2주차

응용 vs 시스템

- 시스템 소프트웨어 : 하드웨어 부팅 및 하드웨어 자원의 운영,관리,동작 작업을 하는 소프트웨어
  - 운영체제, 장치드라이버, 유틸리티, 컴파일러,DBMS 등
- 응용 소프트웨어 : 사용자가 필요로 하는 작업을 수행하는 소프트웨어
  - 압축 프로그램, 워드프로세서, 엑셀 등등

소프트웨어 특성

- 비가시성: 구조 노출x, 코드에 구조가 내재
- 복잡성 : 정형적 구조가 없음, 복잡하고 비규칙적,비정규적
- 변경성 : 상황에 따라 항상 수정과 진화 가능
- 순응성 : 사용자의 요구, 환경의 변화에 따라 적절히 변형 가능
- 비마모성 : 외부의 환경에 마모x, 품질이 나빠질순 잇음
- 견고성 : 구조의 변경 혹은 수정이 용이하지 않음, 일부의 수정으로 소프트웨어에 영향을 줌
- 복제성 : 쉽게 복제됨
- 비제조성 : 논리적 절차에 의해 생성

생명주기 모델

- 개발관리에 기본적인 틀 제공
- 프로세스 계획 및 모델링에 기초가 됨
- 이정표와 기준선 => 진행의 가시성을 보여줌

1. 폭포수 모델
   - 요구사항 분석 => 설계 => 구현 => 테스트
   - 장점
     - 널리 알려짐
     - 순차적인 단계진행으로 정형화됨
     - 단계별 명확한 이정표, 기준선
   - 단점
     - 진행중 요구사항 변경 및 추가가 힘듬
     - 피드백이 어려움 -> 완벽한 설계 요구
   - 적합한 프로젝트 : 기술적 위험 부담이 적고, 비슷한 프로젝트 개발경험이 있는경우 혹은 요구사항이 명확한경우.
2. 시제품화 모델
   - 시제품화를 빨리만들어 고객에게 피드백을 받고, 받은 피드백을 반영시켜 다른 시제품을 만든다
   - 장점
     - 진행 중간에 요구사항 반영이 쉬움
     - 실질적으로 수행되는 물리적 모형을 볼 수 있음
   - 단점
     - 완제품화 하고 싶은 유혹
     - 부족한 문서화 -> 유지보수가 힘듬
     - 시제품 폐기 -> 비경제적

        - 반복적인 시제품화 -> 종료 시기 문제
3. 나선 모델
    - 위험 분석
        - 위험 식별
        - 위험 발생확률, 손실 예측
        - 우선순위 식별
        - 위험관리 대상 결정
    - 해결방안
        - 시제품
        - 시뮬레이션
        - 모델 벤치마킹
4. 증분 개발 모델
    - 여러 부분으로 나눠서 개발
    - 증분 i 단계에서는 전단계의 전달물 i-1 을 통합해 전달물 i를 개발
    - 규모가 큰 개발조직 -> 병행개발을 통해 개발 기간 단축
    - 증분 수가 많고 병행개발이 빈번하면 개발 프로젝트 관리가 어려움
5. 진화적 개발 모델
    - 분석,설계,구현,시험,통합-> 전개
    - 위의 과정을 수행하고 피드백을 받아 다음 단계 진화를 함
    - 1단계 진화에는 핵심부분을 포함한 최소의 시스템 개발->다음 단계에서 개선
    - 전체 진화과정에 대한 개요가 필요
6. RAD모델
    - 모델링 -> 어플리케이션 생성 -> 테스트 및 수락
    - 60~90일 소요
    - 최소한의 활동만 수행 , 프로세스 간결화
    - 기술적 위험이 적고 빠른 개발이 요구되는 경우
7. V모델
    - 테스트 설계, 테스트 작업을 코딩 이후가 아닌 프로젝트 시작과 동시에 시작함
    - 프로젝트 비용 및 시간 감소
    - 단계마다 상세한 문서화

