# SW 공학

하드웨어를 동작시켜 사용자가 작업을 편리하게 수행하도록하는 프로그램과 자료구조
프로그램 개발, 운용, 유지보수 관련 모든 문서 및 정보 포함

상품성 : 개발 SW는 상품화되어 판매된다.
견고성 : 일부 수정은 SW 전체 영향 가능
복잡성 : 개발과정이 복잡, 비표준화
순응성 : 사용자 요구나 환경 변화에 적절히 변경
비가시성 : SW 구조는 외관으로 나타나지 않고 코드로 Hidden
비마모성 : 마모되거나 소멸되지 않는다.
비제조성 : 하드웨어처럼 제작이 아닌 논리적 절차에 맞게 개발
비과학성 : 과학적이 아니라 조직, 인력, 시간, 절차 등 중심

# 분류

기능에 의한 분류 : 시스템, 응용
사용 분야에 의한 분류 : 프로그래밍, 문서, 통신, 분산처리, 멀티미디어, 개발, 인공지능
개발 과정 성격에 따른 분류 : 프로토타입, 프로젝트 산출물, 패키지
정보처리 방법에 따른 분류 : 일괄처리, 온라인, 실시간

# 시스템 구성요소

입력 : 처리 방법, 처리할 데이터, 조건을 시스템에 투입하는 것
처리 : 입력된 데이터를 처리 방법과 조건에 따라 처리하는 것
출력 : 처리된 결과를 시스템에서 산출하는 것
제어 : 자료를 입력하여 출력될 때까지의 처리 과정이 올바르게 진행되는지 감독하는 것
피드백 : 출력된 결과가 예정된 목표를 만족시키지 못할 경우 목표 달성을 위해 반복 처리 하는 것

# SW 위기

여러가지 원인 기반 개발 속도가 하드웨어의 개발속도를 따라가지 못해 SW에 대한 사용자들의 요구사항을 처리할 수 없는 문제가 발생함을 의미
SW의 특징에 대한 이해 부족 : 물리적이지 않고 논리적인 SW 특징을 이해하지 못함
SW의 관리 부재 : SW에 대한 관리를 소홀히 하여 효율적인 자원 통제가 이루어지지 못했다.
프로그래밍에만 치중 : SW 품질이나 유지보수는 고려하지 않고, 프로그래밍만 하려하므로 다양하고 복잡해지는 SW의 요구사항을 처리하지 못함
개발 인력 부족과 그로 인한 인건비 상승
성능 및 신뢰성 부족
개발 기간 지연 및 개발 비용 증가
유지보수가 어려워져 비용 증가
SW 생산성 저하
SW 품질 저하

# SW 공학

SW의 위기를 극복하기 위한 방안으로 연구된 학문
SW의 품질과 생산성 향상을 목적
IEEE 정의 : SW의 개발, 운용, 유지보수, 폐기 처분에 대한 체계적인 접근 방안
Fairley 정의 : 지정된 비용과 기간 내의 SW를 체계적으로 생산하고 유지보수하는 데 관련된 기술적이고 관리적인 원리
Boehm 정의 : 과학적인 지식을 SW 설계와 제작에 응용하는 것이며 이를 개발 운용, 유지보수하는 데 필요한 문서 작성 과정
제품을 단지 생산하는 것이 아니라 가장 경제적인 방법으로 양질의 제품을 생산하는 것
계층화 기술을 사용한다.

# 계층화 기술

**도구, 방법, 절차**

- 도구 : Tool 절차와 방법을 자동 또는 반자동으로 처리하는 기능을 제공, 대표적으로 CASE를 사용
- 방법 : Method SW를 구축하는 기술적인 방법을 제공
- 절차 : Process
  - SW 개발에 사용되는 개발 방법과 도구가 사용되는 순서
  - 계층화 기술들을 결합시켜 합리적이고 적절한 방법으로 SW를 개발하고 유지

# 기본 원칙

- 현대적 프로그래밍 기술을 계속적 적용
- 개발된 SW 품질 유지위해 지속적 검증
- SW 개발 관련 사항 및 결과의 명확한 기록 유지

# 발전 과정

- 1960 : SW 공학의 시작, 구조적 프로그래밍
- 1970 : 구조적 분석/설계 개념 도입, 상품화
- 1980 : 하드웨어 가격 하락
- 1985~ : 객체지향 기술 사용, CASE 등의 활용, 재공학

# 품질과 생산성

## 품질과 생산성

# 품질

- 사용자 요구 동작 구현
- 하드웨어 자원 효율적 이용
- 일정 시간 내, 주어진 조건 하, 원하는 기능 실행
- 처리 절차 기반 정확한 결과 산출
- SW 개발, 유지보수 등 초기 예상 비용 이내 수행
- 편리한 사용을 위한 적당한 사용자 인터페이스 제공
- 유지보수 용이, 높은 신뢰성 제공
- 에러 최소화
- SW 사용법, 구조의 설명, 성능, 기능 이해용이성 필요
- 빠른 실행 속도, 적은 기억 용량 사용

# 생산성

투입된 비용, 노력에 대한 생산량

- 개발자 능력
- 원활한 의사 소통
- 프로젝트 복잡도와 성격
- 기술 수준
- 관리 기술

# 생명 주기

- SW 수명 주기
- SW 개발 방법론의 바탕
- SW를 개발하기 위해 정의하고 운용, 유지보수 등의 과정을 단계별로 나눈 것
- 프로젝트 비용 산정과 개발 계획을 수립할 수 있는 기본 골격
- 프롲게트 진행 방향을 명확하게 파악
- 용어 및 기술의 표준화를 가능하게 한다.
- 프로젝트 관리를 용이하게 한다.
- 여러 SW 간 상호 일관성을 유지하게 한다.

