1장.

1) SoftWare-Engineering

- 최소의 경비로 신뢰도[1] 높은 소프트웨어를 생산하기 위한 기법과 도구에 대한 내용을 다루며, 과학적 지식을 컴퓨터 소프트웨어개발에 도입하고, 이를 개발,운영,유지보수 하는데 필요한 문서화 과정을 포함한다.

2) Software-Crisis

- 소프트웨어에 대한 사용자의 불만족, 비용증가, 신뢰도 결여, 소프트웨어의 수요에 대한 증가로 인해 발생함.
- 위기를 인식하고 위기의 원인을 규명한 후 위기에 대한 원인을 제거하는데 집중하여 소프트웨어 위기를 해결한다.

* Loc(Line of cord) ... 소프트웨어의 라인 수가 얼마나 되는가

3) 소프트웨어 개발방법론

① SDLC 방법 (↔ Bulit-Fix)

- Software Development Life Cycle
- Waterfall Model (폭포수 모델)
- Bohem이 제시한 전통적인 소프트웨어 개발방법으로, 소프트웨어 규모가 큰 경우에 적용한다. 정확성을 중시하고 사용자의 요구사항은 개발 초기 단계에 많이 수용한다. 오랜 기간이 소요되고 참여자가 많다.
- 단계

(1) 타당성 검토 : 소프트웨어의 필요성을 파악한다. 소프트웨어 개발 계획[2] 수립

(2) 계획과 요구 : 분석 단계, 사용자 요구사항에 대한 구체적인 서비스, 제약조건, 목적등을 파악하고 문서화

(3) 개략 설계 : 전체적인 하드웨어, 소프트웨어, 데이터, 제어구조 파악

(4) 상세 설계 : program으로 구현 가능한 전단계로 변환.
순서도,모듈구조 → 시스템 구조 모듈 설계 내용 작성

(5) 코딩 : program의 명령어로 변환하여 작성

(6) 통합 검사 : 모듈 검출, 해당 소프트웨어에 대한 품질 보증 확인
사용자의 요구사항과 프로그램의 실행 결과를 확인[3]

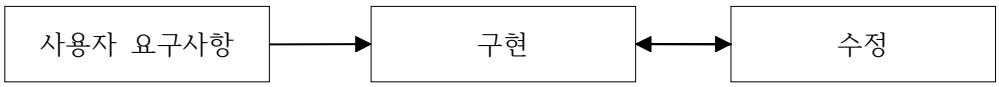
(7) 실행

(8) 운용 및 유지보수

---

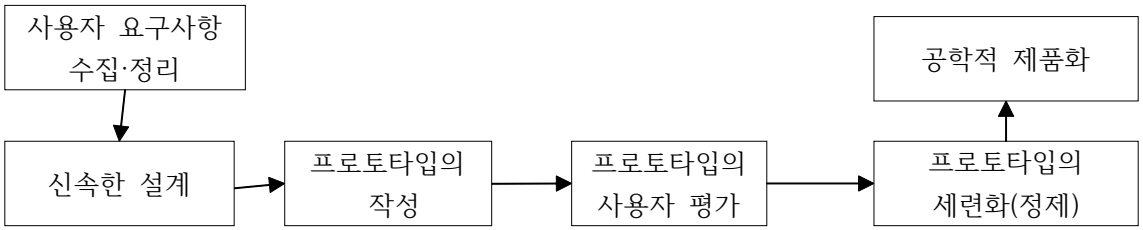
1) 정확하고 안정된 결과를 보장하는 것.
2) 개발과정, 생산제품에 대한 내용(개발 영역, 기능, 정보, 범위, 내용, 전체적인 작업계획..) 포함
3) 세부적 내용, 기능 등을 검토

② Bulit-Fix Model (구축 수정 모델 = 주먹 구구식 모델)



- 분석, 설계 과정 생략. 바로 소프트웨어 개발을 진행하며 체계적인 SW 개발팀이 없는 상태에서 SW 개발
- 구현된 상태에서 계속 수정 사항이 발생하며, 개발 경험이 없는 상태에서 수행되므로 SW에 대한 가독성과 수정 문제 발생

③ Prototype 모델



- 개발자가 구축할 모델을 사전에 작성. 사용자의 요구사항을 개발자가 미리 파악하기 위해 사용
- 개발자와 사용자 간의 의사소통 도구로 사용
  · Paper Prototype : 사용자와 시스템 사이의 인터페이스에 중점
  · Working Prototype : 소프트웨어의 기능에 중점
  · Existing Program : 작성된 소프트웨어를 향상하기 위함

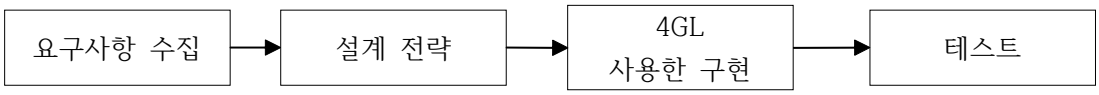
④ Spiral Model

- 대규모 시스템 개발에 적합한 방법, SDLC + 프로토타입 + 위험성 분석에 대한 내용 포함. 점진적으로 소프트웨어 시스템 완성

(1) 계획화 : 사용자의 요구사항을 파악. 시스템의 기능, 성능, 정보, 내용 등을 포함
(2) 위험성 분석 : 위험에 따른 방안에 대한 분석
(3) 공학화 : 시스템을 반복적으로 수정·개선하여 소프트웨어 품질 향상
(4) 사용자 평가

⑤ 4세대 기법(Fourth Generation Language)

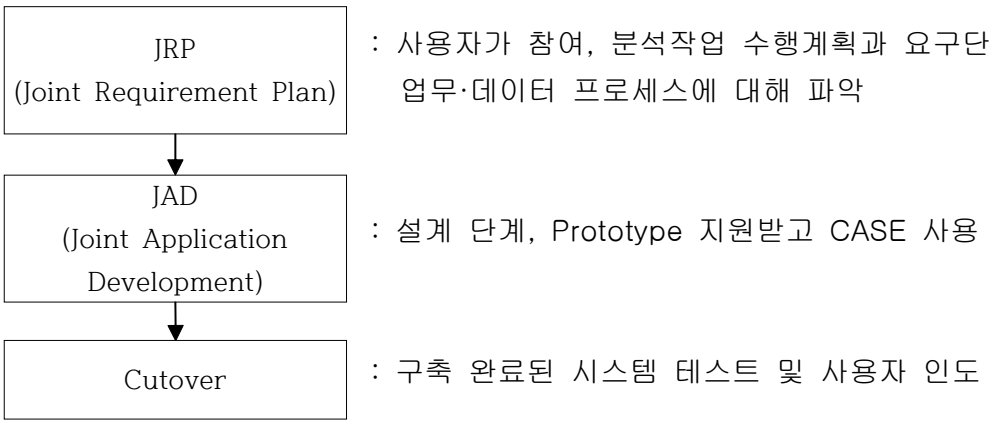
- 4세대 언어[4]를 사용하여 구현. CASE[5]를 사용.
- 기업 정보 시스템, 정보 분석 등에 한정. 대규모 프로젝트에는 부적합



4) 데이터베이스 접근, 비절차적 코딩, 비전문가 위한 언어, 판독성 좋음
5) CASE (Computer Aided Software Engineering) : 소프트웨어 개발의 자동화, 소프트웨어 생명주기의 모든 단계와 연계되어 동작하며, 이들을 자동화 시켜주는 통합된 도구

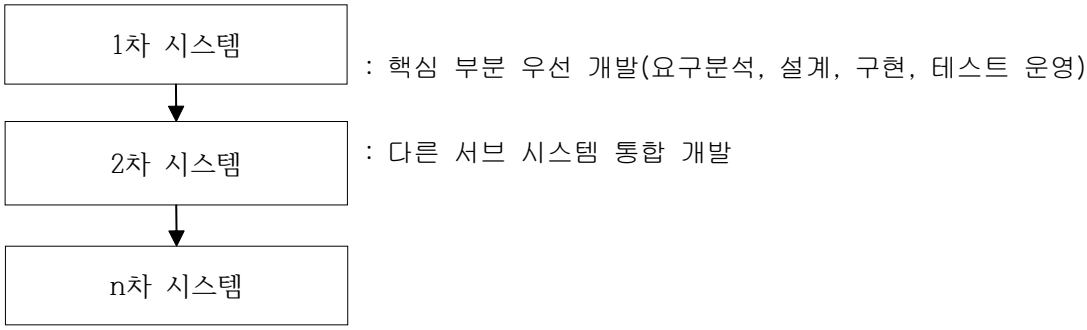
⑥ RAD 모델 (Rapid Application Development)

- 매우 빠른 개발 주기를 가지는 선형 순차 개발 프로세스 모델
- 4세대 언어, 소프트웨어 재사용. 팀 단위 병행 개발의 특성



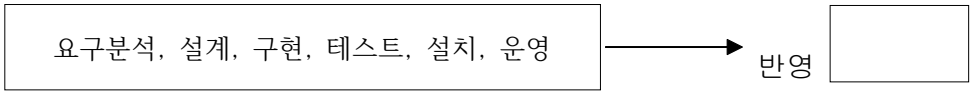
⑦ ICM 모델 (Incremental Complete Model)

- 시스템의 규모가 큰 경우, 여러 개의 서브 시스템[6] 단위로 구분하고 서브 시스템을 단계적으로 개발하여 통합하는 방법
- 핵심적인 부분만을 우선 개발하고, 개발에 대한 확신을 가질 수 있으나, 다른 서브 시스템과 통합할 때 문제 발생.
- 과거 경험이 있는 유사 프로젝트 개발에 유용한 방법



⑧ 진화적 개발 모델 (EDM, Evolutionary Development Model)

- 사용자의 요구사항이 불명확한 경우, 구현하고자 하는 시스템을 n단계 개발계획으로 구분하고, 각 단계에서 요구분석, 설계, 구현, 테스트, 설치, 운영을 반복하면서 사용자가 만족할 때까지 반복적으로 계속 진화시켜 나가는 방식
- 초기 모델이 프로토 타입 방법처럼 이해될 수 있음. 복잡한 요구사항으로 인한 개발지연이 있다.