소프트웨어 생명 주기 프로세스

- ISO 12207 : ISO 에서 정한 표준 소프트웨어 생명주기 프로세스
- 기본 생명주기 프로세스
    - 획득,공급,개발,운영,유지보수
- 지원 생명주기 프로세스
    - 성공 및 고품질에 기여하도록 하는 목적, 다른 프로세스 지원
    - 문서화, 형상관리,품질보증, 검증,확인,합동검토,감사,문제해결
- 조직 생명주기 프로세스
    - 특정 프로젝트 및 계약영역 밖에서 적용
    - 관리,기반구조, 개선,훈련
- 조정 프로세스

CMMI

- 소프트웨어 개발 및 전상장비 운영 업체들의 업무 능력 및 조직의 성숙도를 평가하기 위한 모델

- 각 레벨
    - 1 : Initial : 프로세스가 없는 상태, 개인의 역량에 따라 성공 실패 좌우
    - 2 : Managed : 프로세스하에 프로젝트가 통제됨, 일정,비용 같은 관리 프로세스 중심
    - 3 : Defined : 조직을 위한 프로세스 존재, 세부 표준 프로세스 존재, 모든 프로젝트는 조직의 프로세스를 가져다 상황에 맞게 조정 하고 승인 받아서 사용함
    - 4 : Quantitatively Managed
        - 소프트웨어 프로세스와 품질에 대한 정량적인 측정 가능
        - 프로세스 데이터베이스 구축 => 결과를 수집 분석하여 품질평가 기준을 만듬
        - 프로젝트 활동이 정략적으로 관리, 통제
        - 성과 예측 가능
    - 5 : Optimizing
        - 지속적인 개선에 치중
        - 피드백을 받아 개선
        - 지속적인 개선활동 정착화

SPICE

- 영국의 런던에서 만든 표준
- 소프트웨어 프로세스 평가에 사용될 국제 표준
- 개발,관리, 고객지원, 품질,소프트웨어 개발 및 유지보수등 사람,기술까지 다룸
- 단계
    - 0 : 아무것도 수행 안함
    - 1 : 비정형 적으로 수행
        - 기본활동은 수행, 작업 산출물 존재, 결과가 특정인의 능력에 의해 좌우
    - 2 : 계획하고 추적하는 단계
        - **미리 정의된 시간, 자원 한도**
        - **명시된 절차**에 따라 계획,추적
        - 조직차원이 아닌 **프로젝트 수준 또는 사례 수준**에서 관리
    - 3 : 잘 정의된 단계
        - **소프트웨어 공학 원칙에 기반**해 프로세스 수행
        - **표준화,문서화된 프로세스**에 따라 기본활동 수행
    - 4 : 정량적으로 관리 단계
        - 프로세스 기본활동을 정량적으로 측정 분석
        - 프로세스 능력 정량적 파악, 개선 목표 설정
        - 성취도가 객관적으로 관리
        - 정의된 프로세스가 한계 내에서 일관성 있게 수행
    - 5 : 지속적으로 개선 단계
        - 지속적 개선 수행
        - 혁신적인 아이디어 및 기술 시험적용
        - 사업목적에 맞게 정략적인 프로세스 효율 효과애 대한 목표 설정

ASPICE

- Automobile SPICE
- 자동차 산업쪽에 쓰이는 SPICE

3주차

소프트웨어 방법론이란?

- 시스템 구축시 필요한 단계들의 수행방법, 수행시 도움되는 기법 및 도구를 이용해 **작업 단계를 체계적으로 정리한 작업 수행의 표준 규범**
- 구조적 방법론, 객체지향 방법론, Agile 방법론

구조적 방법론

- 시스템을 기능 중심으로 나누어 모듈 개발
- 시각적 표현(모델,도형,도표) - 분석 작업 이해가 쉬움
- 하향식 방법
- 추상화의 원리 = 어떻게가 아닌 무엇에 집중
- 분할 정복의 개념 = 큰 시스템을 작은 시스템으로 나눔
- 계층 구조 = 트리 형태로 전개, 분할 정복 개념과 밀접
- 개체관계도 : 시스템에 저장될 데이터 객체들과 이들간의 관계
    - 각 데이터 객체의 속성은 자료객체 명세에 정의
    - 표기법은 ppt참조
- 자료흐름도 : 시스템을 통해 데이터가 변환되는 과정 기술
    - 문맥도 : 가장 상위 단계의 자료흐름도

