Chapter 9. 정리

1. 요구공학 모델링 활동의 모델의 종류및 특징

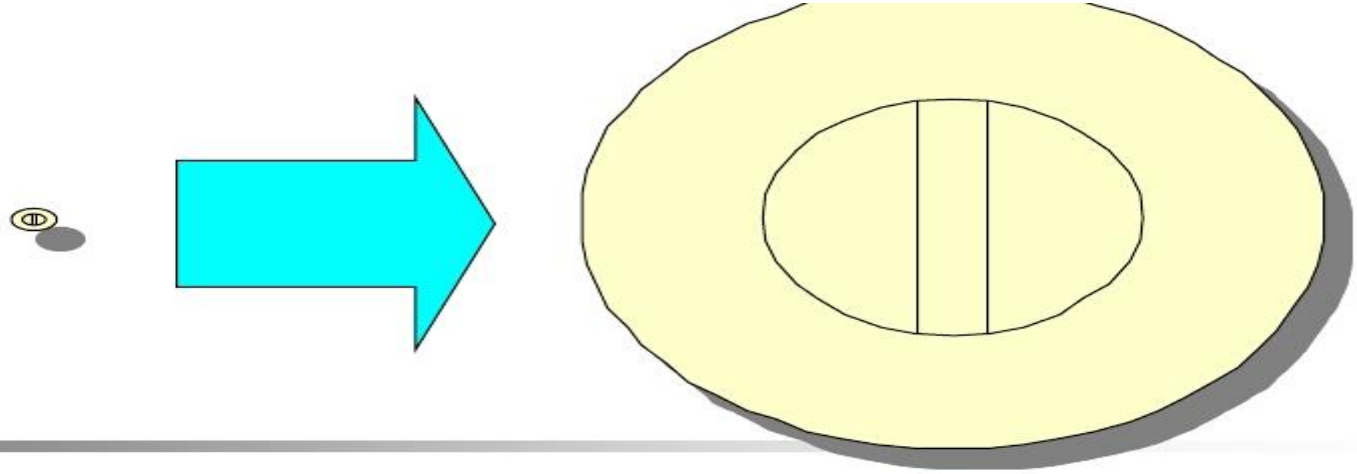
2. 분석 모델링 중 초점을 두는 point

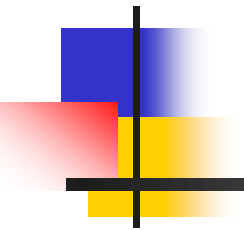
3. 요구사항 모델링 요소

4. 시나리오 기반 모델링 프로세스

나는 힘이 센 강자도 아니다 ,
그렇다고 뛰어난 천재도 아니다 ,
날마다 새롭게 변했을 뿐이다 .

단지 Change 에서 한자만 변경하면 Chance 가 된다





제 12장. 설계 개념

April. 2018
Young-gon, Kim
ykkim@kpu.ac.kr
Department of Computer Engineering
Korea Polytechnic University

Topics covered

- 소프트웨어공학 맥락 안에서의 설계
- 설계 프로세스
- 설계 개념
- 설계 모델

설계 개념

◆ 잘 설계된 소프트웨어 특성 (Software design manifesto)

- **견고함 (Firmness)**
  - 프로그램은 기능을 **방해하는 결함**을 갖지 않아야 함
- **편리함 (Commodity)**
  - 프로그램은 **의도한 목적**을 적절하게 수행해야 함
- **즐거움 (Delight)**
  - 프로그램을 **사용하는 경험**은 즐거워야 함.

◆ 설계의 목표

- 견고함, 편리함 및 즐거움을 줄 수 있는 **모델/표현**을 만들어 내는 것.

1.소프트웨어공학 맥락 안에서의 설계

◆ 설계 완성을 위한 필요한 4 가지 설계 모델

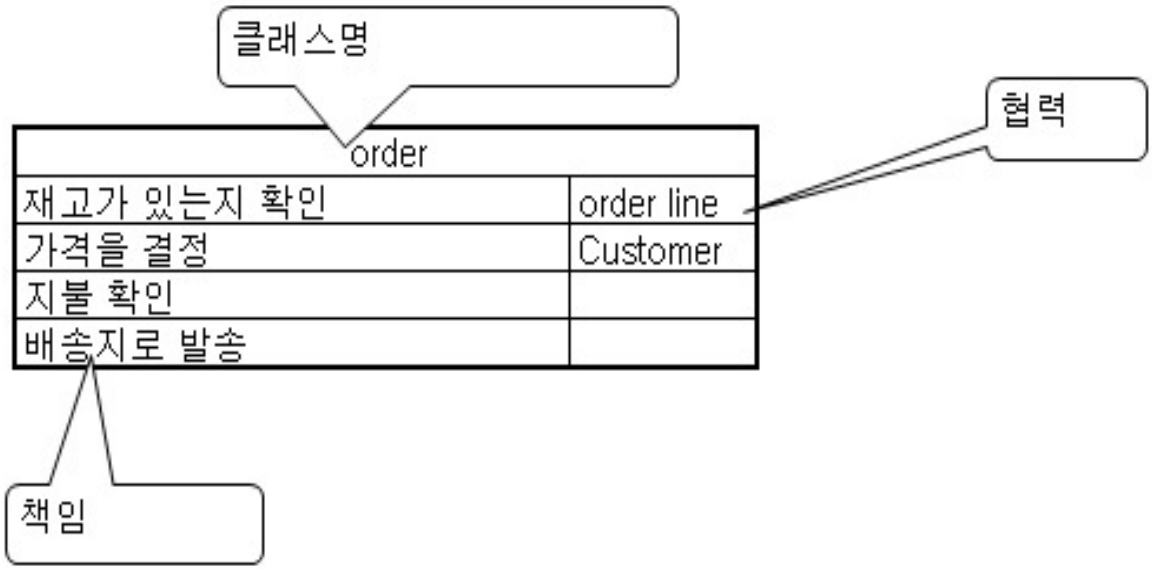
- **1) 클래스 모델**
  - SW 구축에 필요한 **클래스 실현**과 필수 데이터구조인 **데이터/클래스 설계**로 변형
  - **CRC 다이어그램** : 객체, 관계, 클래스 속성, 상세정보 -> **데이터 설계 활동** 기반
  - **상세한 클래스** 설계 : 각 소프트웨어 **컴포넌트 설계**시 작성
- **2) 아키텍처 설계**
  - **SW 주요요소사이 관계** , 시스템에 정의된 **요구사항을 달성**하는데 사용
    - **아키텍처 스타일** 과 아키텍처가 **구현될 수 있는 방법에 영향**을 주는 **제약조건**을 정의
  - 요구사항 모델로부터 도출 -> **아키텍처 설계 표현** (컴퓨터 기반 **프레임 워크**)
- **3) 인터페이스 설계**
  - 소프트웨어를 포함하는 **시스템** 및 이를 사용하는 **사람**과 소프트웨어가 **상호작용** 하는 방법을 기술
    - 인터페이스 : **정보의 흐름** (데이터 / 제어)과 특정한 **행동 유형**
  - **사용 시나리오 모델** -> 인터페이스 설계에 필요한 많은 정보 제공
- **4) 컴포넌트 수준 설계**
  - 소프트웨어의 **구조적 요소** -> 소프트웨어 **컴포넌트의 절차적 기술** 로 변환
  - **클래스 기반 모델**과 **행동 모델**로 부터 얻은 정보 -> 컴포넌트 설계의 기반 .