6) 사용자의 요구사항이 명확

⑨ 통합 처리 모델 (UPM, Unified Process Model)

- 객체 지향 방법론을 적용하여 시스템을 개발 ← 대규모 개발에 적합
- 도입(incept), 정제(elaborate), 구현(construct), 인도(transit) 단계를 수행
  · 도입 단계 : 타당성 검토, 비용, 일정, 계획 수립에 중점을 둔다.
  · 정제 단계 : 설계와 사용자 인터페이스에 대한 사항에 중점을 둔다,
  · 구현 단계 : 사용자 출력에 대한 보고서 작성에 중점
  · 인도 단계 : 완성된 시스템을 사용자에게 설치, 운영
- 각 단계에서 요구분석, 설계, 구현, 테스트 단계를 반복하며 시스템을 확인하는 특성을 가진다.

⑩ Agile 방법론

- Pair programming : 사용자 경험(UX) 중시
- 시스템 개발 시 사용자와 협력하여 유연하게 소프트웨어 개발
- 사용자가 개발과정 직접 참여하며, 개발과정의 우선순위를 정하고 개발된 시스템을 평가하며, 사용자의 UX를 중요항목으로 고려하는 방법
- 익스트림 프로그래밍 (eXtreme Programming, XP)
  개발과정이 얼마 남지 않은 극한의 상황에서 소프트웨어를 개발하는 방법.
  소규모 시스템 단위, 사용자의 요구사항 빈번하게 발생하므로, 개발자·사용자간 소통 중시, 테스트 위주로 시스템 개발.
- 스크럼 (Scrum)
  Product Backlog (구체적 작업목록 결정), Sprint Backlog (특정 기간 동안 이루어져야 하는 목표, 필요작업 명세) 30일 단위로 개발 기간을 가지고 소프트웨어 결과물을 산출함.

Agile 방법론	SDLC(폭포수 모델)
사용자 요구를 일단 적용. 신속성, 짧은 개발 기간	타당성 검토 이후 사용자 개입 x 정확성, 오랜 개발 기간
특정기능 중시, 상향식, 소규모	시스템의 조화, 하향식, 대규모

2장. 프로젝트 관리

1) 프로젝트 개념

- 시스템에 대한 개발 계획을 수립하고, 사용자의 요구사항에 의해 시스템에 대한 분석, 설계, 구현, 테스트, 유지보수 등의 작업단계를 수행하는 과정.
- 프로젝트에 대한 소프트웨어 개발 영역 자원, 비용, 일정, 품질 등을 고려.
- 프로젝트 계획 수립 시 개발단계(과정)와 생산 제품에 대한 내용을 포함하여 프로젝트 복잡도, 규모, 위험성, 과거 정보의 가용성, 구조적 불확실성의 정도를 고려
- 프로젝트에 대한 3P를 고려

※3P

People	Senior Manager	프로젝트 방향 설정
	Project Manager	프로젝트 계획 수립, 동기화 부여, 조직구성, 실무자 관리
	Engineer	실제 프로젝트 세부 내용 구현하는 프로그래머
	Customer	고객 (프로젝트 요청)
	User	최종 사용자
Problem	프로젝트에 대한 내용, 정보, 기능, 성능 등을 파악. 복잡한 문제를 분할&정복함으로써 해결.	
Process	전체적인 작업계획 수립(Framework 파악 단계)	

2) 프로젝트 구조

① Project Organization(프로젝트 조직)

- 프로젝트 수행에 제한된 기간을 가지며, 각 팀들간의 독립적인 구현을 수행
- 프로젝트 관리자에게 많은 권한을 부여

② Functional Organization (기능식 조직, 직능식 조직)

- 각 프로젝트 팀들이 목적에 따라 서로 다른 프로젝트 단계를 수행하고, 프로젝트가 진척됨에 따라 각 팀의 작업 내용을 다른 팀으로 전달함
- 하나의 전문 업무에 집중하며, 특정 소프트웨어에 집중하여 업무를 수행

③ Matrix Organization (행렬 조직, ①+②)

- 프로젝트 팀에 대해 고유의 관리팀과 그 기능과 연계된 전문가들로 구성되며 조직의 구성체계가 복잡하고, 의사결정이 복잡한 구조

3) 팀 구조(구성)

① Democratic Team (민주주의식, 분산형 팀)

- 전체 구성원의 여론에 따라 프로젝트가 진행되며, 구성원 간의 학습 효과, 장기간 프로젝트에 적합하며, 링 모양의 구조를 가짐(의사소통 경로의 수 : n*n-1/2)
- 구성원이 동등한 레벨을 가지고 프로젝트를 수행함

② Chief Programmer Team (중앙집중형, 책임 프로그래머 팀)

- 한 명의 책임자가 모든 중요한 의사결정을 수행하며, 의사소통 경로多
- 한 명의 고급 프로그래머와 여러 명의 중급 프로그래머로 구성됨.
- 단기간·소규모 프로젝트에 적합, 성공 여부는 책임 프로그래머의 능력에 좌우됨

③ Hierachy Team (계층형, ①+②)

- 고급 프로그래머가 5~7명의 프로그래머를 관리하며, 구성원 간의 의사전달 경로가 제한되며, 서로 교신할 필요가 있는 구성원 간의 의사전달 경로만 허용되며, 프로젝트 리더와 고급 프로그래머에게 지휘권한을 부여.

4) 프로젝트 비용 산정

① 전문가의 감정

- 프로젝트 내에 경험이 있는 2명 이상의 전문가에게 프로젝트 비용을 산정하도록 의뢰하는 방법으로, 새로운 프로젝트에 대한 경험이 없으며, 새 프로젝트는 새로운 요소가 있다는 것을 간과, 개인적·주관적이다.

② 델파이 비용산정

- 전문가의 감정에 의한 단점을 보완, 여러 명의 전문가와 한 명의 조정자가 비용을 산정한다.

③ LOC에 의한 방법 (Line Of Code)

- 프로젝트에 대한 각 모듈의 세부적인 작업 단위별로 비용을 산정. 이들 각각을 집계하여 총 비용을 산정하는 방법으로, 시스템을 분석하여 구성되는 전체 모듈에 대한 구조도를 파악 후 작업

* 노력(인/월) = 개발기간 * 투입인원 (또는 LOC / 인당 평균 생산코드 라인)
* 개발비용 = 노력(인/월) * 단위비용 (인당 인건비)
* 개발기간 = 노력(인/월) / 투입인원
* 생산성 = LOC / 노력(인/월)

예) 30000라인, 5명, 평균생산코드 300라인일 때, 소요시간은?
노력(인/월) = 30000 / 300 = 100
개발기간 = 100 / 5 = 20개월

④ COCOMO에 의한 비용산정 (Constructive Cost Model by Boehm)

- Loc에 기반, Loc 예측한 후(프로그래밍 언어에 따라 산정비용 차이) 이를 기반으로 적용되는 비용 산정 방정식에 의해 프로젝트 비용을 산정
- Organic Mode(기본형) : 총 라인 수가 수만 라인(Basic)
  과학 기술 및 사무처리, 응용 프로그램
- Semi-Detatched Mode(중간형) : 총 라인 수가 수십만 라인(Intermediate)
  DB,OS개발 위한 소프트웨어
- Embedded Mode(내장형) : 총 라인 수가 수백만 라인(Detailed)
  우주 항공, 대륙간 탄도미사일

⑤ Putnam에 의한 방법

- 소프트웨어 생명 주기 동안 사용될 노력 분포를 가정해주는 모형으로,
  SDLC 모델에 적용(대형 프로젝트)
- 개발 기간이 길어질수록 투입 인원의 노력이 감소함

⑥ FP에 의한 방법 (Functional Point)

- Functional Point에 기반, Software의 기능을 증가시키는 요인별로 가중치를 부여, 요인별 가중치를 합산하여 총 비용을 산정하는 방법
- 기능별 요인에 대해 개인적·주관적인 방법 ← 신뢰성 문제, 계산 복잡

5) 프로젝트 스케줄링

- 프로젝트의 프로세스에 대한 작업단위(모듈, 구성요소, 컴포넌트, 프로그램..)파악
- 작업 단위의 순서와 일정을 파악하고, 소프트웨어에 대한 개발의 지연을 방지.
- 일정 관리에서는 작업단위 분할, 상호관계, 노력, 책임성, 정의된 산출물을 고려.
- Brooks law : 일정 지연시 더 많은 인력을 투입하면 투입 인원의 적응 기간과 side-effect로 인해 개발이 더욱 지연됨
- 작업 단위에 대한 통제의 기능, 각 작업 단위에 대한 관계 파악, 작업단위에 대한 책임문제를 파악하기 위해 사용.

