Chap. 5
임베디드 소프트웨어 요구사항 분석

Software Engineering

요구사항의 개요(1)

- 요구사항(Requirements) 이란?
  - 사용자의 필요성, 계약상에 명시된 내용, 표준 사양서, 작성된 공식 문서 등에서 찾아낸 시스템이 반드시 수행해야 할 조건과 기능을 나타낸 것

✓ *고객이 시스템으로부터 원하는 서비스*

✓ *시스템을 개발하는 동안 만족시켜야 할 제약*

✓ *시스템이 사용되는 동안 만족시켜야 할 제약*

  - 요구사항은 시스템 설계의 기반
  - 요구사항은 구현의 정확성을 판단하는 기준
  - 요구사항은 시험 시 테스트 케이스를 (자동으로) 생성하는 기반

요구사항의 개요(2)

- 요구공학(Requirements Engineering) 이란?
  - 요구사항을 정의하고 문서화하는데 필요한 요구사항을 추출, 분석, 기술, 검증, 유지보수 및 관리를 포함한 제반 공정에 대한 체계적인 접근 분야

✓ *요구사항을 확립해 나가는 프로세스*

  - 요구공학의 중요성
    - 고객과 개발자가 요구사항의 의미를 서로 다르게 이해하는 경우(의사소통의 실패) 시스템 개발이 실패할 가능성 증대
    - 시스템이 사용되는 동안 발견되는 오류의 상당 부분이 요구사항의 오류에서 파생된 것

요구사항의 개요(3)

- 소프트웨어 요구사항 분석(Requirement Analysis)
  - 소프트웨어에 관한 요구가 무엇인가(즉, 소프트웨어가 무엇을 해야 하는가)를 추출하여 소프트웨어 요구사항 명세를 작성하는 작업
    - 명세를 바탕으로 해서 소프트웨어의 설계 및 구현 작업을 전개
  - 사용자의 요구를 추출하여 목표를 정하고 어떤 방식으로 해결할 것인지를 결정하는 단계 – 무엇(what)에 초점
    - 구현될 시스템의 기능이나 목표, 제약사항 등을 정확히 파악해야 함
    - 사용자로부터 시스템 개발에 필요한 정보를 얻어내는 단계
  - 소프트웨어 개발의 출발점 -> 실질적인 첫 단계
  - 목표 : 소프트웨어 요구사항 명세서 또는 시스템의 모형을 획득

요구사항의 개요(4)

- 요구사항 명세서 (Requirement Specification)
  - 소프트웨어 제품이 무엇(what)을 어느 정도 잘 (how well) 달성해야 하는가를 기술
  - 소프트웨어 개발에 참여하는 사람들(사용자, 시스템 분석가, 프로젝트 관리자 등)의 공통 기반이 되는 것
    - 구성원의 다양성을 인식하고 원활한 의사소통을 위해 사용자 관점의 용어 선정이 필요
  - 명세서를 바탕으로 소프트웨어의 설계 및 구현 작업이 전개

요구사항의 개요(5)

- 요구사항 관리
  - 변경된 요구사항을 찾아서 문서화하고 추적성을 관리하는 시스템적 접근
  - 시스템을 바르게 정의,구축, 테스트, 문서화하는지를 확실하게 하는 행위
  - 요구사항 도출, 유스케이스화, 컴포넌트 식별, 애플리케이션 간의 추적성 보장이 필요

- 추적 가능성(traceability)
  - 요구사항의 추적 가능하다는 것은 각각의 요구사항은 그 출처(source) 및 그와 관련이 있는 다른 요구사항과 서로 연결되어 있어야 한다는 의미
  - 요구사항 명세가 가져야 할 특성의 하나로, 관련된 요구사항을 얼마나 쉽게 찾아낼 수 있는가를 나타냄
  - 추적을 가능하게 하기 위한 기법
    - 모든 요구사항에 고유한 번호 부여
      - 관련된 요구사항들을 이 고유 번호를 사용하여 서로 참조하도록 함

요구사항 유형

- 기능적(functional) 요구사항
  - ✓ *시스템이 수행해야 하는 행위들을 구체화한 것*

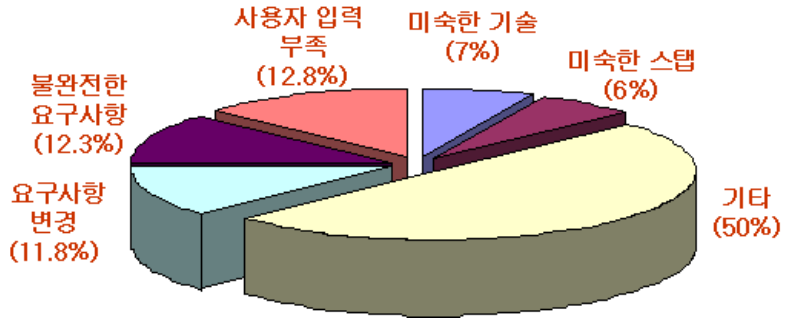
- 비기능적(non-functional) 요구사항
  - ✓ *시스템이 가져야 하는 기능 이외의 요구사항*
  - 시스템이 만족해야 하는 특성과 제약을 규정
    - 시스템이 가져야 하는 신뢰도, 반응 시간, 메모리 요구 등에 대한 요구사항
    - 제약은 입출력 장치의 수용능력, 시스템의 표현 형태 등
  - 일반 애플리케이션의 예
    - 응답시간, 무결성, 품질요구사항, 제약사항, 시스템 특성 및 속성, 보안 등
  - 임베디드 소프트웨어의 예
    - 가격, 신뢰성, 물리적 크기, 전력 소모, 발열, 편리한 사용자 인터페이스, 높은 재생품질, 저전력 프레임워크, 빠른 부팅, 펌웨어 업그레이드 기능, 다양한 디지털 기기와의 연결 지원 등

요구사항 정의의 중요성(1)

- 자연어의 문제점
  - 요구사항의 기술에는 주로 자연어가 사용되며, 이와 더불어 그림과 표가 보충적으로 사용되는 것이 일반적임
  - 자연어 기술의 문제점
    - 모호성 잠재
      - 막연하고 애매한 것들이 많음
        - “적당한”, “충분한”, “실시간”, “융통성이 있는”, “최적”, “99.9% 신뢰가 있어서”, …
      - 같은 단어라도 상황에 따라 다른 의미를 지님
      - 사람에 따라 다른 의미로 사용됨
      - 따라서 동일한 요구사항을 서로 다르게 이해할 위험성
    - 요구사항의 혼동
      - 기능적 요구사항과 비기능적 요구사항이 뒤섞여 기술되는 경우가 많음
    - 요구사항의 융합
      - 여러 가지 요구사항이 한꺼번에 기술되어 설계 및 구현과 연결이 어려운 경우가 발생

요구사항 정의의 중요성[2]

- Bell
  - 소프트웨어 요구에 관한 에러의 30%이상 : 처음부터 잘못된 요구 정의
  - 누락, 불완전한 요구 정의 : 각각 10%
  - 나머지 : 불확실한 정의, 시스템의 대상 범위 이외의 요구
- Standish
  - 프로젝트 실패 원인의 대부분이 요구사항 관련 문제



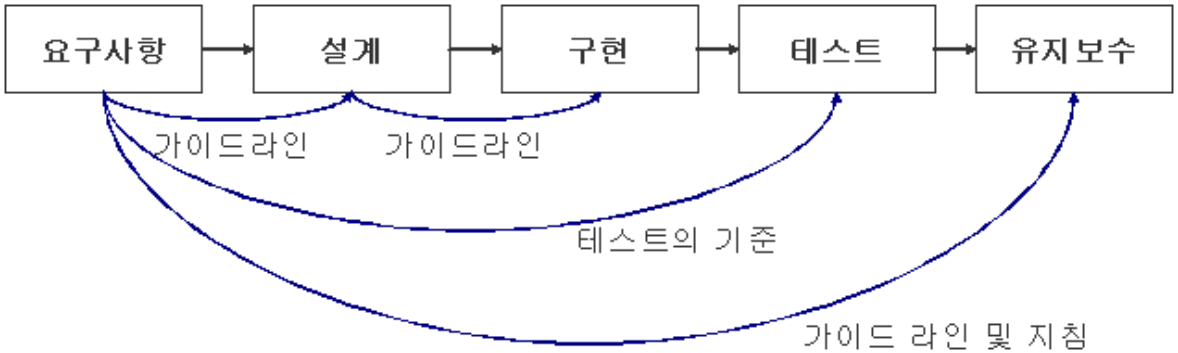
프로젝트 실패 원인	비 율
사용자 입력의 부족	12.8%
불완전한 요구사항	12.3%
요구사항 변경	11.8%

- 요구사항 정의 에러의 비중

- 불명확하거나 애매모호한 요구	10%
- 누락되거나 불완전한 요구	20%
- 부정확한 요구, 테스트 불가능한 요구, 과도한 제약조건을 가진 요구	35%
- 추적 불가능한 요구, 대상범위외의 요구	15%
- 모순된 요구, 호환성이 없는 요구	10%
- 타이핑 에러	10%

요구사항 정의의 중요성[3]

- 소프트웨어 개발의 모든 단계는 요구사항을 만족시키는 방향으로 진행



- 요구사항과 관련된 활동과 이해당사자들이 많음

사람	요구사항 활동
고객/사용자	요구사항 검증
관리	비용 및 일정 결정
시스템 엔지니어	소프트웨어 작업 할당
테스터	테스팅을 위한 기본 지침 제공
소프트웨어 엔지니어	상위 단계 설계를 위한 가이드
형상 관리자	기능적 베이스라인 설정
그 외 모든 관련자	프로젝트 수행을 위한 가이드

요구사항 정의의 중요성(4)

- 요구사항 정의 에러의 원인
  - 이해부족
    - 사용자와 개발자들이 그들이 원하는 것을 잘 모르거나, 정리되어 있지 않음
  - 의사소통의 부족
    - 사용자들의 요구가 개발자들이 이해할 수 있도록 적절하게 표현되지 못하고, 개발자들의 이해가 사용자들에게 적절하게 확인되지 못함
  - 관점의 차이
    - 사용자들이 보는 시스템에 대한 관점과 개발자들이 보는 관점이 다름
  - 시간, 예산 부족
    - 대부분의 소프트웨어 개발 시, 요구사항 분석은 다른 단계에 비하여 덜 중요한 것으로 고려됨
    - 개발하고자 하는 소프트웨어에 대한 충분한 이해 및 상호 합의 없이 다음 단계로 진행

요구사항 정의의 중요성[5]

- 요구 사항 정의의 중요성
  - 요구 분석 : 소프트웨어 개발의 첫 단계
  - 명확하고 분명하게 작성자의 의도가 전달되어야 함
  - 요구 정의단계에서 에러 발생

    ⇒ 나중에 설계를 변경하거나 프로그램을 수정

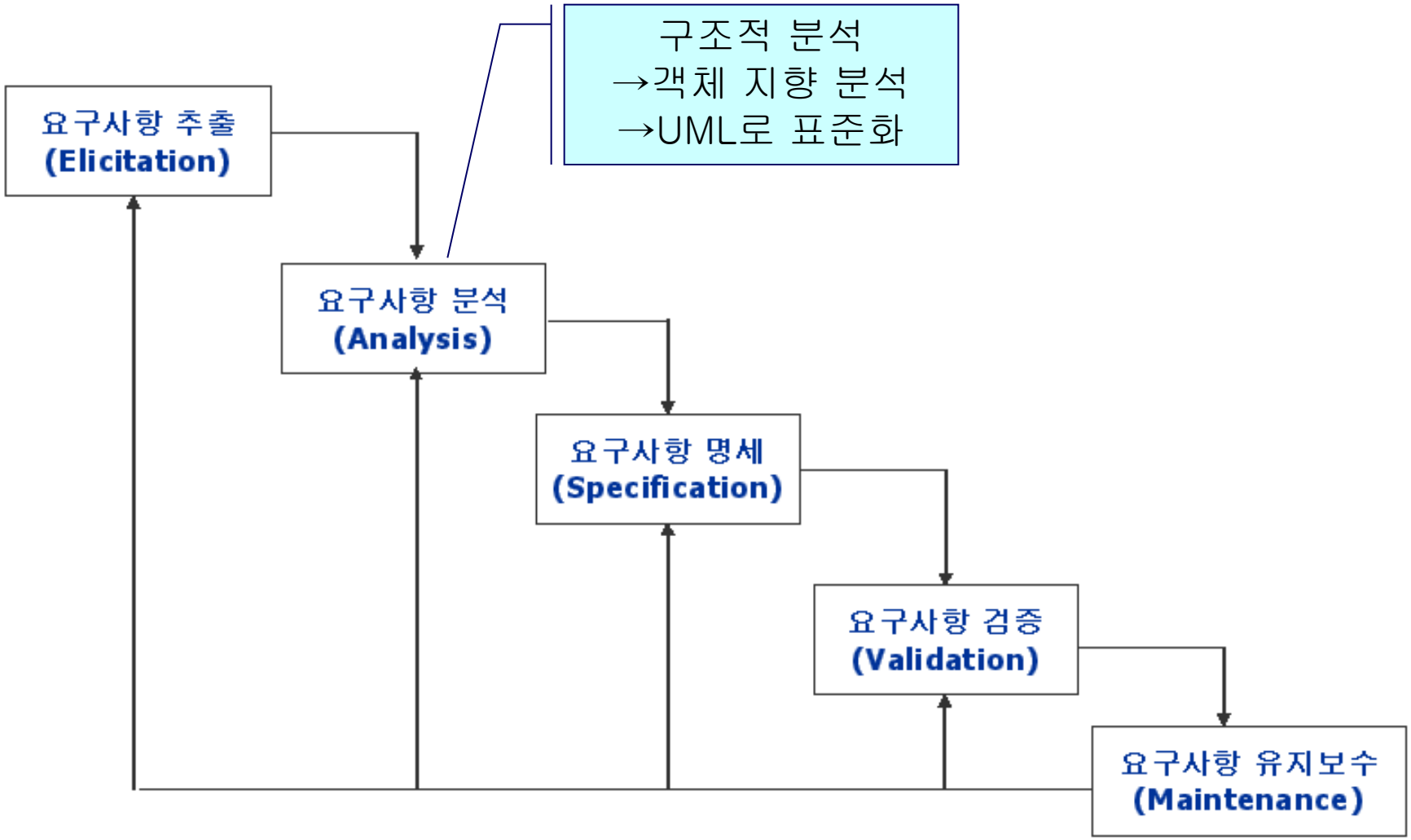
    - 에러 발견 시점이 늦을수록 에러 수정에 필요한 비용 급상승

    ⇒ 최악의 경우,

    - 시스템을 처음부터 다시 개발하는 경우도 발생

    ⇒ 소프트웨어 개발 계획의 지연 또는 비용의 초과

요구사항 관련 프로세스


구조적 분석
→객체 지향 분석
→UML로 표준화
요구사항 추출
(Elicitation)
요구사항 분석
(Analysis)
요구사항 명세
(Specification)
요구사항 검증
(Validation)
요구사항 유지보수
(Maintenance)

