
④ 깊이 우선 방식이나 넓이 우선 방식에 의해 통합한다.

하향식 통합 기법은 처음부터 독립된 프로그램 구조를 가지고 있다.

4. 애플리케이션 테스트 관리-SEC_06(애플리케이션 테스트 프로세스)

1) 애플리케이션 테스트 프로세스

; 애플리케이션 테스트 프로세스는 개발된 소프트웨어가 사용자의 요구대로 만들어졌는지, 결함은 없는지 등을 테스트하는 절차로, 다음과 같은 순서로 진행된다.



● 애플리케이션 테스트를 마치면 테스트 계획서, 테스트 케이스, 테스트 시나리오, 테스트 결과서가 산출된다.

- 테스트 계획서 : 테스트 목적, 범위, 일정, 수행 절차, 대상 시스템 구조, 조직의 역할 및 책임 등 테스트 수행을 계획한 문서
- 테스트 케이스 : 사용자의 요구사항을 얼마나 준수하는지 확인하기 위한 입력 값, 실행 조건, 기대 결과 등으로 만들어진 테스트 항목의 명세서
- 테스트 시나리오 : 테스트를 수행할 여러 개의 테스트 케이스의 동작 순서를 기술한 문서
- 테스트 결과서 : 테스트 결과를 비교·분석한 내용을 정리한 문서

4. 애플리케이션 테스트 관리-SEC_06(애플리케이션 테스트 프로세스)

2) 테스트 계획

; 테스트 계획 단계에서는 프로젝트 계획서, 요구 명세서 등을 기반으로 테스트 목표를 정의하고 테스트 대상 및 범위를 결정한다.

- 테스트 대상 시스템의 구조를 파악한다.
- 테스트에 투입되는 조직 및 비용을 산정한다.
- 테스트 시작 및 종료 조건을 정의한다.
  - 테스트 시작 조건 : 테스트 계획, 일정, 환경 구축, 사용자 요구사항에 대한 테스트 명세서, 투입 조직 및 참여 인력의 역할과 책임 등이 완료되면 테스트가 시작되도록 조건을 정의할 수 있으며, 모든 조건을 만족하지 않아도 테스트를 시작하도록 지정할 수 있다.
  - 테스트 종료 조건: 정상적으로 테스트를 완료한 경우, 테스트 일정이 만료된 경우, 테스트 비용이 모두 소진된 경우 등 업무 기능의 중요도에 따라 테스트 종료 조건을 다르게 지정할 수 있다.
- 테스트 계획서를 작성한다.

4. 애플리케이션 테스트 관리-SEC_06(애플리케이션 테스트 프로세스)

3) 테스트 분석 및 디자인

; 테스트 분석 및 디자인 단계에서는 테스트의 목적과 원칙을 검토하고 사용자의 요구 사항을 분석한다.

- 테스트에 대한 리스크 분석 및 우선순위를 결정한다.
- 테스트 데이터, 테스트 환경, 테스트 도구 등을 준비한다.

; 테스트 데이터는 시스템의 기능이나 적합성 등을 테스트하기 위해 만든 데이터 집합으로, 소프트웨어의 기능을 차례대로 테스트할 수 있도록 만든 데이터입니다.

- 잘못된 데이터는 잘못된 결과를 도출하기 때문에 효율적인 테스트를 위해서는 올바른 테스트 데이터를 준비해야 한다.
- 테스트 데이터 종류
  - 실제 데이터 : 선행된 연산에 의해 만들거나 실제 운영되는 데이터를 복제한 데이터
  - 가상 데이터 : 스크립트를 통해서 인위적으로 만든 데이터

4. 애플리케이션 테스트 관리-SEC_06(애플리케이션 테스트 프로세스)

4) 테스트 케이스 및 시나리오 작성

; 테스트 케이스 및 시나리오 작성 단계에서는 테스트 케이스의 설계 기법에 따라 테스트 케이스를 작성하고 검토 및 확인한 후 테스트 시나리오를 작성한다.

● 테스트용 스크립트를 작성한다.

테스트 스크립트는 테스트 실행 절차나 수행 방법 등을 스크립트 언어로 작성한 파일을 말한다.
※ 스크립트 언어 : 소스 코드를 컴파일 하지 않고도 내장된 번역기에 의해 번역되어 바로 실행할 수 있는 언어(Perl, Javascript 등)

4. 애플리케이션 테스트 관리-SEC_06(애플리케이션 테스트 프로세스)

5) 테스트 수행

; 테스트 수행 단계에서는 테스트 환경을 구축한 후 테스트를 수행한다.

- 테스트의 실행 결과를 측정하여 기록한다.
- 테스트 데이터, 테스트 환경, 테스트 도구 등을 준비한다.
- 테스트 환경 구축은 개발된 소프트웨어가 실제 시스템에서 정상적으로 작동하는지 테스트하기 위해 실제 시스템과 동일하거나 유사한 사양의 하드웨어 소프트웨어, 네트워크 등의 시설을 구축하는 것이다.
  - 하드웨어 : 서버, 클라이언트, 네트워크 등의 관련 장비 설치
  - 소프트웨어 : 구축된 하드웨어 환경에 테스트 할 소프트웨어 설치
  - 가상 시스템 : 독립된 테스트 환경을 구축하기 힘든 경우, 가상 머신(Virtual Machine) 기반의 서버 또는 클라우드 환경을 구축하고 네트워크는 VLAN과 같은 기법을 이용하여 논리적인 분할 환경을 구축한다.

가상 머신(Virtual Machine) : 가상 머신은 하드웨어 환경을 소프트웨어로 구현한 것으로 시스템에 설치된 운영체제와 다른 운영체제를 사용해야 하거나 독립된 작업 공간이 필요한 경우에 사용된다.
클라우드 환경 : 서로 다른 물리적인 위치에 존재하는 컴퓨팅 자원을 가상화 기술로 통합하고 인터넷상의 서버를 통하여 네트워크, 데이터 저장, 콘텐츠 사용 등의 서비스를 한 번에 사용할 수 있는 환경을 의미한다.
VLAN(Virtual Local Area Network) : VLAN은 LAN을 물리적인 배치와는 상관없이 논리적으로 분리하는 기술이다.

4. 애플리케이션 테스트 관리-SEC_06(애플리케이션 테스트 프로세스)

6) 테스트 결과 평가 및 리포팅

; 테스트 결과 평가 및 리포팅 단계에서는 테스트 결과를 비교 분석하여 테스트 결과서를 작성한다.

- 테스트 결과서는 결함 내용 및 결함 재현 순서 등 결함을 중점적으로 기록한다.
- 테스트가 종료되면 테스트 실행 절차의 리뷰 및 결과에 대한 평가를 수행하고, 그 결과에 따라 실행 절차를 최적화하여 다음 테스트에 적용한다.

4. 애플리케이션 테스트 관리-SEC_06(애플리케이션 테스트 프로세스)

7) 결함 추적 및 관리

; 결함 추적 및 관리 단계에서는 테스트를 수행한 후 결함이 어디에서 발생했는지, 어떤 종류의 결함인지 등 결함을 추적하고 관리한다.

- 결함 추적 및 관리를 통해 동일한 결함 발견 시 처리 시간 단축 및 결함의 재발을 방지할 수 있다.
- 결함 관리 프로세스

① 에러 발견 : 에러가 발견되면 테스트 전문가와 프로젝트 팀이 논의한다.

② 에러 등록 : 발견된 에러를 결함 관리 대장에 등록한다.

③ 에러 분석 : 등록된 에러가 실제 결함인지 아닌지를 분석한다.

④ 결함 확정 : 등록된 에러가 실제 결함이면 결함 확정 상태로 설정한다.

⑤ 결함 할당 : 결함을 해결할 담당자에게 결함을 할당하고 결함 할당 상태로 설정한다.

⑥ 결함 조치 : 결함을 수정하고, 수정이 완료되면 결함 조치 상태로 설정한다.

⑦ 결함조치 검토 및 승인 : 수정이 완료된 결함에 대해 확인 테스트를 수행하고, 이상이 없으면 결함 조치 완료 상태로 설정한다.

에러(Error)/오류 : 결함(Defect)의 원인이 되는 것으로, 일반적으로 소프트웨어 개발자, 분석가 등 사람에 의해 발생한 실수를 의미한다.
결함/결점/버그(Bug) : 에러/오류로 인해 소프트웨어 제품에 발생한 결함을 의미하며, 결함을 제거하지 않으면 소프트웨어 제품에 문제(Problem)가 발생할 수 있다.

4. 애플리케이션 테스트 관리- SEC_06(애플리케이션 테스트 프로세스) 출제 예상 문제

출제 예상 문제(애플리케이션 테스트 프로세스)

1. 다음 중 애플리케이션 테스트의 과정으로 옳게 나열된 것은?

㉮ 테스트 분석

㉯ 테스트 계획

㉰ 테스트 실행

㉱ 테스트 케이스 설계

㉲ 테스트 결과 분석

① ㉮ -> ㉯ -> ㉰ -> ㉱ -> ㉲

② ㉮ -> ㉯ -> ㉱ -> ㉰ -> ㉲

③ ㉯ -> ㉮ -> ㉱ -> ㉰ -> ㉲

④ ㉯ -> ㉮ -> ㉰ -> ㉱ -> ㉲

애플리케이션 테스트의 과정을 올바르게 나열하면 계획 -> 분석 -> 설계 -> 실행 -> 결과 분석 -> 결함 관리 순이다.

2. 다음 중 테스트를 마치면 산출되는 문서가 아닌 것은?

① 테스트 계획서

② 테스트 케이스

③ 테스트 다이어그램

④ 테스트 결과서

3. 테스트를 진행한다고 할 때 언제 오류를 발견하는 것이 가장 좋은가?

① 요구사항 분석 단계

② UI 설계 단계

③ UI 구현 단계

④ 결함 추적 및 관리 단계

오류는 될 수 있으면 빨리 발견하는 게 가장 좋다.

4. 다음 중, 결함관리 순서로 나열할 경우 올바른 것은?