① PERT (Program Evaluation & Review Techniques)

- 각 작업 단위에 대한 낙관치, 평균(기대)치, 비관치에 대한 값을 설정하고, 각 작업 단위의 종료 시기를 예측하는 방법
- 노드와 간선으로 각 작업 단위와 진행 과정을 나타내며, 과거 경험이 없는
- 대규모 프로젝트에 적용

② CPM (Critical Path Method)

- 과거 경험이 있는 유사 프로젝트에 적용
- 노드는 작업 단위, 간선은 화살표로 표현하며, 간선 위에 작업완료 예상 시간을 나타냄
- 시스템 전체의 규모를 파악하고, 필요로 하는 작업 단위를 분해하여 작업 간 상호관계를 표시
- 중요한 목표 업무별로 완성해야 하는 시점 표시할 때 이정표를 사용

③ WBS (Work Breakdown Structure)

- 프로젝트를 세부적인 기능과 역할에 의해 작은 작업 단위로 구분하고, 이들에 대한 계층적인 구조
- 각 작업 단위의 구조를 이해하고 각 작업에 필요한 자원을 예측하여 전체 시스템에서 사용되는 자원을 파악
- 작업 단위는 업무지시를 위한 명세로 사용될 수 있으며, 각 작업 단위에는 작업 소요일, 책임자, 작업시간, 마감일, 각 작업의 세부적인 상세 내용과 기능에 대한 설명을 포함

④ Gantt Chart

- 프로젝트에 대한 진도를 파악하며, 계획 대비 작업 단위에 대한 진척도를 나타내어 프로젝트 일정에 도움
- 좌측에는 작업, 상단에는 시간(일)을 표현
- 작업의 길이 = 막대의 길이, 작업시간과 비례
- 프로젝트에 대한 계획과 통제의 기능을 수행하지만 각 작업간의 관계, 작업의 중요도, Critical Path는 파악 불가.

6) 소프트웨어 품질(Software Quality)

- 품질 : 주어진 기능을 만족시키는데 필요한 소프트웨어의 측정 가능한 기능.
- 효율성(Efficiency) : 사용자의 요구사항에 의해 최소한의 시간과 기억장소를 가지고 작업 단위가 수행되는 능력
- 융통성(Flexibility) : 사용자 요구사항의 변경, 수정 등의 작업에 의해 소프트웨어 변경에 대한 유연성
-무결성(Integriting) : 시스템에 대한 독단적인 접근에 대해 소프트웨어가 침해되지 않는 정도
-사용용이성(Useability) : 사용자가 쉽게 배우고, 소프트웨어를 이해하여 사용되는 정도
-신뢰성(Reliability) : 사용자의 요구사항에 대해 항상 정확하고 일관된 기능 제공
-이식성(Portability) : 하나 이상의 하드웨어 환경에서 소프트웨어가 설치, 사용될 수 있는 능력
-정확성(Correct ability) : 작성된 소프트웨어의 KLoc당 오류에 대한 비율로 측정
-유지보수성(maintain ability) : 소프트웨어에 대한 변경 작업에 대해 소요되는 평균 시간의 비율로 측정

* SQA (소프트웨어 품질 보증, Software Quality Assurance)

-소프트웨어가 사용자의 요구사항과 일치하는가를 확인하는 체계적인 과정

* FTR (정형 기술 검토, Formal Technique Review)

-소프트웨어 공학자에 의해 수행되는 품질보증활용. Workthrough와 Inspection 수행

* Workthrough

- 상세 설계 내용을 가지고 소프트웨어의 각 단계에서 실시하는 기술 검토 회의
- 오류검출에 목적. 발견된 오류는 문서화 + 특정 내용 검토 목적
- 3~5명이 참가, 2-3시간의 회의 시간 내 종료

* Inspection (검열)

- 각 단계에서 나온 결과를 여러 전문가가 확인, 전문가 중 책임자가 조정함

7) 소프트웨어 신뢰성과 가용성

-신뢰성: 특정 기간동안 특정 조건 하에서 시스템이 그 기능을 얼마나 발휘할 수 있는지에 대한 확률

-하드웨어 신뢰도 : 어떤 고장이 발생한 다음 다른 고장이 발생하는데 소요되는 평균시간

· MTBF(Mean Time Between Failure, 평균고장시간)
· MTTR(Mean Time To Repair, 평균수리시간)
· MTTF(Mean Time To Failure, 평균가동시간)
· MTBF = MTTR + MTTF
· MTTF = $\frac{\text{가동}1 + \ldots + \text{가동}n}{n}$
· MTTR = $\frac{\text{고장}1 + \ldots + \text{고장}n}{n}$
· 신뢰도 (가용도) = $\frac{MTTF}{MTTF + MTTR} = \frac{MTTF}{MTBF} = \frac{\text{평균가동시간}}{\text{평균고장시간}}$

8) 소프트웨어 복잡도 측정

- 원시 코드의 복잡도 측정은 수정의 전후에 프로그램의 복잡도를 결정하기 위해 사용됨. 명세화를 위한 작업을 할 때 관여한 루틴들 식별하는데 사용.
- 홀스테드의 노력 방정식

: 소프트웨어에 대한 최초의 복잡도 분석 법칙. 소프트웨어 과학의 정성적 법칙을 컴퓨터 소프트웨어 개발에 적용

[7]프로그램 길이(N) = $n_1 \log_2 n_1 + n_2 \log_2 n2$

[8]프로그램 부피(V) = $N \log_2 (n_1 + n_2)$ ← 프로그램을 기술하는데 필요한 정보의 양. 언어마다 다르다.

-McCabe의 사이클로메틱 복잡도 측정

상세 설계 기반으로 사용. 프로그램의 제어 흐름도를 기반으로 분석함. 그래프를 사용하여 제어 흐름 나타내고, 각 노드는 처리 작업 표시

* 복잡도 (V) = 화살표 수(A) - 노드의 수(N) + 2

9) 위험 관리

- 프로젝트 수행 시 예상되는 위험을 미리 파악하고, 이에 대한 적절한 대책을 수립하는 활동이며, 위험은 불확실성과 손실을 포함

-위험 분류

① Project Risk : 프로젝트 수행 중 발생할 수 있는 위험으로, 인력, 비용, 일정, 요구사항 변경
② Technical Risk : 소프트웨어 품질과 연계된 위험 (구현 불가능, 보안, 위협, 사용자 만족도...)
③ Business Risk : 프로젝트의 생존 가능성 위협 (원치 않는 제품을 개발)

-위험 종류

① Known Risk : 프로젝트 계획서, 기술적 환경, 정보 등에서 파악되는 위험
② Predictable Risk : 과거 유사한 프로젝트에 대한 경험으로 유도되는 위험
③ Unpredictable Risk

-위험 관리 절차

① Risk identification : 알려지거나 예측 가능한 위험을 risk item checklist[9]를 사용하여 구분함
② Risk Evaluation & Analysis : 위험 종류 내용, 위험 발생확률, 영향력, 조건 등 따짐
③ Risk Management : 위험의 예방, 대책, 문서화

---

7) n1 = 프로그램 내 고유 연산자 수. n2 = 프로그램 내 고유 연산수의 수
8) N1 = 연산자 발생 총 수. N2 = 연산시 발생 총 수
9) 위험 범위, 내용, 타이밍....

10) 형상 관리(Configuration Management)

- 소프트웨어 개발에서 소프트웨어에 대한 변경사항을 관리하기 위한 일련의 활동
- 소프트웨어 변경 원인을 파악하고 이에 대한 제어 절차, 변경 내용을 담당자에게 통보하는 작업까지를 포함
- 소프트웨어 개발의 전체 비용을 줄이고, 여러 방해 요인들이 최소화되도록 보증
- 단계

① 형상 식별(Configuration identification)
형상 관리 대상에 관리번호와 이름을 부여. 이들에 대한 계층적 관리 지원

② 버전 제어 (Version Control)
소프트웨어의 변경 사항에 대한 delta 정보 관리 지원 및 meta 정보 관리

* delta 관리 : 소프트웨어 개발 도중 발생한 모든 데이터 변경사항
* meta 관리 : 프로그램의 보조정보(기능, 알고리즘, 입출력 데이터, 주요 내용, 작성자, 개발시기..)

