

요구 사항 확인

SW Engineering

- ❖ SW 위기: HW의 발전속도를 SW의 발전속도가 따라잡지 못하는 현상
- ❖ SW 공학: SW 위기를 극복하기 위한 방법으로 SW 개발을 공학적으로 접근하자고 하는 학문으로 소프트웨어의 생산성과 품질 향상이 목표



개발 방법론

❖ 소프트웨어 개발 방법론

- ✓ 소프트웨어 개발, 유지보수 등에 필요한 여러 가지 일들의 수행 방법과 이러한 일들을 효율적으로 수행 하려는 과정에서 필요한 각종 기법 및 도구를 체계적으로 정리하여 표준화한 것
- ✓ 소프트웨어의 생산성과 품질 향상이 목적
- ✓ 방법론의 종류
 - ❑ 구조적 방법론(정형화된 분석 절차에 따라 사용자의 요구사항을 문서화하여 처리)
 - ❑ 정보공학 방법론(데이터 중심의 개발 방법론)
 - ❑ 객체지향 방법론(클래스와 객체 중심의 방법론으로 Encapsulation, Information Hiding, Inheritance, Polymorphism)
 - ❑ 컴포넌트 기반 방법론 (CBD – 객체 생성 과 수명주기 관리를 시스템이나 프레임워크가 하는 방법론으로 프로그램은 컴포넌트의 조합)
 - ❑ 애자일(Agile) 방법론
 - ❑ 제품 계열 방법론(Embedded Application 개발에 적합한 방법으로 공통된 기능을 먼저 만들고 제품에 적용하는 방식) 등이 있음

❖ Software Life Cycle – ISO/IEC 12207

- ✓ 소프트웨어 개발 방법론의 바탕이 되는 것으로, 소프트웨어를 개발하기 위해 정의하고 운용, 유지보수 등의 과정을 각 단계별로 나눈 것
- ✓ 소프트웨어 개발 단계와 각 단계별 주요 활동, 그리고 활동의 결과에 대한 산출물로 표현

개발 방법론

❖ 폭포수 모델

✓ 개발 순서

- 타당성 검토 단계(법적 타당성, 경제적 타당성, 기술적 타당성)
- 계획과 요구 분석 단계
- 개략 설계 단계
- 상세 설계 단계
- 코딩 단계
- 통합 검사 단계
- 운용 단계
- 유지 보수 단계

✓ 폭포수 모형은 앞 단계가 끝나야 다음 단계로 넘어갈 수 있음

✓ 폭포수 모형의 이점

- ❑ 가장 오래되서 적용 사례가 많음
- ❑ 단계별 정의가 분명해서 단계별 산출물이 명확

✓ 폭포수 모형은 피드백(feed back)이 없어서 요구사항 변경이 용이 하지 않음

개발 방법론

❖ 프로토타이핑 모델

- ✓ 요구사항을 미리 파악하기 위한 것으로 개발자가 구축한 S/W 모델을 사전에 만듦(시제품, 견본, 샘플)으로써 최종 결과물이 만들어지기 전에 사용자가 최종 결과물의 일부 또는 모형을 볼 수 있는 방법으로 일반적으로 사용자와 시스템 사이의 인터페이스 중심으로 구현
- ✓ 장점 : 요구사항 변경이 용이
- ✓ 문제점
 - 실제 제품과 혼동(신뢰성이 낮다.)
 - 비효율적인 알고리즘이나 언어로 구현될 가능성이 있음

❖ 나선형 모델

- ✓ 나선형 모델(Spiral Model)은 라이프사이클 모델과 프로토타이핑의 장점만을 수용하고 이에 추가로 위험성 분석(Risk Analysis)등을 포함한 대규모 시스템과 소프트웨어 개발에 가장 현실적인 패러다임
- ✓ 계획(Planning) --> 위험분석(Risk Analysis) --> 구현(Engineering) --> 고객평가(Customer Evaluation)

❖ TDD(Test-driven development)

- ✓ 테스트 주도 개발은 매우 짧은 개발 사이클을 반복하는 소프트웨어 개발 프로세스 중 하나
- ✓ 개발자는 먼저 요구사항을 검증하는 자동화된 테스트 케이스를 작성한 후에, 그 테스트 케이스를 통과하기 위한 최소한의 코드를 생성

개발 방법론

❖ Agile

- ✓ 신속한 반복 작업을 통해 실제 작동 가능한 소프트웨어를 개발하여 지속적으로 제공하기 위한 소프트웨어 개발 방식으로 요구 사항 변화에 유연하게 대응할 수 있도록 일정한 주기를 반복하면서 개발하는 모형
- ✓ 협업과 워크플로우를 바라보는 하나의 관점이며 우리가 무엇을 어떻게 만들지에 관한 선택을 안내하는 가치 체계
- ✓ 애자일 개발의 핵심은 작동하는 소프트웨어의 작은 구성 요소를 신속하게 제공하여 고객의 만족도를 개선하는 것
- ✓ 애자일 소프트웨어 개발은 소프트웨어 개발자와 비즈니스 담당자의 작은 팀으로 이루어지며, 이들은 소프트웨어 개발 라이프사이클 전체에 걸쳐 정기적으로 직접 만나 협업
- ✓ 애자일 개발은 소프트웨어 문서화에 대한 경량화 방식을 선호하며 라이프사이클의 모든 단계에서 변화를 적극 수용
- ✓ 고객의 변화하는 요구사항과 환경 변화에 능동적이 소프트웨어 개발 방법
- ✓ 핵심 가치
 - ❑ 프로세스와 도구 보다는 개인과 상호 작용에 가치를 부여
 - ❑ 문서보다는 실행되는 SW에 가치를 부여
 - ❑ 계약 협상보다는 고객과 협업에 가치를 부여
 - ❑ 계획을 따르기보다는 변화에 반응하는 것에 가치를 부여

개발 방법론

❖ Agile 개발 모형

✓ 스크럼(Scrum)

- ❑ 팀 중심으로 개발의 효율성을 높이는 기법
- ❑ 애자일 모형을 기반으로 하는 상호 점진적 개발 방법론으로 일정한 주기로 실제 동작하는 제품(Backlog-요구사항, Sprint-개발 주기)을 만들면서 개발을 진행
- ❑ 팀의 구성
 - PO(Product Owner): Backlog를 작성하는 주체로 요구사항을 책임지고 의사 결정을 하는 제품 책임자
 - SM(Scrum Master): 스크럼이 잘 수행될 수 있도록 가이드 역할을 수행하는 마스터
 - DT(Development Team): PO 와 SM을 제외한 제품 개발을 수행하는 팀원
- ❑ 개발 프로세스
 - Sprint Planning Meeting: 제품 백로그 중에서 이번 스프린트에서 수행할 작업의 단기 일정을 수립하는 회의
 - Sprint: 실제 작업을 수행하는 과정(2 ~ 4주)
 - Daily Spring Meeting: 15분 정도에 걸친 일일 진행 상황 점검 회의
 - Sprint Review: 요구 사항에 부합하는지 테스트 하는 회의
 - Spring Retrospective: 정해진 규칙 준수 여부 및 개선할 점을 확인하고 기록하는 작업

개발 방법론

❖ Agile 개발 모형

✓ XP 모형

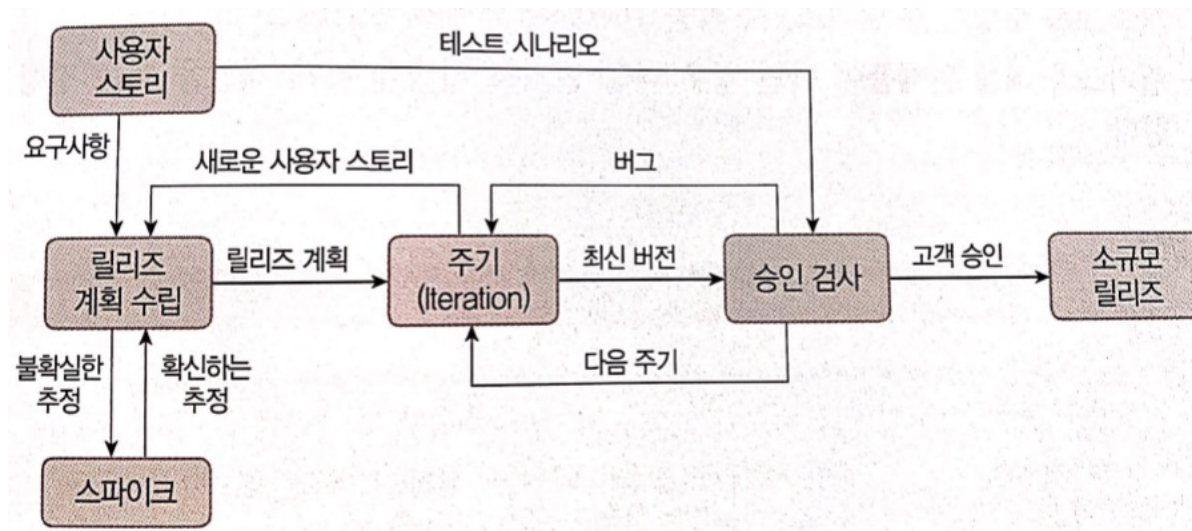
- ❑ 수시로 발생하는 고객의 요구사항에 유연하게 대응하기 위해 고객의 참여와 개발 과정의 반복을 극대화 하여 개발 생산성을 향상시키는 모형으로 릴리즈 테스트마다 고객을 직접 참여시킴으로써 요구한 기능이 제대로 작동하는지 고객이 직접 확인할 수 있는 모형
- ❑ 핵심 가치
 - Communication
 - Simplicity
 - Courage
 - Respect
 - Feedback

개발 방법론

❖ Agile 개발 모형

✓ XP 모형

□ XP 개발 프로세스



- Release Planning: 개발 완료 시점에 대한 일정 수립
- Iteration: 실제 개발 작업을 진행하는 과정으로 보통 1~3주
- Acceptance Test: Iteration 안에서 부분 완료 제품이 구현되면 수행하는 테스트
- Small Release: 요구 사항 변경에 유연하게 대처하기 위해 릴리즈의 규모를 축소하는 것