# SW 단계

정의 단계, 개발단계, 유지보수 단계

# 정의 단계

- SW를 개발할 것인지 정의하는 단계
- 관리자와 사용자가 가장 많이 참여하는 단계
- 타당성 검토단계 › 개발 계획단계 › 요구사항 분석 단계

# 개발 단계

## 실제적 SW 개발 단계

- 설계 단계 : 구조, 알고리즘, 자료구조 작성 단계, 에러가 가장 많이 발생
- 구현 단계 : 설계 단계 문서를 기초로 코딩, 번역하는 단계
- 테스트 단계 : 구현 SW에 내재된 오류 발견 단계

# 유지보수 단계

SW를 적응 및 유지시키는 단계
SW 생명 주기 단계 중에서 시간과 비용이 가장 많이 든다.
정의 : 개발 계획, 요구사항 분석
설계 : 구조, 알고리즘
구현 : 코딩
테스트 : 오류 검출

# 생명 주기 모형

# 폭포수 모형

- SW 개발이 각 단계를 확실히 매듭짓고 그 결과를 철저히 검토하여 승인한 뒤 다음 단계로 진행
- 이전 단계로 넘어갈 수 없는 방식
- 가장 오래되고 가장 폭넓게 사용된 전통적인 SW 생명 주기 모형
- 고전적 생명 주기 모형
- 선형 순차적 모형: 앞 단계 완료 후 다음 단계 진행
- 제품의 일부가 될 매뉴얼을 작성
- 타당성 검토 => 계획 => 요구 분석 => 설계 => 구현(코딩) => 테스트(검사) => 유지보수
- 모형 적용 경험과 성공 사례가 많다.
- 단계별 정의가 분명하고 전체 공조의 이해가 용이

# 프로토타입 모형

- 사용자의 요구사항을 정확하게 파악하기 위해 프로토타입(견본품)을 만들어 최종 결과물을 예측하는 모형
- 시제품은 사용자와 시스템 사이의 인터페이스에 중점을 두어 개발
- 폭포수 모형 단점 보완
- 프로토타입은 요구 분석 단계에서 사용
- SW 생명주기에서 유지보수가 없어지고, 개발 단계 안에서 유지 보수가 이뤄지는 것으로 볼 수 있다.
- 요구 수집 => 빠른 설계 => 프로토타입 구축 => 고객 평가 => 프로토타입 조정 => 구현
- 요구사항을 충실히 반영하며 요구사항 변경 용이
- 최종 결과물이 만들어지기 전에 의뢰자가 최종 결과물의 일부 또는 모형 제공

# 나선형 모형

- Boehm 제안, 폭포수와 프로토타입 모형의 장점에 위험 분석 기능을 추가한 모형
- 반복적인 개발 과정을 거쳐 점진적으로 완벽한 최종 SW 개발
- 점진적 모형
- SW 개발 시 발생하는 위험을 관리하고 최소화 목적
- 계획 및 정의 => 위험 분석 => 공학적 개발 => 고객평가의 반복
- Planning => Risk Analysis => Engineering => Customer Evalutation
- 가장 현실적 모형
- 대규모 프로젝트나 큰 시스템에 적합
- 개발 과정이 반복되므로 누락되거나 추가된 요구사항을 추가 가능, 정밀하며 유

# 4GT 모형

- 사용자와 개발자 쉽게 접근, 4G 언어, 요구사항 명세서로부터 원시코드 자동생성 모형
- 설계 단계 축소, 요구 분석단계에서 코딩단계로 전환 가능한 비절차적 모형
- 요구사항 수집 => 설계 전략 => 4GL 구현 => 제품화
- 중소형 SW 개발 시간 절감, 대규모에는 자동화로 인해 분석/설계 단계에 많은 시간 필요

# 프로젝트 관리

주어진 기간 내에 최소의 비용으로 사용자를 만족시키는 시스템을 개발하기 위한 활동
SW 개발 계획을 세우고 분석, 설계, 구현 등 작업을 통제하는 것
SW 생명 주기의 전 과정에 걸쳐 진행된다.

# 프로젝트 관리 대상

- 계획관리 : 프로젝트 계획, 비용산정, 일정 계획, 조직 계획
- 품질관리
- 위험관리

# 프로젝트 3대 요소

- People : 프로젝트 관리에 가장 기본인 인적자원
- Problem : 사용자 관점 문제 분석 및 인식
- Process : SW 개발에 필요한 전체적 작업 계획 및 구조

# 프로젝트 구성 단계

- 프로젝트 계획 수립
- 프로젝트 가동
- 프로젝트 통제
- 프로젝트 종료

# 프로젝트 계획 수립

- 프로젝트 수행 전 SW 개발 영역 결정, 필요한 자원, 비용, 일정 등을 예측하는 작업
- 관리자가 합리적으로 예측할 수 있도록 프레임워크 제공
- SW 개발 과정 내 발생 가능 위험성 최소화
- 계획 수립 후 시스템 정의서와 프로젝트 계획서 산출
- 프로젝트 관리자 임무
- SW 개발 영역 결정
  - 프로젝트 계획 수립의 첫 번째 업무
  - 개발될 SW의 영역을 결정
  - 주요 요소 : 처리될 데이터, SW에 대한 기능, 성능, 제약 조건, 신뢰도, 인터페이스 등

# SW 프로젝트 추산

비용을 예측하는 작업
가장 어렵고 오차 발생이 심한 작업

- 프로젝트 비용 결정 요소
  - 프로젝트 요소
    - 제품의 복잡도
    - 시스템의 크기
    - 요구되는 신뢰도 : 일정한 기간 내에 주어진 조건 하에서 필요한 기능을 수행하는 정도
  - 자원 요소
    - 인적 자원 : 관리자, 개발자의 자질

# 비용 산정 기법

# 하향식

- 전문가 감정 기법
  - 경험이 많은 두 명 이상의 전문가에게 비용 산정 의뢰
  - 개인적이고 주관적
  - 편리하고 신속하게 비용 산정
  - 의뢰자에게 신뢰를 얻을 수 있음
- 델파이 기법
  - 전문가 감정 기법의 주관적 편견을 보완하기 위함
  - 많은 전문가의 의견을 종합해 선정하는 방법
  - 한 명의 조정자와 여러 명의 전문가

# 상향식

프로젝트의 세부적인 작업 단위별로 비용을 산정한 후 집계하여 전체 비용을 산정하는 방법

- LOC 기법
  - 원시 코드 라인수 기법
  - SW 각 기능의 원시 코드 라인 수와 비관치, 낙관치, 기대치를 측정하여 예측치를 구하고 이를 이용하여 비용을 산정하는 기법
  - 측정이 용이하고 이해하기 쉬워 가장 많이 사용된다.
  - 예측치 = (낙관치 + (4 X 기대치) + 비관치) / 6 = (a + 4m + b) / 6
  - ManMonth = 개발 기간 X 투입 인원 = LOC / 1인당 월평균 코딩량
  - 개발 비용 = ManMonth X 1인 인건비
  - 개발 기간 = ManMonth / 투입 인원

# 수학적 산정 기법

```
상향식 비용 산정 기법
경험적 추정 모형 = 실험적 추정 모형
COCOMO, Putnam, FP 모형
```

- COCOMO 모형
  - COnstructive COst MOdel
  - Boehm이 제안
  - 원시 프로그램 규모인 LOC에 의한 비용 산정 기법
  - 개발할 SW의 규모를 예측한 후 SW의 종류에 따라 다르게 책정되는 비용 산정 방정식에 대입하여 비용을 구한다.
  - 비용 견적의 강도 분석 및 유연성이 높아 널리 사용된다.