요구사항 추출(1)

- 인터뷰(Interview)
  - 1:1의 관계에서 사용자 및 사용자측 의사 결정권자와 시스템에 대한 요구사항을 추출
  - 철저한 사전 준비 작업 필요
  - 인터뷰를 통해 무엇을 얻어야 할지 면밀히 검토

- 질문(Question) – 설문조사
  - 설문지 또는 여론조사 등을 이용해 간접적으로 정보를 수집
  - 개발될 시스템의 사용자가 다수인 경우 의견 수렴에 용이
  - 시스템에 의해 많은 사람들이 영향을 받고, 다양한 기능에 대한 사용자들의 의견이 필요한 경우
  - 사용자의 시스템 요구사항, 기대수준, 현행 시스템의 불만족 사항 및 정도 등에 대한 전체적인 의견 수렴

요구사항 추출[2]

- 브레인스토밍(Brainstorming)
  - 회의의 부정적인 면을 없애고 회의를 즐겁고 말을 꺼내기 쉬운 분위기로 만들어, 회의 참석자들이 내놓은 아이디어들을 비판 없이 수용할 수 있도록 하는 회의
  - 목적: 짧은 시간 내에 많은 양의 아이디어를 획득하는 것
  - 판단과 비판을 유보 → 창의성 극대화
  - 다양한 아이디어를 바탕으로 문제 해결의 실마리를 찾음

- 워크샵(Workshop)
  - 단기간의 집중적인 노력을 통해 다양하고 전문적인 정보를 획득하고 공유하기 위한 방법
  - 프로젝트에 참여하는 모든 핵심 인물의 참여가 필요
  - 성공적인 워크샵을 위해 참석자들은 해당 전문 영역별로 팀 협력이 필요하며 사전 준비가 요구됨

요구사항 추출(3)

- 프로토타이핑(Prototyping)
  - 개발하고자 하는 시스템 또는 그 일부를 신속하고 개략적으로 구축하는 기법
  - 사용자와 설계자의 이해도 증진에 도움이 됨
  - 사용자에게 시스템에 대한 조기 경험을 제공함으로써 비평이 가능하며 시스템이 오류와 약점 파악이 용이

- 스토리보드(Storyboard)
  - 사용자 인터페이스 관점에서 사건의 흐름을 보여주는 것
  - 고객이 원하는 완성된 화면이 아니라, 사건이 어떻게 잘 다루어지는지를 보여줌
  - 사용자는 화면 뒤의 기능에 대해서는 전혀 이해하지 못하고, 이미 시스템이 구축되어 있는 것으로 여김
  - 155page, 운송 주문 스토리보드 예

요구사항 추출(4)

- 의사소통(communication)
  - 발주자와 개발자의 효과적인 의사소통 방안
  - 인터뷰 기술과 프로토타입, 요구 취합 방법(설문지, 유저 그룹, 워크샵 등), 정형적 방법들이 필요
  - 개발자 사이의 의사소통은 man-month와 문서화, 기술회의를 통해서 이루어짐
  - 회의를 통한 회의록은 변경사항에 대한 추적과 프로젝트 인수의 기초 자료로 활용

- 기타, 현재 사용중인 각종 문서의 검토 등 다양한 방법 존재

요구사항 분석

- 요구사항 분석 기법
  - FOA(Function-Oriented analysis)
    - 기능(DFD)과 데이터(ERD)를 분리하여 분석
  - 자료 흐름 지향 분석(Data Flow-Oriented Analysis)
    - 데이터 흐름(DFD)으로부터 소프트웨어 구조를 유도하는 방법
    - 구조적 분석(SA)/구조적 설계(SD)
  - 객체 지향 분석(OOA : Object-Oriented Analysis)
    - 시스템의 기능과 데이터를 함께 분석
    - UML (Unified Modeling Language)로 표준화

요구사항 명세(1)

- 요구사항 명세
  - 분석된 요구사항을 명확, 정확하게 기록하여 명세화하는 과정
  - 명세 방법
    1) <u>자연어에 의한 방법</u>
       - 사용자와 개발자의 이해가 용이
       - 명확성 및 검증에 문제
    2) <u>정형화된 기법을 사용하는 방법</u>
       - 명확성 및 검증이 용이
       - 기법의 이해가 어려움
       - ASM(Abstract State Machines), LOTOS(Language of Temporal Ordering Specification), VDM(Vienna Development Method), Z 언어
  - 요구사항 명세가 갖춰야 할 특성(page 176~179)

요구사항 명세(2)

- 관련 산출물
  - 요구사항 문서

    : *요구사항 결정 단계에서 나온 유형의 산출물*

    - 요구사항 명세서, Usecase 모형 기술서 등으로 구성
    - 산출물을 통해 요구사항 검증 가능
      - 이해당사자들이 자신이 요구하는 내용이 충분히 반영되었는지를 확인
      - 요구공학 엔지니어(비즈니스 분석가, 시스템 분석가)는 이해당사자들의 요구사항이 적절히 초기 시스템 설계에 반영되었는지를 검증
    - 산출물
      - 요구사항, Usecase, Class, Component, Application
    - 산출물 세부 유형
      - 비전 기술서, 요구사항 기술서, 요구사항 명세서, 시스템 휘처 명세서

요구사항 검증(1)

- 요구사항 검증
  - 소프트웨어 요구사항이 사용자들의 요구사항에 맞게 정확하고, 완벽하게, 연계성 있게 명세화 되었는지 검증하는 과정
  - 요구사항들이 고객이 원하는 시스템의 모습을 정확히 반영하고 있는지 확인하는 작업
    - 요구명세서가 내부적으로 일관성을 유지하고 있는지 검사
    - 외부적으로 사용자나 고객이 명세서를 검증
    - 요구사항 문서가 만들어지면 요구사항들이 형식에 맞게 작성되었는지를 검사
  - 검증 방법
    - 리뷰(Reviews), 리허설(Walkthrough), 점검(Inspection) 등
    - 애매모호하지 않고 명확하게 필요한 사항만을 확실하게 구분

요구사항 검증(2)

- 4가지 검증 사항
  - ① 정당성
    - 사용자는 시스템이 어떤 기능을 수행하기 위해 막연히 필요하다고 생각
    - 보다 깊은 생각과 분석은 서로 다른 기능 식별, 추가적인 기능을 찾아냄
    - 사용자들간의 타협안
  - ② 일관성
    - 어떠한 요구라도 다른 요구와 충돌을 일으켜서는 안됨
  - ③ 완전성
    - 모든 기능과 제약 사항을 포함
  - ④ 현실성
    - 현존하는 HW/SW 기술을 사용하여 해결할 수 있는 요구사항인지 예측

요구사항 유지보수

- 요구사항 유지보수
  - 새로운 요구사항의 출현과 변경이 발생했을 경우, 이를 체계적으로 관리하는 활동
  - 요구사항 관리 영역
    - 요구사항 양식의 표준화, 요구사항의 변화 관리, 요구사항 간의 연계성 관리
  - 요구사항 변경 시,
    - 시스템 개발에 관련된 사람들로부터 변경된 요구사항이 어떻게 시스템 전반에 걸쳐 영향을 미치는가를 분석, 관련된 사람들에게 알리고 추적

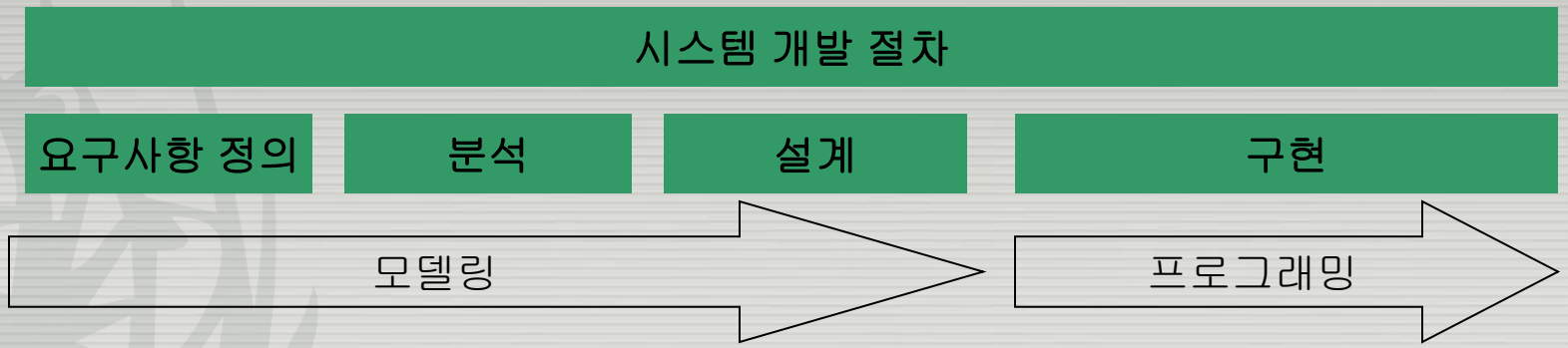
      --> 분석 도구 활용이 필요 : DOOTS, RTM, SACHER

