애플리케이션 테스트 관리

상용 소프트웨어의 특성 및 유형

- 소프트웨어 유형
  - ✓ 산업 범용 소프트웨어
    - 시스템 소프트웨어: 운영체제, DBMS, 프로그래밍 언어, 가상화 도구, 스토리지 소프트웨어, 소프트웨어 공학 도구
    - 미들웨어: 웹 애플리케이션 서버, 실시간 데이터 처리, 네트워크 관리, 시스템 관리, 클라우드 서비스, 접근 제어 소프트웨어 등
    - 응용 소프트웨어
  - ✓ 산업 특화 소프트웨어: 특정한 산업 분야에서 요구하는 기능만을 구현하기 위한 목적의 소프트웨어
  - ✓ 서비스 제공 소프트웨어: 시스템 통합 소프트웨어
    - 신규 개발 소프트웨어
    - 기능 개선 소프트웨어
    - 추가 개발 소프트웨어
    - 시스템 통합 소프트웨어

애플리케이션 테스트

- 애플리케이션에 잠재되어 있는 결함을 찾아내는 일련의 행위 또는 절차
- 애플리케이션 테스트는 개발된 소프트웨어가 고객의 요구사항을 만족시키는지 확인(Validation)하고 소프트웨어가 기능을 정확히 수행하는지 검증(Verification)
- 애플리케이션 테스트의 기본 원리
  - ✓ 완벽한 테스트 불가능: 애플리케이션 테스트는 소프트웨어의 잠재적인 결함을 줄일 수 있지만 소프트웨어에 결함이 없다고 증명할 수는 없음. 즉 완벽한 소프트웨어 테스팅은 불가능
  - ✓ 결함 집중(Defect Clustering): 애플리케이션의 결함은 대부분 개발자의 특성이나 애플리케이션의 기능적 특징 때문에 특정 모듈에 집중되어 있으며 애플리케이션의 20%에 해당하는 코드에서 전체 결함의 80%가 발견된다고 하여 파레토 법칙(Pareto Principle)을 적용하기도 함
  - ✓ 살충제 패러독스(Pesticide Paradox) : 애플리케이션 테스트에서는 동일한 테스트 케이스로 동일한 테스트를 반복하면 더 이상 결함이 발견되지 않는 '살충제 패러독스(Pesticide Paradox)현상이 발생하며, 살충제 패러독스를 방지하기 위해서 테스트 케이스를 지속적으로 보완 및 개선해야 함
  - ✓ 테스팅은 정황(Context) 의존: 애플리케이션 테스트는 소프트웨어 특징, 테스트 환경, 테스터 역량 등 정황 (Context)에 따라 테스트 결과가 달라질 수 있으므로 정황에 따라 테스트를 다르게 수행해야 함
  - ✓ 오류-부재의 궤변(Absence of Errors Fallacy): 소프트웨어의 결함을 모두 제거해도 사용자의 요구사항을 만족시키지 못하면 해당 소프트웨어는 품질이 높다고 말할 수 없음을 의미
  - ✓ 테스트와 위험은 반비례
  - ✓ 테스트의 점진적 확대
  - ✓ 테스트는 개발 팀이 아닌 별도의 팀이 수행

애플리케이션 테스트

- 소프트웨어 테스트 프로세스
  - ✓ 테스트 계획: 프로젝트 계획서, 요구 명세서 등을 기반으로 테스트 목표를 정의하고 테스트 대상 및 범위를 결정함
  - ✓ 테스트 분석 및 디자인: 테스트의 목적과 원칙을 검토하고 사용자의 요구 사항을 분석함
  - ✓ 테스트 케이스 및 시나리오 시나 작성: 테스트 케이스의 설계 기법에 따라 테스트 케이스를 작성하고 검토 및 확인한 후 테스트 리오를 작성
  - ✓ 테스트 수행: 테스트 환경을 구축한 후 테스트를 수행
  - ✓ 테스트 결과 평가 및 리포팅: 테스트 결과를 비교 분석하여 테스트 결과서를 작성함
  - ✓ 결함 추적 및 관리: 테스트를 수행한 후 결함이 어디에서 발생 했는지, 어떤 종류의 결함인지 등 결함을 추 적하고 관리함