# 프로젝트 일정 계획

- 프로젝트 프로세스를 이루는 소작업의 순서와 일정을 정하는 것
- SW 개발 기간의 지연을 방지하고 프로젝트가 계획대로 진행되도록 일정을 계획
- 계획된 일정은 프로젝트의 진행을 관리하는데 기초 자료가 된다.
- 계획된 일정과 프로젝트의 진행도를 비교하여 차질이 있을 경우 조정할 수 있다.
- WBS, PERT/CPM, 간트 차트가 사용된다.

# 사람-노력 관계

- 소규모 개발 프로젝트에서는 한 사람이 요구사항을 분석하고 설계, 코딩, 테스트까지 수행할 수 있다.
- 프로젝트의 크기가 증가할수록 더 많은 사람들이 참여해야 한다.
- Brooks의 법칙 : 프로젝트 중 새로운 인력을 투입할 경우 작업 적응기간과 부작용으로 인해 일정이 더 지연된다.

# 노력 분배

- 노력을 개발과정에 분배할 때는 40-20-40 규칙을 권장한다.
- 분석 설계에 40, 코딩에 20, 테스트에 40

# WBS

- Work Breakdown Structure = 업무 분류 구조
- 개발 프롲게트를 여러 개의 작은 관리 단위로 분할하여 계층적으로 기술한 업무 구조

# PERT/CPM

- 프로젝트 지연을 방지하고 계획대로 진행되게 하기 위한 일정을 계획하는 것
- 초단시간 내 계획 완성을 위한 프로젝트 일정 방법
- 프로젝트 개발 기간을 결정하는 임계 경로를 제공한다.
- 통계적 모델을 적용해 개별 작업에 대한 가장 근접한 시간을 측정하는 기준이 된다.
- 각 작업에 대한 시작 시간을 정의하여 작업들 간의 경계 시간을 계산할 수 있게 한다.
- 가장 빠른 완료시간, 가장 늦은 완료시간, 총 자유시간을 구할 수 있다.

# PERT

- Program Evaluation and Review Technique = 프로그램 평가 및 검토 기술
- 프로젝트에 필요한 전체 작업의 상호 관계를 표시하는 네트워크
- 낙관적인 경우, 가능성이 있는경우, 비관적인 경우로 나누어 각 단계별 종료 시기를 결정하는 방법
- 과거에 경험이 없어 소요 기간 예측이 어려운 SW에서 사용
- 노드와 간선으로 구성되며 원 노드에는 작업을 화살표 간선에는 낙관치, 기대치, 비관치를 표시한다.
- 결정 경로, 작업에 대한 경계시간, 작업 간의 상호관련성을 알 수 있다.
- 작업 예측치 = (비관치 + (4 X 기대치) + 낙관치) / 6

# CPM

- Critical Path Method = 임계 경로 기법
- 프로젝트 완성에 필요한 작업을 나열하고 작업에 필요한 소요 기간을 예측하는 데 사용하는 기법
- 노드와 간선으로 구성된 네트워크로 노드는 작업을, 간선은 작업사이의 전후 의존 관계를 나타낸다.
- 원형 노드는 작업과 소요기간을 표시하고, 박스 노드는 이정표를 의미하며 예상 완료 시간을 표시한다.
- 전 작업이 완료된 후 다음 작업을 진행할 수 있다.
- 각 작업의 순서와 의존관계, 작업의 동시성을 한 눈에 볼 수 있다.
- 프로젝트 규모 추정 => 단계별 필요작업 분할 => 작업의 상호 의존 관계를 CPM 네트워크로 나타냄 => 일정 계획을 간트 차트로 나타냄

# Gantt Chart

- 간트 차트 = 시간선 = Time Line
- 프로젝트의 각 작업들이 언제 시작하고 언제 종료되는지에 대한 작업 일정을 막대 도표를 이용하여 표시하는 프로젝트 일정표
- 중간 목표 미달성시 그 이유와 기간을 예측 가능
- 사용자와의 문제점이나 예산의 초과 지출 등을 관리
- 자원 배치와 인원 계획에 사용 가능
- 다양한 형태로 변경 가능
- 작업 경로를 표시할 수 없다.
- 계획의 변화에 대한 적응성이 약하다.
- 계획 수립 또는 수정 때 주관적 수치에 기울어지기 쉽다.

# 프로젝트 조직 구성 계획

# 분산형 팀 구성

- 팀원 모두가 의사 결정에 참여하는 비이기적인 구성 방식
- 민주주의식 팀 구성
- 팀 구성원의 참여도와 만족도를 높이고 이직률을 낮게 한다.
- 팀 구성원 각자가 서로의 일을 검토하고 다른 구성원이 일한 결과에 대해 같은 그룹의 일원으로 책임을 진다.
- 여러 사람의 의사를 교류하므로 복잡하고 이해되지 않는 문제가 많은 장기 프로젝트 개발에 적합
- 링 모양의 구조를 가진다.
- 팀 구성 방법 중 가장 많은 의사 소통 경로를 갖는다.