③ 변경 제어 (Change Control) : 식별된 형상 항목의 변경 요구를 검토하여 Baseline이 잘 반영될 수 있도록 조정하는 작업

* Baseline : 프로그래머에 의해 정의되고 명세된 합의서, 제품 내용

④ 형상 검사(Configuration Audit): Baseline의 Integrity(무결성)를 평가하기 위한 확인, 검증작업

⑤ 형상 상태 보고서(Configuration Status Reports): 각 SCI 내용에 대한 형상상태보고서 작성

* SCI (Software Configuration Item)
사용자의 요구사항 분석 명세서, 설계서, 순서도, UML, 시스템 구조도, 프로그램, 테스트 문서

11) 소프트웨어 재사용

-장점 : 개발 시간의 단축과 비용 절감, 소프트웨어의 품질 향상, 생산성의 증가, 시스템과 시스템 개발에 관한 지식의 공유

-단점 : 시스템에 공통인 컴포넌트 발견의 어려움, 프로그램 내의 표준화의 부족, 프로그래밍 언어에의 종속, 라이브러리화에 대한 결정과 대상을 찾는 어려움, 소프트웨어 컴포넌트의 기술과 분류의 복잡함, 재사용을 위한 관리 지원의 부재

- 작성한 소프트웨어를 반복 사용하여 소프트웨어의 생산성을 향상하는 방법으로, 프로그램 코드를 자동적으로 생성하는 것을 의미
- 소프트웨어 개발 기간을 단축하며, 신뢰도 향상, 소요 인력의 감소와 프로그래머의 능력 증진시킬 수 있다.
- 구성 요소

① 빌딩 블록(building block)

- 소프트웨어 구성 요소들을 라이브러리에 저장하여 놓고 새로운 개발에 필요한 구성 요소들을 검색하여 적절히 수정한 후 다른 구성 요소들과 결합하여 목적 프로그램을 개발하는 방법
- 잘 정의된 인터페이스를 필요로 하며, 전통적인 소프트웨어 개발 방법과 비교하여 볼 때 질적으로 우수하며, 재사용 비율에 정비례하여 생산성이 향상되고 짧은 개발 기간을 가진다.
- 재사용 가능한 소프트웨어의 검색과 등록을 위한 라이브러리를 제공하고, 라이브러리로부터 재사용 구성 요소를 검색하여 새로운 소프트웨어를 생성하기 위한 환경을 제공해야 한다.
- 새로운 모듈을 구축하는 데 필요한 시간과 노력이 감소되고, 프로그래머의 능력을 증폭시킬 수 있다. 재사용 가능한 모듈을 도구를 통해 검색하므로 효율적이며, 시스템의 유지보수가 용이하다.
- 소프트웨어의 검색과 명세가 자동화되어 있지 않으면 재사용 가능한 소프트웨어를 찾기 위해 요구되는 시간의 양이 증가된다
- 선택 -> 개조 -> 개조 -> 결합 -> 등록 -> 평가

② 패턴 재사용

- 구성 요소 자체가 능동적 요소.

12) 소프트웨어 재공학

- 유지보수 측면에서 소프트웨어의 위기를 해결하려고 한 방법. 유지보수성을 향상시키고, 소프트웨어에서 사용하고 있는 기술을 업그레이드하여 소프트웨어의 수명을 연장함.
- 기존 소프트웨어를 새로운 기술을 사용할 수 있도록 조정하며, 소프트웨어의 유지보수성을 향상시키는 예방적 형태의 유지 보수 활동이다.
- 자동화된 도구를 사용하여 소프트웨어를 분석하고 수정하는 과정을 포함
- 단계

① 분석 : 소프트웨어 동작을 이해, 재공학 대상을 선정
② 재구성 : 소프트웨어의 기능을 변경하지 않으면서 소프트웨어 형태를 목적에 맞게 수정
③ 역공학 : 원시코드를 분석하여 소프트웨어 관계를 파악하고 기존 시스템의 설계정보를 다시 제작, 기존 시스템의 문서화를 다시 작성
④ 이식 : 기존 소프트웨어 시스템을 새로운 기술 또는 하드웨어 환경에 이식함.

13) 클라이언트 / 서버 소프트웨어 공학

-서버 : 정보나 자원을 일원적으로 관리하고 제공하는 역할의 하드웨어와 소프트웨어

① 파일 서버 : 특정 레코드의 요청과 전송을 담당.
② 데이터베이스 서버 : SQL을 통한 요청의 송신과 처리 후 결과를 전달.
③ 트랜잭션 서버 : 원격 프로시저의 호출과 전송을 담당.
④ 그룹웨어 서버 : 서버가 클라이언트와 통신할 수 있는 어플리케이션 집합을 제공.

-클라이언트 : 서버에 요구한 정보와 자원을 이용하는 하드웨어와 소프트웨어

- 클라이언트/서버 시스템의 소프트웨어 컴포넌트

· user interaction / presentation component
시스템에서 사용되는 GUI와 연관된 모든 기능을 담당한다.

· application component
어플리케이션에서 수행되는 작업들을 처리한다. (데이터 입력, 데이터베이스 작업, 보고서 생성 등)

· database management
어플리케이션이 요구하는 데이터의 처리와 제어를 담당한다.

* ORB(Object Request Broker)
클라이언트에 있는 객체가 서버에 있는 객체로 캡슐화시키는 방법으로 메시지를 송신하는 미들웨어 컴포넌트

14) CASE (Computer Aided S/W Engineering)

- **소프트웨어 개발의 자동화를 의미하며, 소프트웨어 생명 주기(SDLC)의 모든 단계와 연계되어 동작하며, 이들을 자동화시켜주는 통합된 도구의 집합을 제공**
- 프로그램의 구현과 유지보수 작업뿐만 아니라 분석과 설계 작업을 자동화함으로써 소프트웨어 생산성 문제를 해결한다.

- CASE의 목표

① 사용자 질의에 의해 시스템 문제의 생성을 자동화하는 것.
② 소프트웨어의 생산성 향상과 품질개선을 목표
③ 시스템의 분석, 설계, 구현, 디버깅 과정 등을 완전히 통합하는 것.

-CASE가 소프트웨어 개발자에게 제공하는 이점

· 구조적 기법의 실용화
· 소프트웨어/정보 공학의 개념을 적용
· 자동 테스트를 통하여 소프트웨어 품질을 향상
· 프로그램의 유지보수를 간단히 한다.
· 시스템 개발 과정의 속도를 빠르게 한다
· 빠른 프로토타입을 가능하게 한다.
· 소프트웨어 부품의 재사용을 가능하게 한다.
· 소프트웨어 형상관리 등의 작업 노력을 최소화한다.
· 소프트웨어 개발의 전 단계에 걸쳐서 표준화 제공
· 문서화를 제공한다.

-단계

① 상위 CASE : 계획과 분석 단계를 컴퓨터로 지원 (분석, 기본설계)
② 중위 CASE : 시스템 설계를 지원 (상세 설계)
③ 하위 CASE : 시스템 개발인 구현과 유지보수에 사용됨 (구축, 테스트)

-분류 by Carma McClure

① CASE Toolkit : 시스템 분석, 데이터베이스, 프로젝트 등의 설계 등 특정한 소프트웨어 개발 공정을 자동화하기 위한 도구 그룹
② CASE Workbench : 시스템의 분석, 설계, 구현까지의 일련의 작업 공정의 자동화를 지원하는 통합 그룹

-CASE의 기능

① 개인적인 컴퓨터 환경을 지원
② 소프트웨어 시스템의 문서화 및 명세화를 위한 그래픽 기능
③ 소프트웨어 생명 주기 전 단계의 연결
④ 소프트웨어 개발 및 유지 보수의 자동화를 수행하는 인공 지능의 사용

3장. 소프트웨어 분석 ... SDLC 2단계

1) 분석의 개요

- 사용자의 요구사항을 명확히 규정하고 이해하여 시스템의 특성을 반영
- 소프트웨어의 목적, 수행할 작업을 요구 조건으로부터 분명하게 명확화하는 단계
- 실제적인 개발과정의 첫 단계이며, 시스템 분석을 수행하기 위해 사용자, 업무, 목적, 타당성 등을 고려해야 함.
- 소프트웨어 분석 순서

① 문제 인식 : 정확한 사용자의 요구사항을 파악하기 위해 면담, 설문, 사용자의 문서 검토

② 평가와 종합 : 사용자의 요구사항 평가, 이에 대한 해결책을 종합

③ 모델 제작 : 구현될 시스템에 대한 데이터와 이들 데이터의 제어 흐름을 분석하고, 각 시스템에 대한 세부 단위에 대한 기능과 역할에 대해 분석, 이를 처리할 정보와 내용을 고려하여 모델을 제작.

④ 문서화와 검토 : 결과물로 요구사항 분석 명세서를 작성, 시스템에 대한 기능, 성능, 제약조건 등을 검토

- *그래픽 기법 단계적 세분화, *하향식 기법을 적용하며, 시스템의 타당성에 대한 검토, 시스템 자원 등에 대한 내용 분석

* 그래픽 기법 : DFD, DD, Mini-Spec, ERD, UML 도구 사용

* 하향식 기법(Top-down)

주어진 문제를 분석하여 시스템의 전체적인 구조와 필요로 하는 인터페이스를 파악하고, 하위 level로 진행하면서 세부적인 내용을 구현하는 방법으로, 상위 level에서 오류가 발생하는 경우에는 하위 시스템의 내용을 수정해야 하는 단점[10]을 가지며, 시스템의 전체적인 인터페이스가 미리 정리된다.

* 상향식 기법

최하위 level에서 사용되는 모듈을 분석하여 구현하고 상위 level로 진행하면서 필요로 하는 인터페이스를 확인해가면서 최상위 level에서 완성된 하나의 시스템을 구축하는 방법이며, 개발팀의 작업 속에 비례하여 시스템을 개발

10) 전체적인 수정이 필요할 수 있다. 이는 객체 지향식으로 문제 해결이 가능함

2) 구조적 방법론

- 시스템을 기능 중심으로 나누어 모듈을 개발하고, 이를 통합해 가면서 시스템을 완성하는 방법
- 전통적 소프트웨어 개발 방법이 주어진 문제를 분석한 다음 설계, 구현, 테스트의 단계를 거쳐서 소프트웨어를 개발하며, 이 과정에서 구조적 방법을 위한 기본 원리에는 추상화, 정형화, 분할과 정복, 계층적 순서화 방법을 사용
- 장점 / 특징

· 기능 중심의 모듈을 개발하고, 각 모듈을 통합하여 시스템을 완성한다.
· 모델, 도형, 도표 등의 시각적 표현을 사용하므로 분석 작업의 이해가 쉽다.
· 하향식 방법을 적용하므로 시스템 전체를 이해하기 쉽다.