테스트 유형

- 프로그램 실행 여부
  - ✓ 정적 테스트: 프로그램을 실행하지 않고 명세서나 소스 코드를 대상으로 분석하는 테스트 방법으로 코드 검사, Walk Through, Inspection 등이 있음
  - ✓ 동적 테스트: 프로그램을 실행하여 오류를 찾는 테스트로 소프트웨어 개발의 모든 단계에서 테스트를 수행할 수 있으며 화이트박스 테스트와 블랙박스 테스트가 있음
- 시각에 따른 테스트
  - ✓ 검증(Verification) 테스트: 개발자의 시각에서 제품의 생산 과정을 테스트하는 것으로 제품이 명세서대로 완성 됐는지를 테스트함
  - ✓ 확인(Validation) 테스트: 사용자의 시각에서 생산된 제품의 결과를 테스트하는 것으로 사용자가 요구한대로 테스트 제품이 완성됐는지, 제품이 정상적으로 동작하는지를 테스트함
- 테스트 기반(Test Bases)에 따른 테스트
  - ✓ 명세 기반 테스트: 사용자의 요구사항에 대한 명세를 빠짐없이 테스트 케이스로 만들어 구현하고 있는지 확인하 는 테스트
  - ✓ 구조 기반 테스트: 소프트웨어 내부의 논리 흐름에 따라 테스트 케이스를 작성하고 확인하는 테스트
  - ✓ 경험 기반 테스트: 유사 소프트웨어나 기술 등에 대한 테스터의 경험을 기반으로 수행하는 테스트

테스트 유형

- 목적에 따른 테스트 유형
  - ✓ 회복(Recovery) 테스트: 시스템에 고의로 실패를 유도하고 시스템이 정상적으로 복귀하는지 테스트한다.
  - ✓ 안전(Security) 테스트: 불법적인 소프트웨어가 접근하여 시스템을 파괴하지 못하도록 소스 코드 내의 보안적 인 결함을 미리 점검하는 테스트이다.
  - ✓ 강도(Stress) 테스트: 시스템에 과다 정보량을 부과하여 과부하 시에도 시스템이 정상적으로 작동되는지를 검증하는 테스트이다.
  - ✓ 성능(Performance) 테스트: 사용자의 이벤트에 시스템이 응답하는 시간, 특정 시간 내에 처리하는 업무량, 사용자 요구에 시스템이 반응하는 속도 등을 테스트한다.
  - ✓ 구조(Structure) 테스트: 시스템의 내부 논리 경로, 소스코드의 복잡도를 평가하는 테스트이다.
  - ✓ 회귀(Regression) 테스트: 변경 또는 수정된 코드에 대하여 새로운 결함 발견 여부를 평가하는 테스트이다.
  - ✓ 병행(Parallel) 테스트: 변경된 시스템과 기존 시스템에 동일한 데이터를 입력 후 결과를 비교하는 테스트이다.

테스트 유형

- 기법에 따른 테스트 유형
  - ✓ White Box Test
    - 모듈의 원시 코드를 오픈 시킨 상태에서 원시 코드의 논리적인 모든 경로를 테스트하여 테스트 케이스를 설계 하는 방법
    - 원시 코드(Source Code)의 모든 문장을 한 번 이상 실행함으로써 수행
    - 모듈 안의 작동을 직접 관찰
    - 검증 기준: 테스트 케이스들이 테스트에 얼마나 적정한지를 판단하는 기준으로 문장 검증 기준(Statement Coverage), 분기 검증 기준(Branch Coverage), 조건 검증 기준(Condition Coverage), 분기/조건 기준(Branch/Condition Coverage) 등이 있음
    - 종류
      - 기초 경로 검사
      - 제어 구조 검사: 조건, 루프, 데이터 흐름 검사