# 중앙 집중형 팀 구성

- 관리자가 의사 결정을 하고 그 결정에 따르는 구성 방식
- 책임 프로그래머 팀 구성
- 프로젝트 수행에 따른 모든 권한과 책임을 한 명의 관리자에게 위임하고 기술 및 관리 지원을 위해 인력을 투입하는 형태
- 소규모 프로젝트에 적합
- 프로젝트의 성공은 책임 프로그래머의 능력에 달렸다.
- 책임 프로그래머에 따라 의사 결정이 이뤄지기 때문에 의사 결정이 빠르고 의사 교환 경로를 줄일 수 있다.
- 책임 프로그래머 : 요구 분석 및 설계, 기술적 판단, 프로그래머 작업 지시 및 배분
- 프로그래머 : 책임 프로그래머 지시에 따른 코딩, 테스트, 디버깅, 문서 작성

# 계층적 팀 구성

- 분산형과 중앙 집중형을 혼합한 형태로 혼합형 팀 구성
- 초급 프로그래머를 작은 그룹으로 만들어 각 그룹을 고급 프로그래머가 관리
- 경험자와 초보자를 구별
- 기술 인력이 관리를 담당하게 되어 좋은 기술력을 사장시킬 수 있고, 기술 인력이 업무 관리 능력을 갖춰야한다.

# SW 품질 보증

# 품질 표준

SW의 운영적인 특성, SW의 변경 수용능력, 새로운 환경에 대한 SW의 적응 능력에 따라 분류

- 운영특성
  - 정확성 : Correctness 사용자의 요구 기능 충족
  - 신뢰성 : Reliability 요구된 기능을 오류 없이 수행하는 정도
  - 효율성 : Efficiency SW가 자원을 쓸데없이 낭비하지 않아야한다.
  - 무결성 : Integrity 허용되지 않는 사용이나 자료의 변경을 제어하는 정도
  - 사용 용이성 : Usability SW는 적절한 UI와 문서를 가지고 있어야한다.
- 변경 수용 능력
  - 유지보수성 : Maintainability 변경 및 오류 사항 교정에 대한 노력을 최소

# 품질 보증

- SQA = Software Quality Assurance
- 어떠한 SW가 이미 설정된 요구사항과 일치하는가를 확인하는 데 필요한 개발 단계 전체에 걸친 계획적이고 체계적인 작업
- SW 개발 초기에 SW 특성과 요구사항을 철저히 파악하여 품질 목표를 설정하고, 개발 단계에서는 정형 기술 검토를 통해 품질 목표의 충족 여부를 점검하며, 개발 후에는 디버깅과 시험 과정을 거친다.

# 정형 기술 검토

- FTR = Formal Technical Review
- 가장 일반적인 검토 방법으로 SW 기술자들에 의해 수행되는 SW 품질 보증 활동
- 검토회의, 검열 등이 있으며 회의 형태로 수행된다.
- 검토중인 SW가 해당 요구사항과 일치하는지를 검증
- 미리 정해진 표준에 따라 표현되는지를 확인
- 기능과 로직에 오류가 있는지 확인
- SW가 균일한 방식으로 개발되도록 한다.
- 프로젝트를 용이하게 관리하도록 한다.
- 정형 기술 검토 지침사항

# SW 신뢰성과 가용성

신뢰성 : 프로그램이 주어진 환경에서 주어진 시간동안 오류 없이 작동할 확률로 측정과 예측이 가능하다.
가용성 : 한 프로그램이 주어진 시점에서 요구사항에 따라 운영되는 확률

# 측정

신뢰성 측정은 MTBF를 이용한다.

- MTBF
  - Mean Time Between Failure
  - 평균 고장 간격
  - 수리가 가능한 시스템이 고장난 후부터 다음 고장이 날 때까지의 평균 시간
  - MTBF = MTTF + MTTR
- MTTF
  - Mean Time To Failure
  - 평균 가동 시간 = 고장 평균 시간

# 위험 관리

Risk Analysis
프로젝트 추진 과정에서 예상되는 각종 돌발 상황을 미리 예상하고 대책을 수립하는 활동
위험은 불확실성과 손실을 가지고 있는데, 위험관리로 대비한다.
**위험 식별 => 위험 분석 및 평가 => 위험 관리 계획 => 위험 감시 및 조치**

# 위험범주

- 프로젝트 위험 : Project Risk
- 기술 위험 : Technical Risk
- 비즈니스 위험 : Business Risk

# 위험종류

- 인력 부족
- 예산 관리
- 일정 관리
- 사용자 요구사항 변경 : 대표적 위험 요소

# Charette가 제안한 종류

- 알려진 위험 : 프로젝트 계획서, 기술적 환경, 정보 등에 의해 발견 될 수 있는 위험
- 예측 가능한 위험 : 과거의 경험으로 예측 가능한 위험
- 예측 불가능한 위험 : 사전 예측이 매우 어려운 위험

# 위험 분석 및 평가

- 프로젝트에 내재한 위험 요소를 인식하고 그 영향을 분석하는 활동
- 위험 추산(Risk Estimation) 작업을 통해 수행된다.
- 가능한 모든 위험 요소와 영향을 분석하여 의사결정에 반영
- 위험표(Risk Table)을 작성하여 활용한다.
  - 위험표
    - 위험 내용
    - 위험 범주
    - 발생 확률
    - 영향력
    - 위험 감시 및 조치

# 위험 감시 및 조치

- 위험 회피 : Risk Avoidance 예상하고 회피
- 위험 감시 : Risk Monitoring 위험 요소 징후에 대해 계속적으로 인지하는 것
- 위험 관리 : Risk Management
- 비상 계획 수립 : Contingency Plan 위험 회피 전략이 실패할 경우 위험에 대해 관리하고 대비책과 비상 계획을 수립한다.