- 단점

· 기능 변경은 사용자 요구사항 명세서의 분석 내용을 변경한다.
· 실행 과정이 유연하지 못하다.
· 데이터 구조보다 사용자 인터페이스를 중시한다.

2-1) 데이터 흐름도 (DFD, Data Flow Diagram)

- 그래픽 기법을 적용하여 시스템에 대한 분석을 단계적으로 세분화하여 나타내는 하향식 방법 (분할 정복)
- 표현을 위해 Process, Flow, Data Store, Terminal을 사용.

· 프로세스 (Process) : 처리하고자 하는 주요 기능
· 흐름 (Flow) : 진행 방향, 화살표 위에 주요 항목
· 데이터 저장 (Data Store) : 데이터 저장소, 데이터 항목들의 집합
· 단말 (Terminal) : 데이터의 입력원과 출력원 표시

- DFD의 작성규칙

· Parsimony rule : 최소한의 데이터 입력
· Persistance rule : 입력 데이터 지속적으로 생김
· Permanent rule : 모든 결과는 데이터 저장소에 저장
· Conservation rule : 입력 데이터 발생 시 이를 처리하여 출력 데이터가 발생할 흐름으로 작성
· Independence rule : 각각의 프로세스에 대한 내용만 확인한다.
· Ordering rule : 모든 데이터 순차적 처리
· Nature of change : 모든 데이터는 프로세스에 의해 합성, 변환되서 처리

- 시스템에 대한 주요 기능 중심으로 분석

2-2) 데이터 사전 (DD, Data Dictionary)
- DFD에 있는 데이터를 자세히 정의하고 기술
- DFD의 데이터 저장소에 대한 정의의 집합
- 소프트웨어가 사용하거나 생산한 모든 데이터에 대한 기술(description)을 포함[11)]
- 데이터의 중복성을 제거하고 모호성을 배제
- 구성

정의(Define)	= ~로 구성되어 있다
연결(And)	+
옵션(Optional)	( ) ← 생략 가능한 데이터
선택(Selection)	[|]
반복(Iteration)	{ }
주석(Comment)	**

2-3) 소단위 명세서 (Mini-Spec)
- DFD의 최하위 프로세스 단계를 지원
- 최하위 단계의 업무 내용에 대한 상세한 조건과 규칙을 상세히 표현
- 조건을 N개 사용 시 2^N개의 규칙을 표현
- 1page 또는 1/2page 정도의 분량을 기술 (1page = 6~80 line)
- 서술문장, 표, 그래프 등으로 나타내며, Decision Tree와 Decision Table 등을 사용

2-4) ERD (개체 간 관계 다이어그램, Entity Relation Diagram)
- 데이터 개체간의 관계를 표현
- 시스템에서 사용되는 개체와 개체의 구성, 속성. 개체 간의 관계를 표현
- 구성

Entity 개체집합	□ 레코드(Record = Tuple), 같은 타입에 속하는 개체들의 집합
Attribute 속성	○ 항목(Item), 현실 세계의 객체들이 가지고 있는 특성
RelationShip 관계집합	◇ 레코드 여러 개 → 파일(=relation, table) 개체들 간의 연결된 개체집합들의 관련, (1:1), (1:n), (n:m)

2-5) STD (State Transition Diagram)
- 시스템이 외부적 사건의 결과로 어떻게 행동하는가를 표현
- 시스템의 행위 정의
- 시스템에서 사건이 발생할 경우 시스템의 상태와 상태 간의 전이를 표현
- 구성

시스템의 상태 (사건)	□
상태의 전이	→

11) 주소파일 = { 이름, 학번, 주소, 성별 }

2-6) 정보공학 방법론
- 기업을 대상으로 분석과 설계에 정형화된 방법을 적용
- 기업 전체의 업무를 관리하고, 데이터 통합하여 처리 (DB에 대한 분석 포함)
- SDLC 기반으로 기업 모델, 데이터 모델, 프로세스 모델을 구축하기 위한 방법
- 단계
① ISP (Information Strategy Plan) : 기업 전체의 업무내용 파악하고 계획 수립
② BAA (Business Area Analysis) : 요구사항 분석, 전체 프로세스와 데이터 파악
③ BSD (Business System Design) : 시스템 구조, 데이터 구조, 사용자 인터페이스, 구체적인 절차(내용)
④ TD (Technical Design) : 세부적인 시스템의 기능
⑤ Implementation : 4세대 언어, Database, CASE 지원
⑥ Transition : 하드웨어 환경에 부합하거나 새로운 환경에 설치

2-7) CBD (Component Based Development 객체 지향 관련된 분석 방법)
- Sun-Java-EJB, MS-COM-C#, MS-MFC-C++
- 부품화된 소프트웨어, 일관된 인터페이스[12], 메타 데이터[13] 사용
- 기본적으로 요구 파악, 분석/설계, 구현, 테스트 단계를 지원
- CBD 기반 개발 환경을 위해 Component development[14], Component 인증, 형상 관리의 지원을 받아 CBD 개발

12) 정보은닉(Information Hiding) : 정보를 특정 인터페이스로만 접근 가능
13) 도움말 기능
14) 컴포넌트 영역 분석, 추출, 설계, 구현

4장. 소프트웨어 설계

1) 설계(Design)의 개요

-분석 단계의 산출물인 사용자 요구사항 분석 명세서에 대한 기능과 내용을 수행하기 위한 알고리즘과 데이터 구조 등을 작성

- 설계단계는 프로그램(모듈)에 대한 구조를 표현

- 프로그램 구조는 어떤 모듈을 호출하는 모듈의 수를 나타내는 팬 인(fan in)과 어떤 모듈에 의해 호출되는 모듈의 수를 나타내는 팬 아웃(fan out)으로 구성된다.

-설계 단계

① 데이터 설계 (Data design) : 시스템에서 사용되는 데이터 구조 정의함. (ERD)

② 구조 설계 (Architecture design) : 프로그램과 데이터, 프로시저 간의 관계 정립함. (DFD, DD, STD)

③ 인터페이스 설계 (Interface design) : 시스템과 사용자 간의 연결 표현. (DFD)

④ 프로시저 설계 (Procedure design) : 모듈에 대한 세부내용 작성 (Mini - spec)

-사용되는 데이터와 프로시저에 대한 분명하고 분리된 표현 포함

-독립적인 기능을 가진 모듈 단위 작성에 초점

-설계 지침

· Capsuling[15]은 약하게, Cohension[16]은 강하게 모듈을 작성.

· 단일 입구와 단일 출구를 가지도록 설계

· 모듈 간의 접속 관계를 분석, 복잡도와 중복성 제거

· 모듈의 기능이 예측 가능하고, 이식이 가능하도록 설계함

· 적절한 크기를 갖고 재사용성을 고려하며 설계

* Structure Programming : GoTo문 없이 순차(Sequence), 선택(Selection), 반복(Iteration)의 세 가지 구성요소로 프로그램을 작성하여 프로그램을 논리적 및 객관적으로 명확하게 작성

15) 결합도, 모듈 간 연관관계 의존 정도

16) 응집도, 하나의 모듈이 얼마나 잘 정의되어 있는가

2) 상향식 설계와 하향식 설계

- 상향식 설계

최하위 단계에서 수행하는 모듈에 대해 먼저 설계하고, 하위 레벨의 모듈들이 완성되면 이들은 결합하여 테스트하면서 상위 모듈과 결합. 상위 수준에서는 하나의 완성된 모듈을 구성

- 장점

· 프로젝트에 참여한 프로그래머의 집단과 관련된 문제를 해결하므로 효율적인 운영

· 시스템에서의 주된 인터페이스인 물리적 입출력 모듈에 대한 테스트가 초기에 이루어짐

- 단점

· 시스템 최상단의 가장 중요한 인터페이스인 중요 연결자와 제어구조를 나중에 처리

· 통합 환경 및 시기는 개발팀의 작업 속도에 비례

- 하향식 설계

시스템의 전체 구성도를 작성하여 모듈에 대한 모든 구조와 데이터를 파악, 하위 레벨로 진행하면서 단계적으로 자세한 기능적 내용 들을 완성해가면서 모듈을 설계

- 상위 레벨의 모듈에서 오류가 발생하면 연관된 하위 레벨 모듈을 모두 수정하는 단점이 존재, 객체지향 방식으로 해결 가능함.
- 구조도에서 *스터브라고 불리는 모델 모듈을 갖는 최상위 모듈을 가장 먼저 수행하고 점차적으로 하위 모듈 순으로 수행해나가며 상향식 설계의 단점을 해결

*스터브(Sturb) : 인터페이스를 확인할 수 있는 최소한의 기능을 가진 모듈

3) 모듈화

- 시스템을 독립적인 각 기능 부분으로 분해하여 나누어진 모듈 단위로 분할하여 주어진 문제를 해결하는 방법. 시스템의 복잡도 감소, 변경 용이, 프로그램 구현을 용이하게 할 수 있다.
- 특징

· 모듈 단위의 독립된 단위로 세분화하여 작성
· 한 프로그램을 독립된 단위로 분할
· 복잡한 문제를 해결 가능한 작은 단위로 분해하는 분할과 정복 개념을 사용
· 모듈 단위로 프로그램에 대한 정확성, 테스트가 용이

- 모듈의 독립성을 높이는 응집도, 결합도

① 응집도 (Cohesion)

· 하나의 모듈 내에 존재하는 기능 요소간의 관계를 나타내며, 하나의 모듈이 얼마나 잘 정의된 기능을 수행하는가에 대한 척도
· 종류 (응집도 위에 있을수록 강함)