테스트 유형

- 기법에 따른 테스트 유형
  - ✓ Black Box Test
    - 소프트웨어가 수행할 특정 기능을 알기 위해서 각 기능이 완전히 작동되는 것을 입증하는 테스트로 기능 테스트
    - 사용자의 요구 사항 명세를 보면서 테스트하는 것으로 주로 구현된 기능을 테스트
    - 소프트웨어 인터페이스에서 실시되는 테스트
    - 종류
      - 동치 분할 검사: 입력 자료에 초점을 맞춰 테스트 케이스를 만들어서 검사하는 기법으로 타당한 입력 자료와 타당하지 않은 입력 자료의 개수를 동등하게 하여 테스트
      - 경계 값 분석: 입력 자료에만 치중한 동치 분할 기법을 보완하기 위한 기법으로 입력 조건의 경계 값을 테스트 케이스로 선정하여 검사
      - 원인-효과 그래프 검사: 입력 데이터 간의 관계와 출력에 영향을 미치는 상황을 체계적으로 분석한 다음 효용성이 높은 테스트 케이스를 선정하여 검사하는 기법
      - 오류 예측 검사: 과거의 경험이나 확인하는 사람의 감각으로 테스트하는 기법
      - 비교 검사: 여러 버전의 프로그램에 동일한 테스트 자료를 제공하여 동일한 결과가 출력되는지 테스트하는 기법

테스트 유형

- 개발 단계에 따른 테스트 유형
  - ✓ 단위 테스트(Unit Test): 모듈 단위로 코딩 직후 소프트웨어 설계의 최소 단위인 모듈이나 컴포넌트에 초점을 맞춰 테스트
  - ✓ 통합 테스트(Integration Test): 단위 테스트가 완료된 모듈들을 결합하여 하나의 시스템으로 완성시키는 과정 에서의 테스트
  - ✓ 시스템 테스트(System Test): 개발된 소프트웨어가 해당 컴퓨터 시스템에서 완벽하게 수행되는 가를 점검하는 테스트
  - ✓ 인수 테스트(Acceptance Test)
    - 개발한 소프트웨어가 사용자의 요구사항을 충족하는지에 중점을 두고 테스트하는 것
    - 알파 테스트: 개발자의 장소에서 사용자가 개발자 앞에서 행하는 테스트 기법으로, 테스트는 통제된 환경 에서 행해지며 오류와 사용상의 문제점을 사용자와 개발자가 함께 확인하면서 기록함
    - 베타 테스트: 선정된 최종 사용자가 여러 명의 사용자 앞에서 행하는 테스트 기법으로 개발자에 의해 제어되지 않은 상태에서 테스트가 행해지며, 발견된 오류와 사용상의 문제점을 기록하고 개발자에게 주 기적으로 보고함
    - 사용자 인수 테스트
    - 운영상의 인수 테스트
    - 계약 인수 테스트
    - 규정 인수 테스트

테스트 유형

- 단위 테스트(Unit Test)
  - ✓ 코딩 직후 소프트웨어 설계의 최소 단위인 모듈이나 컴포넌트에 초점을 맞춰 테스트
  - ✓ 테스트에 필요한 입력 데이터, 테스트 조건, 예상 결과 등을 모아 테스트 케이스를 생성
  - ✓ 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 해당
  - ✓ 구조 기반 테스트: 프로그램 내부 구조 및 복잡도를 검증하는 화이트박스(White Box) 테스트 시행
  - ✓ 명세 기반 테스트: 목적 및 실행 코드 기반의 블랙박스(Black Box) 테스트 시행
  - ✓ 구성 요소
    - ❑ 식별자(Identifier): 항목 식별자, 일련번호
    - ❑ 테스트 항목(Test Item): 테스트 대상(모듈 또는 기능)
    - ❑ 입력 명세(Input Specification): 입력 데이터 또는 테스트 조건
    - ❑ 출력 명세(Output Specification): 테스트 케이스 수행 시 예상되는 출력 결과
    - ❑ 환경 설정(Environmental Needs): 필요한 하드웨어나 소프트웨어의 환경
    - ❑ 특수 절차 요구(Special Procedure Requirement): 테스트 케이스 수행 시 특별히 요구되는 절차
    - ❑ 의존성 기술(Inter-case Dependencies): 테스트 케이스 간의 의존성

통합 테스트

- 통합 테스트
  - ✓ 단위 테스트가 끝난 모듈을 통합하는 과정에서 발생하는 오류 및 결함을 찾는 테스트 기법
  - ✓ 통합 방식
    - ❑ 비점진적 통합 방식
      - ○ 단계적으로 통합하는 절차 없이 모든 모듈이 미리 결합되어 있는 프로그램 전체를 테스트
      - ○ 규모가 작은 소프트웨어에 유리하며 단시간 내에 테스트가 가능함
      - ○ 오류 발견 및 장애 위치 파악 및 수정이 어려움
      - ○ 종류
        - ▪ 빅뱅 통합 테스트 : 모듈 간의 상호 인터페이스를 고려 하지 않고 단위 테스트가 끝난 모듈을 한꺼번에 결합시켜 테스트하는 방법