1.소프트웨어공학 맥락 안에서의 설계

◆ CRC 카드(Candidate, Responsibility, Collaboration)

- **클래스를 정의**하기위한 **키워드 산출** 및 **클래스간의 관계**를 간단하게 표기하는 문서

- CRC카드의 형태



1.소프트웨어공학 맥락 안에서의 설계

◆ CRC 카드(Class, Responsibility, Collaboration)

Problem

당신의 라이브러리 내용을 파악합니다.
각 책의 (하드커버, 종이커버, 등)에는 자세한 형식 그리고 제목과 저자명이 있고, 또한 당신은 당신의 책이 어느정도 양인지 파악합니다. 이전 조건에 해당하는 클래스 책의 설계, 방법을 증명합니다.

- 위에서 뽑은 키워드 들로 간단한 CRC 카드를 만들어보면 다음과 같은 형태

Libray	
자세한 형식 그리고 제목과 저자명 확인 책의 개수 확인	Book

Book
책의 형식 확인 책의 제목 확인 책의 저자명 확인

- class 정의

Library

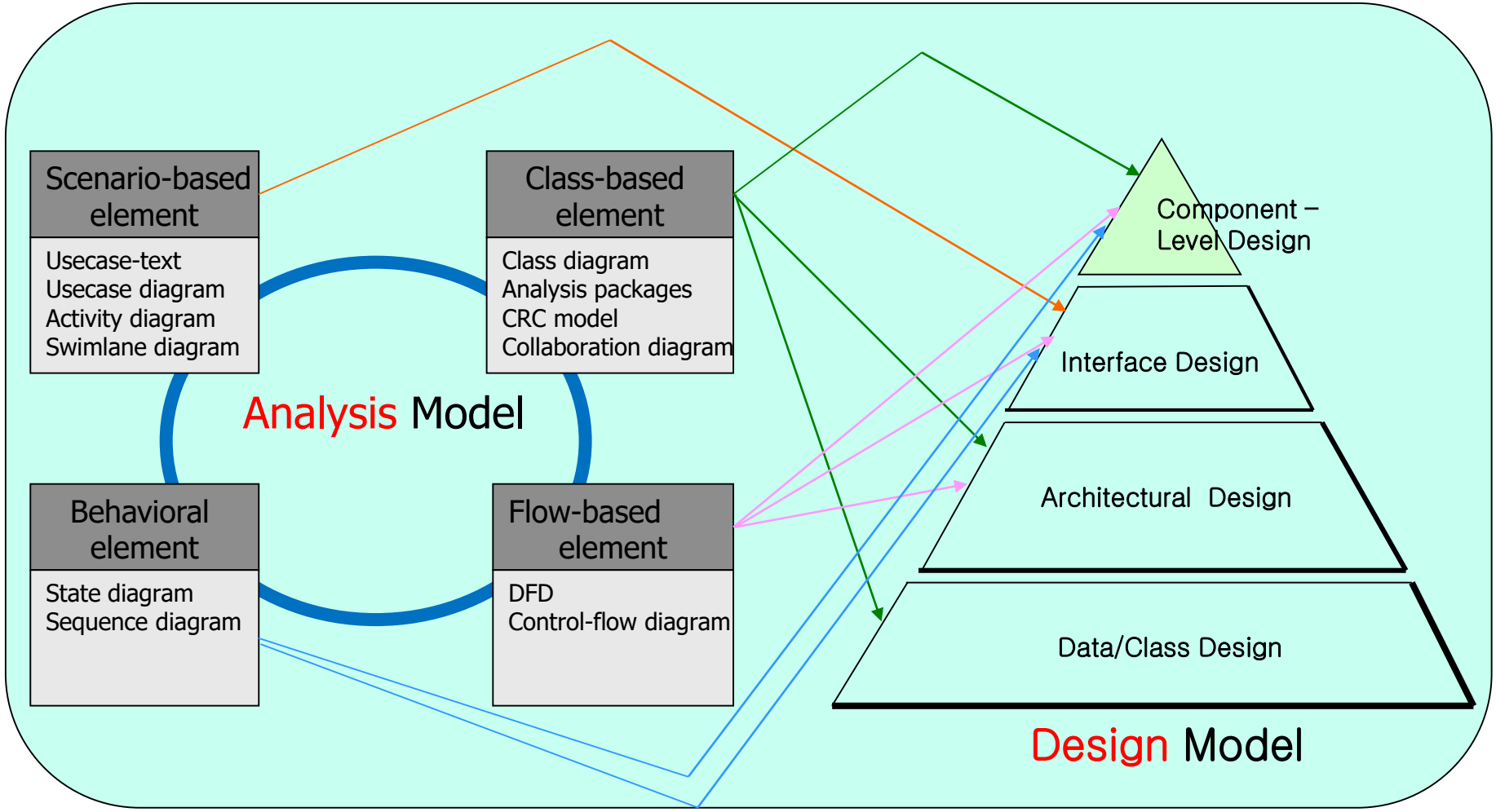
vector BookArray //Book 클래스의 집합,벡터,해쉬,맵 등등.
getBookInfo(Book) //자세한 형식 그리고 제목과 저자명 확인
getBookCount(BookArray)책의 개수 확인