(1) 기능적 (Functional) : 하나의 기능 요소만을 수행하는 경우
(2) 순차적 (Sequential) : 출력값이 다른 기능 요소의 입력으로 사용
(3) 통신적 (Communicational) : 2개 이상의 기능 요소에 동일한 입력을 사용하여 서로 다른 출력값을 가지는 경우
(4) 절차적 (Procedural) : 2개 이상의 기능 요소들이 순차적으로 수행
(5) 시간적 (Temporal) : 특정 시간대에 수행되는 기능 요소들을 모아놓고 이들을 제어하며 사용
(6) 논리적 (Logical) : 기능 요소들이 유사한 특성을 가지거나 특정 부류에 속하는 내용 포함
(7) 우연적 (Coincidental) : 서로 전혀 연관이 없는 기능 요소들을 모아놓고 제어하며 사용

② 결합도 (Coupling)

· 모듈 간의 상호의존관계, 연관관계를 나타내며, 설계에 대한 품질 평가 방법.
· 모듈 간의 결합도는 약해야 좋으며, 모듈 설계 시 의존 관계 고려
· 종류 (결합도 위에 있을수록 강함)

(1) 내용 (Content) : 한 모듈이 다른 모듈에 대한 무조건 분기, 변수의 내용을 강제 참조 및 변경
(2) 공통 (Common) : 2개 이상의 모듈이 공동 데이터 영역(변수)등 사용
(3) 제어 (Control) : 한 모듈의 인수가 제어 기능 수행하는 경우
(4) 스탬프 (Stamp) : 모듈 간의 인수 전달 시 구조체 값을 전달하는 경우
(5) 데이터 (Data) : 모듈에서 전달되는 값이 하나의 field거나 배열의 대표명 등을 사용하는 경우

4) 그래픽 설계 도구

① HIPO (Hierachy Input Process Output)

- 시스템 분석, 설계를 위한 문서화에 사용
- 소프트웨어 개발에 대한 문서화 기법으로 사용되며, 입력, 처리, 출력으로 표현
- 하향식 기법 적용. 소프트웨어에 대한 기능 구조 계층적으로 표현
- 소프트웨어에 대한 변경, 유지보수 용이
- 각 기능과 이에 대한 데이터의 의존 관계를 동시에 표현
- 구성

(1) 도식 목차 (VTOC, Visual Table Of Contents)

· 시스템 전체에 대한 기능과 데이터의 흐름을 표현

(2) 총괄 다이어그램 (OD, Overview Diagram)

· 각 모듈 단위에 대한 입력, 처리, 출력으로 구성

(3) 상세 다이어그램 (DD, Detailed Diagram)

· 총괄 다이어그램에 대한 상세 내용과 조건 등을 세부적으로 명세

② NS Chart (Nassi-Shneiderman)

- 프로그램 논리를 구조화시킨 계층적인 형태로 제공, 논리의 기술에 중점을 둠
- 하나의 입구와 하나의 출구를 가지며 순차, 제어, 반복의 제어구조로 구성되는 프로그램 구조를 표현
- 프로그램의 상세 설계를 위하여 사용하는 다이어그램 작성 방법
- 프로그램 순서도, HIPO의 상세 다이어그램, 의사 코드에 대한 대체 방법

③ PDL (Program Design Language)

- 프로그래밍에 사용되는 언어와 유사한 서술적인 표현에 의하여 프로그램, 설계 및 시스템의 검토에 사용하고 문서화로도 사용할 수 있도록 하며, 프로그램의 기능과 순서를 표현한 것으로 의사 코드를 사용하여 프로그램을 작성

④ 구조 차트 (Structure Chart)

- 시스템을 구성하는 처리 모듈들을 단계적으로 표현.
- 전체적인 프로그램의 구조를 파악하는데 도움, 프로그램을 상세하게 개발하는데에 중점을 둠
- 모듈의 종속 관계를 나타내어 프로그램의 관리를 효율적으로 수행

5장. 소프트웨어 구현

1) 구현의 개요

- 설계 단계에서 작성된 설계 명세서를 원시 코드로 변화하고, 디버깅과 원시 코드에 대한 단위 테스트(unit test)를 수행함.
- 코딩에 사용하는 프로그래밍 언어의 특성이나 코딩 스타일은 소프트웨어 품질과 유지 보수성에 영향을 미치는 중요한 요소
- 코딩 단계에서는 상세 설계를 프로그래밍 언어를 사용하여 코드로 변환하고, 이렇게 작성된 프로그램은 컴파일러에 의해 목적 코드로 변환. 목적 코드는 주 기억 장치에서 실행될 수 있는 명령어로 바뀜

Source Program	명령어(institution)로 구성됨. 객체지향 언어의 경우, [17]메시지
↓	컴파일러 ← 원시 프로그램을 컴퓨터의 기계어로 1:1 대응하여 변환 ① 렉시컬 분석 : 원시 코드의 각 문장을 토큰 단위로 변환 ② 구문 분석 : 문법적인 오류를 파악 ③ 중간 코드 생성 : 고급 언어의 명령어와 기계어 명령어 중간 형태 ④ 코드 최적화 : 사용되는 기억 장소의 양과 변수 고려해 중간 코드 수정 ⑤ 기억장소 할당 ⑥ 목적 코드 생성 : 중간코드를 어셈블리어, 기계어로 변환 ⑦ 에러 처리
Object Program	실행 불가능한 기계어
↓	링커(Linkage editor)
Load Module	실행 가능한 기계어
↓	로더(Loader) 주 기억장소 할당
주 기억장소	주 기억 장치에 적재되어 실행

17) 객체가 특정한 동작을 수행하도록 지시하는 것

6장. 소프트웨어 테스트

1) 개념

- 작성된 프로그램의 오류(error)를 찾기 위해 수행되는 과정
- 설계 내용의 명세화, 코딩 단계에 대한 궁극적인 검토를 수행
- 프로그램에 대한 사용자의 요구사항 만족도, 예상과 실제 결과의 차이점을 분석하는 과정
- 소프트웨어에 대한 품질 보증 활동으로 사용

2) 종류

① White Box Test

- 프로그램 내의 단계적 내용을 자세히 검토하며, 프로그램 상의 논리적 조건, 특정 반복문의 수행 조건, 특정 조건에 대한 명령어에 대해 확인하는 과정

(1) 기초 경로 테스트 (by Mcabe, Basic Path Test)

· 프로그램을 open시킨 상태에서 프로그램 상의 독립경로, 프로그램의 수행과 연계된 test case를 적용하여 검사를 수행

· 프로그램에 대한 복잡도를 계산 (복잡도 = 화살표 개수 - 노드 수 + 2)

· 프로그램에 대한 순서도(흐름도)를 작성하고, 논리적 복잡도를 측정하며, 각 독립된 경로들에 대한 기초 집합을 확인하고, 이들에 test case를 적용하여 프로그램에 대한 검사 수행

(2) 조건 테스트 (Condition Test)

· 프로그램 모듈 내에 있는 모든 논리적인 조건에 대해 검사

(3) 데이터 흐름 테스트 (Data flow test)

· 프로그램에서 사용된 변수와 변수의 변경 내용을 추적하여 명령어에 의해 정확히 변경되었는지 검사를 수행

(4) 루프 테스트 (Loop Test)

· 프로그램 상에 사용된 반복문에 초점을 맞추어 검사를 수행하고, 반복문의 초기값, 증가값, 반복 조건, 반복 조건의 경계에 대한 값을 분석

② Black Box Test

- 각 모듈에 대한 기능을 검사하는 단계

(1) 동치 분할 검사 (Equivalence Partitioning Test)

· 입력 데이터에 초점을 두고 검사 사례를 작성하여 검사를 수행. 타당 데이터와 비타당 데이터로 구분하여 검사

(2) 경계값 분석 (Boundary Value Analysis Test)

· 입력 데이터의 경계값에서 오류가 발생되는 빈도가 많으므로 유효 균등(동치)데이터와 무효 균등 데이터로 구분하여 검사.

(3) 원인 효과 그래프 (Cause Effect Graph Test)

· 특정 입력 데이터가 출력에 미치는 영향을 분석하여 그래픽으로 나타내고 특정 부류에 해당되는 데이터를 사용하여 검사를 수행

(4) 비교 검사 (Compare Test)

· 서로 다른 작성자가 작성한 동일한 기능을 수행하는 프로그램에 대해 입력에 대한 출력의 결과를 확인하는 검사

3) 위험 단계

① 단위 테스트 (Unit Test)

- 각 모듈에 대한 검사를 수행하며, 코딩(구현) 단계와 병행하여 모듈에 대한 에러를 검사하고, 각 모듈의 기능이 정확히 구현되었는가에 집중하여 검사를 수행함
- White Box 검사를 시행
- Driver 와 Stub를 확인하는 단계이며, Driver는 모듈에 전달할 데이터를 수집하고, 이를 모듈에 전달하여 결과를 출력하는 작업
- Stub는 인터페이스를 확인할 수 있는 최소한의 기능을 가진 모듈이며, 하위 모듈과의 인터페이스를 확인하기 위해 사용

② 통합 테스트 (Intergration Test)

- 각 모듈들에 대한 통합검사를 수행하는 단계로서, 설계 단계와 연계되어 테스트를 수행
- 통합 과정에서 발생되는 오류와 모듈들의 인터페이스와 연관된 오류를 테스트하며 Black Box Test를 수행
- 통합 검사를 위한 단계는 Driver와 Stub를 확인하고, Stub를 실제 모듈로 대체하고, 모듈을 통합해 가면서 테스트를 수행하며, 일부 통합된 모듈에 대해 다른 Stub와 통합하고, 하위 Level로 진행하면서 오류를 테스트