통합 테스트

- ❖ 통합 테스트
  - ✓ 통합 방식
    - ❑ 점진적 통합 방식
      - ○ 모듈 단위로 단계적으로 통합하면서 테스트하는 방법
      - ○ 오류 수정이 용이하고 인터페이스와 연관된 오류를 완전히 테스트할 가능성이 높음
      - ○ 방법
        - ▪ 하향식 통합 테스트(Top Down Integration Test)
          - ➢ 프로그램의 상위 모듈에서 하위 모듈 방향으로 통합하면서 테스트하는 기법으로 깊이 우선 통합과 너비 우선 통합이 있음
          - ➢ 절차
            - • 주요 제어 모듈을 드라이버로 사용하고 주요 제어 모듈의 종속 모듈들은 스터브(Stub)로 대체
            - • 깊이 우선, 넓이 우선 방식에 따라 종속 스터브들이 실제 모듈로 교체
            - • 모듈이 통합될 때마다 검사 실시
            - • 새로운 오류가 생기지 않음을 보증하기 위해 회귀 검사 실시
          - ➢ 드라이버(Driver): 검사 자료 입출력 제어 프로그램
          - ➢ 스터브(Stub): 임시 제공되는 가짜 모듈

통합 테스트

- ❖ 통합 테스트
  - ✓ 통합 방식
    - ❑ 점진적 통합 방식
      - ○ 방법
        - ▪ 상향식 통합 테스트(Bottom Up Integration Test)
          - ➢ 프로그램의 하위 모듈에서 상위 모듈 방향으로 통합하면서 테스트하는 기법
          - ➢ 절차
            - 하위 모듈들을 클러스터(Cluster)로 결합
            - 검사 사례 입출력 조정을 위해 드라이버(Driver) 작성(제어 모듈이 없으므로)
            - 클러스터 검사
            - 드라이버 제거 후 클러스터는 프로그램 구조의 상위로 이동하여 결합
        - ▪ 혼합식 통합 테스트 : 하위 수준에서는 상향식 통합, 상위 수준에서는 하향식 통합을 사용하여 최적의 테스트를 지원하는 방식으로 샌드위치(Sandwich)식 통합 테스트 방법이라고도 함
        - ▪ 회귀 테스트(Regression Test): 이미 테스트된 프로그램 의 테스팅을 반복하는 것으로, 통합 테스트로 인해 변경된 모듈이나 컴포넌트에 새로운 오류가 있는지 확인 하는 테스트

Application Test Process

- Test Process 절차
  - ✓ 테스트 계획
  - ✓ 테스트 분석 및 디자인
  - ✓ 테스트 케이스 및 시나리오 작성
  - ✓ 테스트 수행
  - ✓ 테스트 결과 평가 및 리포팅
  - ✓ 결함 추적 및 관리
- 결함 관리 프로세스
  - ✓ 에러 발견
  - ✓ 에러 등록
  - ✓ 에러 분석
  - ✓ 결함 확정
  - ✓ 결함 할당
  - ✓ 결함 조치
  - ✓ 결함 조치 검토 및 승인

Test Case

- 구현된 소프트웨어가 사용자의 요구사항을 정확하게 준수 했는지를 확인하기 위해 설계된 입력 값, 실행 조건, 기대 결과 등으로 구성된 테스트 항목에 대한 명세서로, 명세 기반 테스트의 설계 산출물
- 테스트 시나리오
  - ✓ 테스트 케이스를 적용하는 순서에 따라 여러 개의 테스트 케이스들을 묶은 집합으로 테스트 케이스들을 적용하는 구체적인 절차를 명세한 문서
  - ✓ 테스트 순서에 대한 구체적인 절차, 사전 조건, 입력 데이터 등이 설정
  - ✓ 시스템 별, 모듈 별, 항목 별 등과 같이 여러 개의 시나리오로 분리하여 작성
  - ✓ 각각의 테스트 항목은 식별자 번호, 순서 번호, 테스트 데이터, 테스트 케이스, 예상 결과, 확인 등을 포함해서 작성
  - ✓ 테스트 시나리오는 유스케이스(Use Case) 간 업무 흐름이 정상적인지를 테스트할 수 있도록 작성해야 한다.