㉠ 에러 발견

㉡ 에러 등록

㉢ 에러 분석

㉣ 결함 확정

㉤ 결함 할당

㉥ 결함 조치

① ㉠ -> ㉡ -> ㉢ -> ㉣ -> ㉤ -> ㉥

② ㉠ -> ㉢ -> ㉡ -> ㉣ -> ㉤ -> ㉥

③ ㉡ -> ㉣ -> ㉤ -> ㉥ -> ㉠ -> ㉢

④ ㉡ -> ㉣ -> ㉤ -> ㉢ -> ㉠ -> ㉥

4. 애플리케이션 테스트 관리- SEC_06(애플리케이션 테스트 프로세스) 출제 예상 문제

출제 예상 문제(애플리케이션 테스트 프로세스)

5. 결함(Defect)의 원인이 되는 것으로, 일반적으로 소프트웨어 개발자, 분석가 등 사람에 의해 발생한 실수를 무엇이라고 하는가?

① 결함

② 결점

③ 에러

④ 버그

에러(Error)/오류 : 결함(Defect)의 원인이 되는 것으로, 일반적으로 소프트웨어 개발자, 분석가 등 사람에 의해서 발생한 실수를 의미한다.

결함/결점/버그(Bug) : 에러/오류로 인하여 소프트웨어 제품에 발생한 결함을 의미하며, 결함을 제거하지 않는다면 소프트웨어 제품에 문제(Problem)가 발생할 수 있기에 반드시 결함을 제거해야 한다.

4. 애플리케이션 테스트 관리-SEC_07(테스트 케이스/테스트 시나리오/테스트 오라클)

1) 테스트 케이스(Test Case)

; 테스트 케이스는 구현된 소프트웨어가 사용자의 요구사항을 정확하게 준수했는지를 확인하기 위해 설계된 입력 값, 실행 조건, 기대 결과 등으로 구성된 테스트 항목에 대한 명세서로, 명세 기반 테스트의 설계 산출물에 해당된다.

- 테스트 케이스를 미리 설계하면 테스트 오류를 방지할 수 있고 테스트 수행에 필요한 인력, 시간 등의 낭비를 줄일 수 있다.
- 테스트 케이스는 테스트 목표와 방법을 설정한 후 작성한다.
- 테스트 케이스는 시스템 설계 단계에서 작성하는 것이 가장 이상적이다.

명세 기반 테스트 : 사용자의 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 구현하고 있는지를 확인하는 것이다.

4. 애플리케이션 테스트 관리-SEC_07(테스트 케이스/테스트 시나리오/테스트 오라클)

2) 테스트 케이스 작성 순서

; 테스트 케이스는 테스트 전략이나 테스트 계획서 등을 기반으로 하여 다음과 같은 순서로 작성된다.

1. 테스트 계획 검토 및 자료 확보	•테스트 계획서를 재검토하여 테스트 대상 범위 및 접근 방법 등을 이해한다. •시스템 요구사항과 기능 명세서를 검토하고 테스트 대상 시스템의 정보를 확보한다.
2. 위험 평가 및 우선순위 결정	결함의 위험 정도에 따른 우선순위를 결정하고, 어느 부분에 초점을 맞춰 테스트 할지를 결정한다.
3. 테스트 요구사항 정의	시스템에 대한 사용자 요구사항이나 테스트 대상을 재검토하고, 테스트 특성, 조건, 기능 등을 분석한다.
4. 테스트 구조 설계 및 테스트 방법 결정	•테스트 케이스의 형식과 분류 방법을 결정한다. •테스트 절차, 장비, 도구, 테스트 문서화 방법을 결정한다.
5. 테스트 케이스 정의	요구사항에 따라 테스트 케이스를 작성하고, 입력 값, 실행 조건, 예상 결과 등을 기술한다.
6. 테스트 케이스 타당성 확인 및 유지 보수	•소프트웨어의 기능 또는 환경 변화에 따라 테스트 케이스를 갱신한다. •테스트 케이스의 유용성을 검토한다.

4. 애플리케이션 테스트 관리-SEC_07(테스트 케이스/테스트 시나리오/테스트 오라클)

3) 테스트 시나리오(Test Scenario)

; 테스트 시나리오는 테스트 케이스를 적용하는 순서에 따라 여러 개의 테스트 케이스들을 묶은 집합으로, 테스트 케이스들을 적용하는 구체적인 절차를 명세한 문서이다.

- 테스트 시나리오에는 테스트 순서에 대한 구체적인 절차, 사전 조건, 입력 데이터 등이 설정되어 있다.
- 테스트 시나리오를 통해 테스트 순서를 미리 정함으로써 테스트 항목을 빠짐없이 수행할 수 있다.

4) 테스트 시나리오 작성 시 유의 사항

- 테스트 시나리오는 시스템 별, 모듈별, 항목별 등과 같이 여러 개의 시나리오로 분리하여 작성해야 한다.
- 테스트 시나리오는 사용자의 요구사항과 설계 문서 등을 토대로 작성해야 한다.
- 각각의 테스트 항목은 식별자 번호, 순서 번호, 테스트 데이터, 테스트 케이스, 예상 결과, 확인 등을 포함해서 작성해야 한다.
- 테스트 시나리오는 유스케이스(Use Case)간 업무 흐름이 정상적인지를 테스트 할 수 있도록 작성해야 한다.
- 테스트 시나리오는 개발된 모듈 또는 프로그램 간의 연계가 정상적으로 동작하는지 테스트할 수 있도록 작성해야 한다.

4. 애플리케이션 테스트 관리-SEC_07(테스트 케이스/테스트 시나리오/테스트 오라클)

5) 테스트 오라클(Test Oracle)

; 테스트 오라클은 테스트 결과가 올바른지 판단하기 위해 사전에 정의된 참 값을 대입하여 비교하는 기법 및 활동을 말한다.

- 테스트 오라클은 결과를 판단하기 위해 테스트 케이스에 대한 예상 결과를 계산하거나 확인한다.
- 테스트 오라클의 특징
  - 제한된 검증 : 테스트 오라클을 모든 테스트 케이스에 적용할 수 없다.
  - 수학적 기법 : 테스트 오라클의 값을 수학적 기법을 이용하여 구할 수 있다.
  - 자동화 가능 : 테스트 대상 프로그램의 실행, 결과 비교, 커버리지 측정 등을 자동화 할 수 있다.
- 오라클의 종류에는 참(True) 오라클, 샘플링(Sampling) 오라클, 추정(Heuristic) 오라클, 일관성(Consistent) 검사 오라클 등이 있다.

4. 애플리케이션 테스트 관리-SEC_07(테스트 케이스/테스트 시나리오/테스트 오라클)

6) 테스트 오라클의 종류

; 참 오라클은 주로 항공기, 은행, 발전소 소프트웨어 등 미션 크리티컬한 업무에 사용되고, 샘플링 오라클과 추정 오라클은 일반적인 업무, 게임, 오락 등에 사용된다.

참(True) 오라클	모든 테스트 케이스의 입력 값에 대해 **기대하는 결과**를 제공하는 오라클로, 발생된 모든 오류를 검출할 수 있다.
샘플링(Sampling) 오라클	특정한 몇몇 테스트 케이스의 입력 값들에 대해서만 기대하는 결과를 제공하는 오라클이다.
추정(Heuristic) 오라클	샘플링 오라클을 개선한 오라클로, 특정 테스트 케이스의 입력 값에 대해 기대하는 결과를 제공하고, 나머지 입력 값들에 대해서는 추정으로 처리하는 오라클이다.
일관성(Consistent) 검사 오라클	애플리케이션의 변경이 있을 때, 테스트 케이스의 수행 전과 후의 결과 값이 동일한지를 확인하는 오라클이다.

기대하는 결과 : 예를 들면, 퇴직금을 계산하는 프로그램에서 근무기간을 5로 넣어 테스트 케이스를 실행하였을 경우 예상되는 퇴직금이 기대 결과가 되는 것이다.

미션 크리티컬(Mission Critical) : 단 한 번이라도 다운되면 시스템 전체에 치명적인 영향을 주므로 절대 다운되면 안되는 시스템으로 항공기 운행, 은행의 온라인 시스템 등이 해당된다.

4. 애플리케이션 테스트 관리- SEC_07(테스트 케이스/테스트 시나리오/테스트 오라클) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(테스트 케이스/테스트 시나리오/테스트 오라클)

1. 다음이 설명하는 테스트 용어는?

- 테스트의 결과가 참인지 거짓인지를 판단하기 위해서 사전에 정의된 참 값을 입력하여 비교하는 기법 및 활동을 말한다.
- 종류에는 참, 샘플링, 휴리스틱, 일관성 검사가 존재한다.

① 테스트 케이스② 테스트 시나리오
③ 테스트 오라클④ 테스트 데이터

테스트 오라클은 테스트 결과가 올바른지 판단하기 위해서 사전에 정의된 참 값을 대입하여 비교하는 기법 및 활동을 말한다.
테스트 오라클은 결과를 판단하기 위해 테스트 케이스에 대한 예상 결과를 계산하거나 확인한다.
테스트 오라클의 특징
1. 제한된 검증 : 테스트 오라클을 모든 테스트 케이스에 적용할 수 없다.
2. 수학적 기법 : 테스트 오라클의 값을 수학적 기법을 이용하여 구할 수 있다.

3. 테스트 케이스와 관련한 설명으로 틀린 것은?

① 테스트의 목표 및 테스트 방법을 결정하기 전에 테스트 케이스를 작성해야 한다.
② 프로그램에 결함이 있더라도 입력에 대해 정상적인 결과를 낼 수 있기 때문에 결함을 검사할 수 있는 테스트 케이스를 찾는 것이 중요 하다.
③ 개발된 서비스가 정의된 요구사항을 준수하는지 확인하기 위한 입력 값과 실행 조건, 예상 결과의 집합으로 볼 수 있다.
④ 테스트 케이스 실행이 통과되었는지 실패하였는지 판단하기 위한 기준을 테스트 오라클(Test Oracle)이라고 한다.