Book
String BookFrom String BookName String BookWriter

getBookForm() //책의 형식 확인
getBookName() //책의 제목 확인
getBookWriter() //책의 저자명 확인
settBookForm(String bookForm) //책의 형식 넣기
setBookName(String bookName) //책의 제목 넣기
setBookWriter(String bookWriter) //책의 저자명 넣기

1.소프트웨어공학 맥락 안에서의 설계

◆ Requirement Model -> Design Model



2. 설계 프로세스

◆ 소프트웨어 품질 가이드 라인 및 속성

- **좋은 설계 평가** 를 위한 가이드라인 -> **설계 프로세스 목표**
  - 설계는 **모든 요구사항을 수용**
    - **요구사항 모델** : 포함된 **명시적인 모든 요구사항**을 구현
    - 이해당사자 : 원하는 **암시적인 모든 요구사항**을 수용
  - 설계는 **읽기 쉽고 이해하기 쉬워**야 함
    - **코드를 생성**하는 사람
    - 소프트웨어를 **시험**하는 사람
    - 향후 **소프트웨어 지원**을 담당할 사람
  - 설계는 소프트웨어 **전반적인 그림** 제공
  - 설계는 **구현 관점에서 데이터, 기능 및 행동 영역**을 다뤄야 함.

2. 설계 프로세스

◆ 설계 품질 가이드 라인 : 설계 프로세스의 실질적 목표

- **설계는 3 가지 특성**을 만족하는 **아키텍처** 생성
  - 잘 알려진 **아키텍처 스타일 또는 패턴**을 사용하여 생성
  - **좋은 설계 특성을 보이는 컴포넌트**들로 구성
  - **구현과 시험이 가능하도록 혁신적인 방법**으로 구현 : 작은 시스템 순차적 설계
- 설계는 **모듈화**
  - 소프트웨어는 요소/하부시스템으로 **논리적 분할**
- 설계는 **데이터, 아키텍처, 인터페이스및 컴포넌트**구분하여 표현
- 설계는 **자료구조**를 생성
  - 자료구조는 구현할 클래스에 적절하고 잘 알려진 **데이터 패턴**으로 부터 도출
- 설계는 **독립적인 기능적 특성을 갖는 컴포넌트**를 도출
- 설계는 **연결 복잡성을 줄이는 인터페이스** 도출
  - 컴포넌트와 외부 환경간
- 결정한 설계 기법을 **반복적으로 적용**하여 설계
  - 소프트웨어 요구사항 분석 중 얻은 정보
- 설계는 의미를 효과적으로 전달하는 **기호(notation) 사용**하여 표현.

2. 설계 프로세스

◆ 설계 품질 속성 (FURPS)