개발 방법론

❖ Agile 개발 모형

✓ XP 모형

□ 주요 실천 방법

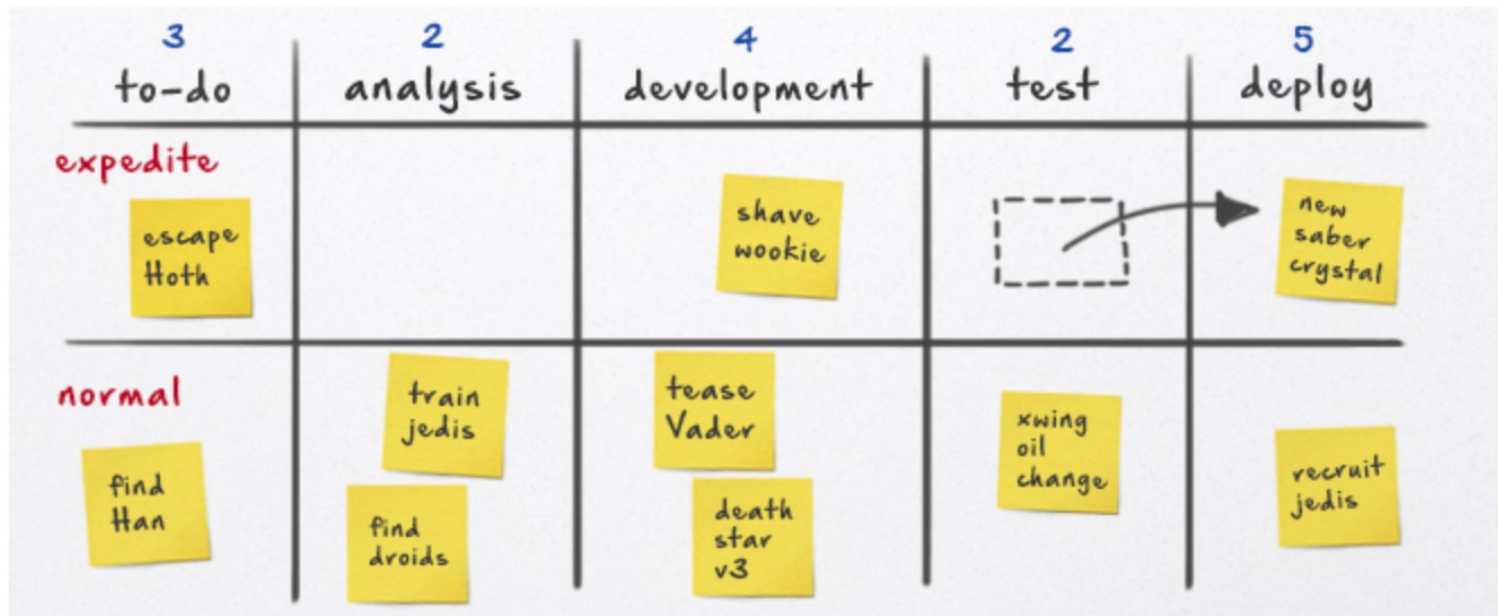
- Pair Programming: 다른 사람과 함께 프로그래밍을 수행함으로써 개발에 대한 책임을 공동으로 나눠 갖는 환경을 조성
- Collective Ownership: 개발 코드에 대한 권한과 책임을 공동으로 소유
- Test-Driven Development (테스트 주도 개발)
 - 개발자가 실제 코드를 작성하기 전에 테스트 케이스를 먼저 작성하므로 자 신이 무엇을 해야 할지를 정확히 파악
 - 테스트가 지속적으로 진행될 수 있도록 자동화된 테스트 도구(구조, 프레임 워크)를 사용
- Whole Team: 개발에 참여하는 모든 구성원(고객 포함)들은 각자 자신의 역할이 있고 그 역 할에 대한 책임을 가져야 함
- Continuous Integration (계속적인 통합): 모듈 단위로 나눠서 개발된 코드들은 하나의 작업이 마무리될 때마다 지속적으로 통합
- Design Improvement (디자인 개선) 또는 Refactoring(리팩토링): 프로그램 기능의 변경 없이 단순화, 유연성 강화 등을 통해 시스템을 재구성
- Small Releases (소규모 릴리즈): 릴리즈 기간을 짧게 반복함으로써 고객의 요구 변화에 신속히 대응

개발 방법론

❖ Agile 개발 모형

✓ Kanban – 칸판의 일본어

- Agile 개발 프로세스 전반에 걸친 적시 개발(Just In Time Development)을 지원하는 방법론



개발 방법론

❖ Agile 개발 모형

✓ Lean

- ❑ 도요타(자동차 제조사)의 프로세스를 S/W 개발에 적용한 방법론
- ❑ 구체적인 개발 프로세스를 정의하지 않고 철학적인 접근 방식을 정의
- ❑ 개발 방법론 이라기 보다는 사고방식이란 용어가 더 적합
- ❑ 린은 낭비를 발견하고 제거함으로써 어떻게 고객에게 가치를 빠르게 제공할 수 있을 것인가에 대한 생각이자 사고방식으로 제조분야에서 생산성 향상을 위해 사용하는 린 원칙을 S/W 개발에 적용하여 낭비요소를 제거하자는 내용으로 S/W개발의 가장 큰 낭비는 결함이고

✓ 기능 중심 개발(Feature Driven Development)

- ❑ 설계 및 구축 기능에 중점을 두는 방식
- ❑ 기능별로 개별적으로 수행해야하는 매우 구체적이고 짧은 단계의 작업을 설명
- ❑ 도메인 둘러보기, 디자인 검사, 빌드 승격, 코드 검사 및 디자인이 포함됨
- ❑ FDD는 목표물을 따라 움직이는 제품을 개발

현행 시스템 파악

❖ 현행 시스템 파악 절차

✓ 1단계

- ❑ 시스템 구성 파악: 현행 시스템의 구성은 조직의 주요 업무를 담당하는 기간 업무와 이를 지원하는 지원 업무로 구분하여 기술
- ❑ 시스템 기능 파악: 현행 시스템의 기능은 단위 업무 시스템이 현재 제공하는 기능들을 주요 기능과 하부 기능, 세부 기능으로 구분하여 계층형으로 표시
- ❑ 시스템 인터페이스 파악 : 현행 시스템의 인터페이스에 는 단위 업무 시스템 간에 주고받는 데이터의 종류, 형 식, 프로토콜, 연계 유형, 주기 등을 명시

✓ 2단계

- ❑ 아키텍처 구성 파악 : 현행 시스템의 아키텍처 구성은 기간 업무 수행에 어떠한 기술 요소들이 사용되는지 최상 위 수준에서 계층별로 표현한 아키텍처 구성도로 작성
- ❑ 소프트웨어 구성 파악: 소프트웨어 구성에는 단위 업무 시스템 별로 업무 처리를 위해 설치되어 있는 소프트웨어들의 제품명, 용도, 라이선스 적용 방식, 라이선스 수 등을 명시

✓ 3단계

- ❑ 하드웨어 구성 파악: 하드웨어 구성에는 단위 업무 시스템들이 운용되는 서버의 주요 사양과 수량, 그리고 이중화의 적용 여부를 명시
- ❑ 네트워크 구성 파악: 네트워크 구성은 업무 시스템들의 네트워크 구성을 파악할 수 있도록 서버의 위치, 서버 간의 네트워크 연결 방식을 네트워크 구성도로 작성

개발 환경 파악

❖ 개발하고자 하는 소프트웨어와 관련된 운영체제(Operating System), 데이터베이스 관리 시스템(DataBase Management System), 미들웨어(Middle Ware) 등을 선정할 때 고려해야 할 사항을 기술하고, 오픈 소스 사용 시 주의해야 할 내용을 제시

❖ 운영체제(Operation System)

- ✓ 컴퓨터 시스템의 자원들을 효율적으로 관리하며 사용자가 컴퓨터를 편리하고 효율적으로 사용할 수 있는 환경을 제공하는 소프트웨어
- ✓ 운영체제 관련 요구사항 식별 시 고려사항: 가용성, 성능, 기술 지원, 주변 기기, 구축 비용
- ✓ Server 운영체제: UNIX, LINUX, Windows
- ✓ Client 운영체제: UNIX, LINUX, Windows, Mac OS, iOS, Android, Tizen 등

❖ 데이터베이스 관리 시스템 (DataBase Management System)

- ✓ 사용자와 데이터베이스 사이에서 사용자의 요구에 따라 정보를 생성해 주고, 데이터베이스를 관리해 주는 소프트웨어
- ✓ DBMS 관련 요구사항 식별 시 고려사항: 가용성, 성능, 기술 지원, 상호 호환성, 구축 비용
- ✓ 관계형 데이터베이스: Oracle, MySQL, MS-SQL Server, Tibero, HANA DB 등
- ✓ No-SQL: Mongo DB, Cassandra, HBase 등

개발 환경 파악

❖ Middle Ware

✓ TP-모니터 미들웨어

- 온-라인 트랜잭션 업무(은행 계정, 항공기/버스 예약 업무 등)에서 트랜잭션을 처리, 감시하는 미들웨어
- 사용자 수가 증가하여도 빠른 응답 속도를 유지해야 하는 OLTP 업무에 적합
- TUXEDO 등이 있음

✓ 웹 애플리케이션 서버(WAS; Web Application Server)

- 정적인 콘텐츠 처리를 하는 웹 서버와 달리 사용자의 요구에 따라 변하는 동적인 콘텐츠를 처리하기 위해 사용되는 미들웨어
- WAS 관련 요구사항 식별 시 고려사항: 가용성, 성능, 기술 지원, 구축 비용
- Java WAS: Tomcat, Jboss, GlassFish, Web Logic, JEUS

❖ IDE: 통합 개발 환경이라고 하는데 소스 코드를 작성하고 실행시켜주는 프로그램

❖ Open Source

- ✓ 별다른 제한 없이 사용할 수 있도록 소스 코드를 공개한 것으로 오픈 소스 라이선스를 만족하는 소프트웨어
- ✓ 오픈 소스 사용에 따른 고려사항 : 라이선스의 종류, 사용자 수, 기술의 지속 가능성

요구 사항 정의

❖ 요구 사항

- ✓ 소프트웨어가 문제를 해결하기 위해 서비스에 대한 설명과 제약조건
- ✓ 소프트웨어 개발에 대한 기준과 근거
- ✓ 개발자와 사용자간의 의사 소통을 원활하게 하기 위한 기준
- ✓ 요구사항을 도출하기 위해서는 설문지, 인터뷰, Field Study, WorkShop 등을 이용

❖ 요구 사항 유형

- ✓ 기능 요구사항: 시스템이 갖춰야할 필수적인 기능에 대한 요구사항
- ✓ 비기능 요구사항: 필수 기능 외의 품질이나 제약조건에 관한 요구사항
- ✓ 사용자 요구사항: 사용자 관점에서 본 시스템이 제공해야 할 요구사항
- ✓ 시스템 요구사항: 개발자 관점에서 본 시스템 전체가 사용자와 다른 시스템에 제공해야 할 요구사항

❖ 요구 사항 개발 프로세스

- ✓ 타당성 조사 -> 도출(Elicitation) -> 분석(Analysis) -> 명세(문서화, Specification) -> 확인(검증, Validation)

요구 사항 정의

❖ 요구 사항

✓ 요구 사항 개발 프로세스

□ 도출(Elicitation)

- 시스템 개발에 관련된 사람들이 서로 의견을 교환하여 요구 사항을 어떻게 수집해서 식별하고 이해할 것인가 하는 과정
- 청취, 인터뷰, 설문, 브레인스토밍, 워크샵, 프로토타입, 유스케이스 등의 방법

□ 분석(Analysis): 도출된 요구 사항 중 명확하지 않거나 이해되지 않는 부분을 발견하고 걸러 내는 과정

□ 명세(문서화, Specification): 요구 사항을 문서화 하는 것

□ 확인(검증, Validation): 요구 사항 명세서가 정확하고 완전하게 작성되었는지를 검토 하는 활동

요구 사항 분석

❖ 요구 공학(requirements engineering)

- ✓ 요구 공학은 소프트웨어의 요구 사항을 정의하고 관리하는 방법을 연구하는 학문
- ✓ 소프트웨어의 규모가 커지면서 요구 사항의 중요성이 날로 커지고 있을 뿐만 아니라 요구 사항을 어떻게 만드는데 따라 소프트웨어 제품 개발의 성공을 좌우함에 따라 요구 사항에 대한 연구가 늘고 있는데 그것이 요구 공학이라는 학문이 탄생하게 된 이유