요구사양 명세 도구 소개

- UML
- 구조적 분석 방법

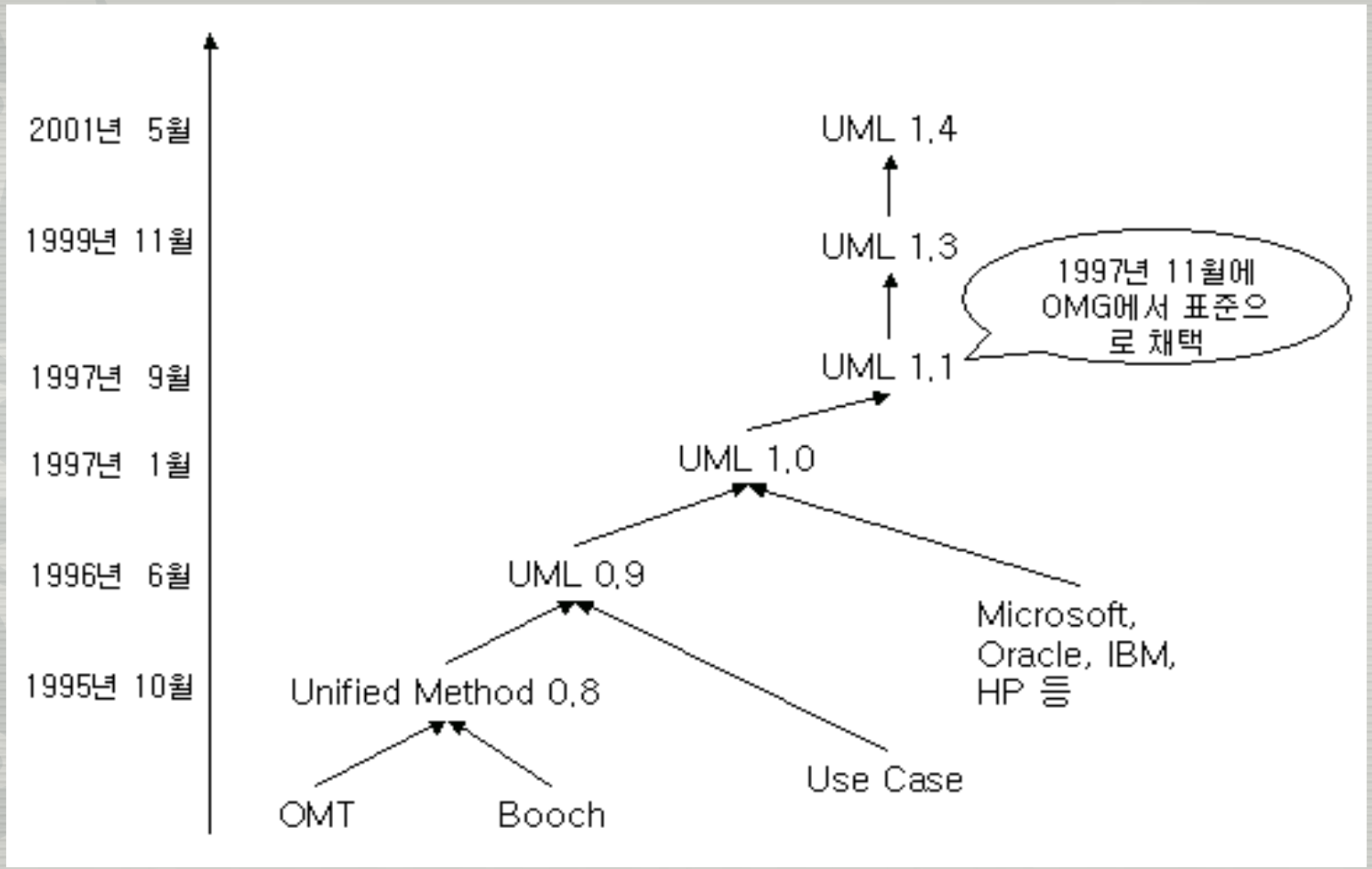
UML 소개

모델, 모델링, 모델링 언어



	모델링	프로그래밍
목적	구축할 시스템에 대한 모델 구성	실제 시스템의 구축
결과물	모델	소스 코드 및 시스템
표현 언어	모델링 언어 예) UML, DFD, ERD	프로그래밍 언어 예) Java, C#
지원 도구	CASE 도구 예) Rose, Together, Star UML	개발 도구 예) Eclipse, VS.NET
지원 도구	CASE 도구	개발 도구

UML의 등장


2001년 5월
1999년 11월
1997년 9월
1997년 1월
1996년 6월
1995년 10월
UML 1.4
UML 1.3
UML 1.1
1997년 11월에 OMG에서 표준으로 채택
UML 1.0
UML 0.9
Unified Method 0.8
OMT
Booch
Use Case
Microsoft, Oracle, IBM, HP 등

UML의 용도

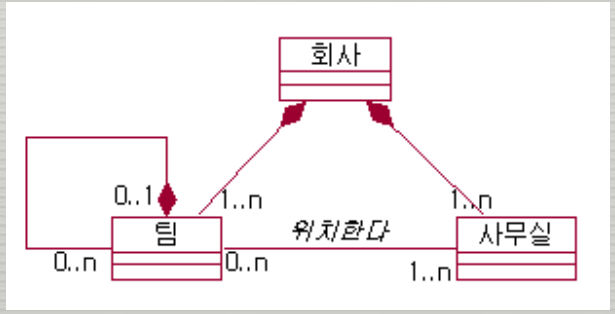
- UML은 다음과 같은 요소에 무관하게 적용된다.
  - 시스템 개발 시 사용되는 개발 방법론에 무관하다.
  - 시스템 개발 시 사용되는 프로그래밍 언어에 무관하다.
  - 시스템 개발 시 사용되는 CASE 도구에 무관하다.

시스템의 유형	설명
정보 시스템	사용자로부터 정보를 입수하고, 이를 가공하여 저장하고 적절한 양식으로 사용자에게 출력하는 작업이 주인 시스템이다. 각종 공공 기관, 금융기관, 회사에서 해당 기관 고유의 업무를 지원하는 시스템들이 이에 해당된다.
기술적 시스템	통신 장비, 군 장비, 공장의 기계 등을 제어하는 시스템이다.
내장 시스템	휴대 전화기, TV 등과 같은 하드웨어에 내장되어 수행되는 시스템이다.

UML의 특징

- UML은 시각적(visual) 언어로서 간결하고 명확한 표현이 가능하다

한 회사에는 여러 팀이 있다. 각 팀에 대해서 해당 팀을 관리하는 하나의 팀이 있으며, 한 팀은 다른 여러 팀을 관리할 수 있다. 한 회사는 여러 사무실을 가지고 있으며, 한 팀은 하나 이상의 사무실에 위치할 수 있고, 반대로 한 사무실에서 여러 팀이 작업할 수 있다.

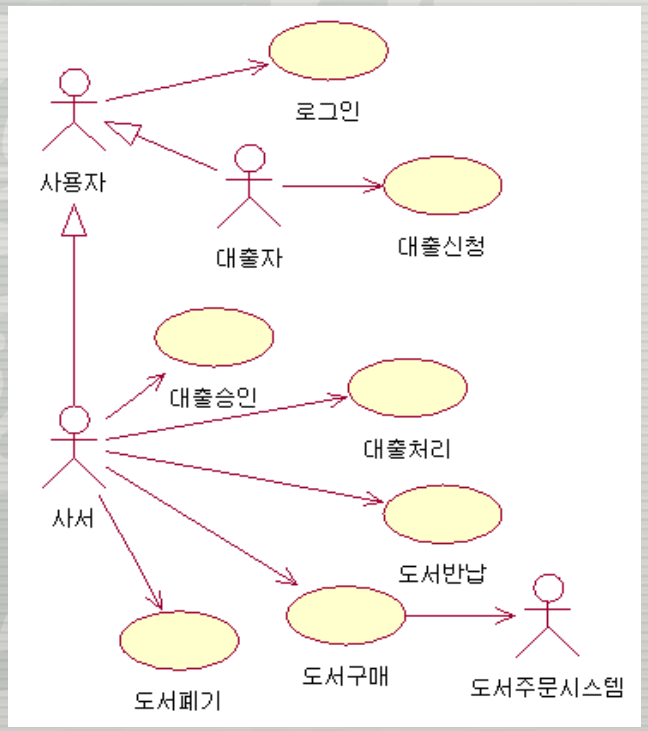


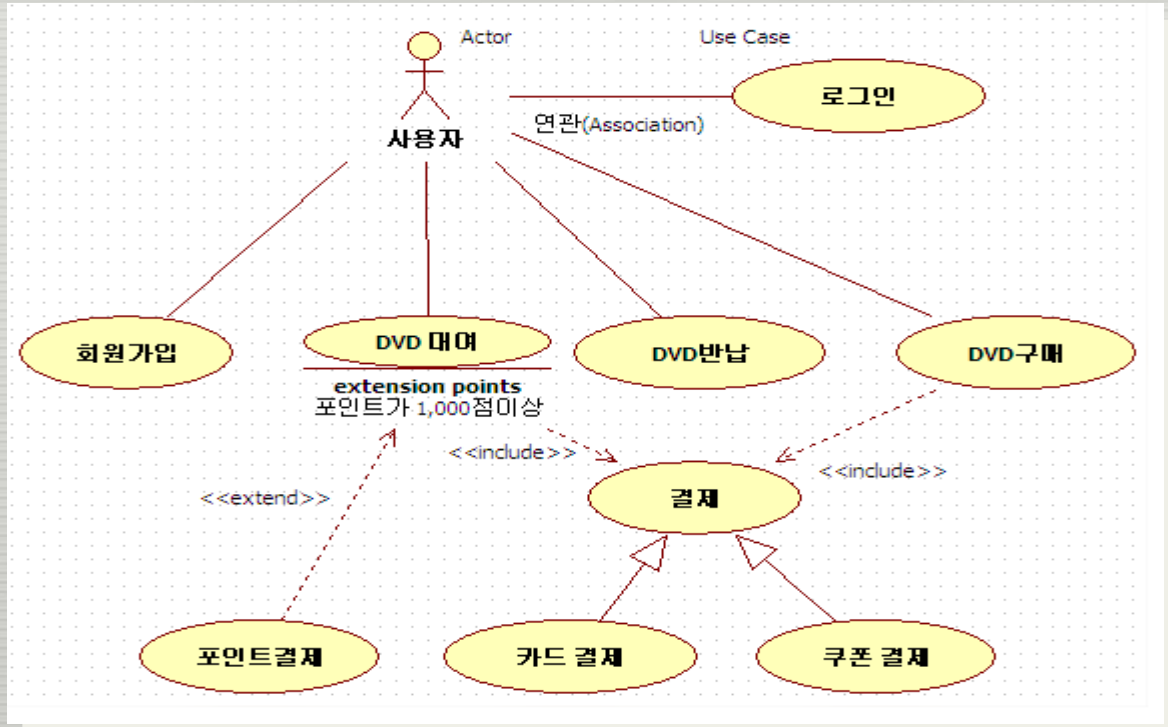
UML의 특징

- UML은 9개의 다이어그램으로 구성된다.
  - 유스케이스 다이어그램
  - 클래스 다이어그램
  - 객체 다이어그램
  - 시퀀스 다이어그램
  - 커뮤니케이션 다이어그램
  - 액티비티 다이어그램
  - 상태차트 다이어그램
  - 컴포넌트 다이어그램
  - 배치 다이어그램
- 시스템 개발 활동의 전 과정에서 사용될 수 있다.
- 시스템의 유형과 규모에 따라서 필요한 다이어그램을 선택하여 사용한다.

유스케이스 다이어그램

- 용도
  - 시스템 개발의 초기 단계에서 시스템에 대한 요구사항을 정의할 때 사용
  - 상호작용을 하는 외부 환경과 시스템이 제공해야 하는 기능을 표현



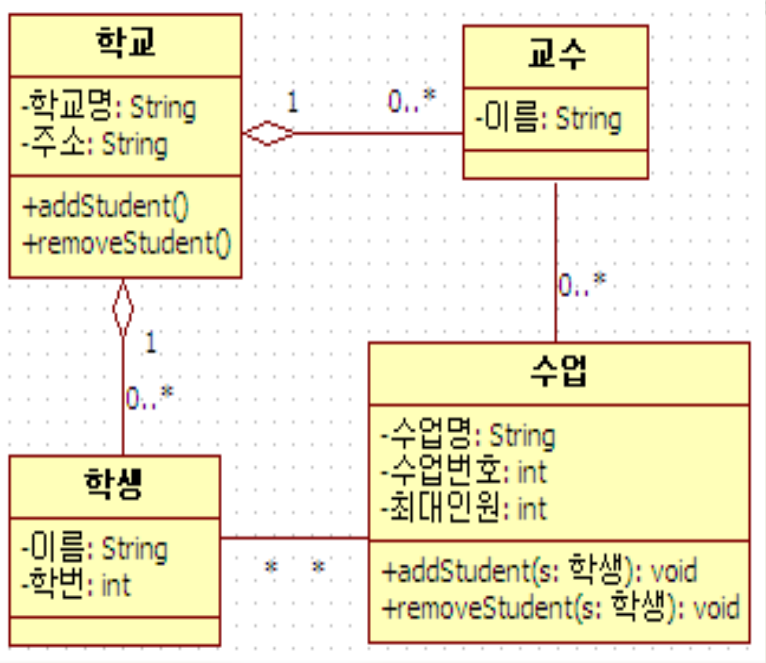


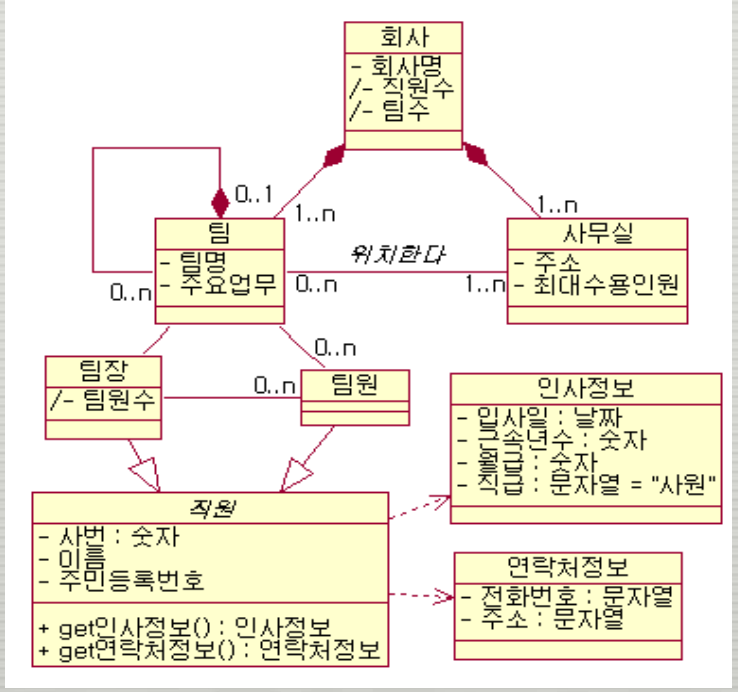
유스케이스 다이어그램

- 액터
  - 개발될 시스템 외부에 존재하는 대상으로서 개발되는 시스템과 상호작용을 하는 모든 것
  - 개발되는 시스템의 기능을 사용하거나, 시스템의 수행 결과를 통보 받는 사용자 또는 시스템의 기능 수행을 위해서 연동이 되는 또 다른 시스템
- 유스케이스
  - 시스템이 제공하는 단위 기능
  - 액터의 관점에서 액터에게 의미가 있는 중요한 결과를 내 보내는 시스템에 의해서 제공되는 기능의 단위
  - 모든 유스케이스를 합하면 시스템이 제공해야 하는 전체 기능이 된다.

