1. 요구사항 확인

소프트웨어 생명 주기

1. 폭포수 모형 (Waterfall Model)
    - 각 단계를 확실히 매듭짓고 그 결과를 철저히 검토 후 승인과정 거친 후 다음단계 진행
    - 고전적 생명 주기 모형
    - 다음단계 수행을 위해 결과물이 명확하게 산출되어야 함

2. 프로토타입 모형 (Prototype Model)
    - 실제 개발될 소프트웨어에 대한 견본품 (Prototype) 제작하여 최종 결과물 예측

3. 나선형 모형 (Spiral Model)
    - 여러 번의 소프트웨어 개발 과정을 거쳐 점진적으로 완벽한 최종 소프트웨어 개발하는 모형
    - 폭포수, 프로토타입 모델에 위험 분석 기능 추가한 모형
    - 요구사항 추가 가능
    - 유지보수 필요 없음
    - 계획 → 위험 분석 → 개발 및 검증 → 고객 평가

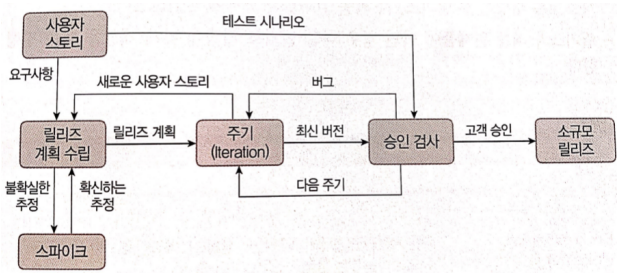
4. 애자일 모형 (Agile Model)
    - 고객의 요구사항 변화에 유연하게 대응하도록 일정 주기를 반복하면서 개발하는 모델
    - 고객과의 소통이 중요
    - 폭포수와는 대조적
    - 기업 활동 전반에 걸쳐 사용
    - 스크럼, XP, 칸반, Lean, 기능 중심 개발 (FDD : Feature Driven Development)
    - 4대 핵심 가치
        1. 프로세스와 도구 < 개인과의 상호작용
        2. 방대한 문서 < 실행되는 SW
        3. 계약 협상 < 고객 가치
        4. 계획 < 변화에 대한 반응

XP (eXtreme Programming) 기법

- 고객의 요구사항에 유연하게 대응하기 위해 고객 참여와 개발 과정의 반복 극대화하여 개발 생상성 향상시키는 방법
- 짧고 반복적인 개발 주기, 단순한 설계, 고객의 적극 참여
- 릴리즈 기간의 짧은 반복

- 5대 핵심 가치
  1. 의사소통 (Communication)
  2. 단순성 (Simplicity)
  3. 용기 (Courage)
  4. 존중 (Respect)
  5. 피드백 (Feedback)

- XP 개발 프로세스



  릴리즈 계획 수립 (계획) → 주기 (진행) → 승인 검사 (검사) → 소규모 릴리즈 (출시)

- XP 주요 실천 방법
  1. Pair Programming (짝 프로그래밍)

     다른사람과 함께 프로그래밍 → 공동의 책임감
  2. Collective Ownership (공동 코드 소유)

     개발 코드에 대한 권한과 책임을 공동으로 소유
  3. Test-Driven Development (TDD : 테스트 주도 개발)

     실제 개발 전 테스트 케이스 작성, 자동화된 테스팅 도구 사용
  4. Whole Team (전체 팀)

     개발에 참여하는 모든 구성원들이 역할에 대한 책임을 가져야함
  5. Continuous Integration (계속적인 통합)

     모듈 단위로 나뉜 코드들을 작업이 끝날때 마다 지속적으로 통합
  6. Refactoring (리팩토링)

     프로그램 기능 변경 없이 지속적으로 재구성

     쉽게 이해하고 수정하여 빠르게 개발 가능
  7. Small Releases (소규모 릴리즈)