❖ 요구 사항 분석: 개발 대상에 대한 사용자의 요구사항 중 명확하지 않거나 모호한 부분을 걸러내기 위한 방법

- ✓ 요구사항 분류(Requirement Classification) : 요구사항을 명확히 확인할 수 있도록 요구사항을 분류
- ✓ 개념 모델링(Conceptual Modeling): 요구사항을 보다 쉽게 이해할 수 있도록 현실 세계의 상황을 단순화하여 개념적으로 표현한 것을 모델이라고 하며 이러한 모델을 만드는 과정이 모델링이라고 하고 이러한 도구들로는 UseCase Diagram, DFD, State Model, Object Oriented Model, User Interaction, Object Model(UML, Class Diagram), Data Model(E-R Diagram) 등 이 있음
- ✓ 요구사항 할당(Requirement Allocation): 요구사항을 만족 시키기 위한 구성 요소를 식별
- ✓ 요구사항 협상(Requirement Negotiation): 요구사항이 서로 충돌될 경우 이를 적절히 해결

요구 사항 분석

❖ 요구 사항 명세 기법

✓ 정형 분석(Formal Analysis)

- ❑ 구문(Syntax)과 의미(Semantics)를 갖는 정형화된 언어를 이용해 요구사항을 수학적 기호로 표현한 후 이를 분석
- ❑ 요구 사항을 정확하고 간결하게 표현
- ❑ 표기법이 어려움

✓ 비정형 분석

- ❑ 상태/기능/객체 중심으로 자연어를 기반으로 Diagram을 작성하는 방식
- ❑ 내용의 이해가 쉬워서 의사 소통이 쉬움
- ❑ 자연어를 기반으로 하기 때문에 일관성이 떨어지고 해석이 달라질 수 있음

구조적 분석

❖ 구조적 분석

- ✓ 자료의 흐름과 처리를 중심으로 요구 사항 분석을 수행하는 기법
- ✓ 하향식 방법을 이용
- ✓ 분석 도구
 - ❑ DFD
 - ❑ DD
 - ❑ Mini-Spec
 - ❑ ERD
 - ❑ STD
 - ❑ 제어 명세서

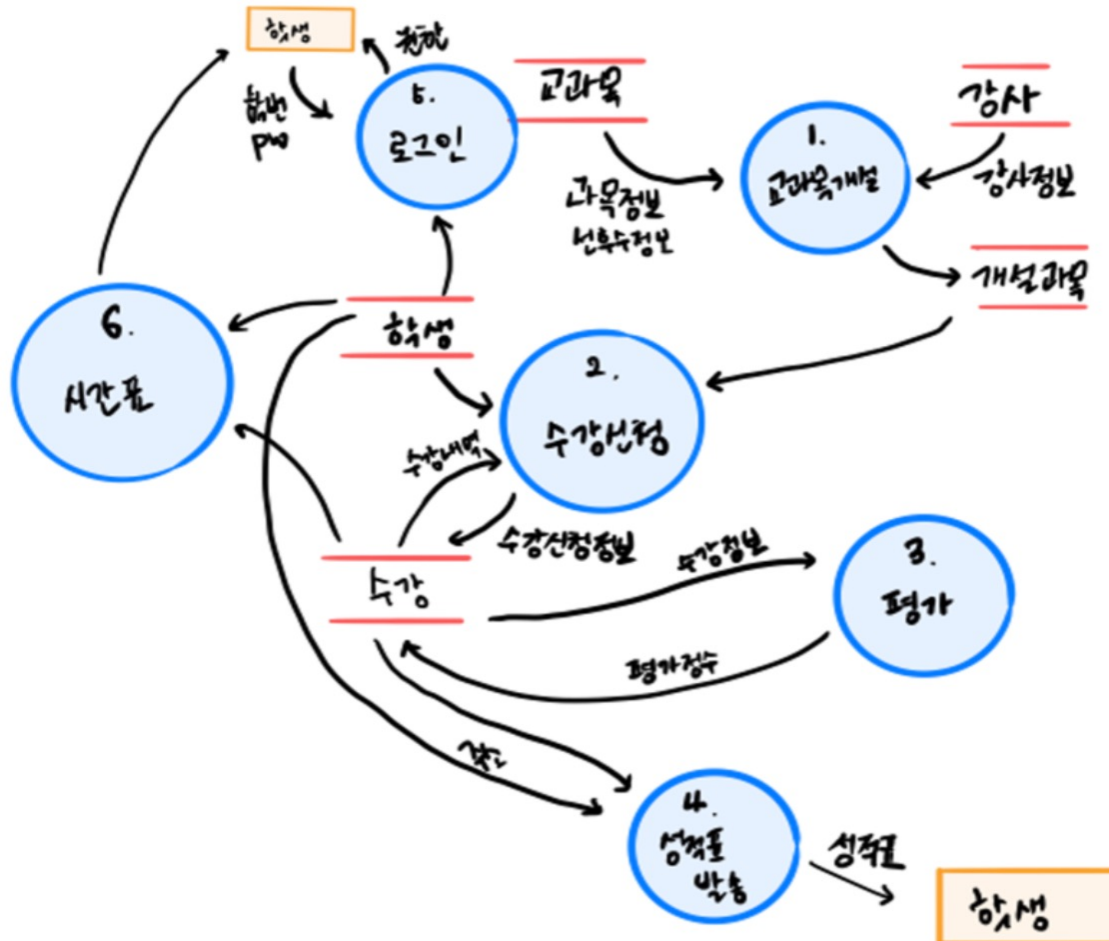
구조적 분석

❖ DFD(Data Flow Diagram)

- ✓ 요구사항 분석에서 자료의 흐름 및 변환 과정과 기능을 도형 중심으로 기술하는 방법으로 자료 흐름 그래프, 버블 차트 라고도 합니다.
- ✓ 표현 방법
 - 프로세스(Process)
 - 자료를 변환시키는 시스템의 한 부분(처리 과정)을 나타내며 처리, 기능, 변환, 버블이 라고함
 - 원이나 둥근 사각형으로 표시하고 그 안에 프로세스 이름을 기입함
 - 자료 흐름(Data Flow)
 - 자료의 이동(흐름)이나 연관관계를 나타냄
 - 화살표 위에 자료의 이름을 기입함
 - 자료 저장소(Data Store)
 - 시스템에서의 자료 저장소(파일, 데이터베이스 등)를 나타냄
 - 사각형에서 좌우가 열린 형태로 그리면 2중선을 이용하기도 함
 - 단말 (Terminator)
 - 시스템과 교신하는 외부 개체로, 입력 데이터가 만들어지고 출력 데이터를 받음(정보의 생산자와 소비자)
 - 사각형 안에 이름을 기입함

구조적 분석

❖ DFD(Data Flow Diagram)



구조적 분석

❖ DD(Data Dictionary)

- ✓ DFD에 나온 자료에 대한 명세
- ✓ 사용되는 기호

기호(symbol)	의미(meaning)
=	정의
+	구성
[]	택일
{ }	반복
()	생략가능
**	설명(comment)

- ✓ 학생증 = 학과 + 학년 + 학번 + 이름 + 생년월일

구조적 분석

❖ HIPO(Hierarchical Input Process Output)

- ✓ Input-Process-Output으로 이루어진 모듈을 계층적으로 나타낸 도표이
- ✓ 시스템의 분석 및 설계나 문서화에 사용 되는 기법으로 계층을 구성하는 각 모듈 별 실행 과정인 입력, 처리, 출력 기능을 표현
- ✓ HIPO Chart
 - ❑ 가시적 도표(Visual Table of Contents): 도식 목차로 시스템의 전체적인 기능과 흐름을 보여주는 Tree형태의 구조도
 - ❑ 총체적 도표(Overview Diagram): 개요 도표로 프로그램을 구성하는 기능을 기술한 것으로 입력, 처리, 출력에 대한 전반적인 정보를 제공하는 도표
 - ❑ 세부적 도표(Detail Diagram): 상세 도표로 총체적 도표에 표시된 기능을 구성하는 기본 요소들을 상세히 기술하는 도표인데 총체적 도표와 같은 모양이지만 내용만 좀 더 복잡하게 들어간 형태

요구 사항 확인 기법

- ❖ 요구사항 개발 과정을 거쳐 문서화된 요구사항 관련 내용을 확인하고 검증하는 방법
 - ✓ 요구사항 검토(Requirement Reviews): 문서화된 요구사항을 보면서 확인하는 것으로 가장 일반적인 요구 사항 검증 방법
 - ✓ 프로토타이핑(Prototyping)
 - 초기 도출된 요구사항을 토대로 프로토타입(Prototype) 을 만든 후 대상 시스템의 개발이 진행되는 동안 도출되는 요구사항을 반영하면서 지속적으로 프로토타입을 재작성하는 방식
 - 프로토타입: 상품이나 서비스가 출시되기 전에 개발 대상 시스템 또는 그 일부분을 개략적으로 만든 원형
 - ✓ 모델 검증(Model Verification): 요구사항 분석 단계에서 개발된 모델이 요구사항을 충족시키는지 검증하는 것
 - ✓ 인수 테스트(Acceptance Test) : 사용자가 실제로 사용될 환경에서 요구사항들이 모두 충족되는지 사용자 입장에서 확인하는 과정

UML



UML

UML

- 분석, 설계를 비주얼 화, 문서화 하기 위한 그래픽 언어
- Unified
 - 이전의 OO 방법들의 통합
- Modeling
 - 객체지향 분석 설계를 위한 비주얼 모델링
- Language
 - 모형화된 지식(의미)을 표현



UML

❖ UML

- ✓ 시스템에 대한 지식을 찾고 표현하기 위한 언어
- ✓ 시스템을 개발하기 위한 탐구 도구
- ✓ 비주얼 모델링 도구
- ✓ 근거가 잘 정리된 가이드 라인
- ✓ 분석, 설계 작업의 마일스톤
- ✓ 실용적 표준
- ✓ 비주얼 프로그래밍 언어나 데이터베이스 표현도구 아님
- ✓ 개발 프로세스나 품질 보증 방안이 아님
- ✓ 시스템의 구조를 나타내는 6개의 구조 Diagram과 시스템 동작을 표현하는 7개의 행위 Diagram으로 표현
- ✓ 구성 요소는 사물, 관계, Diagram

UML

❖ 사물(Things)

- ✓ 모델을 구성하는 가장 중요한 기본 요소로, Diagram 안에서 관계가 형성될 수 있는 대상들
- ✓ 구조(Structural) 사물 : 시스템의 개념적, 물리적 요소를 표현
- ✓ 행동(Behavioral) 사물 : 시간과 공간에 따른 요소들의 행위를 표현
- ✓ 그룹(Grouping) 사물 : 요소들을 그룹으로 묶어서 표현
- ✓ 주석(Annotation) 사물 : 추가적인 설명이나 제약조건 등을 표현

❖ 관계(Relationship)

- ✓ 사물과 사물 사이의 연관성을 표현하는 것
- ✓ 연관(Association) 관계: 2개 이상의 사물이 서로 관련되어 있음을 표현함
- ✓ 집합(Aggregation) 관계: 하나의 사물이 다른 사물에 포함되어 있는 관계를 표현함
- ✓ 포함(Composition) 관계: 집합 관계의 특수한 형태로 포함하는 사물의 변화가 포함되는 사물에게 영향을 미치는 관계를 표현함
- ✓ 일반화(Generalization) 관계: 하나의 사물이 다른 사물에 비해 더 일반적인지 구체적인지를 표현함
- ✓ 의존(Dependency) 관계: 연관 관계와 같이 사물 사이에 서로 연관은 있으나 필요에 의해 서로에게 영향을 주는 짧은 시간 동안만 연관을 유지하는 관계를 표현함
- ✓ 실체화(Realization) 관계: 사물이 할 수 있거나 해야 하는 기능(행위, 인터페이스)으로 서로를 그룹화 할 수 있는 관계를 표현함