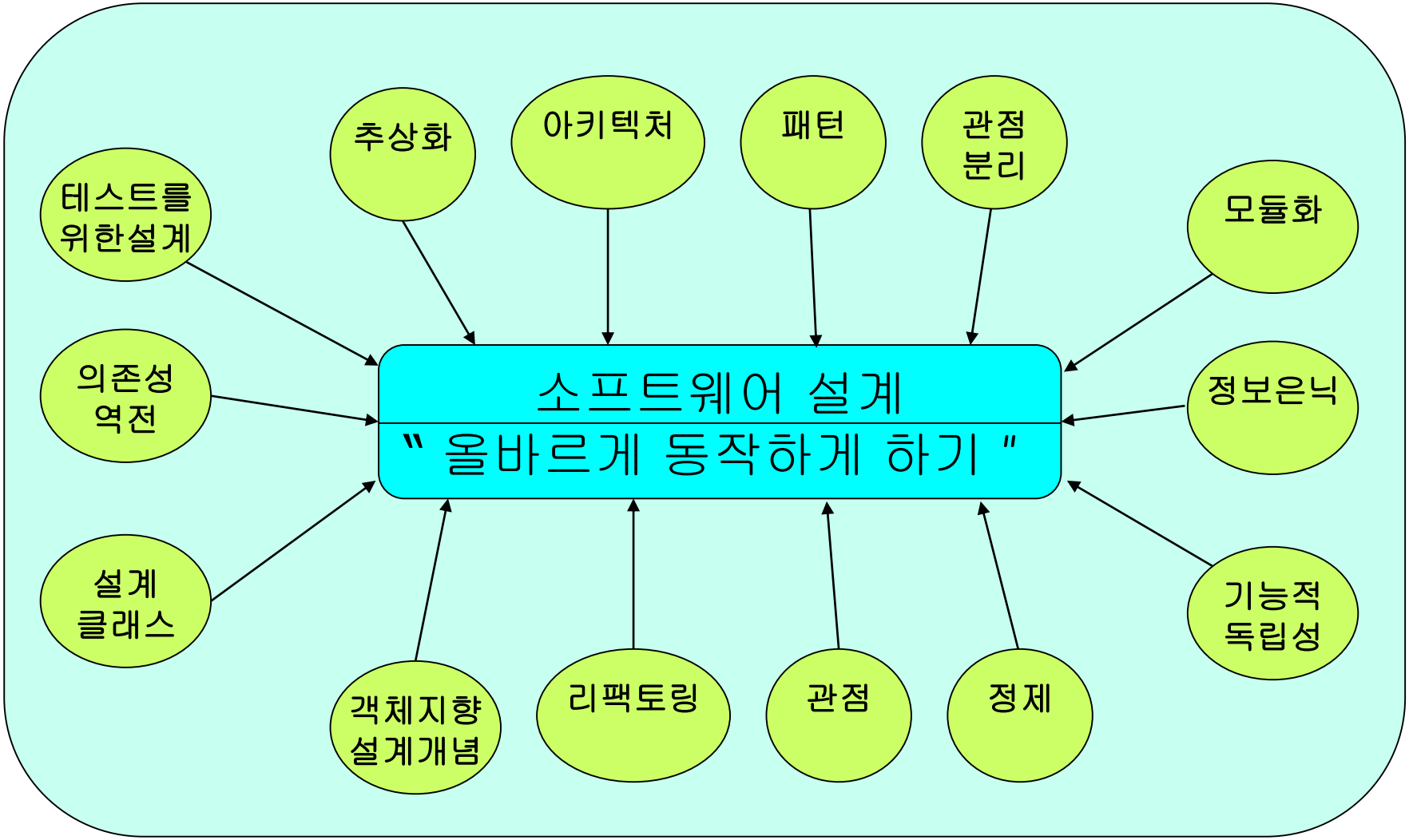
- **기능성** (Functionality)
  - 프로그램의 **특징 및 기능**으로 평가
  - 시스템이 제공하는 **기능 및 보안**으로 평가
- **사용성** (Usability)
  - **인적 요소, 전체적인 심미성, 일관성, 문서**를 평가
- **신뢰성** (Reliability)
  - 출력 결과의 **정확성**, 프로그램의 **예측 가능성**을 측정하여 평가
  - **장애 빈도및 심각도, 평균 장애시간, 장애로 부터 복구하는 능력** 측정하여 평가
- **성능** (Performance)
  - **처리속도 , 응답시간 , 자원 소비 , 처리 능력 및 효율성** 측정하여 평가
- **지원성** (Supportability)
  - **확장성 , 적응성 , 사용성**
  - **시험 가능성 , 호환성 , 구성용이성**(컴포넌트 조직 제어 능력)
  - 시스템 **설치가 가능한 용이성 , 문제 식별의 용이성**.

2.2 소프트웨어 설계의 진화

◆ 설계를 위한 일반 작업(프로세스)

1. 데이터 구조 설계
   - 정보 도메인 모델을 조사하고 데이터 객체 및 그 속성
2. 아키텍처 스타일 (패턴) 선정
   - 분석 모델 을 사용하여 소프트웨어에 적합한 것
3. 분석 모델 / 분석 모델 요소를 하부시스템으로 할당
   - 하부시스템 (기능적으로 응집성), 하부시스템 (인터페이스 설계)
   - 분석 클래스 또는 기능 을 각 하부시스템에 할당
4. 설계 클래스 또는 컴포넌트 생성
   - 분석 클래스 설명 (설계 클래스로 변환) : 클래스와 연관된 메소드와 메시지 정의
5. 외부 시스템 또는 장치와의 필요한 인터페이스 설계
6. 사용자 인터페이스 설계
   - 인터페이스 행동 모델 생성 , 인터페이스 객체 / 제어 메커니즘 정의
7. 컴포넌트 수준 설계를 수행
   - 컴포넌트 수준 데이터 구조 정의
8. 배치 모델 개발.