③ 정당성 테스트 (Validation Test)

- 사용자의 요구 사항이 구현된 모듈과 일치하는가를 test하며, Black Box Test를 수행
- Alpha Test, Beta Test, Gamma Test로 구분

* Alpha Test : 개발자의 위치에서 고객이 시스템 test
* Beta Test : 고객의 위치에 시스템이 설치된 후 사용자가 test
* Gamma Test : 고객의 위치에서 다수의 사용자가 시스템을 test

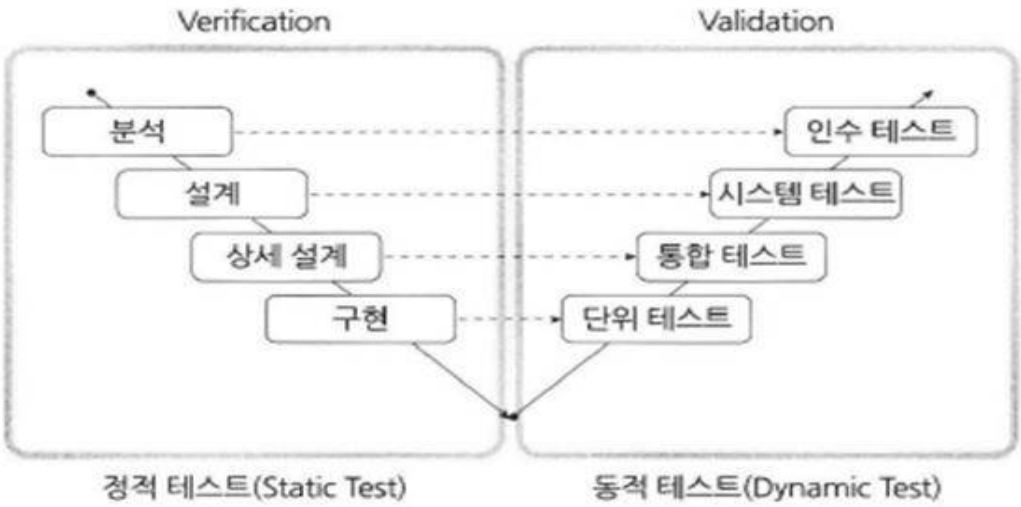
④ 시스템 테스트 (System Test)

- 구현된 시스템에 대한 test를 수행함
- 복구 테스트(recovery) : 소프트웨어가 여러 방법에 의해 손상되도록 한 다음 정상적으로 복구가 되는가 확인
- 보안 테스트 (security) : 시스템 내의 보호 메커니즘이 부적당한 침투로부터 보호될 수 있는가 확인
- 강도 테스트 (stress) : 프로그램을 비정상적인 상황에서 실행시키는 테스트로, 허용 가능한 최대 사용자의 수보다 많은 수를 처리하는 방법
- 성능 테스트 (performance) : 통합 시스템에서 소프트웨어에 대한 실시간 성능에 대해 테스트, 모든 단계에서 수행
- Big Bang Test : 모든 모듈을 통합하여 테스트를 수행
- Regression Test : 새로운 모듈을 테스트 수행할 때, 이와 연계된 다른 모듈을 다시 테스트 하는 것.

- Experience-based test : 테스터, 개발자, 사용자들의 경험을 모두 갖고 시스템에 대한 테스트를 수행 (test case 작성)
- 비기능 테스트 : 시스템의 운영을 위해 사용되는 기준 (품질과 연관, 인터페이스, 성능, 요구사항)

* 기능 테스트 : 설계와 연관, 입출력, 저장, 연산, 업무기능
* Throughput (출력양): 단위 시간 동안에 처리하는 작업의 양(성능과 연관)

* V 모델 기반 테스트



라이프 사이클 모든 단계가 테스트와 연결되어 있음을 나타내며, 소프트웨어 라이프 사이클 단계인 분석, 설계, 코딩 단계를 테스트 단위의 단위 테스트, 통합 테스트, 인수 테스트와 연계하여 V자 형태로 연결한 모델

* 위험 기반 테스트 (Risk based test)

시스템에서 각 부분별로 발생할 위험을 점수화하여 위험이 높게 책정된 항목이나 자원을 집중적으로 테스트

4) 디버깅

-오류를 발견하여 정정하는 과정

- 강행법 (Brute Force)

오류가 발생 될 때까지 방치 해두었다가 오류 발생 시 수정하는 작업.

- 백 트래킹 (BackTracking)

오류가 발생된 시점에서 역으로 오류 발생 원인을 추적해가면서 오류를 수정하는 작업.

-원인 효과 제거 (Cause Effect Elimination) (Black Box와 연계)

오류 발생의 원인에 대한 가정을 작성. 오류 발생과 연관된 데이터를 사용하여 오류를 수정하는 작업

7장. 소프트웨어 유지보수

1) 유지보수 (Maintenance)

- 시스템에 대한 개선 및 수정 사항에 대한 시스템의 수정, 감시, 평가
- 시스템에 대한 계획의 목적 달성과 계획의 실행 상태를 점검하는 단계
- 시스템의 적절한 유지에 대한 감사(audit)를 수행
- 유지보수성을 증대시키기 위한 설계 방법이 중요
- 종류

· 수정 (Corrective) (21%)

소프트웨어에서 발생되는 일반적인 오류에 대한 수정 작업

· 적응 (Adaptive) (25%)

소프트웨어에 대한 새로운 하드웨어와 소프트웨어 환경의 변화에 대한 유지보수 작업

· 완전 (Perfective) (50%)

시스템 사용 중에 발생되는 새로운 기능의 추가와 현 기능에 대한 개선 작업

· 예방 (Preventive) (4%)

소프트웨어의 유지보수성과 신뢰성 증대를 위한 예방적 형태의 유지보수 활동

- 소프트웨어 진화의 원리

· 친숙성의 법칙 : 소프트웨어는 진화에 의해 버전이 변경되어 사용자가 계속 사용

· 안전상태 보존의 법칙 : 시스템이 많은 의사 결정으로 인해 생산되며,
일정한 생산성 향상과 안정적인 인력 지원

· 자신 규제의 법칙 : 소프트웨어는 자신만의 고유한 특성이 존재

· 복잡도 증가의 법칙 : 소프트웨어에 대한 수정과 변경이 많이 발생하면 복잡도 ↑

· 지속적인 변경의 법칙 : 사용자 요구사항에 의해 변경이 수행되며 이에 의해 시스템이 진화

· 지속적인 성장의 법칙 : 소프트웨어에 대한 기능이 다양하게 추가되어 사용자의 편의성 증대

· 품질 쇠퇴의 법칙 : 유지보수가 적절히 수행되지 않으면 소프트웨어 품질 저하

· 시스템 피드백의 법칙 : 사용자의 요구사항에 대한 지속적인 유지보수에 의해
소프트웨어에 대한 위기 감소

-소프트웨어에 대한 유지보수 비용 산정

· Belady-Lehman의 방법 (M : 유지보수)

$$M = P + K^{(c-d)}$$

(P: 생산성(productirity) | K: 경험적 상수 c: 복잡도(complexity) d: 친밀도(density))

·COCOMO에 의한 방법

$$M = ACT * DE * EAF$$

(ACT : 연간 수정 발생, Annual Change Traffic)

(DE : 노력(인/월), Development Efforts)

(EAF : 노력 조정 수치 , Effort Adjustment Factor)

8장. 객체지향 방법론

1) 객체지향의 개요

-객체 지향 방법에서 가장 중요한 내용은 객체(object)이며, 객체의 구체적인 행위를 지시하기 위한 메시지, 그리고 상속에 대한 개념.
-상속으로 인해 상위 모듈에서 수정을 하면 하위 모듈에서는 수정할 필요가 없으므로 하향식 대규모 프로젝트에 적합한 방법
-객체지향 방법론은 데이터와 연산을 분리하여 다루는 기존의 절차 언어와 차별화
-객체지향 방법론의 장점
· 실세계를 모델화하여 분석자와 사용자 양측의 상호 이해가 쉬워짐
· 여러 객체를 동시에 실행하는 방법을 사용
· 데이터와 연산을 묶은 캡슐화로 인해 유지보수에 대한 관리가 용이
· 데이터 구조와 관련된 연산을 은닉하여 불필요한 구조 숨김
· 클래스를 이용한 시스템 확장이 용이
· 인터페이스가 잘 정의됨, 상속으로 인해 특정 메소드를 상속 및 변경 가능
· 구현과 무관하게 클래스를 선택하게 하여 재사용을 증진함.