UML

❖ Diagram

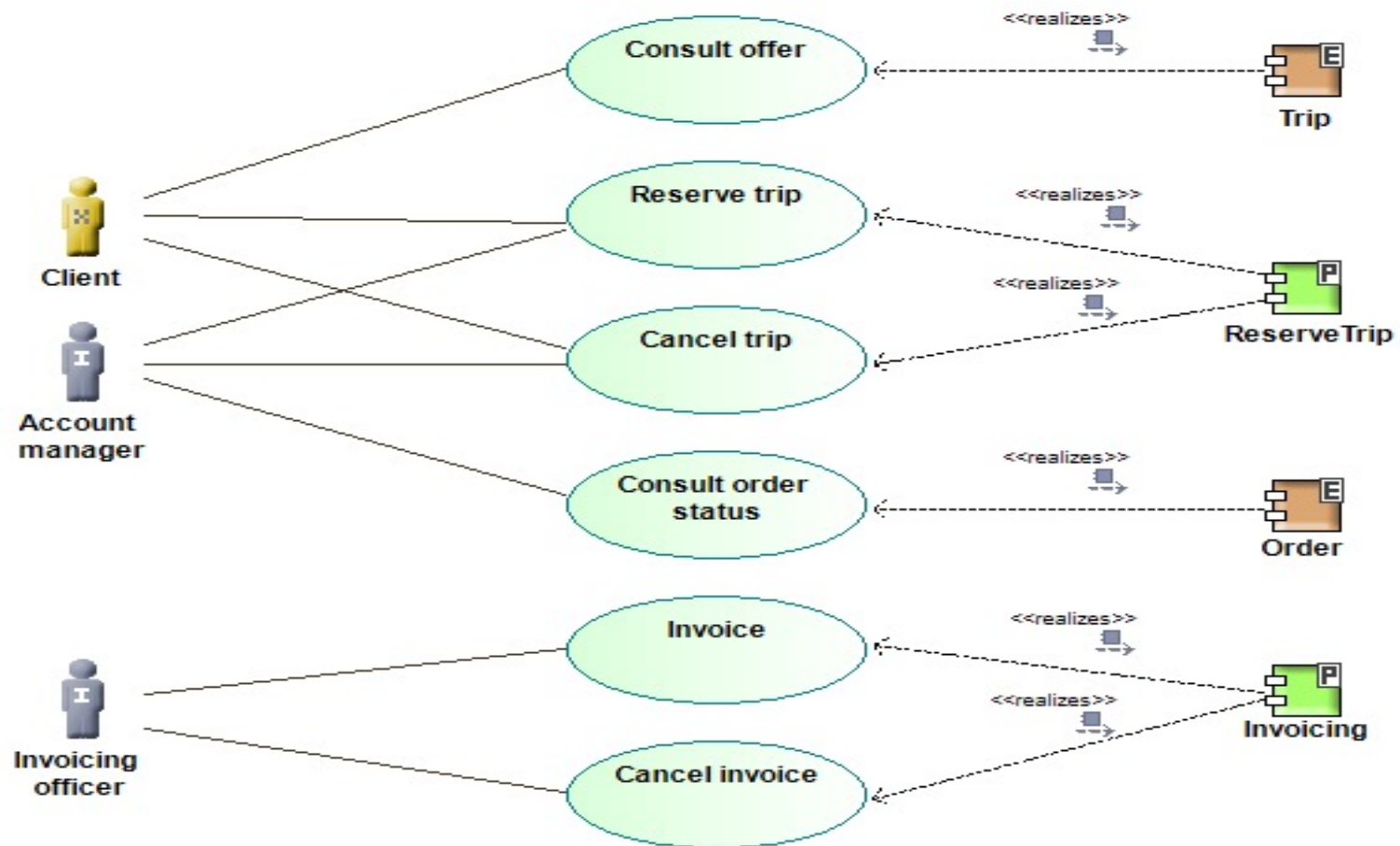
- ✓ 사물과 관계를 도형으로 표현한 것
- ✓ 정적 모델링에서는 주로 구조적 Diagram을 사용하고 동적 모델링에서는 주로 행위 Diagram을 사용
 - 정적 모델링: 사용자가 요구한 기능을 구현하는데 필요한 자료들의 논리적인 구조를 표현한 것
 - 동적 모델링: 시스템의 내부 구성 요소들의 상태가 시간의 흐름에 따라 변화하는 과정과 변화하는 과정에서 발생하는 상호 작용을 표현한 것
- ✓ 구조적 Diagram: Class Diagram, Object Diagram, Component Diagram, Deployment Diagram, Composite Structure Diagram, Package Diagram
- ✓ 행위 Diagram: Use Case Diagram, Sequence Diagram, Communication Diagram, State Diagram, Activity Diagram, Interaction Overview Diagram, Timing Diagram
- ✓ 스테레오타입(stereotype)
 - UML 확장 메커니즘의 한 부분으로써 매우 중요한 역할을 수행하는데 스테레오타입은 UML 모델링 요소들을 모델러의 기준에 따라 새로운 분류를 적용할 수 있도록 허용하는 메커니즘이며 스테레오타입은 모델러 마음대로 정의해서 적용하면 되는 것이고 특별한 제약이나 규칙은 없음
 - 스테레오타입은 각 요소에 "<<" ">>" 사이에 이름을 부여하면 되는데 "<<", ">>"는 꺾쇠 괄호(angle-bracket) 두개가 아니라 guillemets라 불리는 하나의 문자('«', '»')이지만 이러한 문자를 사용하기가 불편하기 때문에 꺾쇠 괄호 두 개를 써도 무방

Use Case Diagram

❖ Use Case Diagram

- ✓ 기능 모델링을 위한 도구
- ✓ 개발될 시스템과 관련된 외부 요소들, 즉 사용자와 다른 외부 시스템들이 개발될 시스템을 이용해 수행할 수 있는 기능을 사용자의 관점(View)에서 표현한 것으로 기능 모델링을 하기 위한 도구 중 하나
- ✓ 구성 요소 : 시스템 범위, 액터, 유스케이스, 관계
 - 시스템 범위 (System Scope): 시스템 내부에서 수행되는 기능들을 외부 시스템과 구분하기 위해 시스템 내부의 유스케이스들을 사각형으로 묶어 시스템의 범위를 표현
 - 액터(Actor): 시스템과 상호작용을 하는 모든 외부 요소로 사람이나 외부 시스템을 의미
 - 유스케이스 (Use Case): 사용자가 보는 관점에서 시스템이 액터에게 제공하는 서비스 또는 기능을 표현한 것
 - 관계 (Relationship): 유스케이스 다이어그램에서 관계는 액터와 유스케이스, 유스케이스와 유스케이스 사이에서 나타날 수 있으며 포함 관계, 확장 관계, 일반화 관계의 3종류가 있음

Use Case Diagram



Activity Diagram



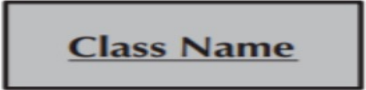




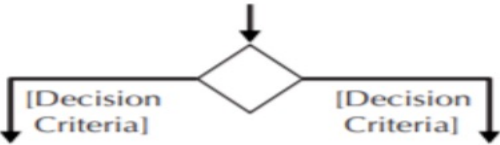
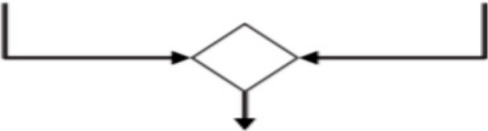
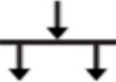
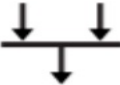
❖ Activity Diagram

- ✓ 기능 모델링을 위한 도구
- ✓ 자료 흐름도와 유사한 것으로 사용자의 관점(View)에서 시스템이 수행하는 기능을 처리 흐름에 따라 순서대로 표현한 것
- ✓ 활동 Diagram의 구성 요소 : 액션, 액티비티, 노드, 스웜레인 등
 - 액션(Action): 더 이상 분해할 수 없는 단일 작업
 - 액티비티(Activity): 몇 개의 액션으로 분리될 수 있는 작업
 - 노드
 - 시작 노드: 액션이나 액티비티가 시작 됨을 의미
 - 종료 노드: 액티비티 안의 모든 흐름이 종료됨을 의미
 - 조건(판단) 노드: 조건에 따라 제어의 흐름이 분리됨을 표현
 - 병합 노드 : 여러 경로의 흐름이 하나로 합쳐짐을 표현
 - 포크(Fork) 노드: 액티비티의 흐름이 분리되어 수행됨을 표현
 - 조인(Join) 노드: 분리되어 수행되던 액티비티의 흐름이 다시 합쳐짐을 표현
 - 스웜 레인 (Swim Lane): 액티비티 수행을 담당하는 주체를 구분