테스트 케이스는 구현된 소프트웨어가 사용자의 요구사항을 정확하게 준수했는지를 확인하기 위해 설계된 입력 값, 실행 조건, 기대 결과 등으로 구성된 테스트 항목에 대한 명세서로, 명세 기반 테스트의 설계 산출물에 해당한다.
1. 테스트 케이스를 미리 설계하면 테스트 오류를 방지할 수 있고 테스트 수행에 필요한 인력, 시간 등의 낭비를 줄일 수 있다.
2. 테스트 케이스는 테스트 목표와 방법을 먼저 설정한 후 작성한다.
3. 테스트 케이스는 시스템 설계 단계에서 작성하는 것이 가장 이상적이다.

4. 애플리케이션 테스트 관리- SEC_07(테스트 케이스/테스트 시나리오/테스트 오라클) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(테스트 케이스/테스트 시나리오/테스트 오라클)

5. 다음 중 테스트 오라클의 종류가 아닌 것은?

① 참(True) 오라클

② 거짓(False) 오라클

③ 샘플링(Sampling) 오라클

④ 추정(Heuristic) 오라클

테스트 오라클 종류는 참, 샘플링, 추정, 일관성 검사 4가지가 존재한다.

6. 다음 중 테스트 시나리오와 테스트 케이스에 대한 설명으로 가장 옳지 않은 것은?

① 테스트 시나리오는 테스트 케이스의 동작 순서를 기술한 문서이다.

② 테스트 케이스는 테스트 절차를 명세한 문서이다.

③ 테스트 시나리오는 개발된 모듈 또는 프로그램 간의 연계가 정상적으로 동작하는지 테스트할 수 있도록 작성해야 한다.

④ 테스트 케이스는 테스트할 시스템이 수행해야 할 액션들로 구성된 일련의 단계이다.

4. 애플리케이션 테스트 관리-SEC_08(테스트 자동화 도구)

1) 테스트 자동화의 개념

; 테스트 자동화는 사람이 반복적으로 수행하던 테스트 절차를 스크립트 형태로 구현하는 자동화 도구를 적용함으로써 쉽고 효율적으로 테스트를 수행할 수 있도록 한 것이다.

● 테스트 자동화 도구를 사용함으로써 휴먼 에러(Human Error)를 줄이고 테스트의 정확성을 유지하면서 테스트의 품질을 향상시킬 수 있다.

테스트 스크립트(Test Script) : 테스트 스크립트는 테스트 실행 절차나 수행 방법 등을 스크립트 언어로 작성한 파일이다.
※스크립트 언어 : 소스 코드를 컴파일 하지 않고도 내장된 번역기에 의해 번역되어 바로 실행할 수 있는 언어
휴먼 에러(Human Error) : 휴먼 에러는 사람의 판단 실수나 조작 실수 등으로 인해 발생하는 에러이다.

4. 애플리케이션 테스트 관리-SEC_08(테스트 자동화 도구)

2) 테스트 자동화 도구의 장점/단점

장점	•테스트 데이터의 재입력, 재구성 같은 반복적인 작업을 자동화함으로써 인력 및 시간을 줄일 수 있다. •다중 플랫폼 호환성, 소프트웨어 구성, 기본 테스트 등 향상된 테스트 품질을 보장한다. •사용자의 요구사항 등을 일관성 있게 검증할 수 있다. •테스트 결과에 대한 객관적인 평가 기준을 제공한다. •테스트 결과를 그래프 등 다양한 표시 형태로 제공한다. •UI가 없는 서비스도 정밀 테스트가 가능하다.
단점	•테스트 자동화 도구의 사용 방법에 대한 교육 및 학습이 필요하다. •자동화 도구를 프로세스 단계별로 적용하기 위한 시간 비용, 노력이 필요하다. •비공개 상용 도구의 경우 고가의 추가 비용이 필요하다.

3) 테스트 자동화 수행 시 고려사항

- 테스트 절차를 고려하여 재사용 및 측정이 불가능한 테스트 프로그램은 제외한다.
- 모든 테스트 과정을 자동화 할 수 있는 도구는 없으므로 용도에 맞는 적절한 도구를 선택해서 사용한다.
- 자동화 도구의 환경 설정 및 습득 기간을 고려해서 프로젝트 일정을 계획해야 한다.
- 테스트 엔지니어의 투입 시기가 늦어지면 프로젝트의 이해 부족으로 인해 불완전한 테스트를 초래할 수 있으므로 반드시 프로젝트 초기에 테스트 엔지니어의 투입 시기를 계획해야 한다.

비공개 상용 도구 : 비공개 상용 도구는 특정 기업체 전용으로 개발되어 독점 공급되는 소프트웨어를 의미한다.

4. 애플리케이션 테스트 관리-SEC_08(테스트 자동화 도구)

4) 테스트 자동화 도구의 유형

정적 분석 도구 (Static Analysis Tools)	•프로그램을 실행하지 않고 분석하는 도구로, 소스 코드에 대한 코딩 표준, 코딩 스타일, 코드 복잡도 및 남은 결함 등을 발견하기 위해 사용된다. •테스트를 수행하는 사람이 작성된 소스 코드를 이해하고 있어야만 분석이 가능하다.
테스트 케이스 생성 도구 (Test Case Generation Tools)	•자료 흐름도 : 자료 원시 프로그램을 입력 받아 파싱한 후 자료 흐름도를 작성함 •기능 테스트 : 주어진 기능을 구동시키는 모든 가능한 상태를 파악하여 이에 대한 입력을 작성함 •입력 도메인 분석 : 원시 코드의 내부를 참조하지 않고, 입력 변수의 도메인을 분석하여 테스트 데이터를 작성함 •랜덤 테스트 : 입력 값을 무작위로 추출하여 테스트함
테스트 실행 도구 (Test Execution Tools)	•스크립트 언어를 사용하여 테스트를 실행하는 방법으로, 테스트 데이터와 테스트 수행 방법 등이 포함된 스크립트를 작성한 후 실행한다. •데이터 주도 접근 방식 - 스프레드시트에 테스트 데이터를 저장하고, 이를 읽어 실행하는 방식이다. - 다양한 테스트 데이터를 동일한 테스트 케이스로 반복하여 실행할 수 있다. - 스크립트에 익숙하지 않은 사용자도 미리 작성된 스크립트에 테스트 데이터만 추가하여 테스트할 수 있다. •키워드 주도 접근 방식 - 스프레드시트에 테스트를 수행할 동작을 나타내는 키워드와 테스트 데이터를 저장하여 실행하는 방식이다. - 키워드를 이용하여 테스트를 정의할 수 있다.
성능 테스트 도구 (Performance Test Tools)	애플리케이션의 처리량, 응답 시간, 경과 시간, 자원 사용률 등을 인위적으로 적용한 가상의 사용자를 만들어 테스트를 수행함으로써 성능의 목표 달성 여부를 확인한다.
테스트 통제 도구 (Test Control Tools)	테스트 계획 및 관리, 테스트 수행, 결함 관리 등을 수행하는 도구로, 종류에는 형상관리 도구, 결함 추적/관리 도구 등이 있다.
테스트 하네스 도구 (Test Harness Tools)	•테스트 하네스는 애플리케이션의 컴포넌트 및 모듈을 테스트하는 환경의 일부분으로, 테스트를 지원하기 위해 생성된 코드와 데이터를 의미한다. •테스트 하네스 도구는 테스트가 실행될 환경을 시뮬레이션 하여 컴포넌트 및 모듈이 정상적으로 테스트 되도록 한다.

4. 애플리케이션 테스트 관리-SEC_08(테스트 자동화 도구)

5) 테스트 하네스(Test Harness)의 구성 요소

- 테스트 드라이버(Test Driver) : 테스트 대상의 하위 모듈을 호출하고, 매개 변수(Parameter)를 전달하고, 모듈 테스트 수행 후의 결과를 도출하는 도구
- 테스트 스텁(Test Stub) : 제어 모듈이 호출하는 타 모듈의 기능을 단순히 수행하는 도구로, 일시적으로 필요한 조건 만을 가지고 있는 테스트용 모듈
- 테스트 슈트(Test Suites) : 테스트 대상 컴포넌트나 모듈 시스템에 사용되는 테스트 케이스의 집합
- 테스트 케이스(Test Case) : 사용자의 요구사항을 정확하게 준수했는지 확인하기 위한 입력 값, 실행 조건, 기대 결과 등으로 만들어진 테스트 항목의 명세서
- 테스트 스크립트(Test Script) : 자동화된 테스트 실행 절차에 대한 명세서
- 목 오브젝트(Mock Object) : 사전에 사용자의 행위를 조건부로 입력해 두면, 그 상황에 맞는 예정된 행위를 수행하는 객체

테스트 슈트와 테스트 시나리오의 차이
테스트 슈트와 테스트 시나리오는 둘 다 테스트 케이스의 묶음이다. 테스트 슈트가 여러 개의 테스트 케이스의 단순한 묶음이라면 테스트 시나리오는 테스트 케이스의 동작 순서에 따른 묶음입니다.

4. 애플리케이션 테스트 관리-SEC_08(테스트 자동화 도구)

6) 테스트 수행 단계별 테스트 자동화 도구

; 테스트 수행 단계를 테스트 계획, 테스트 분석/설계, 테스트 수행, 테스트 관리로 분류하였을 경우 각 단계에 해당하는 테스트 자동화 도구는 다음과 같다.