클래스 다이어그램

- 용도
  - 분석 및 설계 활동에서 시스템을 구성하는 클래스 및 각 클래스의 연산과 속성 그리고 클래스들 사이의 관계를 표현
  - 객체지향 또는 컴포넌트 기반 시스템 개발 시 가장 중요한 역할





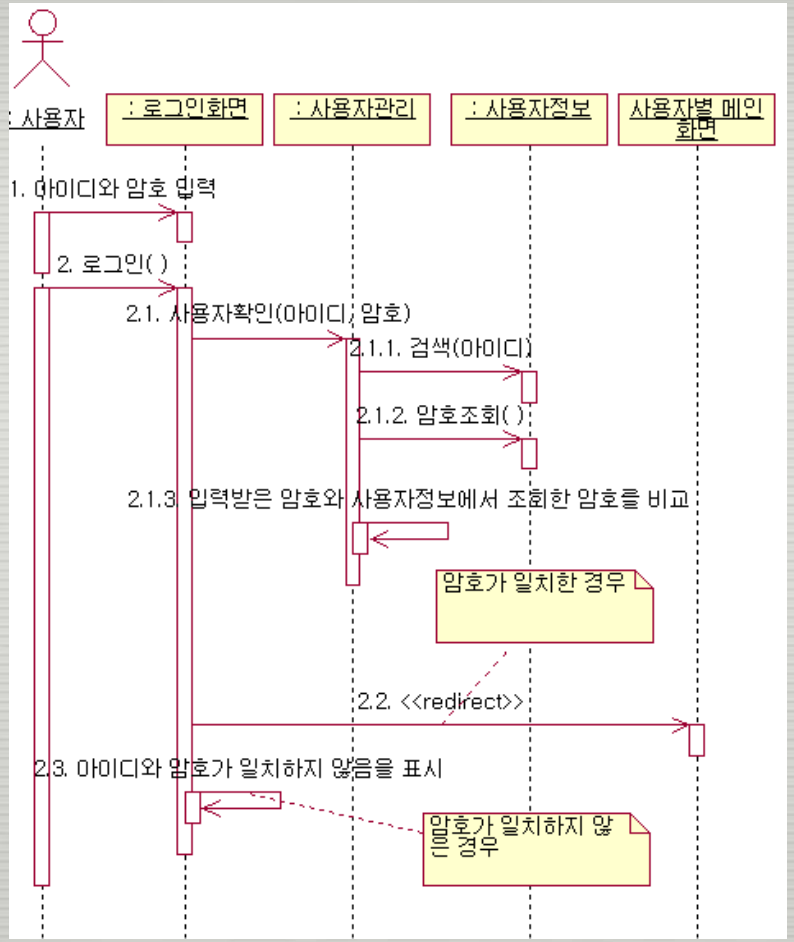
클래스 다이어그램

- 클래스
  - 동일한 유형의 객체들의 일반화
  - 클래스 이름, 속성, 연산을 표현
- 관계

관계의 종류	표기법	설명
연관 (association)		클래스들 사이에 가장 광범위한 관계로서 상대 객체에 대한 인지를 나타낸다.
집합 (aggregation)		연관의 특별한 경우로서, 한 클래스의 객체가 다른 클래스의 객체를 부분으로서 포함하는 것을 나타낸다.
일반화 (generalization)		클래스 간의 계승을 통한 개념의 일반화를 나타낸다.
의존 (dependency)		한 클래스의 변경이 다른 클래스에 영향을 미칠 수 있음을 나타낸다.

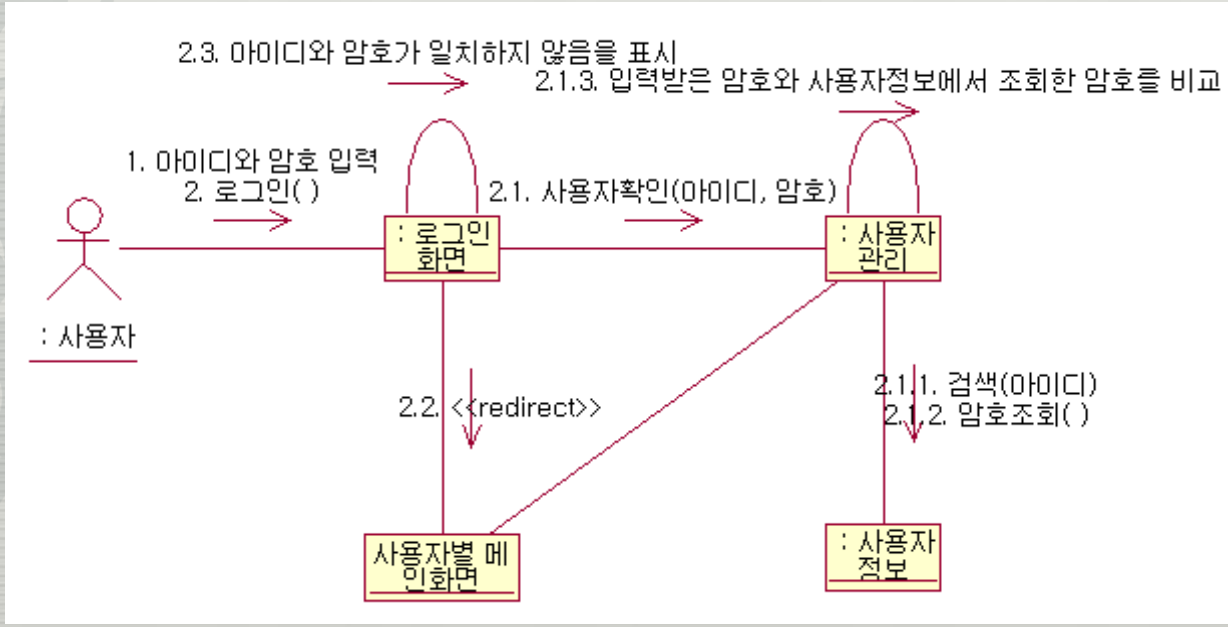
시퀀스 다이어그램

- 용도
  - 분석 및 설계 활동에서 객체들 상의 메시지 상호작용을 표현
- 객체
  - 객체이름: 클래스이름
- 메시지
  - 메시지 이름은 메시지 수신 객체가 수행할 행동을 의미
  - 상위에 배치된 메시지가 하위에 배치된 메시지 보다 우선 전송



커뮤니케이션(협력) 다이어그램

- 용도
  - 분석 및 설계 활동에서 객체들 상의 메시지 상호작용을 표현



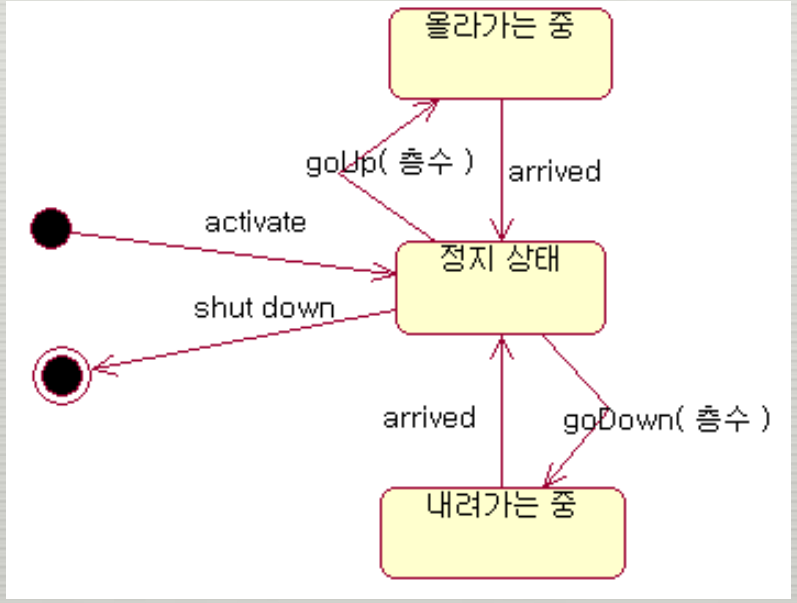
시퀀스 다이어그램과 커뮤니케이션 다이어그램 비교

- 동일한 표현능력을 가진다. 즉, 동일한 상황을 표현할 수 있다.
- 상호작용(interaction) 다이어그램: 시퀀스 다이어그램 및 협력 다이어그램
- 두 다이어그램 사이의 자동적인 변환이 가능

	시퀀스 다이어그램	협력 다이어그램
객체 간의 관계 표시	표시 안됨	명시적으로 링크가 표시됨
메시지 전송의 순서	메시지의 상하 위치와 동일	각 메시지에 부여된 순번을 통해서 구분
주요 용도	객체들 사이의 상호작용을 시간적인 면에서 보다 쉽게 파악하고 싶은 경우	시간적인 면보다는 객체들 사이의 관계를 보다 쉽게 파악하고 싶은 경우

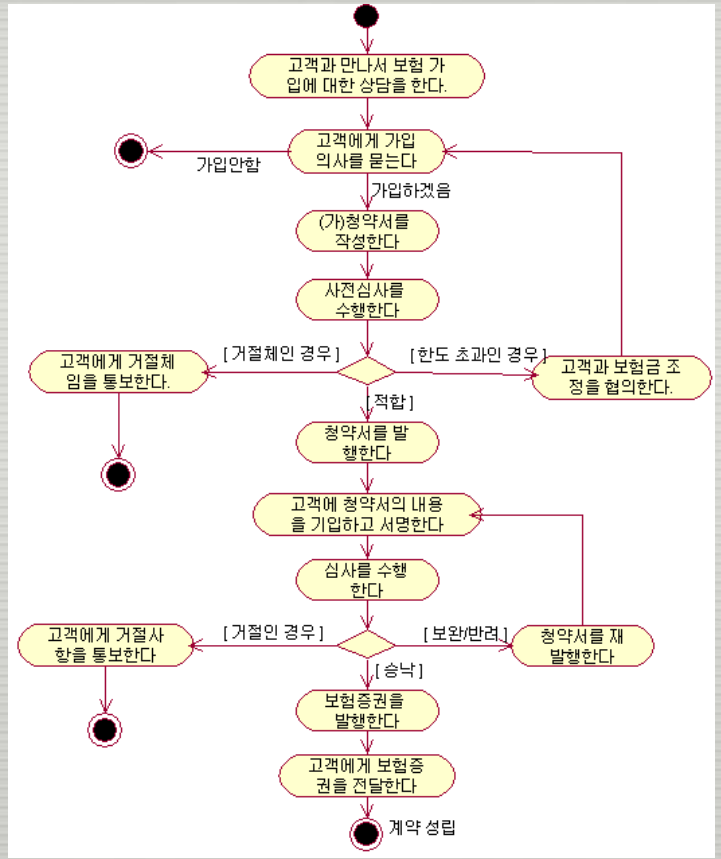
상태차트 다이어그램

- 용도
  - 반응적인(reactive) 객체 및 시스템의 내부의 복잡한 행동을 표현
  - 정보 시스템 보다는 기술적 시스템 또는 내장 시스템에서 주로 이용
- 상태
- 전이와 이벤트