Activity Diagram

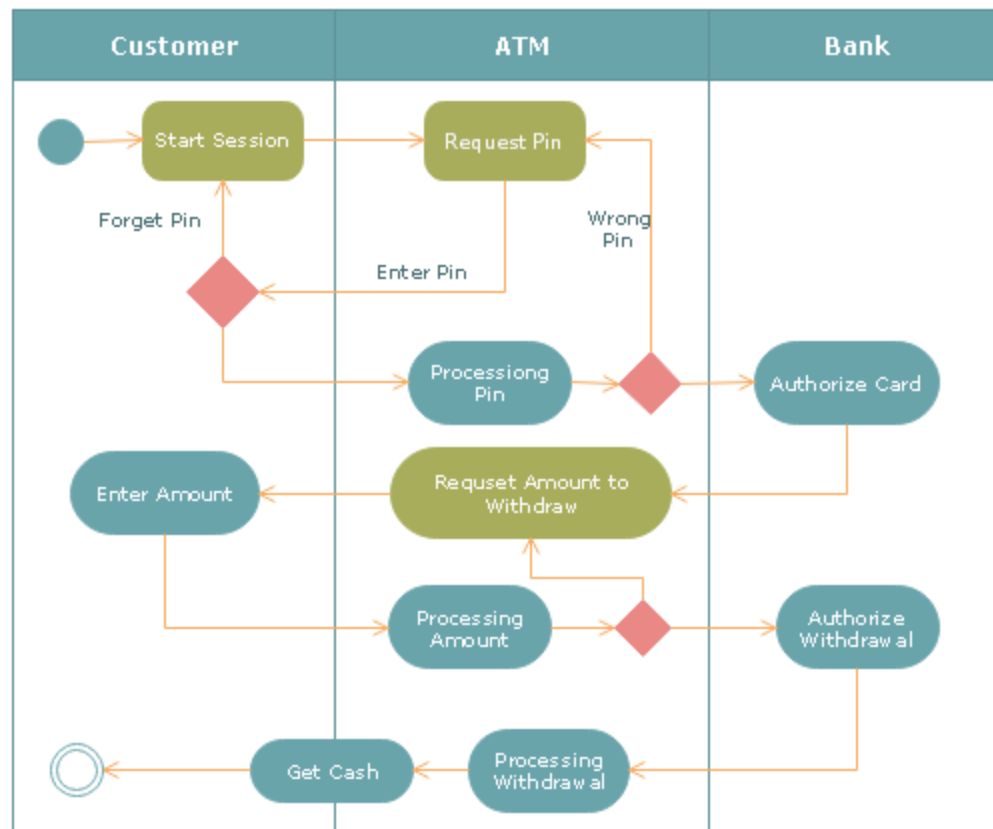
❖ Activity Diagram

- ✓ 활동 Diagram의 구성 요소 : 액션, 액티비티, 노드, 스웜레인 등

		
Action	Activity	Object node
		
Control flow	Object flow	
		
Initial node	Final Node	
		
Decision node	Merge node	
		
Fork node	Join node	

Activity Diagram

ATM Withdrawal Activity Diagram



Class Diagram

❖ Class Diagram

- ✓ 클래스가 가지는 속성 및 메소드 그리고 클래스 사이의 관계를 표현
- ✓ UML을 이용한 정적 모델링의 대표적인 것이 클래스 Diagram
- ✓ 클래스 Diagram의 구성 요소
 - 클래스(Class): 각각의 객체들이 갖는 속성과 오퍼레이션 (동작)을 표현
 - 일반적으로 3개의 구획(Compartment)으로 나누어서 클래스의 이름, 속성, 오퍼레이션을 표기
 - 속성(Attribute): 클래스의 상태나 정보를 표현
 - 오퍼레이션(Operation, 연산): 클래스가 수행할 수 있는 동작으로, 함수(메소드, Method)라고도 함
 - 제약조건(Constraint): 속성에 입력된 값에 대한 제약조건이나 오퍼레이션 수행 전 후에 지정해야 할 조건
 - 관계(RelationShip)
 - Association(연관): 하나의 클래스가 다른 클래스 안에서 사용되는 경우
 - Aggregation(집합): 다른 클래스에 존재하는 메소드의 매개변수로만 사용되는 경우
 - Composition(포함): 두 개의 클래스가 서로 다른 클래스를 사용하는 경우
 - Generalization(일반화): 상속 관계
 - Dependency(의존): 특정 메소드를 호출할 때만 영향을 받는 클래스와의 관계

Class Diagram

클래스 나타내기

□ 박스 위에 클래스 이름

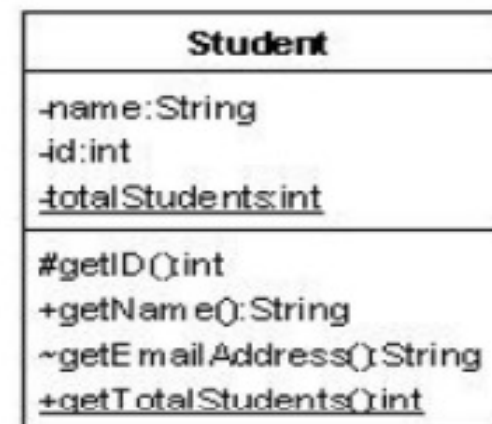
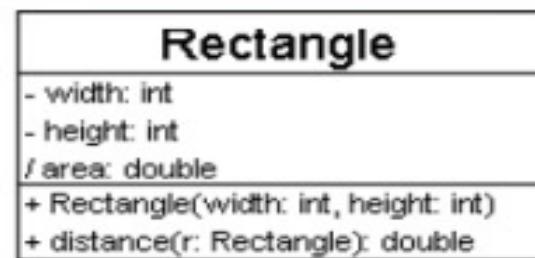
- 추상 클래스는 이탤릭체
- 인터페이스 클래스는 <<interface>> 추가

□ 속성

- 객체가 가지는 모든 필드를 포함

□ 오퍼레이션/메소드

- 아주 흔한 메소드(get/set)는 생략
- 상속된 메소드도 포함할 필요 없음



Class Diagram

클래스 속성

속성(필드, 인스턴스 변수)

visibility name: type[count] = default value

- visibility: + public
- # protected
- private
- ~ package(디폴트)
- / derived

Underline static variable

파생된 속성: 저장되지 않고 다른 속성값으로부터 계산됨

Rectangle
- width: int
- height: int
/ area: double
+ Rectangle(width: int, height: int)
+ distance(r: Rectangle): double

Student
-name:String
-id:int
<u>-totalStudents:int</u>
#getID():int
+getName():String
~getEmailAdress():String
<u>+getTotalStudents():int</u>

Class Diagram

클래스 오퍼레이션/메소드

□ 오퍼레이션/메소드

○ **visibility** name(**parameters**) : *return_type*

○ visibility: + public
 # protected
 - private
 ~ package(디폴트)

○ Underline static method

○ 파라메타 타입 (name: type)

○ 생성자나 리턴 타입이 void인 경우는
return_type 생략

Rectangle
- width: int - height: int / area: double
+ Rectangle(width: int, height: int) + distance(r: Rectangle): double

Student
-name:String -id:int <u>-totalStudents:int</u>
#getID():int +getName():String ~getEmailAdress():String <u>+getTotalStudents():int</u>

Class Diagram

클래스 사이의 관계

□ 일반화(generalization): 상속(isa) 관계

- 클래스 사이의 상속
- 인터페이스 구현

□ 연관(association): 사용(usage) 관계(3 종류)

- 의존
- 집합(aggregation): 어떤 클래스가 다른 클래스의 모임으로 구성
- 합성(composition): 포함된 클래스가 컨테이너 클래스가 없이는 존재할 수 없는 집합관계의 변형

Class Diagram

일반화 관계

□ 일반화(상속)

- 부모를 향한 화살표로 표시되는 하향 계층 관계
- 선/화살표는 부모 클래스의 종류에 따라 다름

❖ 클래스:

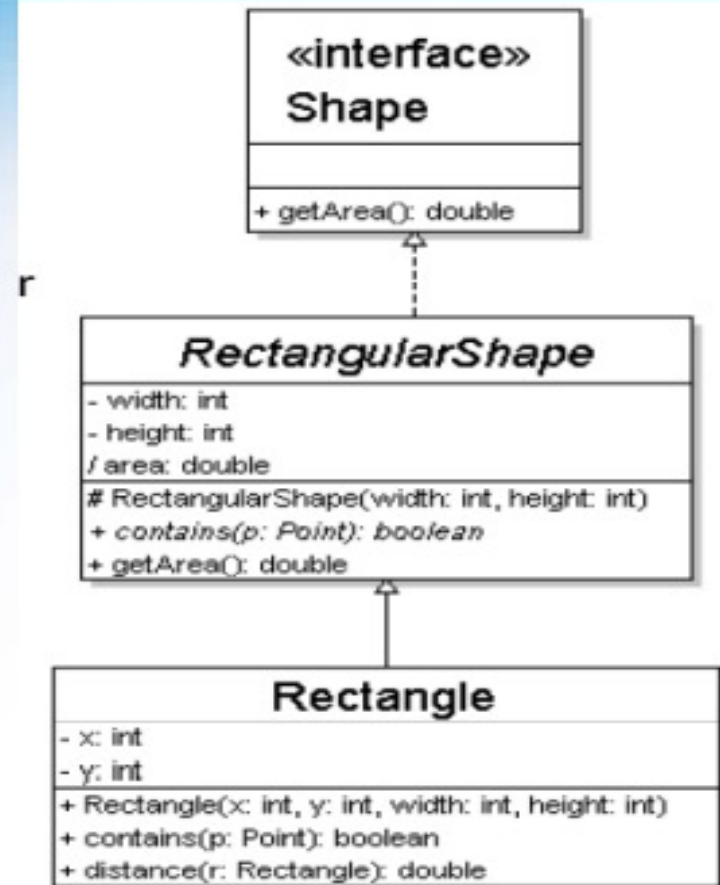
실선/검은 헤드 화살표

❖ 추상 클래스:

실선/흰 헤드 화살표

❖ 인터페이스:

점선/흰 헤드 화살표



Class Diagram

연관 관계

연관(association): 어떤 클래스의 인스턴스가 작업을 수행하기 위하여 다른 클래스를 알아야 하는 함

1. 다중도(multiplicity)

- * ⇔ 0, 1, or more
- 1 ⇔ 정확히 1개
- 2..4 ⇔ 2개 내지 4개
- 3.. * ⇔ 3개 이상

2. 이름 – 객체들의 관계 이름

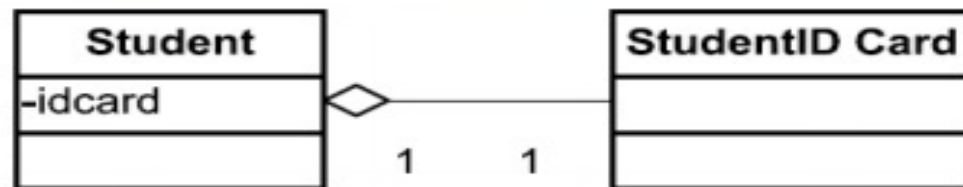
3. 방향성(navigability) – 질의의 방향, 객체 사이의 선으로 표시하며 양쪽 방향인 경우는 화살표시 없음

Class Diagram

연관 관계의 다중도

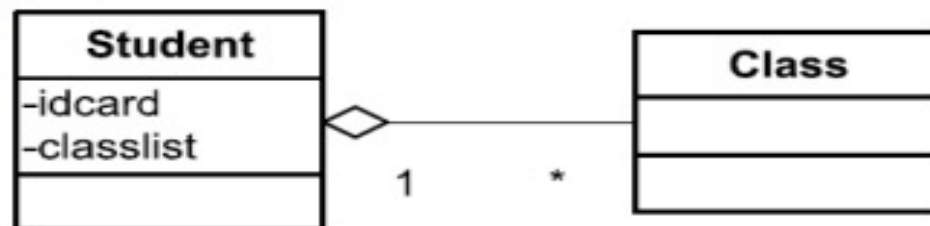
□ 1 대 1

○ 학생 1명이 학생증(id card) 한 개만을 가진다.



□ 1 대 다

○ 학생 1명이 여러 클래스를 수강할 수 있다.