3. 설계 개념

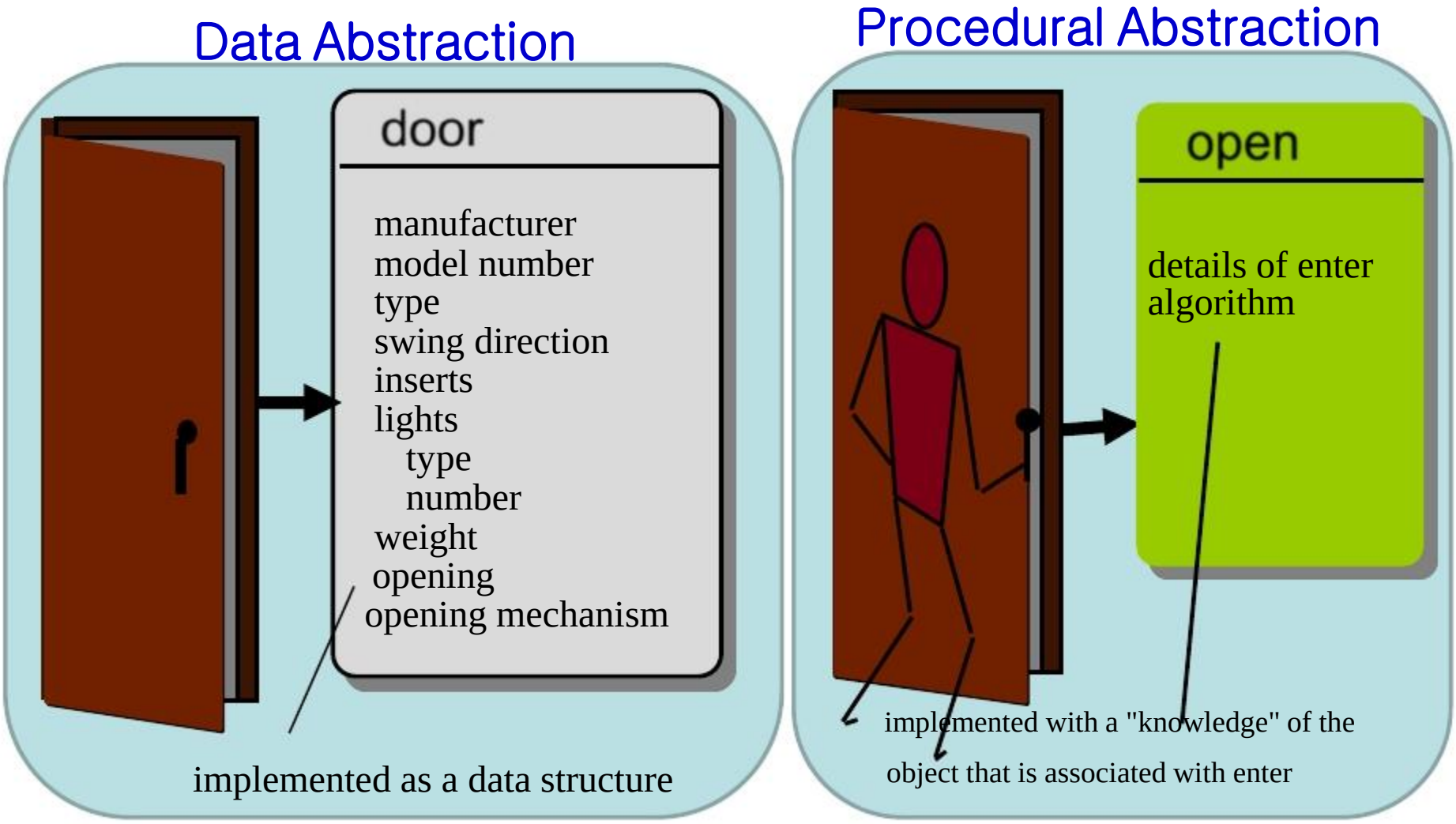

추상화
아키텍처
패턴
관점 분리
테스트를 위한설계
모듈화
의존성 역전
소프트웨어 설계
" 올바르게 동작하게 하기 "
정보은닉
설계 클래스
기능적 독립성
객체지향 설계개념
리팩토링
관점
정제

3. 설계 개념

1. 추상화

- 수준
  - **최고 추상화** 수준 : 해결책은 문제가 속한 영역의 언어 사용 -> **개요수준 , 설명**
  - **낮은 추상화** 수준 : 추상화에서는 보다 **상세한 설명**
- 추상화 개발 과정
  - 1) **절차적 추상화**
    - **구체적이고 제한적인 기능** 을 갖는 일련의 명령문
    - 사용하는 이름 : 수행할 기능들을 암시, 구체적인 내용 미표현
    - 예) 문에 대해 열다 : "열다"는 내부적으로 여러 **절차적 단계**
      - 도어로 걸어간다-> 도달하여 손잡이를 잡는다 -> 손잡이를 돌리고 도어를 당긴다 -> 움직이는 도어로부터 물러선다 . 등
  - 2) **데이터 추상화**
    - 데이터 객체를 설명하는 **데이터 집합에 이름을 부여** 하는 것
    - 예) 문에 대해 열다 : "문"이 갖고 있는 **속성**들
      . 문 종류 , 회전 방향 , 개방방식, 무게, 치수
    - 문을 열다
      . 데이터 추상화 : "문"
      . 절차적 추상화 : "열다"는 데이터 추상화 "문"이 정의한 **속성들의 값**.

3. 설계 개념


Data Abstraction
door
manufacturer
model number
type
swing direction
inserts
lights
type
number
weight
opening
opening mechanism
implemented as a data structure
Procedural Abstraction
open
details of enter algorithm
implemented with a "knowledge" of the object that is associated with enter

3. 설계 개념

◆ 2. 아키텍처

- 소프트웨어 아키텍처
  - 소프트웨어의 **전체적인 구조** 와 해당 구조가 시스템에 대한 **개념적 무결성**을 제공하는 방법
  - 아키텍처 **설계의 속성**

    1) **구조적 속성**

    – 시스템 컴포넌트와 컴포넌트를 **묶는 방법과 상호작용**하는 방법

    2) **추가 기능적 속성**

    – 설계 아키텍처가 **성능, 용량, 신뢰성, 보안, 적응성, 시스템 특성 요구사항** 달성 방법

- **아키텍처 설계 모델**

  1) **구조적 모델** : 프로그램 컴포넌트의 조직화된 집합으로 아키텍처 표현

  2) **프레임워크 모델** : 유사한 유형의 응용분야 에서 나타나는 반복적인 아키텍처 설계 프레임워크(패턴)을 식별하여 설계 추상화 수준 향상

  3) **동적 모델** : 프로그램 아키텍처의 행동 관점, 구조 또는 시스템 구성이 외부 이벤트 의 함수에 따라 변화 모습 표현

  4) **프로세스 모델** : 시스템이 수용해야 하는 업무 기술 프로세스의 설계에 중점

  5) **기능적 모델** : 시스템의 기능적 계층 을 표현하는 데 사용.

3. 설계 개념

◆ 3. 패턴

- **설계 패턴**
  - 관점들이 상충하는 가운데 특정한 문맥 안에서 **반복적인 문제로 입증된 해결방법**의 본질을 만들어내기 위해 통찰로 얻어진 이름
  - 특정한 상황에 있는 **설계 문제를 해결할 설계구조**를 제시하는 것
  - 설계 패턴 기술 시 결정할 사항
    1) 패턴이 현재 **작업에 적용** 될 수 있는지 여부
    2) 패턴이 **재사용** 될 수 있는지 (설계시간을 절약 목적)
    3) 패턴이 유사하지만 **기능/구조적**으로 상이한 패턴을 개발하는데 **가이드 역할** 제공 여부.