테스트 단계	자동화 도구	설명
테스트 계획	요구사항 관리	사용자의 요구사항 정의 및 변경 사항 등을 관리하는 도구
테스트 분석/ 설계	테스트 케이스 생성	테스트 기법에 따른 테스트 데이터 및 테스트 케이스 작성을 지원하는 도구
테스트 수행	테스트 자동화	테스트의 자동화를 도와주는 도구로 테스트의 효율성을 높임
	정적 분석	코딩 표준, 런타임 오류 등을 검증하는 도구
	동적 분석	대상 시스템의 시뮬레이션을 통해 오류를 검출하는 도구
	성능 테스트	가상의 사용자를 생성하여 시스템의 처리 능력을 측정하는 도구
	모니터링	CPU, Memory 등과 같은 시스템 자원의 상태 확인 및 분석을 지원하는 도구
테스트 관리	커버리지 분석	테스트 완료 후 테스트의 충분성 여부 검증을 지원하는 도구
	형상 관리	테스트 수행에 필요한 다양한 도구 및 데이터를 관리하는 도구
	결함 추적/관리	테스트 시 발생한 결함 추적 및 관리 활동을 지원하는 도구

4. 애플리케이션 테스트 관리-SEC_08(테스트 자동화 도구) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(테스트 자동화 도구)

1. 테스트 케이스 자동 생성 도구를 이용하여 테스트 데이터를 찾아내는 방법이 아닌 것은?

① 스텁(Stub)과 드라이버(Driver)

② 입력 도메인 분석

③ 랜덤(Random) 테스트

④ 자료 흐름도

스텁(Stub)과 드라이버(Driver)는 테스트 하네스 도구 요소에 속한다.

테스트 케이스 생성 도구(Test Case Generation Tools)

자료 흐름도 : 자료 원시 프로그램을 입력 받아 파싱한 후 자료 흐름도를 작성함

기능 테스트 : 주어진 기능을 구동시키는 모든 가능한 상태를 파악하여 이에 대한 입력을 작성함

입력 도메인 분석 : 원시 코드의 내부를 참조하지 않고, 입력 변수의 도메인을 분석하여 테스트 데이터를 작성함

랜덤 테스트 : 입력 값을 무작위로 추출하여 테스트함

2. 애플리케이션의 처리량, 응답 시간, 경과 시간, 자원 사용률에 대해

3. 단위 테스트에서 테스트의 대상이 되는 하위 모듈을 호출하고, 파라미터를 전달하는 가상의 모듈로 상향식 테스트에 필요한 것은?

① 테스트 스텁(Test Stub)

② 테스트 드라이버(Test Driver)

③ 테스트 슈트(Test Suites)

④ 테스트 케이스(Test Case)

테스트 하네스(Test Harness)의 구성 요소

1. 테스트 드라이버 : 테스트 대상의 하위 모듈을 호출하고, 파라미터(Parameter, 매개변수)를 전달하고, 모듈 테스트 수행 후의 결과를 도출하는 상향식 테스트에 사용되는 도구

2. 테스트 스텁 : 제어 모듈이 호출하는 타 모듈의 기능을 단순히 수행하는 도구로, 일시적으로 필요한 조건만을 가지고 있는 테스트용 모듈(dummy객체)이며 하향식 테스트에 사용되는 도구

3. 테스트 슈트 : 테스트 대상 컴포넌트나 모듈 시스템에 사용되는 테스트 케이스의 집합

4. 테스트 케이스 : 사용자의 요구사항을 정확하게 준수했는지 확인하기 위한 입력 값, 실행 조건, 기대 결과 등으로 만들어진 테스트 항목의 명세서

4. 애플리케이션 테스트 관리- SEC_08(테스트 자동화 도구) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(테스트 자동화 도구)

5. 다음 중 테스트 자동화에 대한 설명으로 가장 옳지 않은 것은?

① 테스트 자동화는 테스트 준비, 구현, 수행, 분석 등을 스크립트 형태로 구현한다.

② 테스트 자동화 도구를 이용하면 통계 작업과 그래프 등 다양한 표시 형태로 테스트 결과를 표시할 수 있다.

③ 테스트 엔지니어는 프로젝트를 완전히 이해한 후 테스트를 수행해야 하므로 프로젝트가 완료된 후 투입해야 한다.

④ 테스트 자동화 도구는 다중 플랫폼 호환성, 소프트웨어 구성, 기본 테스트 등 향상된 테스트 품질을 보장한다.

테스트 자동화 수행 시 고려사항

1. 테스트 절차를 고려하여 재사용 및 측정이 불가능한 테스트 프로그램은 제외한다.

2. 모든 테스트 과정을 자동화 할 수 있는 도구는 없으므로 용도에 맞는 적절한 도구를 선택해서 사용한다.

3. 자동화 도구의 환경 설정 및 습득 기간을 고려해서 프로젝트 일정을 계획해야 한다.

4. 테스트 엔지니어의 투입 시기가 늦어지면 프로젝트의 이해

4. 애플리케이션 테스트 관리-SEC_09(결함 관리)

1) 결함(Fault)의 정의

; 결함은 오류 발생, 작동 실패 등과 같이 소프트웨어가 개발자가 설계한 것과 다르게 동작하거나 다른 결과가 발생되는 것을 의미한다.

● 사용자가 예상한 결과와 실행 결과 간의 차이나 업무 내용과의 불일치 등으로 인해 변경이 필요한 부분도 모두 결함에 해당된다.

2) 결함 관리 프로세스

; 결함 관리 프로세스는 애플리케이션 테스트에서 발견된 결함을 처리하는 것으로, 처리 순서는 다음과 같다.

① **결함 관리 계획** : 전체 프로세스에 대한 결함 관리 일정, 인력, 업무 프로세스 등을 확보하여 계획을 수립하는 단계이다.

② **결함 기록** : 테스터는 발견된 결함을 결함 관리 DB에 등록한다.

③ **결함 검토** : 테스터, 프로그램 리더, 품질 관리(QA) 담당자 등은 등록된 결함을 검토하고 결함을 수정할 개발자에게 전달한다.

④ **결함 수정** : 개발자는 전달받은 결함을 수정한다.

⑤ **결함 재확인** : 테스터는 개발자가 수정한 내용을 확인하고 다시 테스트를 수행한다.

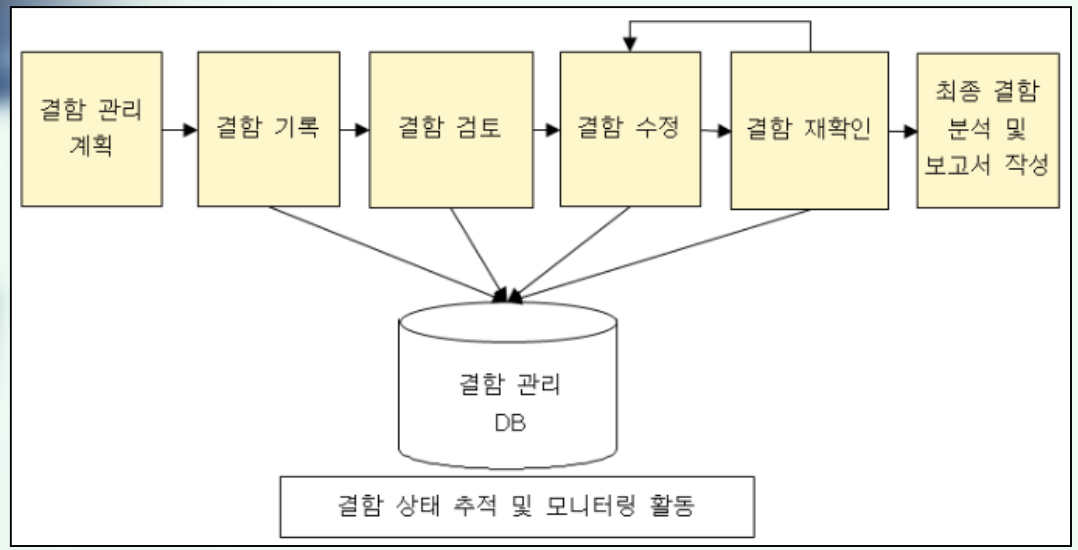
프로그램 리더 : 소프트웨어 설계, 구현 등 소프트웨어의 기술 분야를 책임지는 사람
품질 관리(QA) 담당자 : 제품에 대한 고객만족을 목표로 제품의 생산부터 판매, 폐기에 이르는 전 과정을 관리하는 사람

4. 애플리케이션 테스트 관리-SEC_09(결함 관리)

2) 결함 관리 프로세스

⑥ **결함 상태 추적 및 모니터링 활동** : 결함 관리 DB를 이용하여 프로젝트 별 결함 유형, 발생률 등을 한눈에 볼 수 있는 대시보드 또는 게시판 형태의 서비스를 제공한다.

⑦ **최종 결함분석 및 보고서 작성** : 발견된 결함에 대한 정보와 이해관계자들의 의견이 반영된 보고서를 작성하고 결함 관리를 종료한다.



SEC_06 '애플리케이션 테스트 프로세스'의 마지막 단계에도 결함 관리 프로세스가 있었는데 이번 섹션에도 결함 관리 프로세스가 있다. '두 결함 관리 프로세스' 모두 결함을 관리하는 것은 동일한데, 결함을 관리함에 있어 초점을 어디에 두느냐에 따라 순서가 조금 다르게 표현된 것 뿐이다. SEC_06 '애플리케이션 테스트 프로세스'의 결함 관리 프로세스'는 단계별 테스트 중 발생한 에러에 대해 이것이 결함 인지 아닌지 판별하는 것에 초점을 뒀다면 이번 섹션의 '결함 관리 프로세스'는 발견된 결함의 처리 과정에 초점을 둔 것이다. 두 경우를 구분하여 알아두도록 하자.
대시보드 : 다양한 데이터를 쉽게 모니터링 할 수 있도록 만든 일종의 상황판을 말한다.

4. 애플리케이션 테스트 관리-SEC_09(결함 관리)

3) 결함 상태 추적

; 테스트에서 발견된 결함은 지속적으로 상태 변화를 추적하고 관리해야 한다.