Test Oracle

- 테스트 오라클
  - ✓ 테스트 결과가 올바른지 판단하기 위해 사전에 정의된 참 값을 대입하여 비교하는 기법 및 활동
  - ✓ 특징
    - ❑ 제한된 검증: 모든 테스트에 사용할 수 없음
    - ❑ 수학적 기법: 수학적 기법을 이용하여 구할 수 있음
    - ❑ 자동화 기능: 테스트 대상 프로그램의 실행, 결과 비교, 커버리지 측정 등을 자동화 할 수 있음
  - ✓ 종류
    - ❑ 참(True) 오라클: 모든 테스트 케이스의 입력 값에 대해 기대하는 결과를 제공하는 오라클로 발생된 모든 오류를 검출할 수 있음
    - ❑ 샘플링(Sampling) 오라클: 특정한 몇몇 테스트 케이스의 입력 값들에 대해서만 기대하는 결과를 제공하는 오라클
    - ❑ 추정(Heuristic) 오라클: 샘플링 오라클을 개선한 오라클로 특정 테스트 케이스의 입력 값에 대해 기대하는 결과를 제공하고 나머지 입력 값들에 대해서는 추정으로 처리하는 오라클
    - ❑ 일관성 검사(Consistent) 오라클: 애플리케이션의 변경이 있을 때 테스트 케이스의 수행 전과 후의 결과 값이 동일한지를 확인하는 오라클

Test 자동화 도구

- 사람이 반복적으로 수행하던 테스트 절차를 스크립트 형태로 구현한 테스트 자동화 도구를 사용함으로써 휴먼 에러(Human Error)를 줄이고 테스트의 정확성을 유지하면서 테스트의 품질을 향상시킬 수 있음
  - ✓ 휴먼 에러(Human Error) : 사람의 판단 실수나 조작 실수 등으로 인해 발생하는 에러
- 정적 분석 도구(Static Analysis Tools): 프로그램을 실행 하지 않고 분석하는 도구로 소스 코드에 대한 코딩 표준, 코딩 스타일, 코드 복잡도 및 남은 결함 등을 발견 하기 위해 사용되는 도구
- 테스트 실행 도구(Test Execution Tools) : 스크립트 언어를 사용하여 테스트를 실행하는 방법으로 테스트 데이터와 테스트 수행 방법 등이 포함된 스크립트를 작성 한 후 실행하는 도구
  - ✓ 데이터 주도 접근 방식: 스프레드시트에 테스트 데이터를 저장하고 이를 읽어서 실행하는 방식
  - ✓ 키워드 주도 접근 방식: 스프레드시트에 테스트를 수행할 동작을 나타내는 키워드와 테스트 데이터를 저장하여 실행하는 방식
- 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 확인함
- 테스트 통제 도구(Test Control Tools): 테스트 계획 및 관리, 테스트 수행, 결함 관리 등을 수행하는 도구로, 종류에는 형상 관리 도구, 결함 추적/관리 도구 등이 있음

Test 자동화 도구

- 테스트 하네스 도구(Test Harness Tools)
  - ✓ 테스트가 실행 될 환경을 시뮬레이션하여 컴포넌트 및 모듈이 정상적으로 테스트 되도록 하는 도구
  - ✓ 테스트 하네스: 애플리케이션의 컴포넌트 및 모듈을 테스트하는 환경의 일부분으로 테스트를 지원하기 위해 생성된 코드와 데이터를 의미
  - ✓ 테스트 하네스의 구성 요소
    - ❑ Test Driver: 하위 모듈을 호출하고 파라미터를 전달하고 모듈 테스트 수행 후의 결과를 도출하는 도구
    - ❑ Test Stub: 제어 모듈이 호출하는 타 모듈의 기능을 단순히 수행하는 도구로일시적으로 필요한 조건만을 가지고 있는 테스트 용 모듈
    - ❑ Test Suites: 테스트 대상 컴포넌트나 모듈, 시스템에 사용되는 테스트 케이스의 집합
    - ❑ Test Case: 사용자의 요구 사항을 정확하게 준수했는지 확인하기 위한 입력 값, 실행 조건, 기대 결과 등으로 만들어진 테스트 항목의 명세서
    - ❑ Test Script: 자동화된 테스트 실행 절차에 대한 명세서
    - ❑ Mock Object: 사전에 사용자의 행위를 조건부로 입력해 두면 그 상황에 맞는 예정된 행위를 수행하는 객체