# SW 형상 관리

- SCM = Software Configuration Management
- SW 변경 사항을 관리하기 위해 개발된 일련의 활동
- SW 변경의 원인을 알아내고 제어하며 적절이 변경되고 있는지 확인하여 담당자에게 통보하는 작업
- 형상 관리는 SW 개발의 전 단계에 적용되는 활동
- 유지보수 단계에서도 수행
- 형상 관리는 SW 개발의 전체 비용을 줄인다.
- 개발 과정의 여러 방해 요인을 최소화시킨다.
- 형상은 SW 각 개발 단계의 결과물

# 형상 항목

- SCI = Software Configuration Item
- 스시템 명세서
- SW 프로젝트 계획서
- SW 요구사항 명세와 실행가능한 프로토타입
- 예비 사용자 매뉴얼
- 설계 명세서
- 원시 코드 목록
- 테스트 계획, 절차, 시험 사례, 결과
- 운영과 설치에 필요한 매뉴얼
- 실행 프로그램

# 관리 기능

- 형상 식별 : 대상에게 이름과 관리 번호를 부여하고 계층(트리)구조로 구분
- 버전 제어 : 다른 버전과의 형상 항목을 관리하려 특정 절차와 도구를 결합시키는 작업
- 변경 제어 : 형상 항목의 변경 요구를 검토해 현재의 기준선이 잘 반영될 수 있도록 조정
- 형상 감사 : 기준선의 무결성을 평가
- 형상 상태 보고

# 전통적 SW 개발 방법

고전적 SW 개발 방법,  구조적 SW 개발 방법
과거의 많은 SW 개발 경험을 토대로하여 성공적으로 평가되는 SW 분석 및 설계 방법들을 집대성하여 하나의 개발 방법으로 정형화한 것

# 요구사항 분석

- SW 개발의 첫 단계
- 개발 대상에 대한 사용자의 요구사항을 이해하고 문서화하는 활동
- 사용자 요구의 타당성을 조사하고 비용과 일정에 대한 제약 설정
- 사용자 요구를 정확하게 추출하여 목표를 정하고 어떤 방식으로 해결할 것인지 결정
- 요구사항 분석을 통한 결과는 SW 설계단계에서 필요한 자료가 된다.
- 사용자의 요구사항을 정확하고 일관성있게 분석하여 문서화
- SW 분석가에 의해 요구사항 분석이 수행

# 요구사항 분석작업

- 문제 인식 : 사용자 면담, 설문조사 및 협조, 문서 검토
- 평가와 종합 : 요구사항에 대한 정보를 평가하고 해결책 종합
- 모델 제작 : 내용을 이해하기 쉽도록 모델로 작성
- 문서화와 검토 : 요구사항 분석 명세서 작성

# 요구사항 분석의 어려움

- 대화 장벽 : 다이어그램 및 프로토타입 이용
- 시스템의 복잡도 : 구조적 분석이나 객체지향 분석 이용
- 요구의 변경 : 수정 요구와 상반된 요구들의 수용 기술 필요
- 요구 명세화의 어려움 : 제도적인 요구 분석 기술 필요

# 분석가의 자질

- 개발 경험이 많아야한다.
- 사용자의 요구를 정확히 수용하고 환경을 이해해야한다.
- 설계에 필요한 자료를 충분히 제공
- 시간 배정과 계획을 빠른 시간내에 파악
- 하드웨어 SW를 포함한 컴퓨터 기술에 대한 이해
- 고객 관점에서의 문제 파악

# 구조적 분석 기법

- 자료의 흐름과 처리를 중심으로 하는 요구사항 분석 방법
- 도형 중심의 도구를 사용하므로 분석가와 사용자 간의 대화가 용이
- 하향식 방법을 사용해 시스템을 세분화하고 분석의 중복을 배제할 수 있다.
- 자료흐름도, 자료사전, 소단위 명세서, 개체 관계도, 상태 전이도, 제어 명세서

# 구조적 분석 도구

- 자료 흐름도
  - DFD = Data Flow Diagram = 자료 흐름 그래프 = 버블 차트
  - 자료의 흐름 및 변환 과정과 기능을 도형 중심으로 기술하는 방법
  - 시스템 안의 프로세스와 자료 저장소 사이의 자료 흐름을 나타내는 그래프
  - 자료흐름과 처리를 중심으로하는 구조적 분석 기법
  - 자료 흐름과 기능을 자세히 표현하기 위해 단계적으로 세분화된다.
  - 자료는 처리(프로세스)를 거쳐 변환될 때마다 새로운 이름이 부여된다.
  - 처리는 입력 자료가 발생하면 기능을 수행한 후 출력 자료를 산출한다.
  - 자료 흐름도를 세분화 할 수 록 SW 설계와 구현작업이 용이해진다.
  - 자료 흐름도 구성요소

# 설계

- 구조적 분석 기법의 결과물인 자료 흐름도 등으로 SW 기능과 프로그램 구조, 모듈 설계 전략, 평가 지침, 문서화 도구를 제공하는 체계화된 기법
- 자료 흐름 중심 설계 기법
- 자료 흐름도, 자료 사전, 개체 관계도, 소단위 명세서가 준비된 이후에 설계
- 설계 모형
  - 데이터 설계, 구조 설계, 인터페이스 설계, 프로시저 설계로 구성된다.
  - SW 품질 평가를 위한 지침
  - 데이터 설계
    - Data Design
    - 요구사항 분석 단계에서 생성된 정보를 SW를 구현하는데 필요한 자료 구

# 구현

- 설계 단계에서 생성된 설계 명세서를 컴퓨터가 알 수 있는 모습으로 변환하는 과정
- 프로그래밍 = 코딩
- 각 모듈을 프로그래밍 언어를 사용해 원시 코드로 작성하고 문서화하는 작업
- 설계를 철저히 반영시키고 원시 코드를 간단 명료하게 작성한다.
- 사용할 프로그램이 언어와 코딩 스타일 등을 결정해야한다.
- 프로그래밍 언어
  - 1세대 : 기계어, 어셈블리어
  - 2세대 : FORTRAN, ALGOL, COBOL, BASIC

# 테스트(Test)

- 검사 사례 설계 고려사항
- 모듈 내의 모든 독립적인 경로가 적어도 한 번은 수행되어야 한다.
- 가능한 복잡한 논리는 배제한다.
- 임의의 조건을 만족시켜야 한다.
- 내부 자료 구조를 사용하여 테스트를 수행한다.
- 화이트 박스 테스트
  - 모듈의 원시 코드를 오픈시킨 상태에서 원시 코드의 논리적인 모든 경로를 검사하여 검사 사례를 설계하는 방법
  - 설계된 절차에 초점을 둔 구조적 테스트
  - 프로시저 설계의 제어 구조를 사용하여 검사 사례를 설계한다.

# 유지보수

- SW 개발 단계 중 가장 많은 노력과 비용이 투입되는 단계
- 시험 용이성, 이해성, 수정 용이성, 이식성이 고려되어야 한다.
- 수리 보수, 적응 보수, 완전화 보수, 예방 보수 활동으로 구분된다.