객체지향 방법론

- 데이터와 그 데이터를 연산하는 함수를 묶어 객체라고 부름
- 주요 개념
    - 캡슐화 : 객체의 데이터와 Operation을 묶음
        - 관계있는 데이터와 Operation을 묶음으로써 유지보수성 증가
    - 정보 은닉 : 데이터나 Operation이 하는 일을 필요 이상으로 보여주지 않음.
        - 한 객체의 수정이 다른 객체에 주는 영향을 최소화
    - 상속 : 상위 객체의 성질을 물려 받아서 그대로 사용
        - 코드 재사용성
        - 다중 상속 : C++은 다중상속 허용, Java나 C#은 interface implement
            - C++은 Name conflict가 발생하지 않도록 operation이름을 정의해야함
    - 다형성 : 각 클래스에 동일한 이름의 함수를 정의해 같은 메세지에 대해 다르게 반응할 수 있도록 하게 함
    - 추상 클래스 : 유사한 클래스들의 특징을 모아 하나의 슈퍼클래스로 추출
        - 인스턴스화 x
- 장점
    - 사용자도 좀 더 직관적으로 이해 가능
    - 여러 개발팀이 독립적으로 설계의 여러부분 참가할 수 있음
    - 상대적으로 적은 양의 코드, 재사용성

Agile 방법론

- 앞을 예측하여 개발하지 않고 일정한 주기로 끊임 없이 프로토타입을 만듬, 그때 마다 요구사항 수정
- 사용자 요구가 다양해지고 정보시스템 수명주기가 짧아짐 -> 기존 개발방법론으로는 변화에 대한 대응이 곤란
- 다음과 같은 경우 적합

    - 요구사항 적출이 힘들 때
    - 소규모나 서브프로젝트나 반복주기에 적합
    - 중간 산출물 작업분을 지속적으로 전달하는 경우
    - 기업 내 단위 시스템
    - 프로토타이핑, RAD를 할 수 있는 프로젝트
    - 품질 수준이 낮고, 산출물의 범위변경이 불가능 한 경우는 힘듬

익스트림 프로그래밍

- 애자일의 한 종류이자 대표 주자
- 고객과 함께 2주 정도의 반복개발, 테스트 우선 개발(TDD)
- 적은 규모의 인원
- 개발 문서보다는 **소스코드**, 조직적인 움직임 보다는 **개개인의 책임과 용기**에 중점
- 고객이 원하는 양질의 소프트웨어를 빠른시간 안에 전달하기 위함
- 프로그래밍 할때 테스트 코드를 같이 작성함 => 동시에 테스트
- 실천
    - Customer Tests
    - Small Releases
    - Simple Design
    - Test-Driven Development
    - Pair Programming

스크럼

- 프로젝트 관리를 위한 상호 점진적 개발방법론
- 솔루션에 포함할 기능/개선 점에 우선순위
- 개발주기는 30일 정도, 실제 동작할 수 있는 결과
- 주기마다 적용할 기능 , 개선 목록 제공 (제품 백로그, 스프린트 백로그)
- 날마다 15분 정도 회의
- 팀 단위로 생각
- 구분 없는 열린 공간

5주차

요구사항 분석

- 시스템이나 소프트웨어 요구사항을 정의하기 위해 사용자 요구사항을 조사하고 확인하는 과정
- 생명 주기의 첫 단계
- 사용자가 요구사항을 파악 후, 소프트웨어에 반영할 사용자의 요구사항 결정
- 어떻게 보다는 **무엇**에 초점
- 만족시켜야 할 기능, 성능, 다른 시스템과의 인터페이스 규명
- 고객들과 협상해 공동의 목표를 끌어내야됨
- 프로젝트 초기단계의 중대한 실수 최소화가 목적
- 최종 산출물 : 요구사항 명세서
    - 사용자가 충분히 이해가능
    - 쉬운 용어 사용
    - 다이어그램으로 작성(이해를 돕기위해)