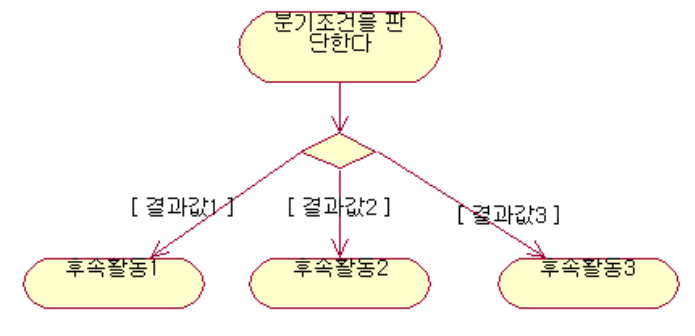
액티비티 다이어그램

- 용도
  - 업무 흐름을 표현
  - 화면 흐름을 표현

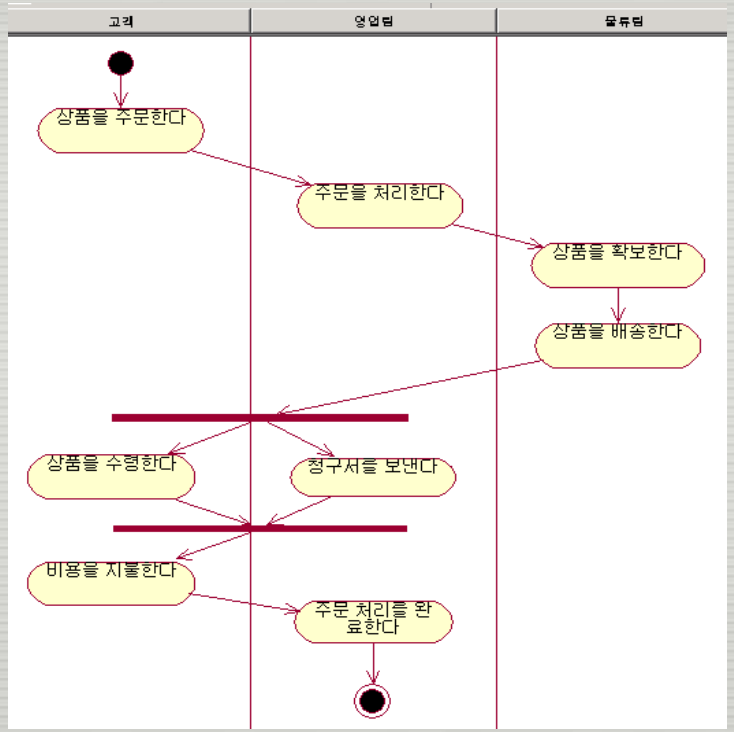


액티비티 다이어그램

- 분기: 조건에 따른 다른 활동의 수행

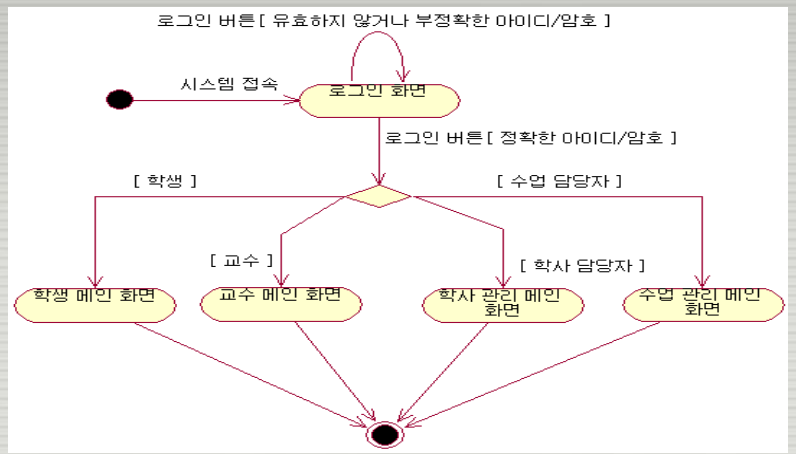


- 구획면: 활동의 수행 주체 표현
- 동기화 막대: 병행적으로 수행되는 활동의 표현

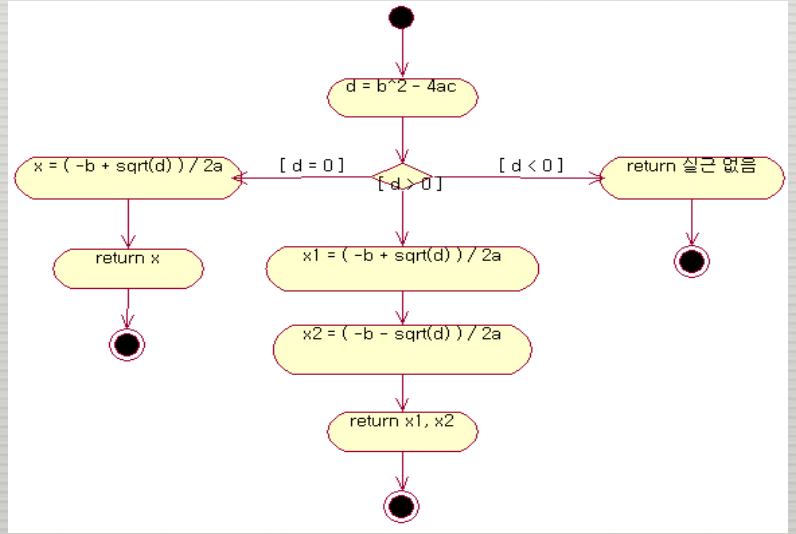


액티비티 다이어그램의 용도

- 업무 흐름의 표현
- 화면 흐름의 표현

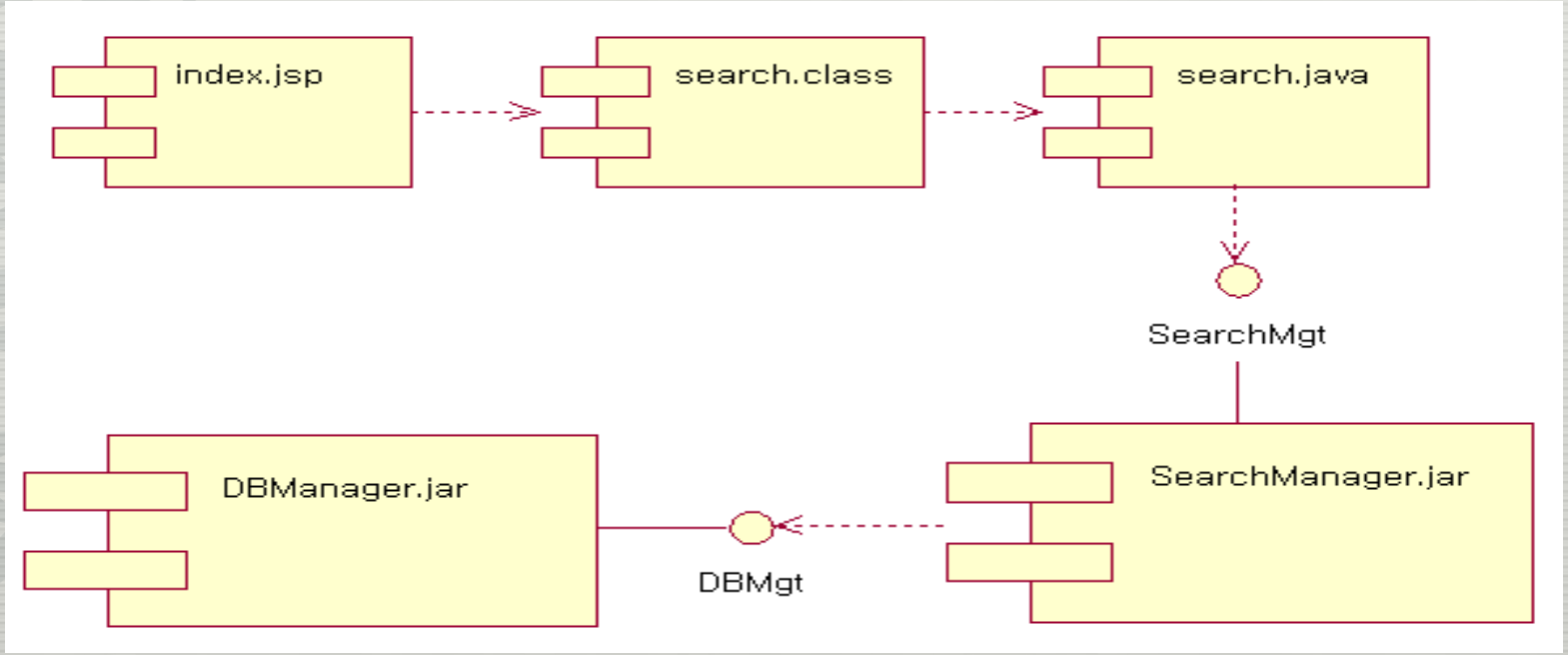


- 알고리즘의 표현



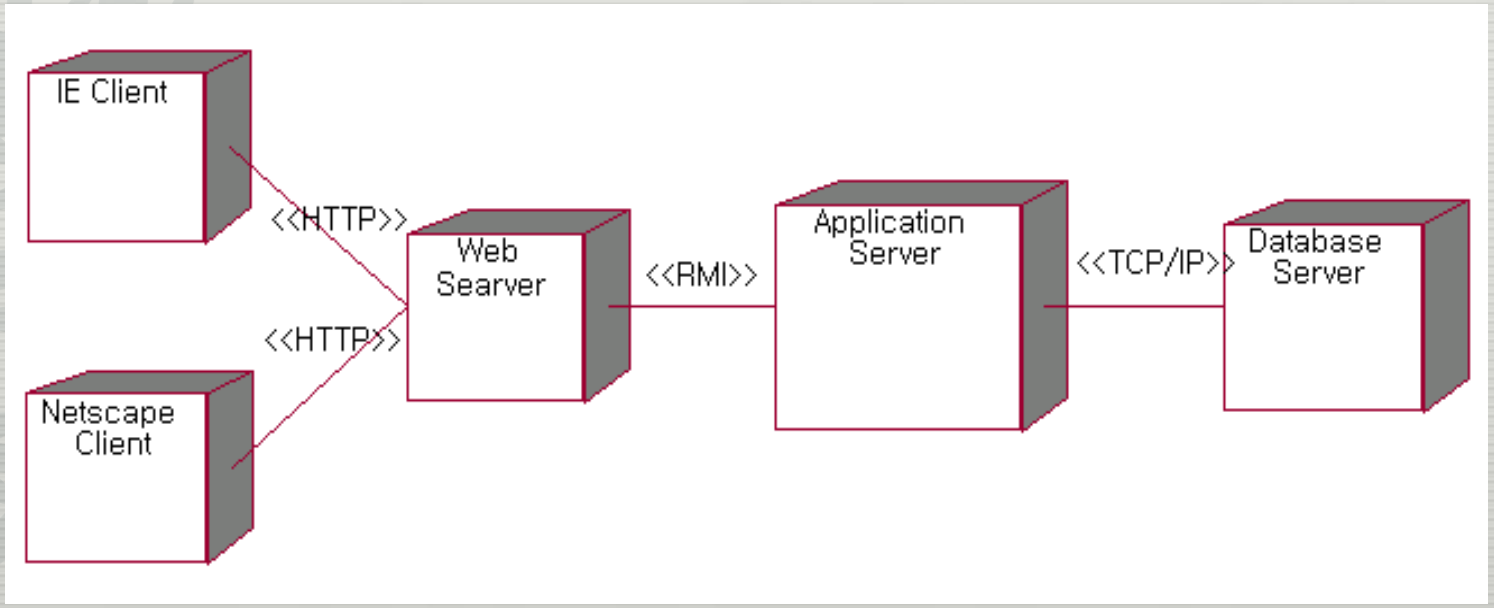
컴포넌트 다이어그램

- 용도
  - 시스템을 구성하는 물리적은 요소(즉, 컴포넌트)와 그들 간의 의존관계를 표현



배치 다이어그램

- 용도
  - 시스템을 구성하는 처리 장치 즉 컴퓨터와 그들 사이의 통신 방법 표현



4+1 관점

- 복잡한 시스템을 5개의 다른 측면에서 바라본다.
  - 각 관점 별로 모델링 한 결과는 1개 이상의 UML 다이어그램으로 표현
  - 모든 관점을 통합적으로 분석해야 시스템 전체의 모습을 알 수 있다.

관점	설명
유스케이스 관점	사용자가 인식할 수 있는 시스템이 제공할 기능 파악에 초점을 둔다.
설계 관점	요구된 기능을 제공하기 위한 시스템의 내부 구조 및 내부 기능에 초점을 둔다.
프로세스 관점	시스템을 구성하는 프로세스/스레드 및 그들 간의 동기화 및 통신 관계에 초점을 둔다.
구현 관점	시스템을 구성하는 물리적인 컴포넌트 및 관계에 초점을 둔다.
배치 관점	시스템을 구성하는 물리적인 처리 장치와 각 처리 장치에 배치되는 컴포넌트에 초점을 둔다.

유스케이스 관점

- 시스템 외부에 있으면서 시스템과 상호작용을 하는 대상과 시스템이 제공할 기능이 무엇인지에 초점
  - 시스템과 상호작용하는 대상 => 액터
  - 시스템이 제공할 기능 => 유스케이스
- 유스케이스 관점은 다른 4개의 관점을 유도하는 중심적인 역할
  - 유스케이스 관점 측면에서의 결과에 따라서 다른 관점의 내용이 결정
- 유스케이스 관점은 개발자 뿐만 아니라, 고객/사용자에게도 중요
  - 개발자: 개발할 시스템의 요구사항에 대한 이해
  - 고객/사용자: 자신들이 원하는 시스템 인지에 대한 확인
- 유스케이스 관점은 시스템을 하나의 black box로 본다.