# 수정 보수

- Corrective
- 시스템을 운영하면서 검사 단계에서 발견핟지 못한 잠재적인 오류를 찾아 수정하는 활동
- 오류의 수정과 진단을 포함한다.

# 적응 보수

- Adaptive = 환경 적응 = 조정 보수
- SW 수명 기간 중 발생하는 환경의 변화(하드웨어, OS)를 기존 SW에 반영하기 위해 수행하는 활동
- 프로그래밍 환경의 변화 또는 주변장치, 시스템 요소의 업그레이드시 대처할 수 있는 유지보수 활동

# 완전화 보수

- Perfective = 기능 개선 = 기능 보수
- SW 본래 기능에 새로운 기능을 추가하거나 성능을 개선하기 위해 SW를 확장시키는 활동
- 유지보수 활동 중 가장 큰 업무 및 비용을 차지한다.

# 예방 보수

- Preventive = SW 재공학
- SW의 오류 발생에 대비하여 미리 예방 수단을 강구하는 활동

# 유지보수 과정

- 유지보수 요구
- 현 시스템에 대한 이해
- 수정 및 시험

# 유지보수 비용

- SW 개발에 필요한 비중 중 약 70%
- Belady, Lehman에 의해 제안된 공식으로 구한다.
- https://i.imgur.com/ml3uWS2.png

# 유지보수 부작용

- 코딩 부작용 : 코딩 내용 변경으로 발생
- 자료 부작용 : 자료나 자료 구조의 변경으로 발생
- 문서화 부작용 : 자료 코드에 대한 변경이 설계문서나 사용자 매뉴얼에 반영되지 않을 때 발생

# 외계인 코드

- Alien Code
- 아주 오래 전에 개발되어 유지보수 작업이 매우 어려운 프로그램
- 일반적으로 15년이 더 된 프로그램
- 문서화로 방지할 수 있다.

# 객체지향 SW 공학

- 현실 세계의 개체를 기계의 부품처럼 하나의 객체로 만들어 부품을 조립하여 제품을 만들 듯이 SW를 개발할 때에도 객체들을 조립해서 작성할 수 있도록 하는 기법
- 구조적 기법의 문제점 해결
- SW 재사용 및 확장을 용이하게 해서 빠르게 개발이 가능하고 유지보수가 쉽다.
- 복잡한 구조를 단계적, 계층적으로 표현한다.
- 멀티미디어 데이터 및 병령 처리를 지원한다.
- 현실 세계를 모형화하여 사용자와 개발자가 쉽게 이해할 수 있다.

# 객체

- Object
- 데이터와 데이터를 처리하는 함수를 묶어 놓은 하나의 SW 모듈
- 데이터
  - 객체가 가지고 있는 정보로 속성이나 상태, 분류를 나타낸다.
  - 속성 = Attribute = 상태 = 변수 = 상수 = 자료 구조
- 함수
  - 객체가 수행하는 기능으로 객체가 갖는 데이터를 처리하는 알고리즘
  - 객체의 상태를 참조하거나 변경하는 수단이 되는 것
  - 메소드 = 서비스 = 동작 = 연산
  - 기존 구조적 기법에서의 함수, 프로시저에 해당하는 연산 기능

# 클래스

- 공통된 속성과 연산을 갖는 객체의 집합
- 객체의 일반적인 타입을 의미
- 각각의 객체들이 갖는 속성과 연산을 정의하고 있는 틀
- 인스턴스 : 클래스에 속한 각각의 객체
- 인스턴스화 : 클래스로부터 새로운 객체를 생성하는 것
- 최상위 클래스는 상위 클래스를 갖지 않는 유일한 클래스
- 슈퍼클래스는 특정 클래스의 상위 클래스
- 서브클래스는 특정 클래스의 하위 클래스

# 메세지

- 객체들 간에 상호작용을 하는 데 사용되는 수단
- 객체에게 어떤 행위를 하도록 지시하는 명령 또는 요구사항
- 메세지를 받은 수신 객체는 요구된 메소드를 수행한다.

# 객체지향 기법의 기본 원칙

캡슐화, 정보은닉, 추상화, 상속성, 다형성

- 캡슐화
  - Encapsulation
  - 데이터와 데이터를 처리하는 함수를 하나로 묶는 것
  - 캡슐화된 객체의 세부내용이 외부에 은폐된다.
  - 캡슐화된 객체는 재사용이 용이하다.
  - 객체 간의 메세지를 주고받을 때 각 객체의 세부 내용은 알 필요가 없으므로 인터페이스가 단순해지고 객체 간의 결합도가 낮아진다.

- 정보 은닉
  - Information Hiding
  - 캡슐화에서 가장 중요한 개념
  - 다른 객체에게 자신의 정보를 숨기고 자신의 연산만을 통하여 접근을 허용하는 것
  - 각 객체의 수정이 다른 객체에 주는 영향을 최소화하는 기술
  - 유지보수와 SW 확장시 오류를 최소화

- 추상화
  - Abstraction
  - 불필요한 부분을 생략하고 객체의 속성 중 가장 중요한 것에만 중점을 두어 모델화하는 것

- 상속성
  - Inheritance
  - 이미 정의된 상위 클래스의 모든 속성과 연산을 하위 클래스가 물려받는 것
  - 다중 상속성 : Multiple Inheritance 한 개의 클래스가 두 개 이상의 상위 클래스로부터 속성과 연산을 상속받는 것

- 다형성
  - Polymorphism
  - 객체들은 동일한 메소드명을 사용하며 같은 의미의 응답을 한다.
  - 응용 프로그램 상에서 하나의 함수나 연산자가 두 개 이상의 서로 다른 클래스의 인스턴스들을 같은 클래스에 속한 인스턴스처럼 수행할 수 있도록하는 것

# 객체지향 기법의 생명주기

- 각 과정이 명확하게 순찾거으로 이루어지지는 않는다.
- 계획 및 분석
- 설계
- 구현
- 테스트 및 검증

# 객체지향 분석

- OOA = Object Oriented Analysis
- 사용자의 요구사항을 분석하여 요구된 문제와 관련된 모든 클래스, 연관된 속성과 연산, 관계 등을 정의하여 모델링 하는 작업
- SW를 개발하기 위한 비즈니스를 객체와 속성, 클래스와 멤버, 전체와 부분 등으로 나눠서 분석한다.
- 분석가에게 주요한 모델링 구성요소인 클래스, 속성, 연선달을 표현해서 문제를 모형화할 수 있게 해준다.
- 객체는 클래스로부터 인스턴스화되고, 클래스를 식별하는 것이 객체지향 분석의 주요한 목적이다.