Sequence Diagram

❖ 시퀀스(Sequence) Diagram

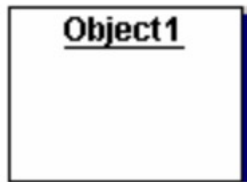
- ✓ 시스템이나 객체들이 메시지를 주고받으며 시간의 흐름에 따라 상호 작용하는 과정을 그림으로 표현한 것
- ✓ 시퀀스 Diagram의 구성 요소: 액터, 객체, 라이프라인, 활성화 상자, 메시지, 객체 소멸, 프레임 등
- ✓ 구성 요소
 - 액터(Actor): 시스템으로부터 서비스를 요청하는 외부 요소로, 사람이나 외부 시스템
 - 객체(Object): 메시지를 주고받는 주체로 콜론(:)을 기준으로 앞쪽에는 객체명을 뒤쪽에는 클래스명 기술
 - 라이프라인 (Lifeline): 객체가 메모리에 존재하는 기간으로, 객체 아래쪽에 점선을 그어 표현
 - 활성화 상자 (Activation Box): 객체가 메시지를 주고받으며 구동되고 있음을 라이프라인 상에 겹쳐 직사각형 형태로 표현
 - 메시지(Message): 객체가 상호 작용을 위해 주고받는 메시지
 - 객체 소멸: 라이프라인 상에서 객체 소멸 표시를 만나면 해당 객체는 더 이상 메모리에 존재하지 않음을 의미
 - 프레임(Frame): Diagram의 전체 또는 일부를 묶어 표현

Sequence Diagram

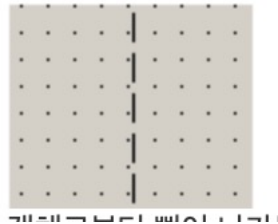
❖ 시퀀스(Sequence) Diagram

✓ 구성 요소

2.1 객체(Object)



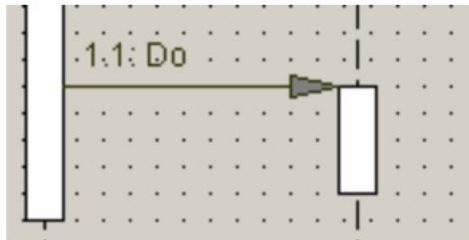
2.2 생명선(Lifeline)



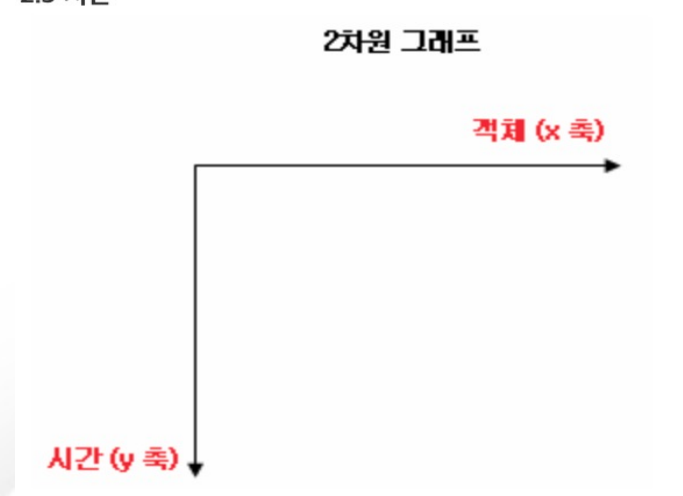
2.3 실행(activation)



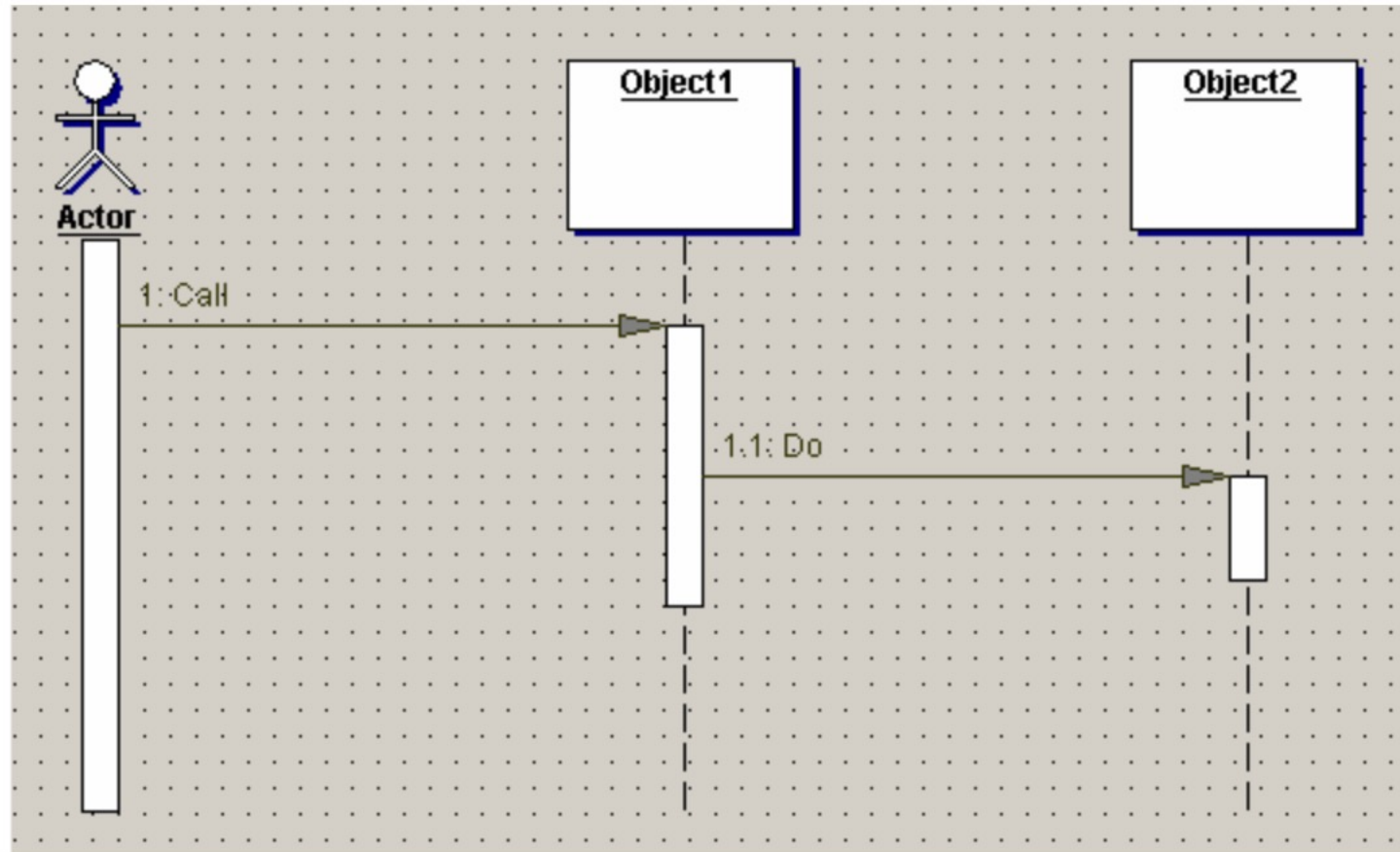
2.4 메시지



2.5 시간



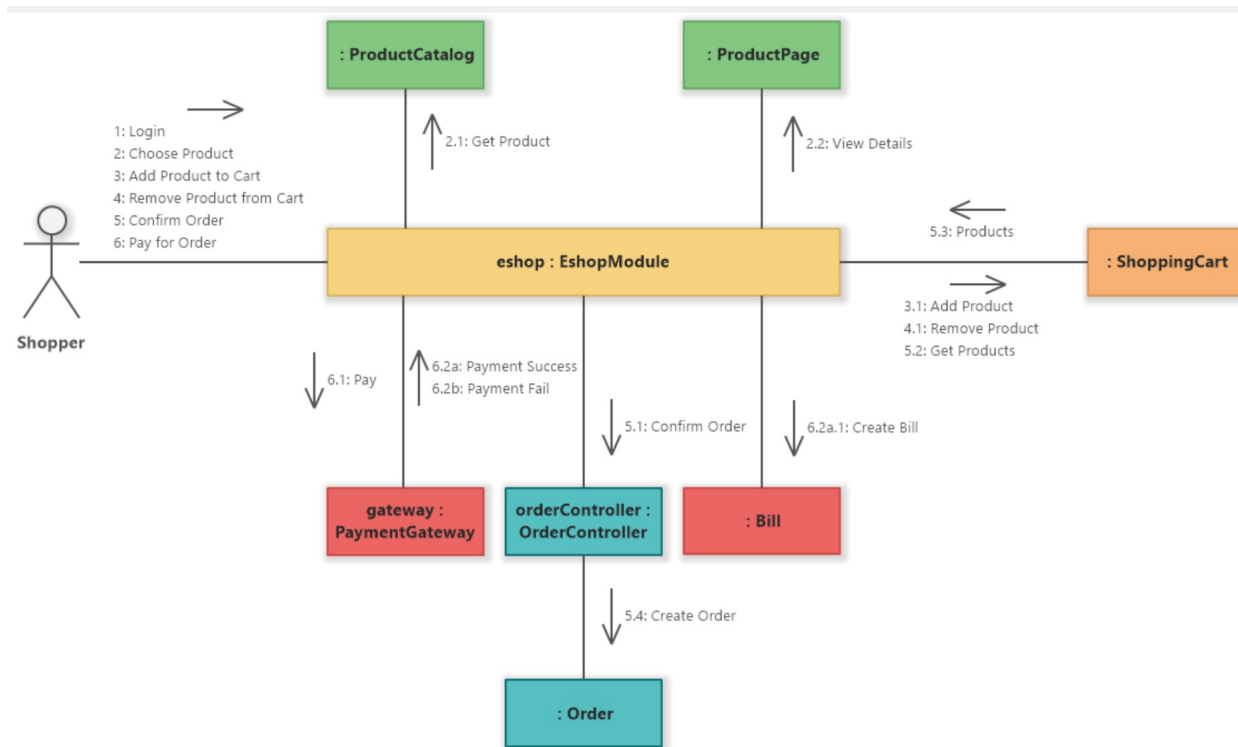
Sequence Diagram



Communication Diagram

❖ Communication Diagram

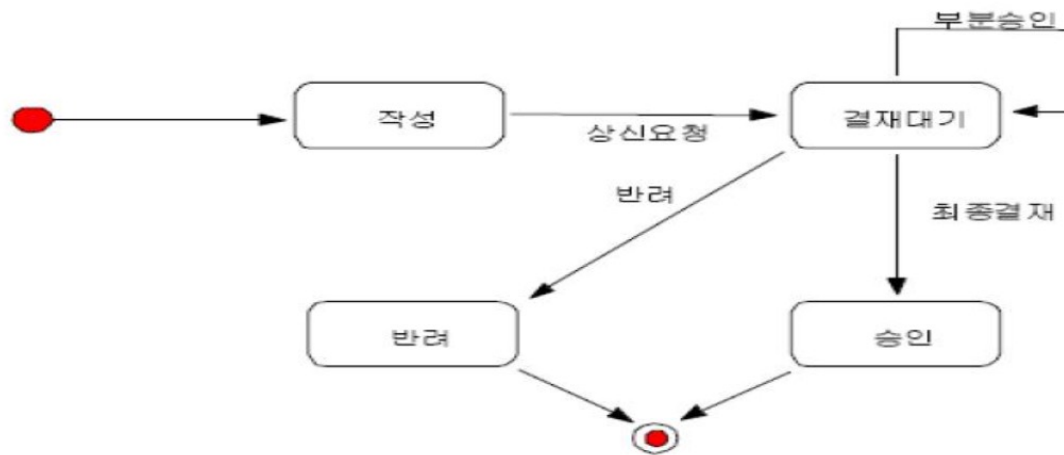
- ✓ 시퀀스 Diagram과 같이 동작에 참여하는 객체들이 주고받는 메시지를 표현하는데, 메시지뿐만 아니라 객체들 간의 연관까지 표현
- ✓ 커뮤니케이션 Diagram의 구성 요소: 액터, 객체, 링크, 메시지 등
 - Link
 - 객체들 간의 관계를 표현하는 데 사용
 - 액터와 객체, 객체와 객체 간에 실선을 그어 표현



State Diagram

❖ State Diagram

- ✓ 하나의 객체가 자신이 속한 클래스의 상태 변화 혹은 다른 객체와의 상호 작용에 따라 상태가 어떻게 변화하는지를 표현
- ✓ 구성요소
 - 상태(State): 객체의 상태를 둥근 사각형 안에 기술
 - 시작 상태: 상태의 시작을 속이 채워진 원(●)으로 표현
 - 종료 상태: 상태의 종료를 속이 채워진 원을 둘러싼 원(⊙)으로 표현
 - 이벤트(Event): 조건, 외부 신호, 시간의 흐름 등 상태에 변화를 주는 현상
 - 상태 전환: 상태 사이의 흐름, 변화를 화살표로 표현
 - 프레임(Frame): 상태 Diagram의 범위를 표현