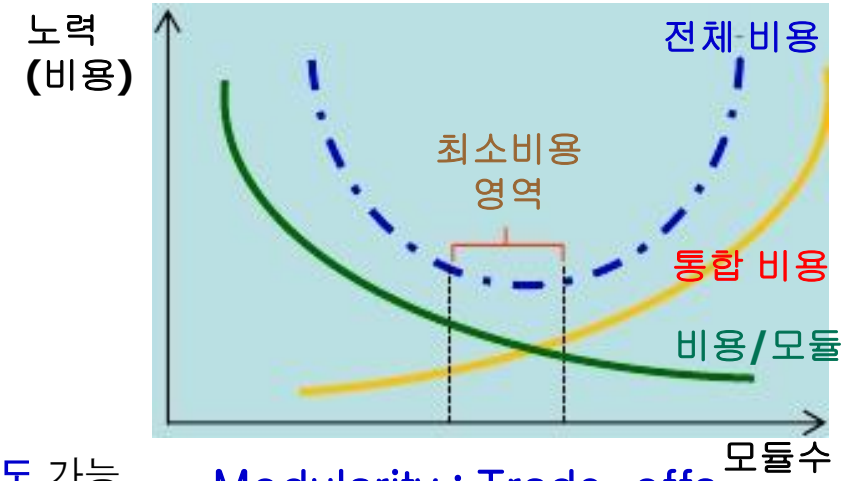
◆ 4. 관점 분리

- 관점 분리 개념 : **독립적**으로 해결 / 최적화될 수 있는 조각으로 세분화
  - 관점 : 소프트웨어 대한 요구사항 모델의 일부로서 명시되는 **특징 및 행동**
- 관점을 **작고 다루기 쉬운 조각** 으로 분리 -> 문제 해결에 필요한 노력/시간 절감
- **분할 정복 전략**
  - 복잡도 : 합쳐진 문제 인식 복잡도 >> **개별 문제로 인식**할 때 복잡도
- 관점분리와 관련된 개념
  - **모듈화, 관점, 기능적 독립성, 정제.**

3. 설계 개념

◆5. 모듈화

- **관점 분리**의 가장 흔한 표현
- 프로그램을 **관리할 수 있는 단위** 로 만들어 주는 소프트웨어 특성
- **최소 개발비**를 유발하는 모듈 개수
  - 모듈 총수가 증가함에 따라 -> **모듈 개발 노력 (비용) 감소**
  - 모듈 총수가 증가함에 따라 -> **통합에 관련된 노력 (비용) 증가**



Modularity : Trade-offs

- **모듈화 원칙**
  - **개발 용이** 하게 계획
  - 소프트웨어 증분 정의하고 **증분별로 인도** 가능
  - **변경 사항을 좀 더 쉽게 수용** 가능
  - **시험및 디버깅을 보다 효율적** 으로 실시
  - **유지보수의 심각한 부분 없이** 장시간 수행가능.