     릴리즈 기간을 짧게 두어 고객의 요구사항에 빠르게 대응

요구사항 정의

1. 기능 요구사항 (Functional Requirements)
    - 시스템의 기능이나 수행에 관련된 요구사항
    - 시스템의 입 출력과 관련
    - 시스템의 데이터 저장 및 연산과 관련/
    - 시스템이 반드시 수행해야 하는 기능
    - 사용자가 시스템을 통해 제공받기를 원하는 기능

2. 비기능 요구사항 (Non-functional Requirements)
    - 시스템의 품질이나 제약사항과 관련된 요구사항
    - 시스템 장비 구성 요구사항
    - 성능, 인터페이스, 테스트, 보안, 데이터 구축에 필요한 요구사항

3. 사용자 요구사항 (User Requirements)
    - 사용자 관점에서 본 시스템이 제공해야 할 요구사항
    - 친숙한 표현으로 이해하기 쉽게 작성

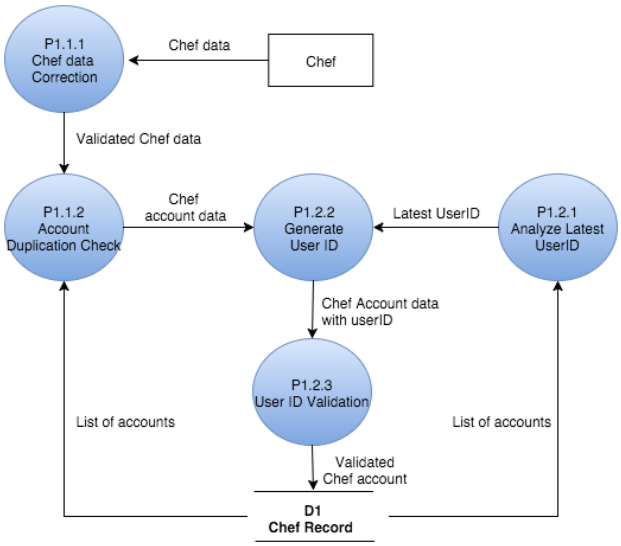
4. 시스템 요구사항 (System Requirements)
    - 개발자 관점에서 본 시스템 전체가 사용자와 다른 시스템에 제공해야 할 요구사항
    - 전문적이고 기술적인 용어로 작성
    - 소프트웨어 요구사항과 같은 의미

요구사항 분석

1. 요구사항 분석 (Requirement Analysis)
    - 개발 대상에 대한 사용자의 요구사항을 이해하고 문서화
    - 사용자 요구의 타당성 조사, 비용과 일정에 대한 제약 설정
    - 사용자 요구를 정확하게 추출하여 목표 정함

2. 구조적 분석 기법
    - 자료의 흐름과 처리를 중심으로 하는 요구사항 분석 방법
    - 도형 중심의 분석용 도구와 분석 절차를 이용하여 요구사항 파악 후 문서화
    - 하향식 방법 사용하여 세분화
    - 분석의 중복 배제 가능
    - 자료 흐름도 (DFD), 자료 사전 (DD), 소단위 명세서 (Mini-Spec), 개체 관계도 (ERD), 상태 전이도 (STD), 제어 명세서

3. 자료 흐름도 (DFD : Data-Flow Diagram)
    - 요구사항 분석에서 자료의 흐름 및 변환 과정과 기능을 도형 중심으로 기술하는 방법
    - = 자료 흐름 그래프 = 버블 차트
    - 구조적 분석 기법에 사용



1. 프로세스 (Process) : 자료를 변환시키는 시스템의 처리과정을 나타냄
2. 자료 흐름 (Data Flow) : 자료의 이동이나 연관 관계
3. 자료 저장소 (Data Store) : 시스템에서 자료 저장소 (File, DB)를 나타냄
4. 단말 (Terminator) : 시스템과 교신하는 외부 개체

4. 자료 사전 (DD : Data Dictionary)
    - 자료 흐름도의 자료를 더 자세히 정의 및 기록
    - = 데이터의 데이터 = 메타 데이터

1. = : 자료의 정의
2. + : 자료의 연결
3. ( ) : 자료의 생략
4. [ ] : 자료의 선택
5. { } : 자료의 반복
6. * * : 자료의 설명