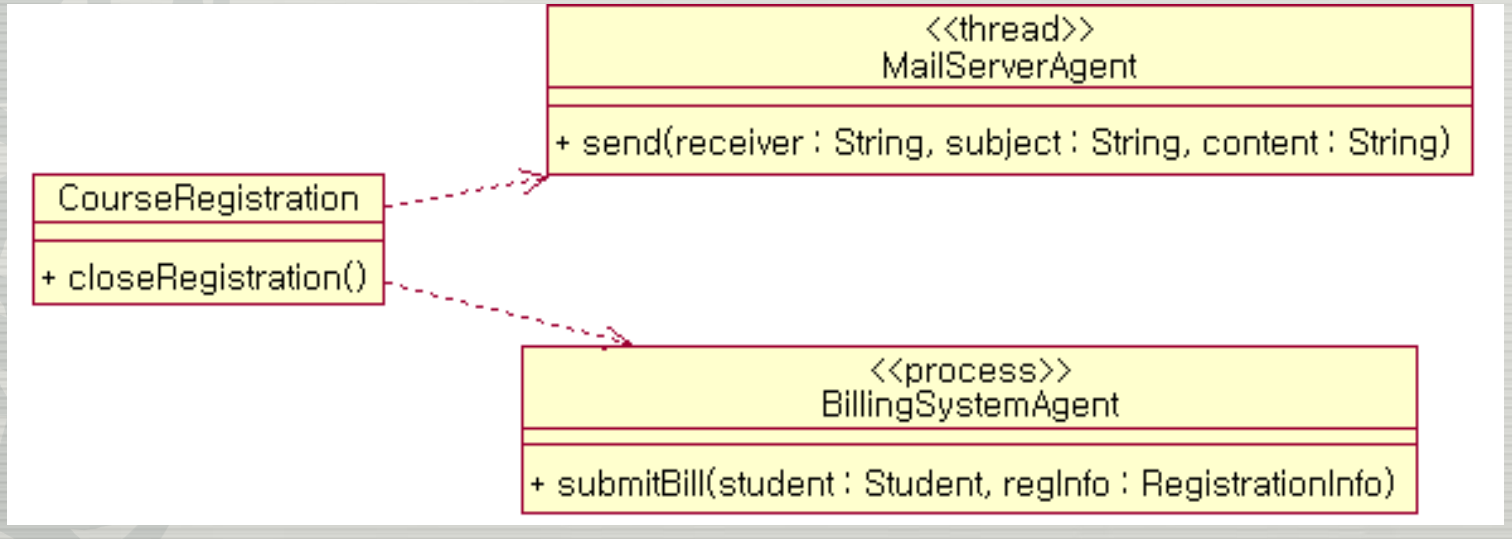
설계 관점

- 유스케이스 관점에서 정의된 기능을 시스템이 제공하기 위해서는 시스템 내부에 어떤 클래스/컴포넌트가 필요하고 이들 클래스/컴포넌트들이 서로를 어떻게 이용/호출하는 지에 초점

구분	관심 사항	사용되는 다이어그램
정적인 측면	클래스 및 클래스 간의 관계	클래스 다이어그램
동적인 측면	클래스 내의 동작	상태차트 다이어그램
	클래스 간의 상호 작용	시퀀스 다이어그램 커뮤니케이션 다이어그램
	클래스 연산의 동작	액티비티 다이어그램

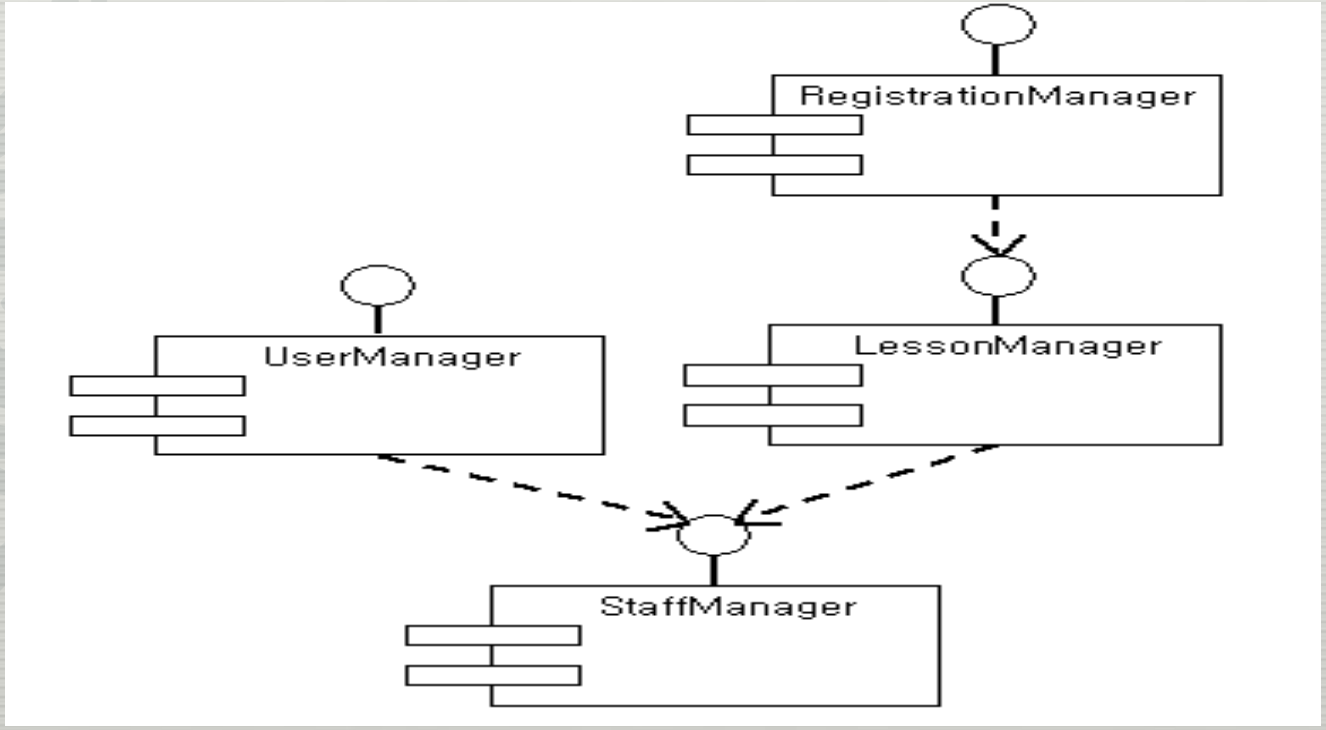
프로세스 관점

- 병행성을 가진 클래스 즉 활성(active) 클래스 및 그들 사이의 동기화와 통신에 초점
- 설계 관점에서 사용된 동일한 UML 다이어그램을 사용



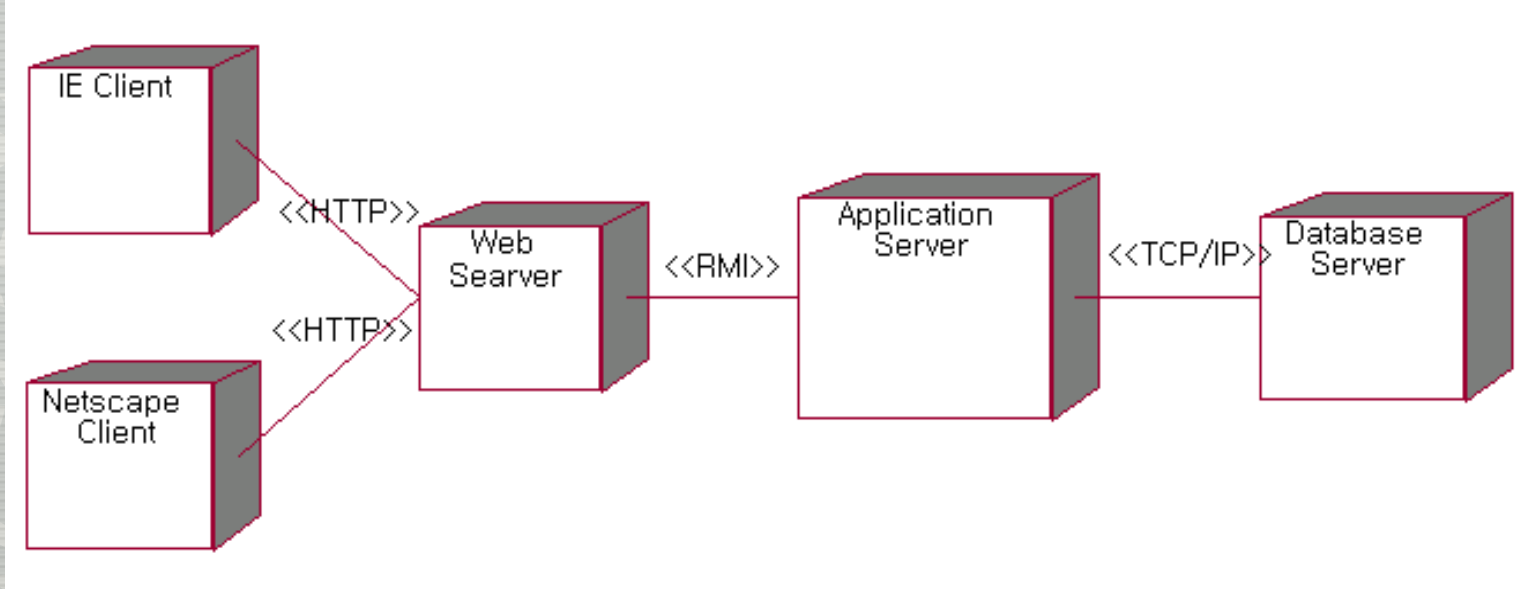
구현 관점

- 시스템에서 물리적으로 구현된 요소 즉 컴포넌트(각종 파일)들과 그들 간의 관계에 초점을 둔다.
- 컴포넌트 다이어그램 사용

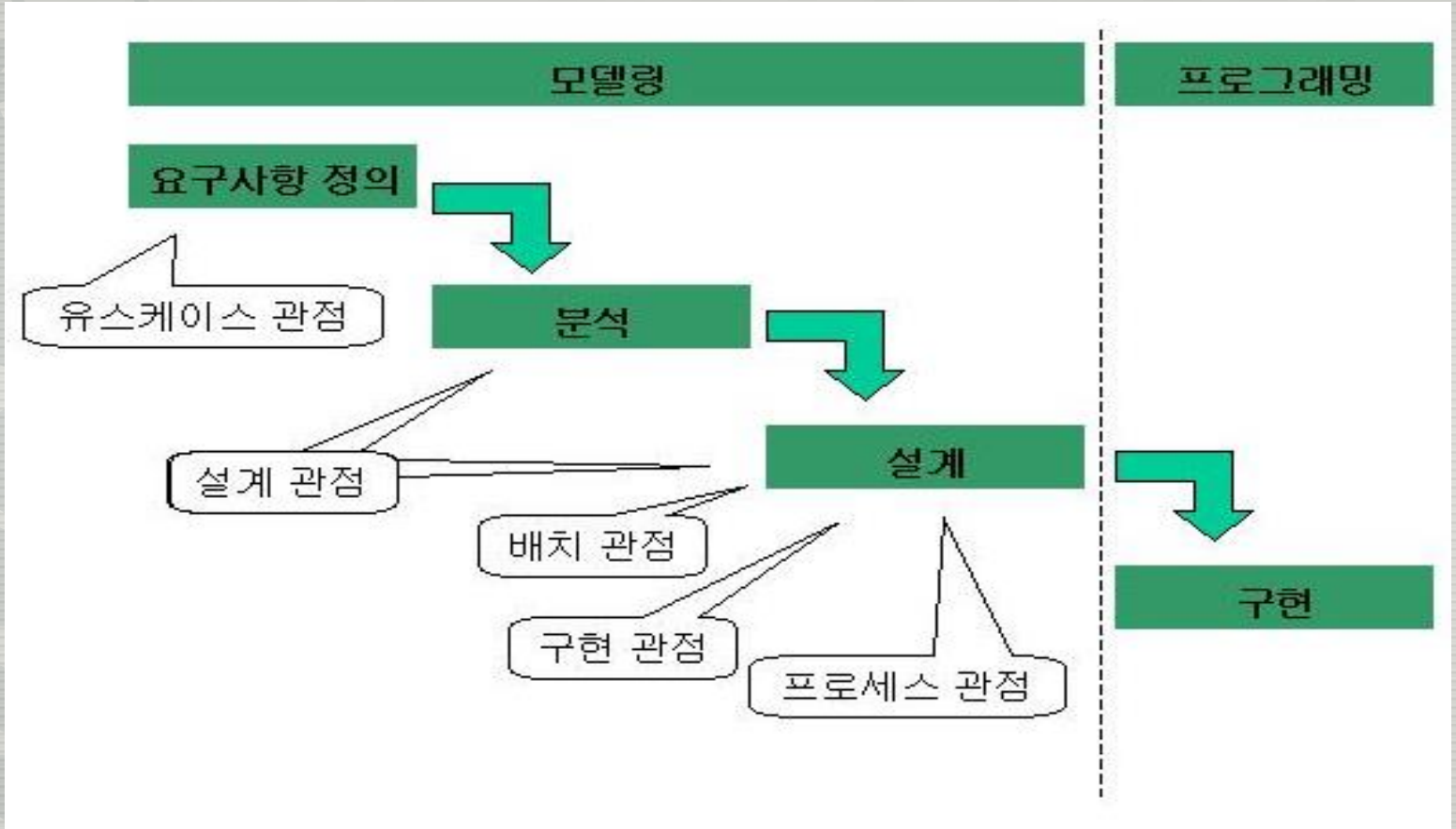


배치 관점

- 시스템을 구성하는 처리장치 즉 컴퓨터와 그 컴퓨터 간의 통신 방법에 초점
- 배치 다이어그램 사용



개발 활동과 관점



시스템 유형 별 UML 다이어그램의 사용

	관점	다이어그램	1	2	3
논리적	유스케이스 관점	유스케이스	√	√	√
	설계 관점	클래스	√	√	√
		상호작용	√	√	√
		상태차트		√	√
	프로세스 관점	클래스			√
		상호작용			√
		상태차트			√
물리적	구현 관점	컴포넌트			√
	배치 관점	배치			√

1. 간단한 애플리케이션
2. 반응적인 시스템
3. 복잡한 분산 시스템

구조적 분석(1)