Test 자동화 도구

- 테스트 단계별 자동화 도구
  - ✓ 테스트 계획: 요구 사항 관리
  - ✓ 테스트 분석/설계: 테스트 케이스 생성
  - ✓ 테스트 수행
    - ❑ 테스트 자동화
    - ❑ 정적 분석
    - ❑ 동적 분석
    - ❑ 성능 테스트
    - ❑ 모니터링
  - ✓ 테스트 관리
    - ❑ 커버리지 분석
    - ❑ 형상 관리
    - ❑ 결함 추적/관리

결함 (Fault) 관리

- 결함은 오류 발생, 작동 실패 등과 같이 소프트웨어가 개발자가 설계한 것과 다르게 동작하거나 다른 결과가 발생되는 것
- 결함 관리 프로세스 처리 순서: 결함 관리 계획 → 결함 기록 → 결함 검토 → 결함 수정 → 결함 재확인 → 결 함 상태 추적 및 모니터링 활동 → 최종 결함 분석 및 보고서 작성
- 결함 관리 측정 지표
  - ✓ 결함 분포: 모듈 또는 컴포넌트의 특정 속성에 해당하는 결함 함수 측정
  - ✓ 결함 추세: 테스트 진행 시간에 따른 결함 함수의 추이 분석
  - ✓ 결함 에이징: 특정 결함 상태로 지속되는 시간 측정
- 결함 추적 순서: 결함 등록(Open) → 결함 검토 (Reviewed) → 결함 할당(Assigned) → 결함 수정(Resolved) → 결함 종료(Closed) → 결함 해제 (Clarified)
- Fixed(고정) : 개발자가 필요한 변경 작업을 수행하여 결함 수정 작업을 완료한 상태

결함 (Fault) 관리

- 결함 분류
  - ✓ 시스템 결함: 애플리케이션 환경이나 데이터베이스 처리에서 발생된 결함
  - ✓ 기능 결함: 애플리케이션의 기획, 설계, 업무 시나리오 등의 단계에서 유입된 결함
  - ✓ GUI 결함: 사용자 화면 설계에서 발생된 결함
  - ✓ 문서 결함: 기획자, 사용자, 개발자 간의 의사소통 및 기록이 원활하지 않아 발생된 결함
- 결함 심각도 의 분류
  - ✓ High: 더 이상 프로세스를 진행할 수 없도록 만드는 결함
  - ✓ Medium: 시스템 흐름에 영향을 미치는 결함
  - ✓ Low: 시스템 흐름에는 영향을 미치지 않는 결함
- 결함 우선순위 : 결정적(Critical), 높음(High), 보통 (Medium), 낮음(Low) 또는 즉시 해결, 주의 요망, 대기, 개선 권고 등

애플리케이션 성능 분석

- 사용자가 요구한 기능을 최소한의 자원을 사용하여 최대한 많은 기능을 신속하게 처리하는 정도
- 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 처리하 는 동안의 CPU 사용량, 메모리 사용량, 네트워크 사용량 등 자원 사용률
- 애플리케이션 성능 관리 도구(Application Performance Management Tool): 정보 기술 및 시스템 관리 분야에서 응용 프로그램 성능 관리는 소프트웨어 응용 프로그램의 성능 및 가용성에 대한 모니터링 및 관리를 위한 소프트웨어나 하드웨어
  - ✓ 성능 테스트 도구
    - ❑ JMeter
    - ❑ LoadUI
    - ❑ OpenSTA
  - ✓ 시스템 모니터링 도구
    - ❑ Scouter
    - ❑ Zabbix