Package Diagram

❖ Package Diagram

- ✓ 유스케이스나 클래스 등의 모델 요소들을 그룹화 한 후 그룹된 단위들 사이의 의존된 관계를 표현하기 위한 Diagram
- ✓ 아이콘 안에 패키지 이름만을 표기한 단순 표기법과 패키지에 포함된 내부 패키지 or 클래스까지 표현한 확장 표기법이 있음
- ✓ 구성 요소
 - Package: 객체들을 그룹화 한 것



□ Object



□ Dependency: 패키지와 패키지 사이의 관계



프로젝트 관리

❖ 프로젝트 관리

- ✓ 특정한 목적을 달성하기 위해 개발 계획 프로그램을 수립하고 프로그램의 분석, 구현 등의 작업을 수행하는 것
- ✓ 주어진 기간 내에 최소의 비용으로 사용자의 요구를 만족시키는 시스템을 개발하기 위해 수행하는 것
- ✓ 프로젝트 관리 대상
 - 사람(People)
 - 문제(Problem)
 - 프로세스(Process)
- ✓ 프로젝트 계획 및 예측
 - 개발 과정에서의 제품 비용 초과, 일정, 성능, 품질 저하 등에 영향을 미치므로 개발 공정과 제품생산에 대한 계획을 세우는데 신중
 - 예측을 뒤로 미루는 것은 가장 비현실적인 방법

프로젝트 관리

❖ 프로젝트 관리

✓ 관리 유형

- 일정 관리: 작업 순서, 작업 기간 산정, 일정 개발, 일정 통제
- 비용 관리: 비용 산정, 비용 예산 편성, 비용 통제
- 인력 관리: 프로젝트 팀 편성, 자원 산정, 프로젝트 조직 정의, 프로젝트 팀 개발 및 관리, 자원 통제
- 위험 관리: 위험 식별, 위험 평가, 위험 대처, 위험 통제
- 품질 관리: 품질 계획, 품질 보증 수행, 품질 통제 수행
- 형상 관리: 프로젝트 내의 모든 과정의 변경 사항을 관리하는 것으로 형상 제어라고도 함

프로젝트 관리

❖ 비용 산정 요소

✓ 프로젝트 요소

- 제품 복잡도: 소프트웨어의 종류에 따라 발생할 수 있는 문제점들의 난이도
- 시스템 크기: 소프트웨어의 규모에 따라 개발해야 할 시스템의 크기
- 요구되는 신뢰도: 일정 기간 내 주어진 조건하에서 프로그램이 필요한 기능을 수행하는 정도

✓ 자원 요소

- 인적 자원: 소프트웨어 개발 관련자들이 갖춘 능력 혹은 자질
- 하드웨어 자원: 소프트웨어 개발 시 필요한 장비와 워드프로세서, 프린터 등의 보조 장비
- 소프트웨어 자원: 소프트웨어 개발 시 필요한 언어 분석기, 문서화 도구 등의 개발 지원도구

✓ 생산성 요소

- 개발자 능력: 개발자들이 갖춘 전문지식, 경험, 이해도, 책임감, 창의력 등
- 개발 기간: 소프트웨어를 개발하는 기간

프로젝트 관리

❖ 비용 산정 방식

✓ 하향식 산정법

- 전문가의 감정: 두 명 이상의 전문가에게 의뢰
- 델파이식 산정: 조정자를 두는 형태

✓ 상향식 산정법

- LOC 기법: 원시 코드 라인수를 측정해서 산정 PERT를 이용
- 개발 단계 별 Man / Month: 인/월 수 기법 몇 명이 몇 개월 동안 작업하느냐에 중점을 둔 방법으로 1명이 1개월 동안 작업 하는 것을 1MM이라고 하고 1명이 10개월 동안 작업하는 것과 10명이 1개월 동안 하는 것을 동일한 것으로 간주하지만 1명이 10개월 동안 하는 것이 바람직 - 브룩스의 법칙(Brooks' law)은 "지체되는 소프트웨어 개발 프로젝트에 인력을 더하는 것은 개발을 늦출 뿐이다"라고 주장한 법칙

프로젝트 관리

❖ 비용 산정 방식

✓ 상향식 산정법

● 수학적 산정법(경험적 측정법)

- COCOMO(CONstructive COst MOdel): 비용 추정 단계 및 적용 변수의 구체화 정도
 - Basic COCOMO(크기와 개발 유형만을 고려), Intermediate COCOMO(제품의 특성, 컴퓨터의 특성, 개발 요원의 특성, 프로젝트 특성을 고려), Detailed COCOMO(대형 시스템에 대하여 서브시스템의 상이한 특성을 반영하여 비용을 개별 산정한 후 합산)로 분류
 - 복잡도에 따라서 기본형(organic model – 5만 라인 이하), 중간형(semi-detached model – 30만 라인 이하) , 내장형(embedded model – 30만 라인 이상) 으로 분류
- Putnam 의 생명주기 예측 모형
- 기능점수(Function Point) 모형

✓ 비용 산정 자동화 도구

- SLIM
- ESTIMACS

프로젝트 관리

❖ 일정 관리

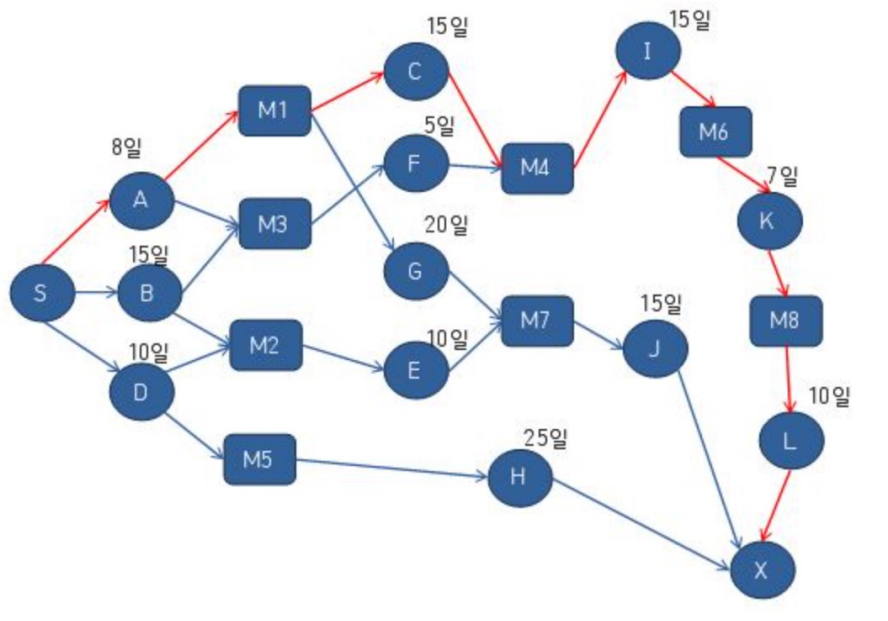
- ✓ 프로젝트 일정 관리(Project Schedule Management)는 프로젝트의 성공적인 완성을 위한 구체적인 계획을 일정에 따라 효율적으로 자원을 배분하는 것
- ✓ 프로젝트 일정을 관리하기 위해서는 프로젝트의 내용과 범위를 기반으로 일정 활동(Schedule Activity)를 식별한 후 일정 별 활동의 의존관계를 파악하고 각 활동의 소요 기간을 추측하여 분배하는 것이 중요
- ✓ 프로젝트 일정 계획은 프로젝트가 시작하는 단계에서부터 종료될 때까지 프로젝트 일정을 바탕으로 정책과 절차를 문서화하기 위한 계획을 의미
- ✓ 프로젝트 일정 계획을 만들기 위해서는 WBS, PERT/CPM, Gantt 차트 등을 이용
- ✓ WBS(Work Breakdown Structure): 업무 분업 구조, 작업 분해 구조, 작업 분류 체계, 작업 분할 구조, 작업 분할 구도는 프로젝트 관리와 시스템 공학 분야에서 프로젝트의 더 작은 요소로 분해시킨 딜리버러블 지향 분업 구조
- ✓ PERT(Program Evaluation Review Technique)
 - 네트워크를 이용하여 사업계획을 효과적으로 수행할 수 있도록 이를 일정, 노력 및 비용 등과의 관련하에 과학적으로 계획하고 관리하는 종합적인 일정관리 기법
 - 시간에 중점을 둔 것이 PERT / time, 비용절감도 동시에 고려할 수 있는 것이 PERT / cost

프로젝트 관리

❖ 일정 관리

✓ CPM(Critical Path Method, 임계 경로)

- Project 관리에서 공기의 단축이 요구될 때 작업자나 설비를 초과 투입하여 주 공정상의 어느 작업을 단축해 최소비용 증가로 공사기간을 최소화하는 기법으로 각 작업의 시간당 비용증가를 비교하여 최소비용증가 작업부터 단계적으로 단축함으로써 공사 비용의 증가를 최소화하는 방법으로 비용과 시간을 동시에 고려

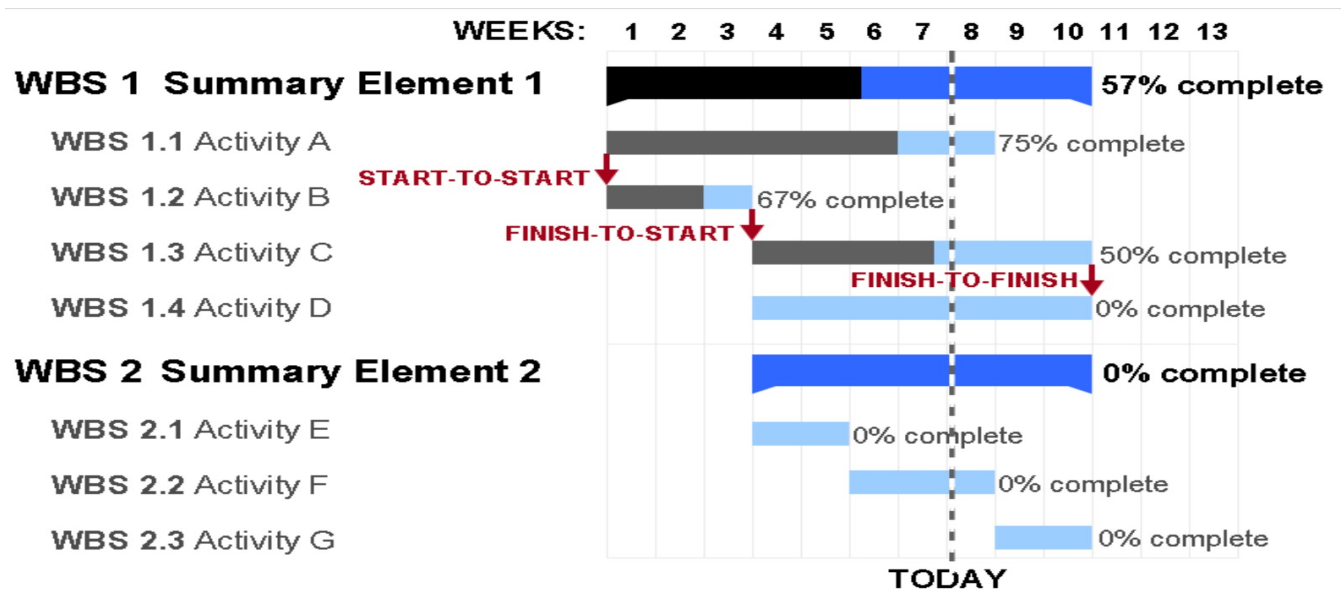


프로젝트 관리

❖ 일정 관리

✓ Gantt Chart

- 프로젝트 일정관리를 위한 바(bar)형태의 도구로서 각 업무별로 일정의 시작과 끝을 그래픽으로 표시하여 전체 일정을 한눈에 볼 수 있으며 각 업무(activities) 사이의 관계를 보여줄 수도 있음