# 객체지향 분석 방법론

Rumbaugh 방법, Booch 방법, jacobson 방법, Coad와 Yourdon 방법, Wirfs-Brock 방법,

- Rumbaugh 방법
  - 럼바우 방법
  - 가장 일반적으로 사용되는 방법
  - 분석 활동을 객체 모델, 동적 모델, 기능 모델로 나누어 수행

- Booch 방법
  - 부치 방법
  - 미시적Micro 개발 프로세스와 거시적Macro 개발 프로세스 모두를 사용하는 분석 방법
  - 클래스와 객체들을 분석 및 식별하고 클래스의 속성과 연산을 정의한다.

- jacobson 방법
  - Use Case를 강조하여 사용하는 분석 방법이다.

- Coad와 Yourdon 방법
  - E-R 다이어그램을 사용하여 객체의 행위를 모델링한다.
- Wirfs-Brock 방법
  - 분석과 설계 간의 구분이 없고 고객 명세서를 평가해서 설계 작업까지 연속적으로 수행하는 기법

# Rumbaugh 분석 기법

- 모든 SW 구성요소를 그래픽 표기법을 이용하여 모델링하는 기법
- 객체 모델링 기법 = OMT = Object Modeling Technique
- 객체 모델링, 동적 모델링, 기능 모델링

- 객체 모델링
  - Object Modeling = 정보 모델링
  - 관계를 규정하여 객체 다이어그램으로 표시하는 것
  - 분석 활동의 모델 중 가장 중요하며 선행되어야 할 모델링
  - 순서
    - 객체와 클래스를 식별
    - 클래스에 대한 자료 사전 작성
    - 클래스 간의 관계를 정의
    - 객체 속성 및 연결 관계 정의
    - 클래스를 계층화하고 모듈로 정의
    - 생성된 모형을 반복적으로 검증

- 동적 모델링
  - Dynamic Modeling
  - 상태 다이어그램(상태도)을 이용하여 객체들 간의 제어 흐름, 상호 작용, 동작 순서 등의 동적인 행위를 표현하는 모델링
  - 객체나 클래스의 상태, 사건을 중심으로 다룬다.
  - 사건 : 하나의 객체로부터 다른 객체에 자극을 주어 객체의 상태를 변화시키는 것
  - 상태 : 특정 시점의 객체에 대한 속성값
  - 순서
    - 시나리오 작성
    - 사건 추적도 작성
    - 사건 흐름도 작성

- 기능 모델링
  - Functional Modeling
  - 자료 흐름도를 이용해 다수의 프로세스 간의 자료 흐름을 중심으로 처리과정을 표현한 모델
  - 순서
    - 입출력 자료를 정의
    - 자료 흐름도를 상세화
    - 기능 명세서 작성
    - 제약 조건 파악
    - 최적화 기준 명세

- 객체 모델링: 객체
- 동적 모델링 : 객체의 흐름, 상태, 행위
- 기능 모델링 : 자료 흐름, 처리 과정

# 객체지향 설계

- Object Oriented Design
- 객체지향 분석을 사용해서 생성한 여러 가지 분석 모델을 설계 모델로 변환하는 작업
- 시스템 설계와 객체 설계를 수행한다.
- 사용자 중심, 대화식 프로그램 개발에 적합하다.
- 시스템을 구성하는 객체와 속성, 연산을 인식하는 것이 중요한 문제
- 추상화, 정보 은닉, 기능 독립성, 모듈화, 상속성을 바탕으로 하며 모듈화가 가장 중요하다.
- 문제 정의 => 요구 명세화 => 객체 연산자 정의 => 객체 인터페이스 결정 => 객체 구현
- 럼바우의 객체지향 설계

# 객체지향 구현

- 구현은 설계 단계에서 생성된 설계 모델과 명세서를 근거로 하여 코딩하는 단계이다.
- 객체지향 프로그래밍을 이용하면 용이하게 구현할 수 있다.
- 객체는 순차적으로 또는 동시적으로 구현될 수 있다.
- 객체지향 프로그래밍
  - Object Oriented Programming
  - 객체 기반 언어 : 객체의 개념만을 지원하는 언어
  - 클래스 기반 언어 : 객체와 클래스의 개념을 지원하는 언어
  - 객체 지향성 언어 : 객체, 클래스, 상속의 개념을 모두 지원하는 언어 (Simula, Smalltalk, C++, Objective C)

# 객체지향 테스트

- 클래스 테스트 : 캡슐화 된 클래스나 객체를 검사하는 것
- 통합 테스트 : 객체 몇개를 결합하여 하나의 시스템으로 완성시키는 과정에서의 검사
  - 스레드 기반 테스트
  - 사용 기반 테스트
- 확인 테스트 : 사용자 요구사항에 대한 만족 여부를 검사
- 시스템 테스트 : 모든 요소들이 적합하게 통합되고 올바른 기능을 수행하는지 검사

# UML

- Unified Modeling Language
- Rumbaugh, Booch, Jacobson 등의 객체지향 방법론의 장점을 통합한 객체 지향 모델의 표준 표현 방법
- 객체지향 분석과 설계를 위한 모델링 언어로 객체 기술에 관한 국제 표준화 기구인 Object Management Group에서 UML을 표준으로 지정했다.
- 어플리케이션을 개발할 때 이해를 도와주는 사용 사례 다이어그램, 순서 다이어그램, 상태 다이어그램, 활동 다이어그램 등 여러 형태의 다이어 그램을 제공
- 사용 사례 다이어그램
  - Use Case
  - 사용자와 사용사례로 구성
  - 사용 사례 간에는 여러 형태의 관계로 이루어진다

# SW 재사용

- 이미 개발되어 인정받은 SW의 전체 혹은 일부분을 다른 SW 개발이나 유지에 사용하는 것
- 클래스, 객체 등의 SW 요소는 SW 재사용성을 크게 향상시켰다.
- SW 재사용에 가장 많이 이용되는 것은 소스코드이다.
- 모듈의 크기가 작고 일반적인 설계일수록 재사용률이 높다.