UML 개요

- 시스템 분석, 설계, 구현 등 시스템 개발 과정에서 원활한 의사소통을 위해 표준화한 객체지향 모델링 언어
- OMG (Object Management Group)에서 표준으로 지정
- 사물 (Things) + 관계 (Relationship) + 다이어그램 (Diagram)

1. 사물 (Things)

   관계가 형성될 수 있는 대상

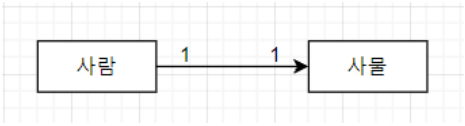
   - 구조 사물 (Structural Things)
   - 행동 사물 (Behavioral Things)
   - 그룹 사물 (Group Things)
   - 주해 사물 (Annotation Things)

2. 관계 (Relationship)

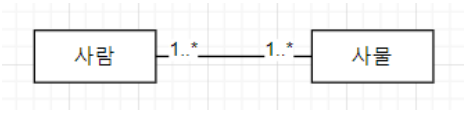
   사물과 사물의 연관성 표현

   - 연관 관계 (Association)

     2개 이상의 사물이 서로 관련되어 있는 것

     : 사람이 사물을 소유하는 관계

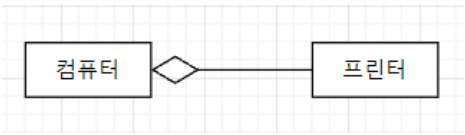
     

     : 사람과 사물은 관계를 가짐

   - 집합 관계 (Aggregation)

     하나의 사물이 다른 사물에 포함되어 있는 관계

     포함하는 쪽과 포함되는 쪽은 독립적

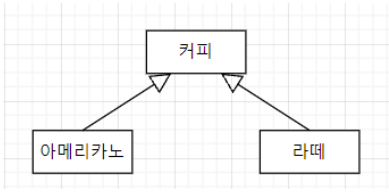
     : 프린터는 컴퓨터에 속해있음

   - 포함 관계 (Composition)

   - 일반화 관계 (Generalization)

     하나의 사물이 다른사물에 비해 더 일반적이거나 구체적인 관계

     부모 - 자식 관계

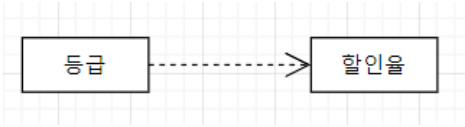
     

     : 커피에는 아메리카노와 라떼가 있다..

   - 의존 관계 (Dependency)

     서로에게 영향을 주는 짧은 시간 동안만 연관을 유지하는 관계

     소유 관계는 아니지만 사물의 변화가 다른 사물에도 영향을 미침

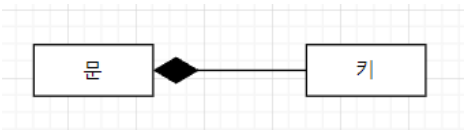
     

     : 등급에 따라 할인율을 다르게 적용

   - 실체화 관계 (Realization)