- 구조적 분석기법
  - 사용자의 요구 사항을 파악하기 위하여 자료의 흐름과 가공 절차를 그림 중심으로 표현하는 방법
    - 처리중심(process-oriented) 분석 기법
  - 세부 작업 순서
    - 배경도 작성
    - 상위/하위 자료 흐름도(Data Flow Diagram : DFD) 작성
    - 자료 사전(Data Dictionary : DD) 작성
    - 미니 명세서(Mini-Specification) 작성
  - 특징
    - 그림 중심의 표현
    - 하향식(top-down)원리를 적용
    - 사용자의 업무 요구 사항을 쉽게 문서화
    - 사용자와 분석가간의 의사소통을 위한 공용어
    - 실체 모형(추상적 표현)을 추출

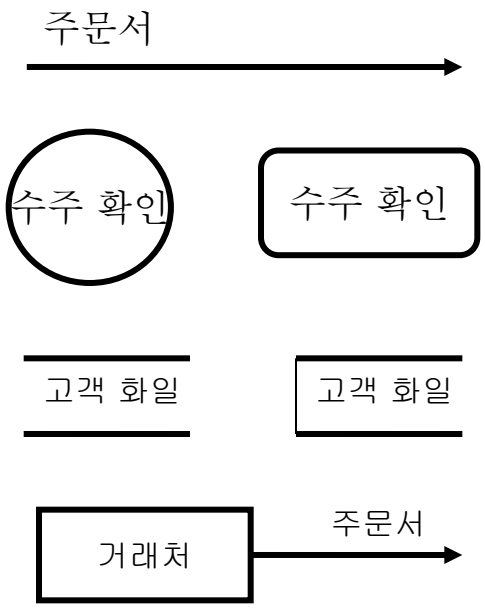
- ✓ 매우 간결
- ✓ 이해하기 쉬움
- ✓ 검증이 가능
- ✓ 체계적

구조적 분석(2)

- 구성 요소
  - 자료 흐름도(Data Flow Diagram : DFD)
    - 자료가 소프트웨어 내의 절차에 따라 흐르면서 변화되는 과정을 도형화한 기법
    - 하나의 process가 진행될 때마다 새로운 레벨을 형성하여 자세하게 기술됨
  - 자료 사전(Data Dictionary : DD)
    - 메타데이터, 즉 데이터에 대한 데이터를 모아 놓는 저장소이며, 데이터 항목, 데이터 흐름에 대한 정의가 포함되어 있음
    - 자료 흐름도에 나타난 데이터에 대한 정보를 한 곳에 모아 놓음으로써 개발자나 사용자들이 편리하게 사용할 수 있도록 해줌
  - 미니 명세서(Mini-Spec.)
    - 자료 흐름도의 프로세스에 대한 자세한 작업 과정을 기술

자료 흐름도(1)

- 자료흐름 그래프, 버블 차트(Bubble chart)
- 정보의 흐름을 나타내는 그래프
- 구성요소
  - 자료흐름(Data Flow) : 화살표(→)
    - 정보의 흐름을 표시하는 자료항목 또는 데이터 단위이며 화살표는 데이터의 흐름을 표시
  - 처리(Process) 공정 : 원(○), 둥근 사각형
    - 시스템에서 정보를 처리하고 변환시키는 변환기이며 버블이라고도 부름
  - 자료 저장소(Data Store) : 직선(단선, 이중선)
    - 오랫동안 보관되는 데이터를 저장해 놓는 파일이나 데이터베이스 시스템을 의미
  - 단말(Terminator) : 정사각형, 직사각형(□)
    - 자료 흐름의 발생지(source)와 최종 출력의 종착지(sink)를 표시

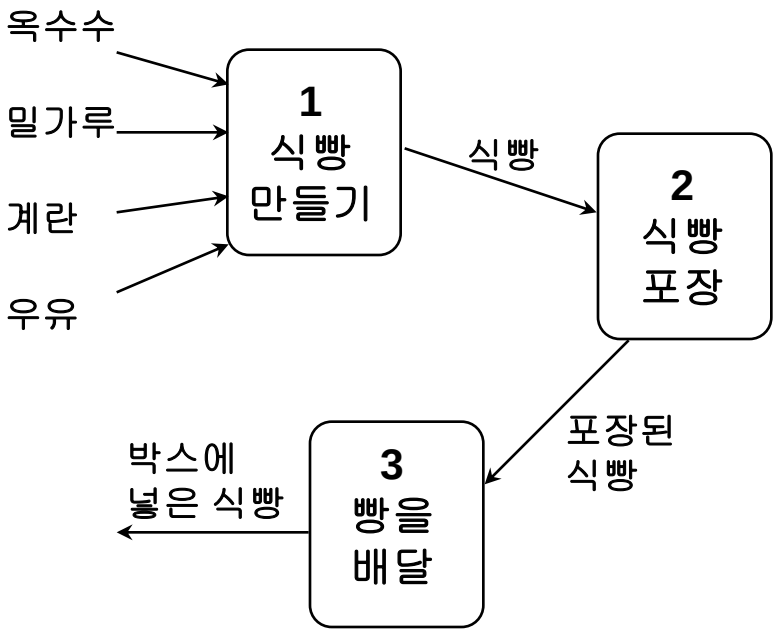


자료 흐름도(2)

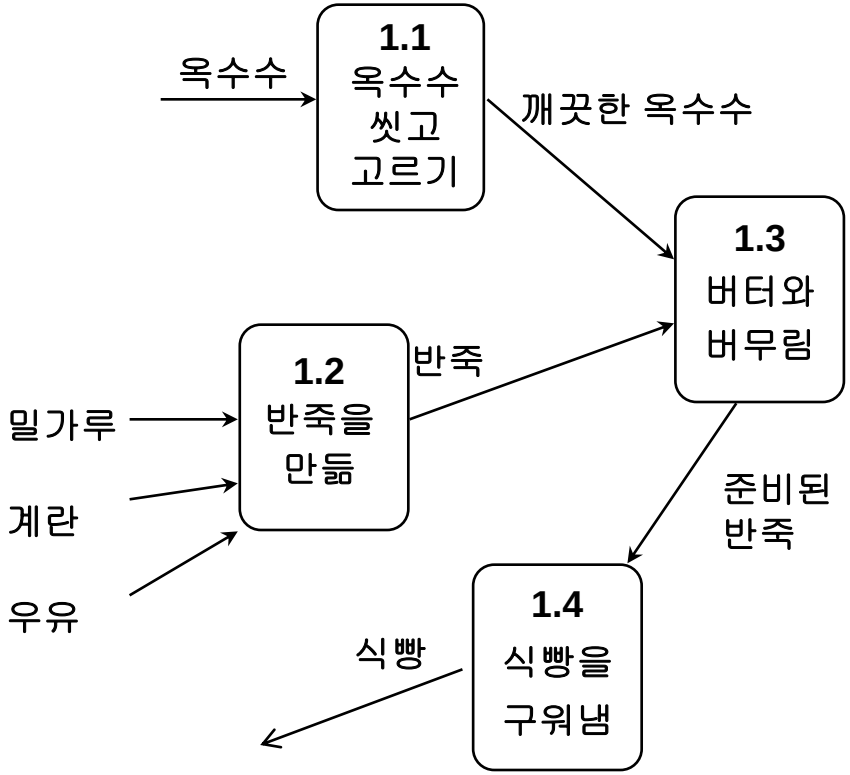
- 단계적 분할에 의하여 단계적으로 표현
  - 배경도(context diagram) 작성
    - 개발하려는 시스템과 외부세계와의 인터페이스를 식별
    - 시스템 분석의 범위를 설정
    - 시스템 전체를 나타내는 하나의 처리와 관련된 단말들로 표시
  - 중간 단계의 자료흐름도 작성
    - 자료흐름도 내의 하나 이상의 처리가 하위 자료흐름도로 분할되는 자료흐름도
  - 최하위 단계의 자료흐름도 작성
    - 자료흐름도 내의 모든 처리가 더 이상 분할되지 않는 자료흐름도
    - 모든 처리들이 소단위 명세서로 설명됨

자료 흐름도-예제

- 베이커리 시스템의 1차 자료흐름도

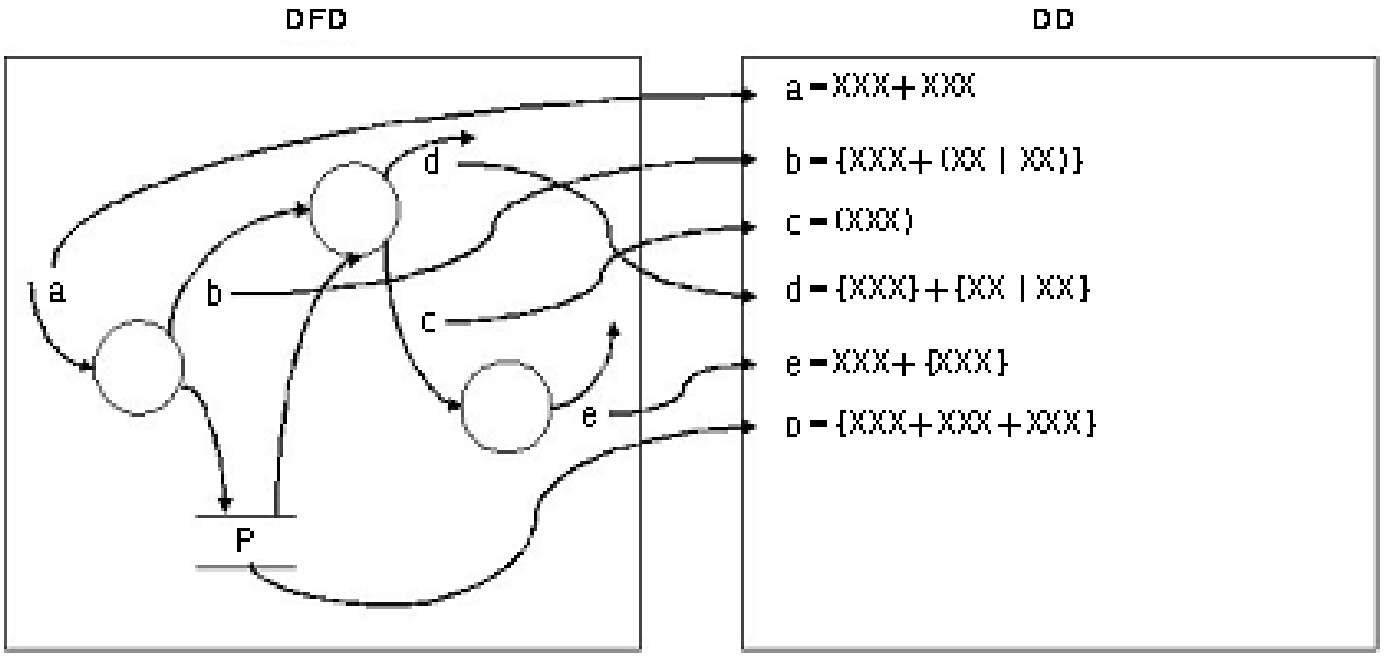


- 1번 처리(식빵 만들기)의 분할



자료 사전(1)

- 자료 흐름도 상의 자료 요소를 수학적으로 정의
- 자료를 계층적으로 단순화시켜 가면서 정의



자료 사전(2)

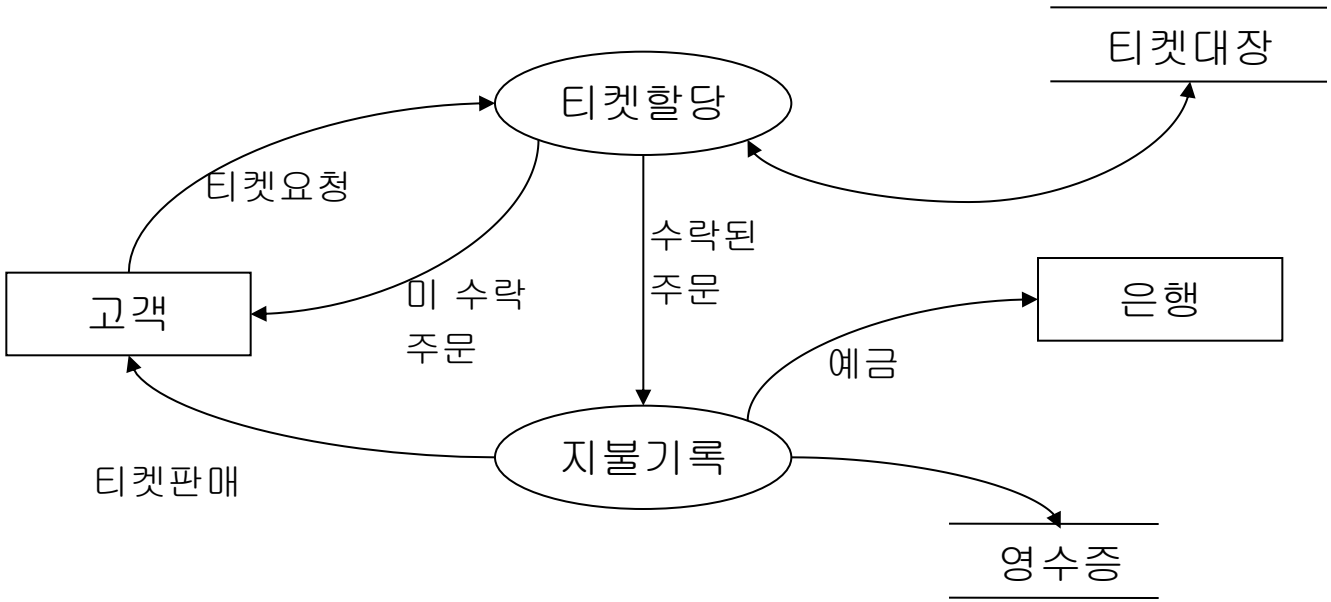
- 자료사전에 사용되는 기호

기호	예	의미
=	a=b+c	a는 b+c로 구성된다
+	a+b	AND : a와 b의 합성
{ }	3{a}5	a를 3번에서 5번 사이 반복
[]	[a|b]	OR : a와 b중 하나 선택
( )	(a)	A는 선택적 (0{a}1과 동일)
**		주석문