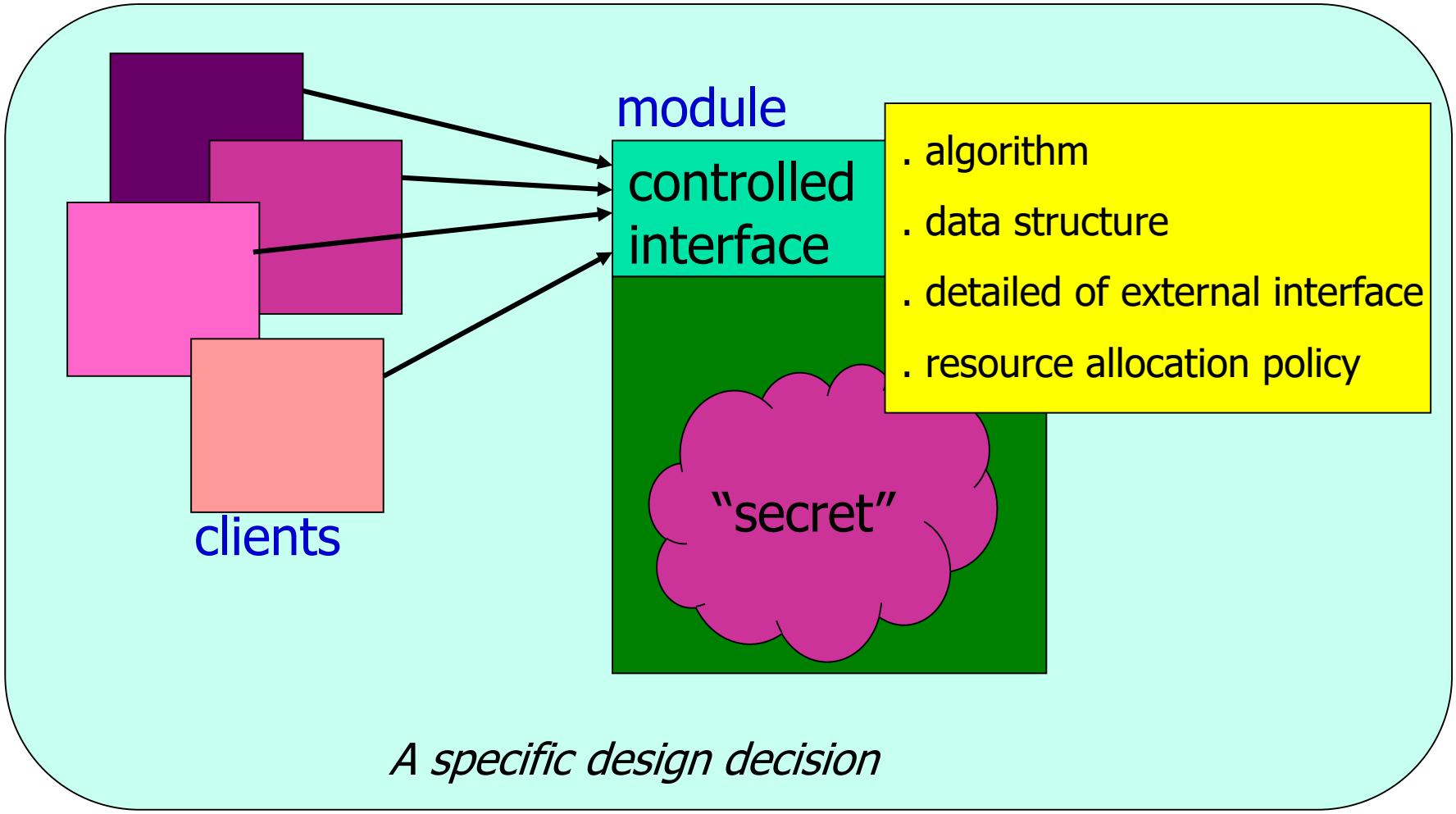
3. 설계 개념

◆6. 정보은닉

- 정보은닉의 **원칙**
  - **다른 모든 것들로부터 숨기는 설계 결정**을 만족하는 것
  - 모듈은 모듈에 포함된 정보가 **해당 정보를 다른 모듈은 접근할 수 없도록** 설계
    - 정보 : 알고리즘 , 데이터
- **은닉**
  - 소프트웨어 기능을 완성 위한 **필요한 정보만 교류** -> **모듈을 독립적** 으로 생성
  - 소프트웨어를 구성하는 **절차적(정보제공) 개체** 를 정의 -> **추상화** 개념 필요
    - 모듈 안에 정의한 절차적 논리와 모듈이 사용하는 **지역 데이터 구조를 접근**하는데 **제약**을 정의하고 반영하는 것
- 정보은닉 적용시 **이점**
  - **테스트 기간** 중과 향후 **유지보수 단계** 에서 **수정이 발생**시 이점
    - **데이터와 절차적 처리**가 다른 부분으로 부터 은닉-> 변경중 발생할 수 있는 **오류가 다른 위치로 파급될 가능성이 적음.**

3. 설계 개념

◆정보은닉


module
controlled
interface
. algorithm
. data structure
. detailed of external interface
. resource allocation policy
"secret"
clients
A specific design decision

3. 설계 개념

◆7. 기능적 독립성

- 기능적 독립성 개념
  - **관점의 분리 , 모듈화 , 추상화 및 정보은닉**으로부터 얻어지는 결과물
  - 정제기법이 **모듈 독립성 강화**하는 방법
  - **단일 기능 수행**하게 하고 , 다른 모듈과의 **상호작용을 최대한 줄여** 모듈 개발
- 기능적 독립성 **이점**
  - 함수 구분이 쉽고 함수간 **인터페이스가 단순** -> 개발이 쉬움
  - **유지보수 (시험) 이 용이**
    - 이후 설계 또는 코드 수정이 발생해도 그 **파급 범위가 제한적**
    - 오류 발생시 **파급효과 적음**
    - 모듈의 **재사용** 이 가능
- **독립성 평가 기준**
  - **응집도 (Cohesion)** : 모듈이 갖는 **단일 기능 수행** 여부에 대한 지표
    - **정보 은닉 개념**을 확장한 개념 .
    - 응집도 높은 모듈 : 단일 기능 수행 , **다른 컴포넌트와 상호작용 거의 불필요**
  - **결합도 (Coupling)** : 모듈 간에 존재하는 상호의존성 에 대한 지표
    - 결합도 발생 : 인터페이스 복잡성 , 모듈에 대한 입력 / 참조 지점 ,
    - 결합도 낮은 모듈 : **이해하기 쉽고 , 한 모듈에서 오류가 발생해도 시스템 전체로 파급 효과 적음** .

3. 설계 개념

8. 정제

- 단계적 정제 : 하향식 설계 전략
  - **절차적 처리의 상세함을 지속으로 정제 과정** -> 응용 프로그램 개발 과정
  - 프로그래밍 언어로 **표현할 문장으로 도출** 할 때까지 기능의 내용을 분할 : **계층구**조 형성
- **정련의 프로세스**
  - 초기 설명문을 정리하면 이어지는 **정제 과정** -> 더 많은 **상세화**
- 추상화와 정제는 보완적 개념
  - **추상화** : **절차와 데이터를 내부적**으로 명세 , 외부요소는 구체적 내용을 알 필요 없게함
  - **정제** : 설계가 진행하면서 **하위 내용을 드러내는 것** .