표준화

❖ 소프트웨어 개발 표준

- ✓ 소프트웨어 개발 단계에서 수행하는 품질 관리에 사용되는 국제 표준

❖ ISO/IEC 12207

- ✓ ISO에서 만든 표준 소프트웨어 생명 주기 프로세스
- ✓ 시스템(소프트웨어) 서비스를 수행하기 위한 절차, 활동, 획득, 설정에 대한 소프트웨어 개발 생명주기 단계별로 필요한 프로세스와 각 프로세스에서 나오는 산출물을 포함
- ✓ 표준에는 23개의 프로세스, 95개의 활동, 224개의 산출물을 정의
- ✓ 구분
 - 기본 생명 주기: 획득, 공급, 개발, 운영, 유지보수
 - 지원 생명 주기: 품질 보증, 검증, 확인, 활동 검토, 감사, 문서화, 형상 관리, 문제 해결
 - 조직 생명 주기: 관리, 기반 구조, 훈련, 개선

표준화

❖ CMMI(Capability Maturity Model Integration)

- ✓ 소프트웨어 개발 및 전산장비 운영 업체들의 업무 능력 및 조직의 성숙도를 평가하기 위한 모델
- ✓ 기존 능력 성숙도 모델(CMM)을 발전시킨 것으로서 기존에 소프트웨어 품질보증 기준으로 사용되던 SW-CMM과 시스템 엔지니어링 분야의 품질보증 기준으로 사용되던 SE-CMM을 통합하여 개발한 후속 평가 모델로 카네기 멜론 대학의 소프트웨어 공학 연구소에서 개발
- ✓ 단계
 - 레벨 1(Initial) -개인의 역량에 따라 프로젝트의 성공과 실패가 좌우된다. 소프트웨어 개발 프로세스는 거의 없는 상태를 의미
 - 레벨 2(Managed) - 프로세스 하에서 프로젝트가 통제되는 수준으로 조직은 프로세스에 대한 어느 정도의 훈련이 되었다고 볼 수는 있지만, 일정이나 비용과 같은 관리 프로세스 중심이다. 기존 유사 성공사례를 응용하여 반복적으로 사용
 - 레벨 3(Defined) - 레벨 2에서는 프로젝트를 위한 프로세스가 존재한다면 레벨 3에서는 조직을 위한 표준 프로세스가 존재한다. 모든 프로젝트는 조직의 프로세스를 가져다 상황에 맞게 조정하여 승인받아 사용
 - 레벨 4(Quantitatively Managed) - 소프트웨어 프로세스와 소프트웨어 품질에 대한 정량적인 측정이 가능해지는데 조직은 프로세스 데이터베이스를 구축하여 각 프로젝트에서 측정된 결과를 일괄적으로 수집하고 분석하여 품질평가를 위한 기준으로 삼는다.
 - 레벨 5(Optimizing) - 이 레벨에서는 지속적인 개선에 치중한다. 조직적으로 최적화된 프로세스를 적용하여 다시 피드백을 받아 개선하는 상위 단계이다.

표준화

❖ SPICE

- ✓ 소프트웨어 프로세스 심사(SPA)를 위한 국제표준인 ISO/IEC 15504의 개발을 지원하기 위하여 진행중인 국제표준화 프로젝트
- ✓ 소프트웨어 심사표준에 대한 노력은 1980년대 미국과 영국의 군사부문에 의해서 최초로 시작
- ✓ 개발중인 소프트웨어의 품질을 개선시키고, 소프트웨어 프로젝트와 관련된 리스를 감소시키며 소프트웨어 개발 업체의 선정 메커니즘을 개선할 목적으로 출발
- ✓ 단계
 - Level 0, Incomplete: 프로세스의 목적 달성에 전반적으로 실패하는 상태
 - Level 1, Performed: 프로세스의 목적이 전반적으로 달성되나, 적극적으로 계획되거나 추적되지 않는 상태
 - Level 2, Managed: 프로세스가 명시된 절차에 따라 WP를 산출하고, 프로세스가 계획되고 추적되는 상태
 - Level 3, Established: "정의된 프로세스(Defined Process)"를 사용하여 프로세스를 수행하고 관리하는 상태
 - Level 4, Predictable: 정의된 프로세스가 일정한 통제범위(Control Limits)내에서 일관되게 수행되는 상태
 - Level 5, Optimizing: 현재 및 미래의 사업 needs에 맞게 프로세스가 최적화되게 변경되어, 정의된 사업목표를 계속적으로 충족시킬 수 있는 상태

테일러링

❖ tailoring

- ✓ 시스템 개발이나 홈페이지 제작 등 프로젝트 진행 시 회사의 표준 방법론이나 표준 산출물을 활용하여 개별 과제의 특성에 딱 맞도록 방법론과 산출물을 보완하는 방식
- ✓ 해당 과제에 맞지 않는 일반적인 방법론이나 산출물 목록은 제거하고 꼭 필요한 방법론과 산출물만 남기는 보완을 통해서 해당 과제에 딱 맞는 최적화된 결과를 생성
- ✓ 수행 절차
 - 프로젝트 특징 정의 – 표준 프로세스 선정/검증 – 상위 수준 커스터마이징 – 세부 커스터마이징 – 테일러링 문서화
- ✓ 고려 사항
 - ❑ 내부적 요소: 요구사항, 프로젝트 규모, 보유 기술
 - ❑ 외부적 요소: 법적 제약 사항, 표준 품질 기준

SW 공학의 3R

- ❖ SW Engineering: 소프트웨어의 개발, 운용, 유지보수 등의 생명 주기 전반을 체계적이고 서술적이며 정량적으로 다루는 학문으로 공학을 소프트웨어에 적용하는 것
- ❖ SW 공학의 3R
 - ✓ 역공학(Reverse Engineering): 자동화된 도구의 도움으로 물리적 수준의 정보를 논리적인 소프트웨어 정보의 서술로 추출하는 프로세스
 - ✓ 재공학(Re-engineering): 자동화된 도구로 현존하는 시스템을 점검 또는 수정하는 프로세스
 - ✓ 재사용(Reuse): 이미 개발되어 기능, 성능 및 품질을 인정받았던 소프트웨어의 전체 또는 일부분을 다시 사용
- ❖ SW 리팩토링은 외부 동작을 바꾸지 않으면서 내부 구조를 개선하는 방법으로 소프트웨어 시스템을 변경하는 프로세스로 소프트웨어를 보다 이해하기 쉽고, 수정하기 쉽도록 만드는 것, 겉으로 보이는 소프트웨어의 기능을 변경하지 않는 것
- ❖ CASE(Comput Aided Software Engineering)
 - ✓ 소프트웨어 위기 현상을 해결하기 위하여 소프트웨어 생산 자체를 컴퓨터의 도움으로 자동화 시키자는 개념으로 탄생한 것
 - ✓ 수작업으로 프로그래밍하고 문서 산출물을 만드는 대신, CASE를 활용하여 산출물을 작성하고 프로그램 코드를 만들어 내는 것
 - ✓ 자동화를 통한 개발 기간의 단축 뿐 아니라 생산하는 소프트웨어의 일관성과 통합성, 완전성을 높임으로서 생산성 향상과 품질의 향상을 가져온다는 개념

SW 개발 프레임워크

- ❖ SW 개발 프레임워크: SW 개발에 공통적으로 사용되는 구성 요소와 아키텍처를 일반화하여 쉽게 구현할 수 있도록 여러가지 기능을 제공하는 소프트웨어 시스템
 - ✓ 프레임워크가 가져야 할 성질
 - ❑ 모듈화
 - ❑ 재사용성
 - ❑ 확장성
 - ❑ 제어의 역전(역흐름, Inversion of Control) – 인스턴스의 관계 및 수명 주기를 프레임워크가 관리하는 것
 - ✓ Spring: 자바(코틀린 지원) 플랫폼을 위한 오픈 소스 경량형 프레임워크
 - ✓ 전자정부 프레임워크: 공공부문 정보화 사업 시 효율적인 정보 시스템 구축을 지원하기 위한 프레임워크
 - ✓ .Net: MS Windows 프로그램의 개발 및 실행 환경을 제공하는 프레임워크

명명 법

❖ 명명하는 방법

✓ PascalCasing (파스칼 케이싱)

- 클래스, 열거형, 이벤트, 메서드 등의 이름을 만들 때에는 대문자로 시작하는 변수명을 사용
- 복합어일 경우 중간에 시작하는 새로운 단어는 대문자로 적는다.

✓ CamelCasing (카멜 케이싱)

- 메서드의 매개변수의 이름에 적용되는데 첫번째 문자는 소문자로 시작하고 복합어 일 경우 파스칼 케이싱과 동일하게 적용

✓ GNU Naming Convention

- Linux의 프로젝트들은 GNU Naming Convention이라는 형태의 명명법을 주로 사용
- 모두 소문자를 사용하고 복합어 사이를 '_'를 사용하여 연결

✓ BREW Naming Convention

- BREW 는 Qualcomm에서 만든 플랫폼으로 국내의 휴대폰 제조사들은 초기부터 현재까지 이 코드들을 많이 사용
- 기존 명명법을 조합한 형태로 변종 명명법이지만 익숙함을 벗어나지 못하는 국내 제조사의 개발자들이 선호하는 형태
- 클래스나 인터페이스를 대문자나 파스칼 케이싱으로 앞에 두고, '_' 이후에 다시 파스칼 케이싱 형태의 메서드 명을 추가

예) IDISPLAY_ClearScreen다.

명명 법

❖ 명명하는 방법

✓ Hungarian notation (헝가리안 표기법)

- Microsoft 의 개발자 중 헝가리 사람의 프로그래머가 쓰던 변수 명명법으로 MS 내부에서 따라쓰기 시작하던 것이 점차 전세계의 프로그래머들에게 널리 퍼져 이젠 프로그램 코딩시 변수 명명의 표준적인 관례가 됨

예) g_bTrue

- 첫글자 g는 전역변수, m은 멤버변수를 의미한다. 전역이나 멤버변수의 경우에는 그 다음에 _ 를 적는다.
- b는 Boolean타입을 의미하고 True가 의미있는 이름이다.

예) nCnt

- 전역이나 멤버변수가 아니므로 g_ 나 m_ 가 없다.
- n과 i는 자연수를 뜻하며 i는 주로 인덱스에 사용하고 n은 카운트의 목적에 주로 사용
- 의미있는 이름이 길 경우에는 자음만을 사용

✓ Constant (상수)

- 거의 모든 명명법에서 상수를 표기하는 방법은 거의 동일
- 모든 문자를 대문자로 사용하는 GNU Naming Convention의 형태를 사용 – Snake 표기법
예) DEFAULT_COUNTRY_CODE