- 자료 수집 -> 요구사항 도출 -> 문서화 -> 검증
- 기능적 요구사항 : 사용자가 원하는 기능
- 비기능적 요구사항 : 환경 ,품질, 제약사항

6주차

- 시퀀스 다이어그램 : 단일 UseCase에 대하여 객체들 간의 메세지 흐름을 시간순서로 나타낸 다이어그램이다.
    - 어떤 행위에 대한 공통의 객체를 어떻게 그룹화 할 것인지 나타냄
    - 단일 usecase의 행위 획득
    - 객체 사이의 전송된 메세지와 객체 수 보여줌
    - 상호작용 동안의 객체의 생명을 나타냄
- 다이어그램 표기법은 ppt에

7주차

소프트웨어 설계란

- 요구사항 명세서를 기준으로 어떻게 구축할 것인가를 결정한다.
- 요구사항 분석이 what에 초점이었다면 설계는 how에 초점
- 비기능적 요구사항 고려
- 운영체제, 미들웨어, 프레임워크 등 플랫폼을 결정한다.
- 다양한 제약조건을 만족시켜야 함.

설계원리

1. 분할과 정복
    - 큰 문제를 소 단위로 나눈다.
    - 나누어진 사람들이 각 부분을 맡을 수 있다.
    - 일을 맡은 개발자가 해당분야에 전문화 될 수 있다
    - 컴포넌트가 작아지고 이해하기 쉬워짐
    - 부품의 교체 혹은 변경이 쉽다 (다른 부분에게 영향을 주지 않음)
2. 추상화
    - 특정한 목적과 관련된 필수정보만 추출해서 강조
    - 세부사항은 넣지 않음
    - 본질적인 문제에 집중
    - 좋은 추상화 = 정보 은닉 제공
    - 서브시스템의 근본을 쉽게 이해
    - 객체지향
        - private 선언시 추상성 높아짐
        - 메서드가 적을수록 추상화는 좋아짐
        - 슈퍼클래스, 인터페이스 사용시 추상성 높아짐
        - 메소드는 절차적 추상 = 좋은 추상이 되려면 적은 매개변수 메소드 제공
3. 단계적 분해
    - 하향식 설계
    - 기능을 점점 작은 단위로 나누어 점차적으로 구체화 함

4. 모듈화
    - 모듈 : 규모가 큰 것을 여러개로 나눈 조각
    - 함수도 모듈임
    - 소프트웨어 구조를 이루는 기본 단위
    - 특징
        - 독립적인 기능을 갖는 단위
        - 유일한 이름
        - 독립적 컴파일 가능
        - 모듈에서 다른 모듈 호출 가능
        - 다른 프로그램에서도 모듈 호출 가능
    - 라이브러리 함수
    - 추상화된 데이터, 객체,메소드
    - 결합도는 느슨해야 됨
    - 응집도는 강해야됨
    - 장점
        - 분할 정복 원리 => 복잡도 감소
        - 문제를 이해하기 쉽게 만듬
        - 변경 쉬움, 변경으로 인한 영향이 적음
        - 유지보수 용이
        - 프로그램 효율적 관리
        - 오류로 인한 파급효과 최소화
        - 설계 및 코드 재사용
5. 정보은닉

응집도

1. 기능적 응집도 : 한 모듈에 단일 기능만이 있는 것
2. 순서적 응집도 : 요소1의 출력이 요소2의 입력으로 쓰이는 경우
3. 대화적 응집도 : 같은 입력이나 같은 출력을 사용하는 요소들 끼리 묶는다, 순서는 상관없음
4. 절차적 응집도 : 시간 순서대로 따라 수행되는 요소들, 즉 요소1->요소2->요소3->요소4 등,
    - 구성요소의 출력이 다음 구성요소의 입력으로 사용되는게 아님
5. 시간적 응집도 : 같은 시간에 수행되는 요소들
6. 논리적 응집도 : 공통점이나 관련된 임무가 존재, 혹은 기능이 비슷한 이유로 묶음
7. 우연적 응집도 : 아무 관련 없는 것끼리 묶음