# 재사용의 이점

- 개발 시간과 비용의 단축
- SW 품질 향상
- 개발 생산성 향상
- 프로젝트 실패 위험 감소
- 시스템 구축 방법에 대한 지식 공유
- 시스템 명세, 설계, 코드 등 문서를 공유

# 재사용의 문제점

- 새로운 개발방법론 도입이 어려움
- 프로그램 표준화가 부족
- 프로그램 언어가 종속적
- SW 요소 내부 뿐아니라 인터페이스 요구사항의 이해가 필요하다.

# 재사용 방법

- 합성 중심 방법
  - Composition Based = 블록 구성 방법
  - 모듈을 만들어 조립하며 SW를 완성시키는 방법
- 생성 중심
  - Generation Based = 패턴 구성 방법
  - 추상화 형태로 쓰여진 명세를 구체화하여 SW를 완성시키는 방법

# SW 재공학

- Software Reengineering
- 새로운 요구에 맞도록 기존 시스템을 이용하여 보다 나은 시스템을 구축
- 새로운 기능을 추가하여 SW 성능을 향상
- 유지보수 생산성 향상을 통해 SW 위기를 해결
- 기존 SW의 기능을 개조하거나 개선하므로 예방 유지보수 측면
- 자동화된 도구를 사용하여 SW를 분석하고 수정하는 과정을 포함
- SW 수명이 연장되고 기술이 향상
- 오류가 줄어들고 비용이 절감
- 예방 유지보수

# 재공학의 목표

- 복잡한 시스템을 다루는 방법 구현 : 자동화 도구 사용
- 다른 뷰의 생성 : 기존 시스템 개발 관점 외에 다른 방향의 관점을 생성
- 잃어 버린 정보의 복구 및 제거
- 부작용의 발견
- 고수준의 추상 : 추상화된 어려운 내용을 여러 형태로 추출해 이해
- 재사용 용이

# 주요활동

- 분석
  - Analysis
  - 기존 SW의 명세서를 확인하여 SW의 동작을 이해하고 재공학 대상을 선정하는 것
- 개조
  - Restructuring = 재구조 - 재구성
  - 상대적으로 같은 추상적 수준에서 하나의 표현을 다른 표현 형태로 바꾸는 것
  - 기존 SW의 구조를 향상시키기 위해 코드를 재구성 하는 것
  - SW의 기능과 외적인 동작은 바뀌지 않는다.
  - IF ELSE를 SWITCH CASE로 변경하듯이

# Client/Server 시스템

- 분산 시스템의 가장 대표적인 모델
- 정보를 제공하는 서버와 정보를 요구하는 클라이언트로 구성
- 클라이언트와 서버가 하나의 작업을 분산 협동 처리한다.

# 요소

- 애플리케이션 요소 : 응용 프로그램에 의해 정의된 요구사항을 구현
- 데이터베이스 요소
- 프리젠테이션/상호작용 요소 : GUI와 관련된 모든 기능

# 미들웨어

- 클라이언트와 서버 사이에 존재해서 데이터 전송 과정을 효율적으로 수행하도록 도와주는 SW
- 통신 미들웨어 : NOS(Network Operating System)
- 데이터베이스 미들웨어 : ODBC
- 분산 객체 미들웨어 : CORBA, DCOM

# 객체 요청 브로커

- ORB = Object Request Broker
- 분산 객체 미들웨어의 일종
- 클라이언트의 객체가 서버 객체의 캡슐화된 메소드에게 메세지를 보낼 수 있게 하는 것

# CORBA

- Common Object Request Broker Architecture
- 가장 많이 사용되는 객체 요청 브로커의 표준
- OMG(Obejct Management Group)라는 개발자 연합에서 인가
- IDL : Interface Description Language CORBA가 클라이언트/서버 시스템에서 구현될 때 필요한 인터페이스 언어

# CASE

- Computer Aided Software Engineering
- SW 개발 과정에서 사용되는 요구 분석, 설계, 구현, 검사 및 디버깅 과정 전체 또는 일부를 컴퓨터와 전용 SW 도구를 사용하여 자동화하는 것
- SW 개발 도구와 방법론이 결합된 것
- 정형화된 구조 및 방법을 SW 개발에 적용하여 생산성 향상을 구현하는 공학 기법
- 자동화 도구를 지원하고 개발자는 SW 개발의 표준화를 지향하며 자동화의 이점을 얻을 수 있다.
- SW 생명주기 전 단계의 연결, 다양한 SW 개발 모형 지원, 그래픽 지원

# 사용 이점

- SW 개발 기간 단축하고 개발 비용 절감
- 자동화된 기법을 통해 SW 품질 향상
- 유지보수 간편하게 수행
- 생산성 향상
- 운용 활동 효과적으로 관리 및 통제
- 품질과 일관성을 효과적으로 제어
- SW 개발 모든 단계에 걸친 표준 확립
- 모듈의 재사용성 향상
- 개발 기법의 실용화, 문서화가 쉬움

# 분류

- 상위 CASE
  - Upper CASE
  - SW 생명 주기 전반부에서 사용
  - 문제를 기술하고 계획하며 요구 분석과 설계 단계를 지원
  - 여러가지 명세와 문서를 작성하는데 사용
  - SREM, PSL/PSA, SERA, FOUNDATION
- 하위 CASE
  - Lower CASE
  - SW 생명 주기 하반부에서 사용
  - 코드의 작성과 테스트, 문서화하는 과정을 지원

# 정보 저장소

- SW를 개발하는 과정 동안 모아진 정보를 보관하여 관리하는 곳
- CASE 정보 저장소 = CASE 데이터베이스 = 요구사항 사전 = 저장소
- 초기에는 사람이 정보 저장소, 오늘은 DB가 정보 저장소
- 도구들의 통합, SW 시스템의 표준화, SW 시스템의 정보 공유, SW 재사용성의 기본
- 시스템의 정보 공유 활성화
- 유지보수성 향상
- CASE 도구간 정보를 쉽게 교환, 사용자가 새로운 도구를 쉽게 추가
- 중복된 공통정보를 통합해 불필요한 정보 제거
- 생명 주기 정보를 재사용