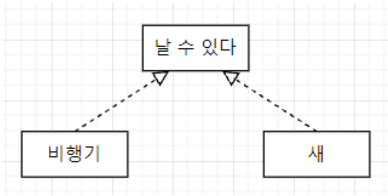
포함하는 사물의 변화가 포함되는 사물에 영향을 미치는 관계

독립적이지 않고 생명주기를 함께함



: 문이 사라지면 키도 필요하지 않게됨

사물이 할 수 있거나 해야 하는 기능, 서로에게 그룹화 할 수 있는 관계



: 비행기와 새는 날 수 있다는 행위로 그룹화 가능

3. 다이어그램 (Diagram)
   - 사물과 관계를 도형으로 표현
   - 가시화된 View를 통해 의사소통에 도움

   1. 구조적 다이어그램 : 정적 모델링에 사용
      a. **클래스 다이어그램** (Class Diagram)
         : 클래스와 클래스 사이의 관계를 표현
      b. **객체 다이어그램** (Object Diagram)
         : 객체와 객체 사이의 관계로 표현
         : Rumbaugh 객체 지향 분석 기법에서 정적 모델링에 활용됨
      c. **컴포넌트 다이어그램** (Component Diagram)
         : 컴포넌트 (구현 모듈) 간의 관계 또는 인터페이스 표현
         : 구현 단계에서 사용
      d. **배치 다이어그램** (Deployment Diagram)
         : 결과물, 프로세스와 같은 물리적 요소들의 위치 표현
         : 구현 단계에서 사용
      e. **복합체 구조 다이어그램** (Composite Structure Diagram)
         : 클래스나 컴포넌트가 복잡한 구조를 갖는 경우 내부 구조 표현
      f. **패키지 다이어그램** (Package Diagram)
         : Use-case나 클래스 등의 모델 요소들을 그룹화한 패키지들의 관계 표현

   2. 행위 다이어그램 : 동적 모델링에 사용
      a. **유스케이스 다이어그램** (Use Case Diagram)
         : 사용자의 요구 분석, 기능 모델링 작업에 사용
         : Actor, Use Case로 구성
      b. **시퀀스 다이어그램** (Sequence Diagram)
         : 상호작용하는 시스템, 객체들이 주고받는 메시지 표현

c. **커뮤니케이션 다이어그램** (Communication Diagram)

: 동작에 참여하는 객체들이 주고받는 메시지와 객체들 간의 연관 관계 표현

d. **상태 다이어그램** (State Diagram)

: 하나의 객체가 자신이 속한 클래스의 상태 변화 혹은 다른 객체와의 상호작용을 통해 변화되는 상태를 표현

: Rumbaugh 객체지향 분석 기법에서 동적 모델링에 활용

e. **활동 다이어그램** (Activity Diagram)

시스템이 어떤 기능을 수행하는지 객체의 처리 로직이나 조건에 따른 처리의 흐름을 순서에 따라 표현

f. **상호작용 개요 다이어그램** (Interaction Overview Diagram)

상호작용 다이어그램의 제어 흐름 표현

g. **타이밍 다이어그램** (Timing Diagram)

객체 상태 변화와 시간 제약을 명시적으로 표현

3. 스테레오 타입 (Stereo Type)

UML에서 기본적인 기능 외 추가적인 기능 표현

1. << include >> : 포함 관계
2. << extend >> : 확장 관계
3. << interface >> : 인터페이스 정의
4. << exception >> : 예외 정의
5. << constructor >> : 생성자 역할 수행

S/W 공학의 발전적 추세

1. 소프트웨어 재사용 (SW Reuse)
   - 이미 개발되어 인정받은 소프트웨어를 다른 소프트웨어 개발이나 유지에 사용
   - 품질과 생산성 높이는 방법
   1. 합성 중심 (Composition-Based)

      블록을 만들어 끼워 맞춰 완성시키는 방법
   2. 생성 중심 (Generation-Based)

      추상화 형태의 명세를 구체화하여 프로그램을 만드는 방법

2. 소프트웨어 재공학 (SW Reengineering)
   - 기존 시스템을 이용하여 더 나은 시스템 구축, 새로운 기능 추가하여 성능 향상
   - 기존 소프트웨어의 데이터 및 기능의 개선을 통해 유지보수성과 품질 향상
   - 이점 : 품질 향상, 생산성 증가, 수명 연장, 오류 감소

3. CASE (Computer Aided Software Engineering)
    - 소프트웨어 개발 과정을 컴퓨터와 소프트웨어 도구로 자동화
    - 소프트웨어 생명 주기의 전체 단계 연결 및 자동화 통합 도구 제공
    - 주요 기능
        1. 소프트웨어 생명 주기 전 단계 연결
        2. 다양한 소프트웨어 개발 모형 지원
        3. 그래픽 지원

비용 산정 기법 (LOC)

1. 상향식 비용 산정 기법
    - 세부적인 작업 단위별 비용을 산정 후 집계하여 전체 비용 산정
    - LOC 기법, 개발 단계별 인월수 기법, 수학적 산정 기법