- 발견된 결함에 대해 결함 관리 측정 지표의 속성 값들을 분석하여 향후 결함이 발견될 모듈 또는 컴포넌트를 추정할 수 있다.
- 결함 관리 측정 지표

결함 분포	모듈 또는 컴포넌트의 특정 속성에 해당하는 결함 수 측정
결함 추세	테스트 진행 시간에 따른 결함 수의 추이 분석
결함 에이징	특정 결함 상태로 지속되는 시간 측정

에이징 : 아무리 우선순위가 낮은 대상이라도 그 대상을 기다리는 다른 대상이 있을 수 있으니 늦게라도 자원이 할당되어 처리되도록 해야 한다.
대기시간에 비례하여 우선순위를 부여함으로써 기아 현상을 방지한다.
기아 현상 : 어떤 우선순위 기준에 따라서 자원을 할당하는데 대상들이 계속 유입되는 상황에서 우선순위가 낮은 자료들은 영영 자원을 할당 받지 못하게 된다.

4. 애플리케이션 테스트 관리-SEC_09(결함 관리)

4) 결함 추적 순서

; 결함 추적은 결함이 발견된 때부터 결함이 해결될 때까지 전 과정을 추적하는 것으로 순서는 다음과 같다.

① 결함 등록(Open) : 테스터와 품질 관리(QA) 담당자에 의해 발견된 결함이 등록된 상태

② 결함 검토(Reviewed) : 등록된 결함을 테스터, 품질 관리(QA) 담당자, 프로그램 리더, 담당 모듈 개발자에 의해 검토된 상태

③ 결함 할당(Assigned) : 결함을 수정하기 위해 개발자와 문제 해결 담당자에게 결함이 할당된 상태

④ 결함 수정(Resolved) : 개발자가 결함 수정을 완료한 상태

⑤ 결함 조치 보류(Deferred) : 결함의 수정이 불가능해 연기된 상태로, 우선순위, 일정 등에 따라 재오픈을 준비중인 상태

⑥ 결함 종료(Closed) : 결함이 해결되어 테스터와 품질 관리(QA) 담당자가 종료를 승인한 상태

⑦ 결함 해제(Clarified) : 테스터, 프로그램 리더, 품질 관리(QA) 담당자가 종료 승인한 결함을 검토하여 결함이 아니라고 판명한 상태

4. 애플리케이션 테스트 관리-SEC_09(결함 관리)

5) 결함 분류

; 테스트에서 발견되는 결함을 유형별로 분류하면 다음과 같다.

시스템 결함	시스템 다운, 애플리케이션의 작동 정지, 종료, 응답 시간 지연, 데이터베이스 에러 등 주로 애플리케이션 환경이나 데이터베이스 처리에서 발생된 결함
기능 결함	사용자의 요구사항 미반영/불일치, 부정확한 비즈니스 프로세스, 스크립트 오류, 타 시스템 연동 시 오류 등 애플리케이션의 기획, 설계, 업무 시나리오 등의 단계에서 유입된 결함
GUI 결함	UI 비 일관성, 데이터 타입의 표시 오류, 부정확한 커서/메시지 오류 등 사용자 화면 설계 에서 발생된 결함
문서 결함	사용자의 요구사항과 기능 요구사항의 불일치로 인한 불완전한 상태의 문서, 사용자의 온라인/오프라인 매뉴얼의 불일치 등 기획자, 사용자, 개발자 간의 의사소통 및 기록이 원활하지 않아 발생된 결함

4. 애플리케이션 테스트 관리-SEC_09(결함 관리)

6) 테스트 단계별 유입 결함

- 기획 시 유입되는 결함 : 사용자 요구사항의 표준 미준수로 인한 테스트 불가능, 요구사항 불명확/불완전/불일치 결함 등
- 설계 시 유입되는 결함 : 설계 표준 미준수로 인한 테스트 불가능 기능 설계 불명확/불완전/불일치 결함 등
- 코딩 시 유입되는 결함 : 코딩 표준 미준수로 인한 기능의 불일치/불완전, 데이터 결함, 인터페이스 결함 등
- 테스트 부족으로 유입되는 결함 : 테스트 수행 시 테스트 완료 기준의 미준수, 테스트 팀과 개발팀의 의사소통 부족, 개발자의 코딩 실수로 인한 결함 등

4. 애플리케이션 테스트 관리-SEC_09(결함 관리)

7) 결함 심각도

; 결함 심각도는 애플리케이션에 발생한 결함이 전체 시스템에 미치는 치명도를 나타내는 척도이다.

● 결함 심각도를 우선순위에 따라 분류하면 다음과 같다.

High	핵심 요구사항 미구현, 장시간 시스템 응답 지연, 시스템 다운 등과 같이 더 이상 프로세스를 진행할 수 없도록 만드는 결함
Medium	부정확한 기능이나 데이터베이스 에러 등과 같이 시스템 흐름에 영향을 미치는 결함
Low	부정확한 GUI 및 메시지, 에러시 메시지 미출력, 화면상의 문법/철자 오류 등과 같이 시스템 흐름에는 영향을 미치지 않는 결함

8) 결함 우선순위

; 결함의 우선순위는 발견된 결함 처리에 대한 신속성을 나타내는 척도로, 결함의 중요도와 심각도에 따라 설정되고 수정 여부가 결정된다.

● 일반적으로 결함의 심각도가 높으면 우선순위도 높지만 애플리케이션의 특성에 따라 우선순위가 결정될 수도 있기 때문에 심각도가 높다고 반드시 우선순위가 높은 것은 아니다.

● 결함 우선순위는 결정적(Critical), 높음(High), 보통(Medium), 낮음(Low) 또는 즉시 해결, 주의 요망, 대기, 개선 권고 등으로 분류된다.

4. 애플리케이션 테스트 관리-SEC_09(결함 관리)

9) 결함 관리 도구

; 결함 관리 도구는 소프트웨어에 발생한 결함을 체계적으로 관리할 수 있도록 도와주는 도구로, 다음과 같은 것들이 있다.

Mantis	결함 및 이슈 관리 도구로 소프트웨어 설계 시 단위별 작업 내용을 기록할 수 있어 결함 추적도 가능하다.
Trac	결함 추적은 물론 결함을 통합하여 관리할 수 있는 도구
Redmine	프로젝트 관리 및 결함 추적이 가능한 도구
Bugzilla	결함 신고 확인, 처리 등 결함을 지속적으로 관리할 수 있는 도구로 결함의 심각도와 우선 순위를 지정할 수도 있다.

4. 애플리케이션 테스트 관리-SEC_09(결함 관리) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(결함 관리)

1. 소프트웨어 개발 활동을 수행함에 있어서 시스템이 고장(Failure)을 일으키게 하며, 오류(Error)가 있는 경우 발생하는 것은?

① Fault ② Testcase

③ Mistake ④ Inspection

결함은 오류 발생, 작동 실패 등과 같이 소프트웨어가 개발자가 설계한 것과 다르게 동작을 하거나 다른 결과가 발생되는 것을 의미한다.

사용자가 예상한 결과와 실행 결과 간에 차이나 업무 내용과의 불일치 등으로 인해 변경이 필요한 부분도 모두 결함에 해당한다.

인스펙션(Inspection) : 인스펙션은 워크스루를 발전시킨 형태로 소프트웨어 개발 단계에서 산출된 결과물의 품질을 평가하며 이를 개선하기 위한 방법 등을 제시하는 것이다.

2. 다음 중 결함에 관한 설명으로 틀린 것은?

① 결함은 사용자의 기대 결과와 실제 소프트웨어를 실행했을 때의 결과 간의 차이를 의미한다.

② 결함 심각도는 우선순위에 따라 High, Medium, Low로 분류하기도 한다.

3. 결함 관리 프로세스 중 "전체 프로세스에 대한 결함 관리 일정, 인력, 업무 프로세스 등을 확보하여 계획을 수립하는 단계"는 어떤 단계에 속하는가?

① 결함 기록

② 결함 관리 계획

③ 결함 수정

④ 결함 검토

결함 관리 프로세스(이미 발견된 결함 처리)

1. 결함 관리 계획 : 전체 프로세스에 대한 결함 관리 일정, 인력, 업무 프로세스 등을 확보하여 계획을 수립하는 단계

2. 결함 기록 : 테스터는 발견된 결함을 결함 관리 DB에 등록한다.

3. 결함 검토 : 테스터, PM, QA담당자 등은 등록된 결함을 검토하고 결함을 수정할 개발자에게 전달한다.

4. 결함 수정 : 개발자가 전달 받은 결함을 수정한다.

5. 결함 재확인 : 테스터는 개발자가 수정한 내용을 확인하고 다시 테스트를 수행한다.

6. 결함 상태 추적 및 모니터링 활동 : 결함 관리 DB를 이용하여 프로젝트 별 결함 유형, 발생률 등을 한눈에 볼 수 있는 대시보드 또는

4. 애플리케이션 테스트 관리-SEC_09(결함 관리) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(결함 관리)

5. 결함 심각도를 우선순위에 따라 분류할 때, "핵심 요구사항 미구현, 장시간 시스템 응답 지연, 시스템 다운 등과 같이 더 이상 프로세스를 진행할 수 없도록 만드는 결함"은 무엇인가?

① High ② Medium

③ Low ④ Minor

High : 핵심 요구사항 미구현, 장시간 시스템 응답 지연, 시스템 다운 등과 같이 더 이상 프로세스를 진행할 수 없도록 만드는 결함

Medium : 부정확한 기능이나 데이터 베이스 에러 등과 같이 시스템 흐름에 영향을 미치는 결함

Low : 부정확한 GUI 및 메시지, 에러시 메시지 미출력, 화면상의 문법/철자 오류 등과 같이 시스템 흐름에는 영향을 미치지 않는 결함

다른 기준으로도 분류할 수도 있는데 치명적(Critical), 주요(Major), 보통(Normal), 경미(Minor), 단순(Simple)으로도 나뉜다.

6. 결함 관리 도구의 종류로 틀린 것은?

① Case ② Trac

4. 애플리케이션 테스트 관리-SEC_10(애플리케이션 성능 분석)

1) 애플리케이션 성능

; 애플리케이션 성능이란 사용자가 요구한 기능을 최소한의 자원을 사용하여 최대한 많은 기능을 신속하게 처리하는 정도를 나타낸다.