결합도

1. 자료 결합도 : 모듈A->B 호출시 데이터만 주고받음
2. 스탬프 결합도 : 레코드나 배열 같은 데이터 구조, 필요 이상의 데이터를 주고 받을 때
3. 제어 결합도 : 모듈 A가 모듈B의 제어를 위해 플래그를 전달 하는 경우
    - 정보 은닉에 위배
4. 공유 결합도 : 전역변수 이용
5. 내용 결합도 : 다른 모듈의 내부자료 변경

4+1 아키텍쳐 뷰

1. Scenarios (Use case view)
    - USECASE 다이어그램, 시퀀스 다이어그램

    - 객체,프로세스 간의 상호작용 기술
    - 아키텍처 구성요소 식별, 설명하고 설계할때 사용
    - 다른 네가지 관점에 사용되는 다이어그램의 근간
    - 아키텍쳐 프로토타입의 테스트를 위한 시작점
2. Logical View
    - 시스템이 사용자에게 제공하는 기능과 관련된 관점
    - 클래스, 컴포넌트의 종류와 이들의 관계 초점
    - 클래스 다이어그램, 상태 다이어그램
3. Development View
    - 프로그래머의 시각에서 시스템 설명
    - 소프트웨어 관리와 관련된 관점
    - 서브시스템 모듈이 어떻게 구조화되있나 관심
    - 컴포넌트 다이어그램, 패키지 다이어그램
4. Process View
    - 시스템의 동적인 측면
    - 실제 구동환경을 살펴봄, 논리적 관점같이 시스템 내부구조 초점
    - 시스템 동시성, 분산, 성능, 확장성 관심
    - 액티비티 다이어그램
5. Physical view
    - 구현 후 물리적으로 어떻게 배치할 것인가
    - 시스템을 구성하는 처리 장치간의 물리적 배치에 초점
    - 서브시스템들이 물리적 환경에서 어떻게 연관되어 실행되는지 나타냄
    - 시스템 분산구조, 실행할 때 컴포넌트들의 배치 상태
    - 배치 다이어그램

아키텍쳐 스타일

1. 데이터 중심형 모델
    - Repository 모델 (DBMS)
    - 주요 데이터를 repository에서 중앙 관리
    - 은행시스템 같은 곳 많이 사용
    - 장점
        - 데이터가 한 군데 있음 -> 일관성
        - 새로운 서브시스템 추가 용이
    - 단점
        - 병목 현상
        - 서브시스템 - repository 강한 결합
            - repository 변경시 서브시스템 영향
2. 클라이언트 - 서버 모델
    - 분산 아키텍쳐에 유용
    - 데이터와 처리 기능을 클라이언트와 서버에 분할하여 사용
3. 계층 모델
    - Layering
    - 기능을 몇개의 계층으로 나누어 배치
    - 하위 계층은 서버, 상위 계층은 클라이언트 역할 수행
    - 사용자 인터페이스 계층 -> 애플리케이션 계층 -> 데이터 계층
4. MVC 모델

- Model, View, Controller
- 일종의 Layering이랑 비슷
- 중앙 데이터 구조
- 같은 모델의 서브시스템에 대해 여러 뷰를 필요로 하는 시스템에 적합
- 장점
  - 관심 분리
  - 데이터를 화면에 표현(뷰) 부분과 로직(모델) 분리함 -> 느슨한 결합
  - 구조 변경 요청시 수정용이
- 단점
  - 속도
  - 설계 클래스 수 증가 -> 복잡도 증가

5. Pipe and filter
   - Filter : 데이터 스트림을 한개 이상 입력받아 처리 후 data stream 하나 출력
   - Pipe :fitler를 거쳐 생성된 data stream으로 다른 filter 의 입력에 연결