2. LOC (원시 코드 라인 수, source Line Of Code) 기법
    - 노력 (인월) = 개발 기간 * 투입 인원 = 총 코드 라인 수 / 1인 월평균 생산 코드라인
    - 개발 비용 (총 인건비) = 노력 (인월) * 1인 월평균 인건비
    - 개발 기간 = 노력 (인월) / 투입 인원
    - 생산성 = 총 코드 라인 수 / 노력 (인월)

> 📌 예측된 총 라인수 30000라인, 개발에 참여할 프로그래머 5명, 평균 생산량 월 300라인 → 개발에 소요되는 기간?
>
> 노력 (인월) = 총 코드라인 수 / 1인 월 평균 생산량 = 30000 / 300
> 개발 기간 = 노력 (인월) / 투입 인원 = 100 / 5 = 20개월

수학적 산정 기법

1. 수학적 산정 기법
    - 상향식 비용 산정 기법중 하나
    - = 경험적 추정 모형 = 실험적 추정 모형

2. COCOMO (COnstructive COst MOdel) 모형
    - LOC에 의한 비용 산정 기법
    - LOC를 예측한 후 소프트웨어 종류에 따라 다르게 책정되는 비용 산정 방정식에 대입하여 비용을 산정
    - 결과는 프로젝트 완성에 필요한 노력으로 나타냄

- 보헴 (Boehm)이 제안

- 소프트웨어 개발 유형
  1. 조직형 (Organic Mode)

     기관 내부에서 개발된 중 소 규모의 소프트웨어

     5만 라인 이하의 소프트웨어 개발

     사무 처리용, 업무용, 과학용 응용 소프트웨어 개발에 적합

  2. 반분리형 (Semi-Detached mode)

     조직형과 내장형의 중간형 소프트웨어

     30만 라인 이하의 소프트웨어 개발

     컴파일러, 인터프리터와 같은 유틸리티 개발에 적합

  3. 내장형 (Embedded Mode)

     초대형 규모의 소프트웨어

     30만 라인 이상의 소프트웨어 개발

     신호기 제어 시스템, 미사일 유도 시스템, 실시간 처리 시스템 등의 시스템 프로그램 개발에 적합

- 모형 종류
  1. 기본형 (Basic COCOMO)

     소프트웨어의 크기와 개발 유형만을 이용하여 비용 산정

  2. 중간형 (Intermediate COCOMO)

     기본형 COCOMO의 공식을 토대로 사용

     제품 특성, 컴퓨터 특성, 개발 요원 특성, 프로젝트 특성에 의해 비용 산정

  3. 발전형 (Detailed COCOMO)

     중간형 COCOMO를 보완하여 만들어짐

     개발 공정별로 자세하고 정확하게 노력 산출하여 비용 측정

     소프트웨어 환경과 구성 요소가 사전에 정의되어 있어야 함

3. Putnam 모형
   - 소프트웨어 생명 주기의 전 과정 동안에 사용될 노력의 분포를 예상하는 모형
   - = 생명 주기 예측 모형
   - Rayleigh-Nordan 곡선의 노력 분포도를 기반
   - 대형 프로젝트의 노력 분포 산정에 이용
   - 개발 기간이 늘어날수록 프로젝트 인원의 노력이 감소

- Putnam이 제안

4. 기능 점수 (FP : Function Point) 모형
   - 총 기능 점수와 영향도를 이용하여 기능 점수를 구한 후, 이를 통해 비용을 산정하는 기법
   - 알브레히트 (Albrecht)가 제안
   - 소프트웨어 기능 증대 요인
     1. 자료 입력
     2. 정보 출력
     3. 명령어
     4. 데이터 파일
     5. 필요한 외부 루틴과의 인터페이스

5. 비용 산정 자동화 추정 도구
   a. SLIM

      Rayleigh-Norden 곡선과 Putnam 예측 모델을 기반
   b. ESTIMACS

      FP 모형을 기반

소프트웨어 개발 표준