● 애플리케이션 성능 측정 지표

처리량(Throughput)	일정 시간 내에 애플리케이션이 처리하는 일의 양
응답 시간(Response Time)	애플리케이션에 요청을 전달한 시간부터 응답이 도착할 때까지 걸린 시간
경과 시간(Turn Around Time)	애플리케이션에 작업을 의뢰한 시간부터 처리가 완료될 때까지 걸린 시간
자원 사용률(Resource Usage)	애플리케이션이 의뢰한 작업을 처리하는 동안의 CPU 사용량, 메모리 사용량, 네트워크 사용량 등 자원 사용률

● 애플리케이션의 성능 분석 도구는 애플리케이션의 성능을 테스트 하는 도구와 시스템을 모니터링 하는 도구로 분류된다.

4. 애플리케이션 테스트 관리-SEC_10(애플리케이션 성능 분석)

2) 성능 테스트 도구

; 성능 테스트 도구는 애플리케이션의 성능을 테스트하기 위해 애플리케이션에 부하나 스트레스를 가하면서 애플리케이션의 성능 측정 지표를 점검하는 도구이다.

● 종류

도구명	도구 설명	지원 환경
JMeter	HTTP, FTP 등 다양한 프로토콜을 지원하는 부하 테스트 도구	Cross-Platform
LoadUI	•서버 모니터링, Drag&Drop 등 사용자의 편리성이 강화된 부하 테스트 도구 •HTTP, JDBC(Java Database Connectivity)등 다양한 프로토콜 지원	Cross-Platform
OpenSTA	HTTP, HTTPS 프로토콜에 대한 부하 테스트 및 생산품 모니터링 도구	Windows

부하(Load) 테스트 : 애플리케이션에 일정 시간 동안 부하를 가하면서 반응을 측정하는 테스트
스트레스(Stress) 테스트 : 부하 테스트를 확장한 테스트로 애플리케이션이 과부하 상태에서 어떻게 작동하는지 테스트 함
Cross-Platform : 크로스 플랫폼(Cross Platform)은 "교차"를 뜻하는 "Cross"와 Platform의 합성어로, "다양한 플랫폼에서 사용할 수 있는"이라는 뜻을 가지고 있다.
HTTP(Hyper Text Transfer Protocol) : 인터넷에서 데이터를 주고받을 수 있는 프로토콜
HTTPS(Hyper Text Transfer Protocol Secure Socket) : 기본 골격이나 사용 목적 등은 HTTP와 거의 동일하지만, 데이터를 주고 받는 과정에 '보안'요소가 추가되었다는 것이 가장 큰 차이점이다. HTTPS를 사용하면 서버와 클라이언트 사이의 모든 통신 내용이 암호화 된다.

4. 애플리케이션 테스트 관리-SEC_10(애플리케이션 성능 분석)

3) 시스템 모니터링(Monitoring) 도구

; 시스템 모니터링 도구는 애플리케이션이 실행되었을 때 시스템 자원의 사용량을 확인하고 분석하는 도구이다.

● 시스템 모니터링 도구는 성능 저하의 원인 분석, 시스템 부하량 분석, 사용자 분석 등 시스템을 안정적으로 운영할 수 있는 기능을 제공한다.

● 종류

도구명	도구 설명	지원 환경
Scouter	•단일 뷰 통합/실시간 모니터링, 튜닝에 최적화된 인프라 통합 모니터링 도구 •애플리케이션의 성능을 모니터링/통제하는 도구	Cross-Platform
Zabbix	웹 기반 서버, 서비스, 애플리케이션 등의 모니터링 도구	Cross-Platform

4. 애플리케이션 테스트 관리-SEC_10(애플리케이션 성능 분석)

4) 애플리케이션 성능 저하 원인 분석

; 애플리케이션의 성능 저하 현상은 애플리케이션을 DB에 연결하기 위해 Connection객체를 생성하거나 쿼리를 실행하는 애플리케이션 로직에서 많이 발생한다.

● 다음은 애플리케이션의 성능 저하 현상을 발생시키는 주요 요인이다.

- DB에 필요 이상의 많은 데이터를 요청한 경우
- 데이터베이스의 락(DB Lock)이 해제되기를 기다리면서 애플리케이션이 대기하거나 타임 아웃된 경우
- 커넥션 풀(Connection Pool)의 크기를 너무 작거나 크게 설정한 경우
- JDBC나 ODBC 같은 미들웨어를 사용한 후 종료하지 않아 연결 누수(Connection Leak)가 발생한 경우
- 트랜잭션이 확정(Commit)되지 않고 커넥션 풀에 반환되거나, 잘못 작성된 코드로 인해 불필요한 Commit이 자주 발생하는 경우
- 인터넷 접속 불량으로 인해 서버 소켓(Server Socket)에 쓰기는 지속되나, 클라이언트에서 정상적인 읽기가 수행되지 않는 경우
- 대량의 파일을 업로드 하거나 다운로드 하여 처리 시간이 길어진 경우
- 트랜잭션 처리 중 외부 호출이 장시간 수행되거나 타임 아웃된 경우
- 네트워크 관련 장비 간 데이터 전송이 실패하거나 전송 지연으로 인해 데이터 손실이 발생한 경우

4. 애플리케이션 테스트 관리- SEC_10(애플리케이션 성능 분석) 출제 예상 문제

출제 예상 문제(애플리케이션 성능 분석)

1. 다음 중 애플리케이션의 성능을 측정하기 위한 지표가 아닌 것은?

① 신뢰도(Reliability)

② 처리량(Throughput)

③ 경과 시간(Turn Around Time)

④ 응답 시간(Response Time)

애플리케이션 성능 측정 지표

처리량(Throughput) : 일정 시간 내에 애플리케이션이 처리하는 양

응답 시간(Response Time) : 애플리케이션에 요청을 전달한 시간부터 응답이 올 때까지 걸린 시간

경과 시간(Turn Around Time) : 애플리케이션에 작업을 의뢰한 시간부터 처리가 완료될 때까지 걸린 시간

자원 사용률(Resource Usage) : 애플리케이션이 의뢰한 작업을 처리하는 동안의 CPU, 메모리, 네트워크 등 사용량 자원 사용률

2. 다음 중 애플리케이션의 성능을 저하하는 원인이 아닌 것은?

① DB에 필요 이상의 많은 데이터를 요청하면 애플리케이션의 성능 저하 현상이 발생할 수 있다.

3. 다음 중 애플리케이션의 성능을 측정하는 지표들에 대한 설명으로 틀린 것은?

① 처리량(Throughput) : 일정 시간 내에 애플리케이션이 처리하는 일의 양

② 응답 시간(Response Time) : 애플리케이션에 요청을 전달한 시간 부터 응답이 도착할 때까지 걸린 시간

③ 경과 시간(Turn Around Time): 애플리케이션이 작업을 처리하기 시작한 시간부터 처리가 완료될 때까지 걸린 시간

④ 자원 사용률(Resource Usage) : 애플리케이션이 의뢰한 작업을 처리하는 동안의 CPU, 메모리, 네트워크 등 의 사용량

경과 시간(Turn Around Time): 애플리케이션이 작업을 의뢰한 시간부터 처리가 완료될 때까지 걸린 시간

4. 소프트웨어 설계 시 구축된 플랫폼의 성능 특성 분석에 사용되는 측정 항목이 아닌 것은?

① 응답시간(Response Time)

② 가용성(Availability)

③ 사용률(Utilization)

④ 서버 튜닝(Server Tuning)

4. 애플리케이션 테스트 관리- SEC_10(애플리케이션 성능 분석) 출제 예상 문제

출제 예상 문제(애플리케이션 성능 분석)

5. 다음 중 성능 테스트 도구가 아닌 것은?

① Scouter

② JMeter

③ LoadUI

④ OpenSTA

성능 테스트 도구의 종류

JMeter : HTTP, FTP 등 다양한 프로토콜을 지원하는 부하 테스트 도구

LoadUI : 서버 모니터링, Drag&Drop 등 사용자 편리성이 강화된 부하 테스트 도구

OpenSTA : HTTP, HTTPS 프로토콜에 대한 부하 테스트 및 생산품 모니터링 도구

시스템 모니터링 도구의 종류

Scouter : 단일 뷰 통합/실시간 모니터링, 튜닝에 최적화된 인프라 통합 모니터링 도구, 애플리케이션의 성능을 모니터링/통제하는 도구

Zabbix : 웹 기반 서버, 서비스, 애플리케이션 모니터링 도구

6. 다음 중 HTTP와 HTTPS의 가장 큰 차이점은 무엇인가?

4. 애플리케이션 테스트 관리-SEC_11(복잡도)

1) 복잡도의 개요

; 복잡도(Complexity)는 시스템이나 시스템 구성 요소 또는 소프트웨어의 복잡한 정도를 나타나는 말로, 시스템 또는 소프트웨어를 어느 정도의 수준까지 테스트해야 하는지 또는 개발하는데 어느 정도의 자원이 소요되는지 예측하는데 사용된다.

- 시스템의 복잡도가 높으면 장애가 발생할 수 있으므로 정밀한 테스트를 통해 미리 오류를 제거할 필요가 있다.
- 주요 복잡도 측정 방법에는 LOC(Line Of Code), 순환 복잡도(Cyclomatic Complexity) 등이 있다.

LOC(Line Of Code) : 소프트웨어의 개별적인 기능에 대해 원시 코드 라인 수의 비관치, 낙관치, 기대치를 측정하여 예측치를 구하고 이를 이용하여 비용을 산정하는 기법이다.
순환 복잡도(Cyclomatic Complexity) : 한 프로그램의 논리적인 복잡도를 측정하기 위한 소프트웨어의 척도로, 맥케이브 순환도(McCabe's Cyclomatic) 또는 맥케이브 복잡도 메트릭(McCabe's Complexity Metrics)라고도 하며, 제어 흐름도 이론에 기초를 두는 기법이다.

4. 애플리케이션 테스트 관리-SEC_11(복잡도)

2) 시간 복잡도

; 시간 복잡도는 알고리즘의 실행시간, 즉 알고리즘을 수행하기 위해 프로세스가 수행하는 연산 횟수를 수치화한 것을 의미한다.