복잡도

- ❖ 알고리즘 분석 – 복잡도
  - ✓ 시스템이나 시스템 구성 요소 또는 소프트웨어 품질의 복잡한 정도
    - 공간 복잡도
      - 알고리즘을 프로그램으로 실행하여 완료하기까지 필요한 총 저장 공간의 양
      - 공간 복잡도 = 고정 공간 + 가변 공간
    - 시간 복잡도
      - 알고리즘을 프로그램으로 실행하여 완료하기까지의 총 소요시간
      - 시간 복잡도 = 컴파일 시간 + 실행 시간
      - 컴파일 시간 : 프로그램마다 거의 고정적인 시간 소요
      - 실행 시간 : 컴퓨터의 성능에 따라 달라질 수 있으므로 실제 실행시간 보다는 명령문의 실행 빈도수에 따라 계산
      - 실행 빈도수의 계산
      - 지정문, 조건문, 반복문 내의 제어문과 반환문은 실행시간 차이가 거의 없으므로 하나의 단위 시간을 갖는 기본 명령문으로 취급
    - Macabe 의 순환 복잡도: 논리적인 복잡도를 측정하기 위한 방법으로 복잡도 = 노드의 수 – 간선의 수 + 2로 계산
    - 알고리즘의 성능 분석에 시간 복잡도가 공간 복잡도보다 더 중요한 평가 기준이기 때문에 알고리즘의 성능 분석은 대부분 시간 복잡도를 대상으로 함

복잡도

- ❖ 복잡도
  - ✓ 시간 복잡도
    - ○ 빅-오(O) 표기법
      - O(f(n))과 같이 표기, “Big Oh of f(n)”으로 읽음
      - 수학적 정의: 함수 f(n)과 g(n)이 주어졌을 때, 모든 $n \geq n0$에 대하여 $|f(n)| \leq c|g(n)|$ 을 만족하는 상수 c와 n0이 존재하면 $f(n) = O(g(n))$
      - 함수의 상한을 나타내기 위한 표기법: 최악의 경우에도 g(n)의 수행 시간 안에는 알고리즘 수행 완료 보장
      - 먼저 실행 빈도수를 구하여 실행 시간 함수를 찾고, 이 함수 값에 가장 큰 영향을 주는 n에 대한 항을 한 개 선택하여 계수는 생략하고 O의 오른쪽 괄호 안에 표시
      - 피보나치 수열에서 실행 시간 함수는 4n+2이고, 가장 영향이 큰 항은 4n인데 여기에서 계수 4를 생략하여 O(n)으로 표기

복잡도

- 복잡도
  - ✓ 시간 복잡도
    - ○ 빅-오메가 표기법
      - Ω(f(n))과 같이 표기, “Big Omega of f (n)”으로 읽음
      - 수학적 정의: 함수 f(n)과 g(n)이 주어졌을 때, 모든 n≥n0에 대하여 |f(n)| ≥ c |g(n)|을 만족하는 상수 c와 n0이 존재하면, f(n) = Ω(g(n))
      - 함수의 하한을 나타내기 위한 표기법: 어떤 알고리즘의 시간 복잡도가 Ω(g(n))으로 분석되었다면, 이 알고리즘 수행에는 적어도 g(n)의 수행 시간이 필요함을 의미

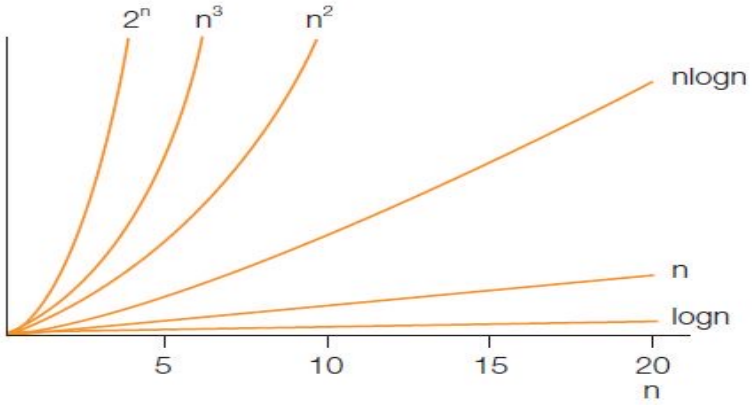
복잡도

- 복잡도
  - ✓ 시간 복잡도
    - ○ 빅-세타 표기법
      - Θ(f(n))과 같이 표기, “Big Theta of f (n)”으로 읽음
      - 수학적 정의: f(n)과 g(n)이 주어졌을 때, 모든 n≥n0에 대하여 c1|g(n)| ≤ f(n) ≤ c2 |g(n)| 을 만족하는 상수 c1, c2와 n0이 존재하면, f(n) = Θ(g(n))
      - 상한과 하한이 같은 정확한 차수를 표현하기 위한 표기법: f(n)= Θ (g(n))이 되려면 f (n)= O(g(n))이면서 f (n)= Ω(g(n))이어야 함