※RUP(Rational Unified Process)
대표적인 객체지향 방법론. UML을 기반으로 하며, 반복 및 점증적으로 프로세스를 진행하고, 샘플 산출물과 시스템에 적용되는 다양한 활동을 바탕으로 한 지식 베이스를 가지고 사용자가 개발과정에서 프로세스 요소들을 선택하여 사용

개념 (Inception)	시스템의 목표, 범위 적용 업무를 규정하며, 요구사항을 파악하고 개발 범위를 정함
↓	
상세 (Elaboration)	시스템에서 사용될 업무 활동을 구체적으로 정의, 필요한 자원을 파악하여 요구사항 명세화를 수행하고 실행 가능한 아키텍처에 대한 프로토타입 구현
↓	
구축 (Construction)	설계와 구현 과정을 통해 시스템을 구축
↓	
전이 (Transition)	구축된 시스템의 교육지원, 유지보수, 사용자에게 시스템을 인도 형상 관리의 지원

* 개체(entity) : 단독으로 존재 가능. 어떠한 의미를 나타냄
* 객체(object) : 필요한 데이터 구조를 기반으로 수행되는 함수들을 가진 소프트웨어 모듈

2) 객체 지향 시스템

-객체(object) : 필요한 데이터 구조와 이의 처리를 위한 함수들을 가진 소프트웨어 모듈
프로그램은 현실 세계를 추상화한 모델, 객체는 현실 세계에서 행동하는 주체인 대상
각 객체는 상태(state)를 가지며, 객체의 수행에 의해 이는 변경된다.
객체의 외부적인 행위는 오퍼레이션, 내부적인 구현을 메소드라고 한다.
객체는 캡슐화(encapsulation), 데이터 추상화(data abstraction)를 지원하며,
객체는 전용 데이터와 이를 처리하기 위한 메소드로 구성됨.
전용 데이터는 객체의 물리적 구성 요소, 메소드는 객체의 행위.

* 캡슐화(encapsulation)
모듈의 구현 내용을 사용자에게 최소한으로 제공하며, 모듈 사이의 인터페이스가 적을수록 상호 종속성은 더욱 줄어든다. 각 객체는 캡슐화되어 존재하므로 외부에서 사용자에게 제공되는 객체 행위는 구체적이 아닌 추상적인 특성을 지니게 된다. 객체는 객체 내부의 상태와 그 객체가 수행하는 오퍼레이션의 구현에 대해서는 감추어져 있다.
* 추상화(abstraction)
사용자가 복잡한 문제를 단순하게 다루기 위해 구현의 모든 사항을 단순화시켜 표현하는것에 중점을 둔다. 객체에 대한 추상화는 객체의 구현 내용을 감추고 어떤 기능에 의해 수행될 것인가를 나타낸다.

-클래스(class)
하나 이상의 유사한 객체들을 모으고, 이들의 공통된 특성을 나타낸 것으로, 객체는 클래스로부터 만들어진 인스턴스(instance)다, 한 클래스에서부터 만들어진 객체들은 모두 동일한 행위를 가지며, 클래스가 동일한 형태의 객체를 생성한다는 의미에서 템플릿이라고도 한다.
- 클래스 설계원칙

개방 폐쇄의 원칙 (OCP, Open Close Principal)	하위 클래스는 상위 클래스를 간섭X
단일 책임의 원칙 (SRP, Single Resposibility Principal)	하나의 클래스는 하나의 책임만 수행하도록 설계한다.
인터페이스 분리 원칙 (ISP, Interface Segregation Principal)	클래스는 자주 변경되는 method에 의존하지 않는다.
의존 관계 역전의 법칙 (DIP, Dependency Inversion Principal)	자주 변경되는 클래스가 아닌 추상 클래스에 의존한다.
리스코프 교환 원칙 (LSP, Liskov Substitution Principal)	상위 클래스가 사용되는 곳에서는 하위 클래스가 사용된다.

* 메시지 : 객체에 특정 동작을 수행하도록 지시하는 명령
* 메소드 : 객체가 받은 메시지를 가지고 수행해야 하는 객체의 구체적인 연산
* 상속 : 한 클래스에서 정의한 메소드들을 다른 클래스에 계승하며, 소프트웨어의 재사용을 가능하게 함.
* 위임 : 상속 이외에 정보와 코드를 공유할 수 있는 방법. 객체가 처리해야 하는 오퍼레이션의 일부를 다른 객체들에게 처리하도록 할당, 그 권한을 부여
* 다형성 : 함수 이름/연산자가 여러 목적으로 사용되는 것
* 관계성 : 두 개 이상의 개체형에서 데이터를 서로 간에 참조하는 관계

3) UML (Unified Modeling Language)

-객체지향 분석과 설계에서 사용되는 시스템에 대한 구체적인 내용과 동작을 표현

-사물(Thing), 관계(Relation), 도표(Diagram)로 구성

Thing	구조 사물 (Structural thing)	객체 지향 모델의 개념적인 내용과 물리적인 요소들을 표현하는 것 클래스, 객체, 인터페이스, [18]유즈 케이스, [19]노드 등의 정적인 내용
	행위 사물 (Behavioral thing)	interaction : 객체가 주고받는 메시지
		state machine : 객체 내의 메소드 수행에 의한 상태와 상태 변화 순서 등을 사각형으로 나타냄.

Realation	연관 관계 (association)	객체들 간의 관계를 의미. 객체들 사이의 대응 관계를 n:n으로 표현하며, ERD 관계 (1:1, 1:n, n:n)와 동일하게 나타냄
	포함 관계 (inclussion)	객체에 포함된 다른 객체들 간의 관계로서 하위 클래스와 상위 클래스 간의 관계를 표현

Diagram	Usecase Diagram	·시스템의 기능적 요구 사항에 대한 [20]baseline을 제공하고 사용자 시각에서 본 시스템의 범위와 기능을 명세 ·프로젝트에 대한 개발 범위, 사용자 요구사항 정의, 시스템의 세부 기능 정의 시 사용 · Actor : 시스템의 교류 및 상호작용을 수행하며, 시스템에 서비스를 요청하거나 정보를 제공하는 대상 · Usecase : 시스템의 행위이며, Actor의 요구에 대해 시스템이 수행되는 서비스 · Relation : Actor와 Actor, Usecase 간의 관계 표현
	Class Diagram	클래스에 대한 기본적인 내용인 클래스 이름, 속성, 메소드에 대한 내용 포함
	Object Diagram	클래스에서 파생된 객체를 표현하며, 객체의 메소드와 객체의 구현 정보 표현
	Sequence Diagram	시스템에서 사용되는 클래스들간의 관계로 나타내며, 클래스들 간의 동적인 관계를 표현하여 class와 class, class와 object, object와 object의 연관관계를 표현
	Collaboration Diagram	객체들 간의 동적 관계를 나타내며, 객체들 간의 상호관계를 표현하므로, interface diagram 이라고도 한다.
	Package Diagram	시스템에서 사용되는 클래스들을 모아 하나의 모델링 개체로 표현
	State Diagram	시스템에서 표현되는 객체들의 상태와 상태 간의 전이에 대해 표현
	Deployment Diagram	시스템 상의 하드웨어 관계를 표현
	Component Diagram	클래스 자신의 세부적인 구현 정보를 명세
	Activity Diagram	시스템의 전체 흐름, 작업절차(분기, 제어) 관계들을 표현

18) 사용자 관점에서 소프트웨어 시스템에 대한 범위와 기능에 대해 설명 및 정의하며, 소프트웨어 시스템의 기능적 요구 사양에 대한 내용을 자세히 표현함

19) 소프트웨어 실행을 위한 물리적인 요소, 컴퓨터 장치와 관련된 하드웨어를 의미함

20) 개발자에 의해 합의된 구체적인 내용을 기술

4) 객체지향 분석과 설계

- 객체 지향 분석은 사용자 요구사항 파악, class 식별, class 계층화, 개체 연결, 객체 관계 작업을 수행
- 객체 지향 설계 단계에서는 객체에 대한 속성, 연산, 속성과 연관된 데이터 구조, 클래스 계층과 연관된 메시지 정의 등의 작업을 수행

① Coad와 Yourdon의 방법

·객체와 클래스에 대한 상속과 연관관계, 포함관계 포함하며, DFD를 사용

·분석 단계에서는 객체와 클래스 식별, 객체와 클래스에 대한 관계표시, 객체들을 주제별로 분류하고 이들에 대한 메시지 정의, 객체에 대한 속성을 정의하고, 이들의 관계를 지정하며, 시스템이 사용할 역할을 명세함

·설계 시에 human interaction, problem domain, test management, data management 작업 수행

② Boach의 방법

·DFD를 사용하여 객체 지향 분석과 설계작업을 수행하며, 전체적인 시스템 가시화를 수행하고, 거시적 개발 프로세스와 미시적 개발 프로세스를 포함하며, 설계를 위한 문서화 기법을 강조

·분석 단계에서는 클래스와 객체에 대한 분석 및 식별을 수행하고, 객체에 클래스의 의미 식별, 이들에 대한 관계를 파악하고 클래스에 대한 계층 정의와 클래스들의 Clustering 작업 후, 객체와 클래스에 대한 구현 작업을 수행

·설계 단계에서는 거시적 및 미시적 개발 프로세스를 포함하여 단계적인 개발작업을 수행하고, 미시적 개발 단계에서는 architecture 세부 설계, 이에 의한 정책설계와 release 단계를 포함

③ Jacobson의 방법

·시스템에 대한 사용자와 그들의 책임 구현 부분을 확인하고, 이들에 대한 내용, 범위, 기능을 확인

·Actor의 책임과 Actor에 대한 usecase를 식별하고, 이들에 대한 내용을 명세

·Usecase 사용한 모델을 구축

④ Rumbaugh의 방법

·모든 소프트웨어 구성요소들을 그래픽 표기법 사용하여 객체를 모델링하며, OMT 기법을 사용 (Object Modeling Technique)

·Object Modeling은 객체에 대한 연관화, 포함관계를 파악하고, dynamic modeling은 STD를 사용하여 객체 수행에 의한 행위를 나타내고, Functional modeling은 DFD를 사용하여 객체들에 대한 프로세스, 데이터, 제어관계 표현