- 시간 복잡도가 낮을수록 알고리즘의 실행시간이 짧고, 높을수록 실행시간이 길어진다.
- 시간 복잡도는 알고리즘의 실행시간이 하드웨어적 성능이나 프로그래밍 언어의 종류에 따라 달라지기 때문에 시간이 아닌 명령어의 실행 횟수를 표기하는데, 이러한 표기법을 점근 표기법이라고 한다.
- 점근 표기법의 종류

빅오 표기법 (Big-O Notation)	•알고리즘의 실행시간이 최악일 때를 표기하는 방법이다. •입력 값에 대해 알고리즘을 수행했을 때 명령어의 실행 횟수는 어떠한 경우에도 표기 수치보다 많을 수 없다.
세타 표기법 (Big-θ Notation)	•알고리즘의 실행시간이 평균일 때를 표기하는 방법이다. •입력 값에 대해 알고리즘을 수행했을 때 명령어 실행 횟수의 평균적인 수치를 표기한다.
오메가 표기법 (Big-ω Notation)	•알고리즘의 실행시간이 최상일 때를 표기하는 방법이다. •입력 값에 대해 알고리즘을 수행했을 때 명령어의 실행 횟수는 어떠한 경우에도 표기 수치보다 적을 수 없다.

4. 애플리케이션 테스트 관리-SEC_11(복잡도)

3) 빅오 표기법(Big-O Notation)

; 빅오 표기법은 알고리즘의 실행시간이 최악일 때를 표기하는 방법으로, 신뢰성이 떨어지는 오메가 표기법이나 평가하기 까다로운 세타 표기법에 비해 성능을 예측하기 용이하여 주로 사용된다.

● 일반적인 알고리즘에 대한 최악의 시간 복잡도를 빅오 표기법으로 표현하면 다음과 같다.

O(1)	입력 값(N)이 증가해도 실행시간은 동일한 알고리즘, index로 접근하여 바로 처리할 수 있는 연산 과정의 시간 복잡도 → 기본 연산 수라고 생각하면 편함 예) 스택의 삽입(Push), 삭제(Pop)
O(log n)	연산이 한 번 실행될 때 마다 데이터의 크기가 절반 감소하는 알고리즘(log의 지수는 항상 2)예) 이진 트리(Binary Tree), 이진 검색(Binary Search)
O(n)	문제 해결에 필요한 단계가 입력 값(n)과 1:1의 관계를 가진다. 입력 값(n)이 증가함에 따라 실행시간도 선형적으로 증가하는 알고리즘 예) 1중 for문
O(n log n)	문제 해결에 필요한 단계가 $O(n\log_2 n)$번만큼 수행된다. 예)힙 정렬(Heap Sort), 2-Way 합병(병합) 정렬(Merge Sort)
$O(n^2)$	문제 해결에 필요한 단계가 입력 값(n)의 제곱만큼 수행된다. 예) 삽입 정렬(Insertion Sort), 쉘 정렬(Shell Sort), 선택 정렬(Selection Sort), 버블 정렬(Bubble Sort), 퀵 정렬(Quick Sort), 이중 for문
$O(2^n)$	문제 해결에 필요한 단계가 2의 입력 값(n) 제곱만큼 수행된다. 예) 피보나치 수열(Fibonacci Sequence)

● 왼쪽에서 오른쪽으로 갈수록 효율성이 떨어진다

$$O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(2^n)$$

4. 애플리케이션 테스트 관리-SEC_11(복잡도)

4) 순환 복잡도

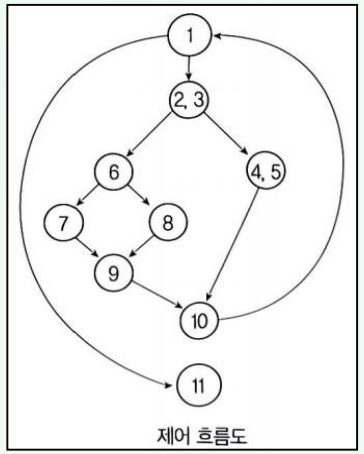
; 순환 복잡도(Cyclomatic Complexity)는 한 프로그램의 논리적인 복잡도를 측정하기 위한 소프트웨어의 척도로, 맥케이브 순환도(McCabe's Cyclomatic) 또는 맥케이브 복잡도 메트릭(McCabe's Complexity Metrics)라고도 하며, 제어 흐름도 이론에 기초를 둔다.

● 순환 복잡도를 이용하여 계산된 값은 프로그램의 독립적인 경로의 수를 정의하고, 모든 경로가 한 번 이상 수행되었음을 보장하기 위해 행해지는 테스트 횟수의 상한선을 제공한다.

● 제어 흐름도 G에서 순환 복잡도 V(G)는 다음과 같은 방법으로 계산할 수 있다.

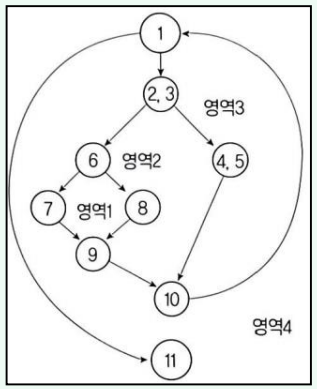
- 방법1) 순환 복잡도는 제어 흐름도의 영역 수와 일치하므로 영역 수를 계산한다.
- 방법2) V(G) = E - N + 2 : E는 화살표 수, N은 노드의 수

예제) 제어 흐름도가 다음과 같을 때 순환 복잡도(Cyclomatic Complexity)를 계산하시오.



제어 흐름도

방법1) 제어 흐름도에서 화살표로 구분되는 각 영역의 개수를 구하면 4이다.



방법2) 순환 복잡도 = 화살표의 수 - 노드의 수 + 2 이므로 11 - 9 + 2 = 4

4. 애플리케이션 테스트 관리-SEC_11(복잡도) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(복잡도)

1. 알고리즘 시간 복잡도 O(1)이 의미하는 것은?

① 컴퓨터 처리가 불가

② 알고리즘 입력 데이터 수가 한 개

③ 알고리즘 수행시간이 입력 데이터 수와 관계 없이 일정

④ 알고리즘 길이가 입력 데이터보다 작음

O(1)

입력 값(n)이 증가해도 실행시간은 동일한 알고리즘, index로 접근하여 바로 처리할 수 있는 연산 과정의 시간 복잡도 -> 기본 연산 수라고 생각하면 편함

예) 스택의 삽입(Push), 삭제(Pop)

O(log n)

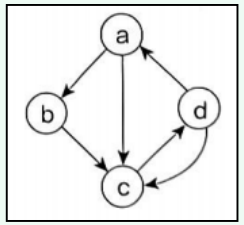
연산이 한 번 실행될 때 마다 데이터의 크기가 절반 감소하는 알고리즘(log의 지수는 항상 2)

예) 이진 트리(Binary Tree), 이진 탐색(Binary Search)

O(n)

문제 해결에 필요한 단계가 입력 값(n)과 1:1의 관계를 가진다. 입력 값(n)이 증가함에 따라 실행시간도 선형적으로 증가하는

3. 제어 흐름 그래프가 다음과 같을 때 McCabe의 Cyclomatic 수는 얼마인가?



① 3 ② 4

③ 5 ④ 6

E : 화살표의 개수, N의 노드의 개수

V(G) = E – N + 2 -> 6 - 4 + 2 = 4

4. n개의 원소를 정렬하는 방법 중 평균 수행시간 복잡도와 최악 수행시간 복잡도가 모두 O(n log n)인 정렬은?

① 삽입 정렬 ② 힙 정렬

③ 버블 정렬 ④ 선택 정렬

삽입, 버블, 선택 정렬은 평균과 최악 수행시간 복잡도가 모두 O(n의 제곱)이다.

4. 애플리케이션 테스트 관리-SEC_11(복잡도) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(복잡도)

5. 빅오 표기법에서 효율성이 최악인 것은?

① O(1)　　② O(n)

③ O(log n)　　④ $O(n^2)$

$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n)$

오른쪽으로 갈수록 알고리즘의 효율성이 떨어진다.

6. 시간 복잡도에 대한 설명으로 틀린 것은?

① 알고리즘을 수행하기 위해 프로세스가 수행하는 연산 횟수를 수치화한 것을 의미한다.

② 시간 복잡도가 높을수록 알고리즘의 실행시간이 짧고, 낮을수록 실행시간이 길어진다.

③ 알고리즘의 실행시간이 하드웨어적 성능이나 프로그래밍 언어의 종류에 따라 달라진다.

④ 시간이 아닌 명령어의 실행 횟수를 표기한다.

시간 복잡도는 알고리즘의 실행 시간, 즉 알고리즘을 수행하기 위해서 프로세스가 수행하는 연산 횟수를 수치화한 것을 의미한다.

시간 복잡도가 낮을수록 알고리즘의 실행시간이 짧고, 높을수록

4. 애플리케이션 테스트 관리-SEC_12(애플리케이션 성능 개선)

1) 소스코드 최적화

; 소스 코드 최적화는 나쁜 코드(Bad Code)를 배제하고, 클린 코드(Clean Code)로 작성하는 것이다.