- 예) 전화번호에 대한 자료 사전
  - 전화번호 = [지역번호] + 국번 + '-' + 가입자_번호
  - 지역번호 = '0' + 첫자리 + 10{십진수}
  - 국번 = 3{십진수} 4
  - 가입자_번호 = 4{십진수} 4
  - 첫자리 = 2|3|4|5|6

자료 사전(3)

- 티켓 발행 DFD와 자료사전의 관계

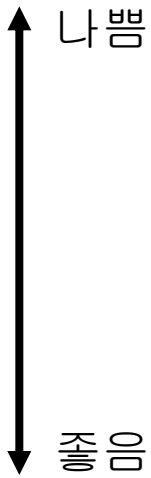


- 수락된 주문 = 이름 + 주소 + 지불 + 실제날짜 + 실제 수량 + 1{티켓}
- 티켓 = 1{날짜 + 가격 + 가능 수량 + {티켓}}
- 티켓 요청 = 이름 + 주소 + 지불 + 1{우선 티켓}
- 미 수락 주문 = 이름 + 주소 + 지불 + 미 수락 편지

미니 명세서(1)

- 자료 흐름도에 있는 한 개의 처리 공정을 대상으로 작업
- 자료 흐름도의 최하위 처리가 어떤 기능을 하는가를 기술한 것

- 미니 명세서의 작성 툴
  - 서술 문장
  - 구조적 언어
  - 의사 결정트리
  - 의사 결정표
  - 표
  - 그래프



미니 명세서(2)

- 의사 결정표(decision table)
  - 여러 가지 다른 조건에 대하여 다른 처리를 해야 할 경우

조 건	규 칙			
	1	2	3	4
1) 송장금액 $ 500	Y	N	Y	N
2) 송장 60일 초과	Y	Y	N	N

출 력	결 과			
1) 확인서 발행	N	Y	Y	Y
2) 송장 발행	N	Y	Y	Y
3) 신용처리 보고서 기입	N	N	Y	N

미니 명세서(3)

- 구조적 언어(Structured Language)
  - 영어에서 쓰이는 단어 중 연산이나 제어구조를 표현하는데 쓰이는 단어(if then else, case, repeat, until, while 등)를 제한해서 사용
  - 예)

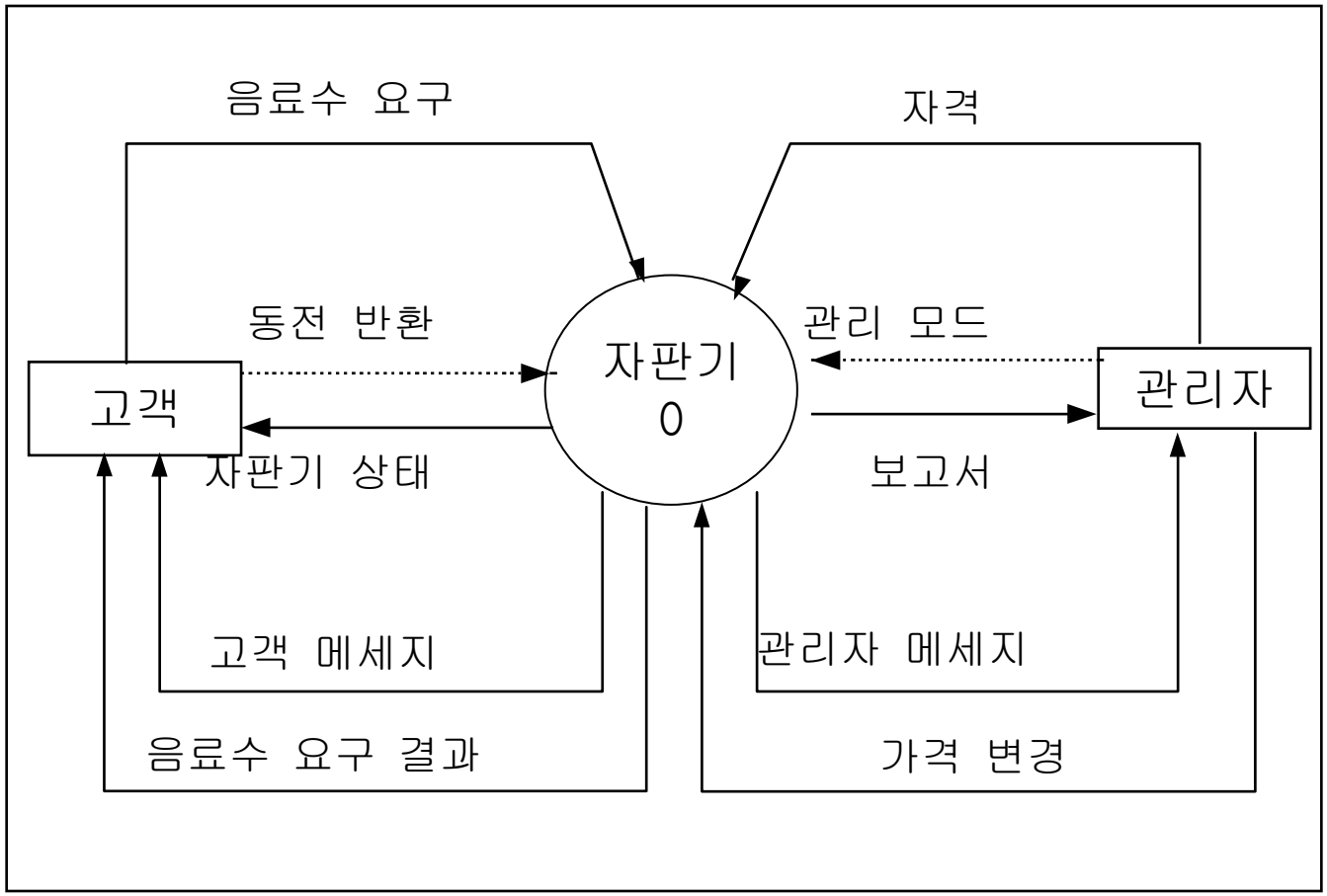
```
IF 청구액 > 50만원
        IF 납입지체일 > 60일
                THEN 사고해결부서에 통고
        ELSE (신용도가 이직은 좋음)재청구서 발송
ELSE
        IF 납입지체일 > 60일
                THEN 재청구서 발송
                     신용평가서에 기록
        ELSE 재청구서 발송
```

미니 명세서(4)

- Mini명세서의 작성기준
  - 반 페이지내지 한 페이지 정도의 분량이 적당

    ⇒미니 명세서의 분량을 염두에 두고 DFD 분할
  - 다음 조건일 때 상기 기준에 얽매이지 않고 DFD를 분할
    - 시간적으로 동일하지 않은 두 개의 기능을 수행할 때
    - 명명이 용이하지 않을 때
    - 분량이 적어져서 사용자의 시스템 대한 이해가 용이할 때

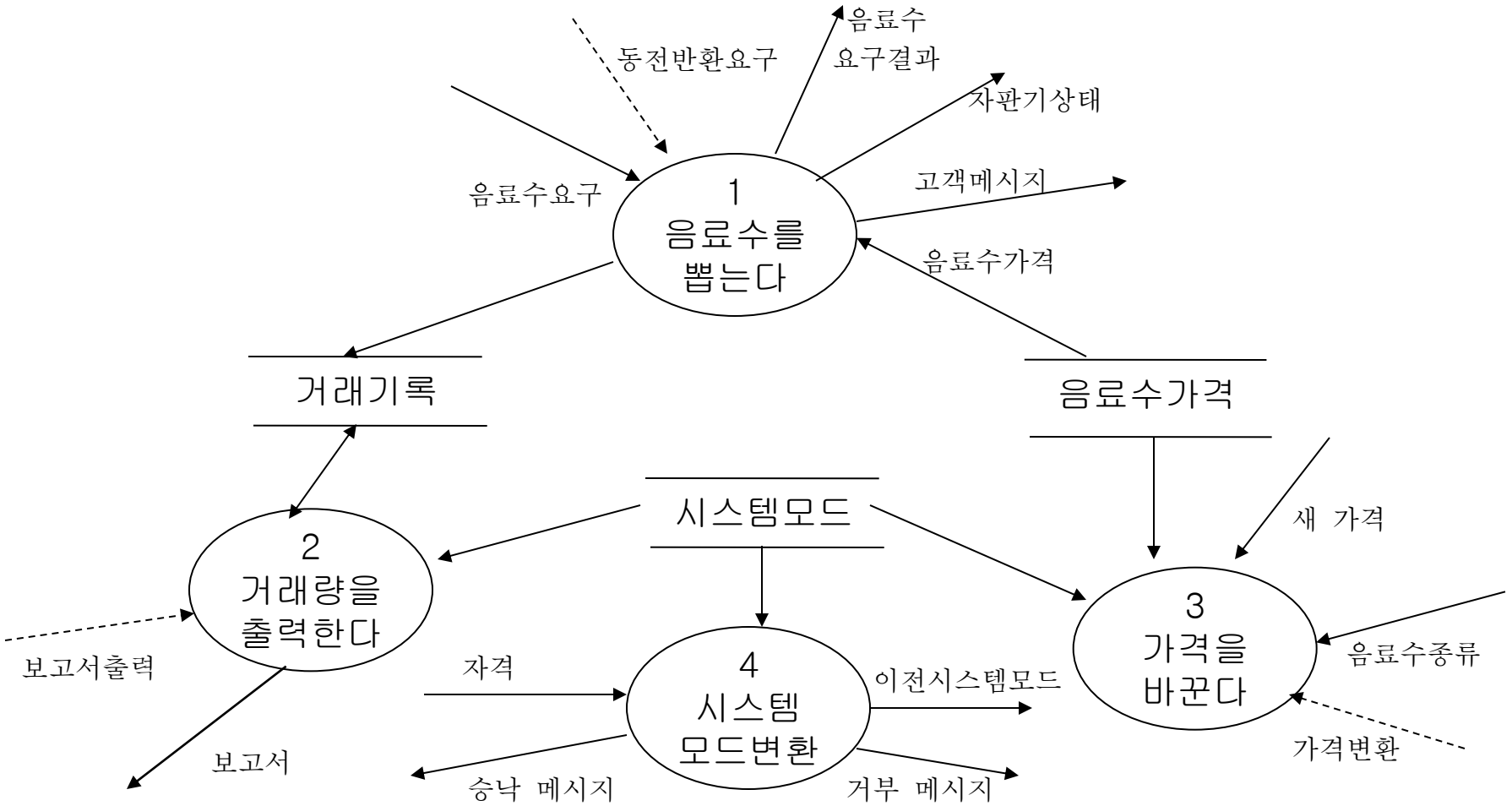
예제-자판기 시스템의 배경도



예제-자판기의 정보 흐름도

- 하나의 정보흐름이 여러 정보흐름의 결합으로 되어 있는 경우 다음과 같이 자료사전에 기록
  - 고객 메시지 = 금액 부족 메시지 + 음료수 없음 메시지 + 잔돈 없음 메시지
  - 가격 변경 = 음료수 종류 + 새 가격
  - 음료수 요구 = 총액 + 음료수 선택
  - 음료수 요구 결과 = (음료수) + (반환 동전) + 잔액
  - 자격 = 암호 + 시스템 모드
  - 자판기 상태 = {음료수 가격 + 음료수 상태} + 잔돈 상태

예제-자판기의 정보 흐름도



요구 분석서(1)

개요	• 목적, 범위, 정의 • 약어, 참고 문헌 등
일반적 사항	• 제품 기능, 사용자 특성 • 제약 사항, 가정 등
세부 요구사항	• 기능 요구사항 • 성능 요구사항(반응시간, 처리 소요 시간, 처리율 등) • 하드웨어 요구 사항(기억장치 규모, 통신 수용도) • 사용자 인터페이스 요구사항 • 운영 요구사항 • 자원 및 인력에 대한 요구사항 • 검증 요구사항 • 인수시험 요구사항, 문서화 요구사항, 보안 요구사항 • 이식성 요구사항, 품질 요구사항, 신뢰성 요구사항 • 유지보수성 요구사항, 안전 요구사항 등

요구 분석서(2)

- 요구 분석서의 평가 기준
  - 무결성과 완벽성(completeness)
  - 일관성(consistency)
  - 명확성(correctness)
  - 기능성(functional)
  - 검증 가능성(verifiability)
  - 추적 가능성(traceability)/변경 용이성(modifiability)