복잡도

- 복잡도
  - ✓ 시간 복잡도
    - ○ 각 실행 시간 함수에서 n값의 변화에 따른 실행 빈도수 비교

$\log n$ <	n <	$n\log n$ <	n^2 <	n^3 <	2^n
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	64	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	4294967296



복잡도

❖ 복잡도

✓ 시간 복잡도

- 시간 복잡도에 따른 알고리즘 수행 시간 비교

입력 크기 n	알고리즘 수행 시간				
	n	nlogn	n^2	n^3	2^n
10	10^{-8}초	3×10^{-8}초	10^{-7}초	10^{-6}초	10^{-6}초
30	3×10^{-8}초	2×10^{-7}초	9×10^{-7}초	3×10^{-5}초	1초
50	5×10^{-8}초	3×10^{-7}초	3×10^{-6}초	10^{-4}초	13일
100	10^{-7}초	7×10^{-7}초	10^{-5}초	10^{-3}초	4×10^{13}년
1,000	10^{-6}초	10^{-5}초	10^{-3}초	1초	3×10^{283}년
10,000	10^{-5}초	10^{-4}초	10^{-1}초	17분	
100,000	10^{-4}초	2×10^{-3}초	10초	12일	
1,000,000	10^{-3}초	2×10^{-2}초	17분	32년	

애플리케이션 성능 개선

- 소스 코드 최적화
  - ✓ 나쁜 코드(Bad Code)를 배제하고 클린 코드(Clean Code)로 작성하는 것
  - ✓ 클린 코드(Clean Code): 누구나 쉽게 이해하고 수정 및 추가할 수 있는 단순, 명료한 코드
  - ✓ 나쁜 코드(Bad Code): 코드의 로직이 서로 얽혀 있는 스파게티 코드 등 프로그램의 로직(Logic)이 복잡하고 이해하기 어려운 코드
  - ✓ 나쁜 코드로 작성된 애플리케이션의 코드를 클린 코드로 수정하면 애플리케이션의 성능이 개선됨
  - ✓ 외계인 코드: 너무 오래되서 설계 관련 정보가 없는 코드 – 역공학을 이용해서 구현된 코드로부터 설계 정보를 추출함
- 클린 코드 작성 원칙 : 가독성, 단순성, 의존성 배제, 중복성 최소화, 추상화
- 소스 코드 최적화 유형
  - ✓ 클래스 분할 배치 : 하나의 클래스는 하나의 역할만 수행하도록 응집도를 높이고, 크기를 작게 작성함
  - ✓ Loosely Coupled(느슨한 결합) : 인터페이스 클래스를 이용하여 추상화된 자료 구조와 메소드를 구현함으로써 클래스 간의 의존성을 최소화함
  - ✓ 코딩 형식 준수, 좋은 이름 사용, 적절한 주석문 사용

애플리케이션 성능 개선

- 소스 코드 품질 분석 도구
  - ✓ 코딩 스타일, 코딩 표준, 코드의 복잡도, 코드에 존재하는 메모리 누수, 스레드 결함들을 발견하기 위해 사용하는 도구로 실행을 하지 않고 확인하는 정적 분석 도구와 코드를 실행해서 확인하는 동적 분석 도구로 나눔
  - ✓ 정적 분석 도구
    - ❑ pmd: 미사용 변수 나 최적화 되지 않은 코드 처럼 결함을 유발할 수 있는 코드를 검사
    - ❑ cppcheck: C/C++ 코드에 대한 메모리 누수 나 오버플로우를 분석
    - ❑ SonarQube: 중복 코드, 복잡도, 코딩 설계 등을 분석하는 소스 분석 통합 플랫폼
    - ❑ checkstyle: 자바 코드의 소스 코드 표준 준수 여부를 검사해주는 라이브러리
    - ❑ ccm: 코드 복잡도 분석
    - ❑ cobertura: 자바의 소스 코드 복잡도 및 테스트 커버리지 측정
  - ✓ 동적 분석 도구
    - ❑ Avalanche: 프로그램에 대한 결함 및 취약점 등을 분석
    - ❑ Valgring: 메모리 및 쓰레드 결함 등을 분석함
    - ❑ Instruments: Xcode에서 성능 분석 도구