- 클린 코드(Clean Code): 누구나 쉽게 이해하고 수정 및 추가할 수 있는 단순 명료한 코드, 즉 잘 작성된 코드를 의미한다.
- 나쁜 코드(Bad Code)
  - 프로그램의 로직(Logic)이 복잡하고 이해하기 어려운 코드로, 스파게티 코드와 외계인 코드가 여기에 해당한다.
  - 스파게티 코드(Spaghetti Code) : 코드의 로직이 서로 복잡하게 얽혀 있는 코드
  - 외계인 코드(Alien Code) : 아주 오래되거나 참고문서 또는 개발자가 없어 유지보수 작업이 어려운 코드

4. 애플리케이션 테스트 관리-SEC_12(애플리케이션 성능 개선)

1) 소스코드 최적화

● 클린 코드 작성 원칙

가독성	•누구든지 코드를 쉽게 읽을 수 있도록 작성한다. •코드 작성 시 이해하기 쉬운 용어를 사용하거나 들여쓰기 기능 등을 사용한다.
단순성	•코드를 간단하게 작성한다. •한 번에 한 가지를 처리하도록 코드를 작성하고 클래스/메소드/함수 등을 최소 단위로 분리한다.
의존성 배제	•코드가 다른 모듈에 미치는 영향을 최소화한다. •코드 변경 시 다른 부분에 영향이 없도록 작성한다.
중복성 최소화	•코드의 중복을 최소화한다. •중복된 코드는 삭제하고 공통된 코드를 사용한다.
추상화	상위 클래스/메소드/함수에서는 간략하게 애플리케이션의 특성을 나타내고, 상세 내용은 하위 클래스/메소드/함수에서 구현한다.

4. 애플리케이션 테스트 관리-SEC_12(애플리케이션 성능 개선)

2) 소스 코드 최적화 유형

- 클래스 분할 배치 : 하나의 클래스는 하나의 역할만 수행하도록 응집도를 높이고, 크기를 작게 작성한다.
- 느슨한 결합(Loosely Coupled) : 인터페이스 클래스를 이용하여 추상화된 자료 구조와 메소드를 구현함으로써 클래스 간의 의존성을 최소화한다.
- 코딩 형식 준수 : 코드 작성 시 다음의 형식을 준수한다.
  - 줄 바꿈 사용
  - 개념적 유사성이 높은 종속 함수 사용
  - 호출하는 함수는 선배치, 호출되는 함수는 후배치
  - 지역 변수는 각 함수의 맨 처음에 선언
- 좋은 이름 사용 : 변수나 함수 등의 이름은 기억하기 좋은 이름, 발음이 쉬운 용어, 접두어 사용 등 기본적인 이름 명명 규칙(Naming Rule)을 정의하고 규칙에 맞는 이름을 사용한다.
- 적절한 주석문 사용 : 소스 코드 작성 시 앞으로 해야 할 일을 기록하거나 중요한 코드를 강조할 때 주석문을 사용한다.

응집도 : 명령어나 호출문 등 모듈의 내부 요소들이 서로 관련되어 있는 정도, 즉 모듈이 독립적인 기능으로 정의되어 있는 정도를 의미한다.
인터페이스 클래스 : 클래스나 객체의 사용 방법을 정의한 것으로 개발 코드와 클래스 사이에서 통신 역할을 한다. 개발 코드가 클래스의 메소드를 직접 호출하지 않고 중간 매체인 인터페이스 클래스를 사용하는 이유는 개발 코드를 수정하지 않고 실행 내용이나 리턴 값을 다양하게 변경할 수 있기 때문이다. 이럴 경우 클래스를 직접 사용하지 않으므로 클래스 간 의존성이 줄어든다.
추상화는 불필요한 부분을 생략하고 객체의 속성 중 가장 중요한 것에만 중점을 두어 개략화하는 것, 즉 모델화 하는 것이다.

4. 애플리케이션 테스트 관리-SEC_12(애플리케이션 성능 개선)

3) 소스 코드 품질 분석 도구

; 소스 코드 품질 분석 도구는 소스 코드의 코딩 스타일, 코드에 설정된 코딩 표준, 코드의 복잡도, 코드에 존재하는 메모리 누수 현상, 스레드 결함 등을 발견하기 위해 사용하는 분석 도구로, 크게 정적 분석 도구와 동적 분석 도구로 나뉜다.

● 정적 분석 도구

- 작성한 소스 코드를 실행하지 않고 코딩 표준이나 코딩 스타일, 결함 등을 확인하는 코드 분석 도구이다.
- 비교적 애플리케이션 개발 초기의 결함을 찾는데 사용되고, 개발 완료 시점에서는 개발된 소스 코드의 품질을 검증하는 차원에서 사용된다.
- 자료 흐름이나 논리 흐름을 분석하여 비정상적인 패턴을 찾을 수 있다.
- 동적 분석 도구로는 발견하기 어려운 결함을 찾아내고, 소스 코드에서 코딩의 복잡도, 모델 의존성, 불일치성 등을 분석할 수 있다.
- 종류 : pmd, cppcheck, SonarQube, checkstyle, ccm, cobertura 등

● 동적 분석 도구

- 작성한 소스 코드를 실행하여 코드에 존재하는 메모리 누수, 스레드 결함 등을 분석하는 도구이다.
- 종류 : Avalanche, Valgrind 등

스레드 : 프로세스 내에서의 작업 단위로서 시스템의 여러 자원을 할당 받아 실행하는 프로그램의 단위를 의미한다.

4. 애플리케이션 테스트 관리-SEC_12(애플리케이션 성능 개선)

4) 소스 코드 품질 분석 도구의 종류

도구	설명	지원 환경
pmd	통상적으로 자바 소스 코드에 대한 미사용 변수, 최적화 되지 않은 코드 등 결함을 유발할 수 있는 코드를 검사한다.	Linux, Windows
cppcheck	C/C++ 코드에 대한 메모리 누수, 오버플로우 등 분석	Windows
SonarQube	중복 코드, 복잡도, 코딩 설계, 개발된 코드의 지속적인 인스펙션을 통해 품질 목표를 달성 등을 분석하는 소스 분석 통합 플랫폼	Cross-Platform
checkstyle	•자바 코드에 대해 소스 코드 표준을 따르고 있는지 검사한다. •다양한 개발 도구에 통합하여 사용 가능하다.	Cross-Platform
ccm	다양한 언어의 코드 복잡도를 분석한다.	Cross-Platform
cobertura	자바 언어의 소스 코드 복잡도 분석 및 테스트 커버리지를 측정한다.	Cross-Platform
Avalanche	•Valgrind 프레임 워크 및 STP 기반으로 구현된다. •프로그램에 대한 결함 및 취약점 등을 분석한다.	Linux, Android
Valgrind	프로그램 내에 존재하는 메모리 누수 및 스레드 결함 등을 분석한다.	Cross-Platform

4. 애플리케이션 테스트 관리-SEC_12(애플리케이션 성능 개선) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(애플리케이션 성능 개선)

1. 소스 코드품질 분석 도구 중 정적 분석 도구가 아닌 것은?

① pmd ② checkstyle

③ valMeter ④ cppcheck

정적 분석 도구

- 작성한 소스 코드를 실행하지 않고 코딩 표준이나 코딩 스타일, 결함 등을 확인하는 코드 분석 도구이다.
- 비교적 애플리케이션 개발 초기의 결함을 찾는데 사용되고, 개발 완료 시점에서는 개발된 소스 코드의 품질을 검증하는 차원에서 사용된다.
- 자료 흐름이나 논리 흐름을 분석하여 비정상적인 패턴을 찾을 수 있다.
- 동적 분석 도구로는 발견하기 어려운 결함을 찾아내고, 소스 코드에서 코딩의 복잡도, 모델 의존성, 불일치성 등을 분석할 수 있다.
- 종류 : pmd, cppckeck, SonarQube, checkstyle, ccm, cobertura 가 있다.

동적 분석 도구

3. 다음 중 클린 코드 작성 원칙으로 거리가 먼 것은?

① 누구든지 쉽게 이해하는 코드 작성

② 중복이 최대화된 코드 작성

③ 다른 모듈에 미치는 영향 최소화

④ 단순, 명료한 코드 작성

가독성

- 누구든지 코드를 쉽게 읽을 수 있도록 작성한다.
- 코드 작성 시 이해하기 쉬운 용어를 사용하거나 들여쓰기 기능 등을 사용한다.

단순성

- 코드를 간단하게 작성한다.
- 한 번에 한 가지를 처리하도록 코드를 작성하고 클래스/메서드/함수 등을 최소 단위로 분리한다.

의존성 배제

- 코드가 다른 모듈에 미치는 영향을 최소화 한다.
- 코드 변경 시 다른 부분에 영향이 없도록 작성한다.

중복성 최소화

- 코드의 중복을 최소화 한다.

4. 애플리케이션 테스트 관리- SEC_12(애플리케이션 성능 개선) 기출 및 출제 예상 문제

기출 및 출제 예상 문제(애플리케이션 성능 개선)

5. 소스 코드 정적 분석(Static Analysis)에 대한 설명으로 틀린 것은?

① 소스 코드를 실행시키지 않고 분석한다.

② 코드에 있는 오류나 잠재적인 오류를 찾아내기 위한 활동이다.

③ 하드웨어적인 방법으로만 코드 분석이 가능하다.

④ 자료 흐름이나 논리 흐름을 분석하여 비정상적인 패턴을 찾을 수 있다.

pmd, cppckeck 는 소프트웨어적인 방법으로 코드를 분석한다.

6. 클린 코드 작성원칙에 대한 설명으로 틀린 것은?

① 코드의 중복을 최소화 한다.

② 코드가 다른 모듈에 미치는 영향을 최대화하도록 작성한다.

③ 누구든지 코드를 쉽게 읽을 수 있도록 작성한다.

④ 간단하게 코드를 작성한다.

클린 코드는 의존성 배제 원칙에 따라 코드가 다른 모듈에 미치는 영향을 최소화 해야 한다.

7. 코드의 간결성을 유지하기 위해 사용되는 지침으로 틀린 것은?

① 공백을 이용하여 실행문 그룹과 주석을 명확히 구분한다.

② 복잡한 논리식과 산술식은 괄호와 들여쓰기(Indentation)를 통해 명확히 표현한다.

③ 빈 줄을 사용하여 선언부와 구현부를 구별한다.

④ 한 줄에 최대한 많은 문장을 코딩한다.

소스 코드는 가독성을 위해 줄 나눔과 들여쓰기, 괄호를 적절하게 사용해야 한다.

8. 아주 오래되거나 참고문서 또는 개발자가 없어 유지보수 작업이 아주 어려운 프로그램을 의미하는 것은?

① Title Code ② Source Code

③ Object Code ④ Alien Code

감사합니다.