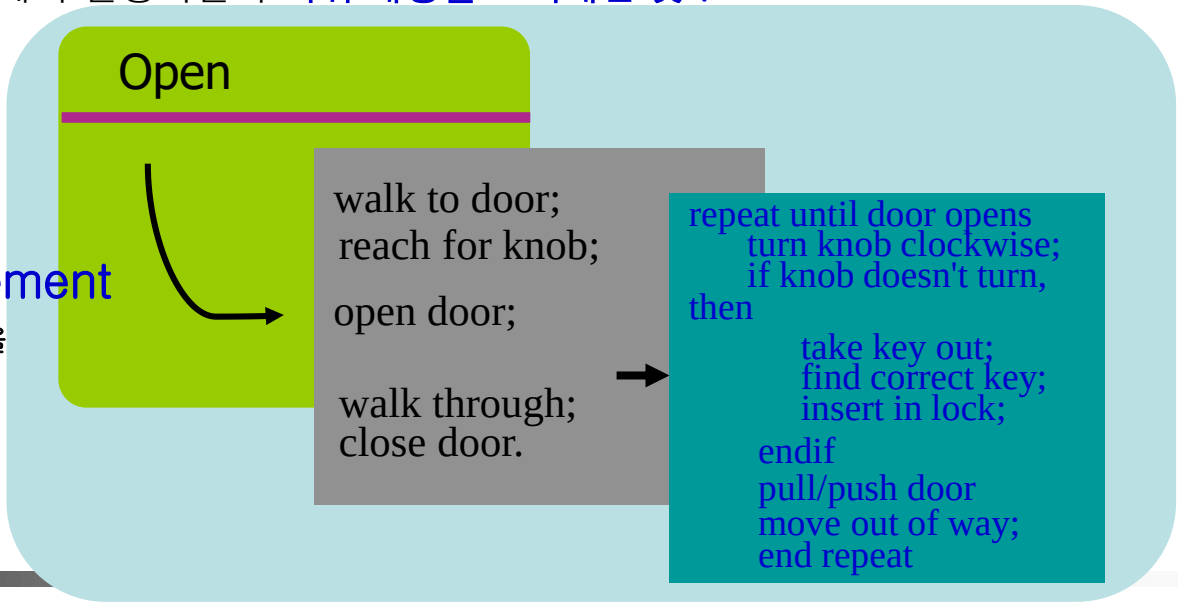
Stepwise Refinement

:데이터 정의/처리 스텝을

우선 크게 정의하고,

상세히 해 가는 개발 방법

:Top-Down



3. 설계 개념

◆9. 관점

- **요구사항 분석**하면 -> 관점
  - 관점 : **요구사항 , 유스케이스 , 특징 데이터 구조 , 서비스 품질문제 , 가변성 , 지적재산권의 범위 , 공동작업 , 패턴및 계약** 등을 포함
  - 요구사항 모델 을 구성
    - 각 관점을 **독립적으로 고려**될 수 있도록 **관점 (요구사항) 분리 허용**하는 방법
  - 관점 : **교차하는 관심사**들을 표현
    - 관점을 제대로 수용하고 , 모듈화가 이루어 지도록 관점을 식별
    - 많은 컴포넌트에 **"분산", "뒤얽힌" 조각** 으로 분리 보다 -> **별도 모듈(컴포넌트)** 구현.

◆10. 리팩토링

- **기능/행위를 변경하지 않고** 컴포넌트 설계(코드)를 단순화하여 **재구성** 기법
- 코드의 외부 행위를 변경하지 않고 **내부 구조를 개선**하는 방법
- 소프트웨어 리팩토링후 **검사**
  - 설계가 **중복성**을 갖고 있는지 ?
  - **사용하지 않은 설계 요소**가 있는지 ?
  - **비효율적이거나 불필요한 알고리즘**이 있는지 ?
  - **잘못 구성했거나 부적절한 데이터 구조**가 있는지 ?
  - 그 밖에 **개선을 통해 좋은 설계**가 될 수 있는지 ?

3. 설계 개념

◆ 11. 객체지향 설계 개념

- 객체지향 설계 **5 개 개념**
  1) Object : 실제존재 , 클래스 구현 / 인스턴스 (실제값)-> 실제 수행 (코드 단위)
  2) Class : 특정 종류의 객체내에 있는 변수 / 메소드를 정의하는 Template
  3) Inheritance : 부클래스는 상위 클래스의 하나 / 그 이상의 정의를 물려받음
  4) Abstract : 객체와 프로시저들의 공통 특징 을 골라내는 과정
  5) Interface : 의사소통 수단 -> 물리적인 / 논리적인 측면
- 객체지향 5 가지 원칙
  1) **개방 – 폐쇄의 원칙 (OCP : Open Closed Principle) : 확장에 열려 , 변경에 닫혀**
     - 변화는 확정을 고려하여 설계 하되 , 클래스 변경없이 **인터페이스만 노출하여 캡슐화**
  2) **리스코프 치환 원칙 (LSP : Liskov Substitution Principle)**
     - 슈퍼클래스를 서브클래스로 대처해도 가동가능 -> **상속**과 인터페이스 개념
  3) **의존관계 역전의 원칙 (DIP : Dependency Inversion Principle)**
     - 추상적으로 의존 -> 추상적이면 LSP 에 따라 교체 가능 (**유연성 높음**)
  4) **인터페이스 분리의 원칙 (ISP : Interface Segregation Principle)**
     - 다양한 인터페이스 조합가능 (세부적 분리)-> **다수** 구체적인 인터페이스 사용
  5) **단일 책임의 원칙 (SRP : Single Responsibility Principle)**
     - 하나의 클래스는 **하나의 책임**만 -> 클래스 방대 / 복잡성 방지 .

3. 설계 개념

◆ 12. 설계 클래스

- **분석 모델 : Sets{ 다수 분석 클래스 }**
  - **클래스** : 사용자가 **문제 이해하는 관점**에 따라 문제영역의 일부 요소를 기술
- 서로 다른 설계 아키텍처를 표현하는 **5 가지 종류의 설계 클래스**

  1) **사용자 인터페이스 클래스** : **HCI 를 구현하는 모든 추상화** 를 은유적으로 정의
  2) **업무 분야 클래스** : **업무영역**의 일부 요소를 구현하는데 필요한 **속성과 방법** 정의
  3) **프로세스 클래스** : 업무영역 클래스를 처리하는데 필요한 **하위업무추상화**를 구현
  4) **저장클래스** : 소프트웨어 실행 이후에도 지속하는 **데이터 저장소**(데이터베이스)
  5) **시스템 클래스** : 컴퓨팅 환경 내 / 외부 세계와 동작하고 교류하기 위해 필요한 **SW 관리 및 제어 기능**

- 좋은 모습의 설계 클래스가 갖는 **4 가지 특성**

  1) **완전 및 충분** : 완전 (**속성과 메소드** 모두 포함), 충분 (클래스 의도대로 **메소드만** 포함)
  2) **원초적** : 메소드는 **하나의 서비스** 만 달성하는데 집중
  3) **높은 응집도** : 응집도높은 클래스
     (작지만 **집중된 책임**, 책임을 구현하는 **속성/메소드는 한가지 목적**만 갖음)
  4) **낮은 결합도** : 협업은 최소화 , **강한 결합 (시스템 구현 , 시험 , 유지관리가 어려움)**
     하부시스템의 클래스 (다른 클래스에 대해 **제한적으로 알아야함**)
     하나의 메소드는 이웃 클래스에 있는 **메소드를 통해서 메시지 전송 .**

3. 설계 개념

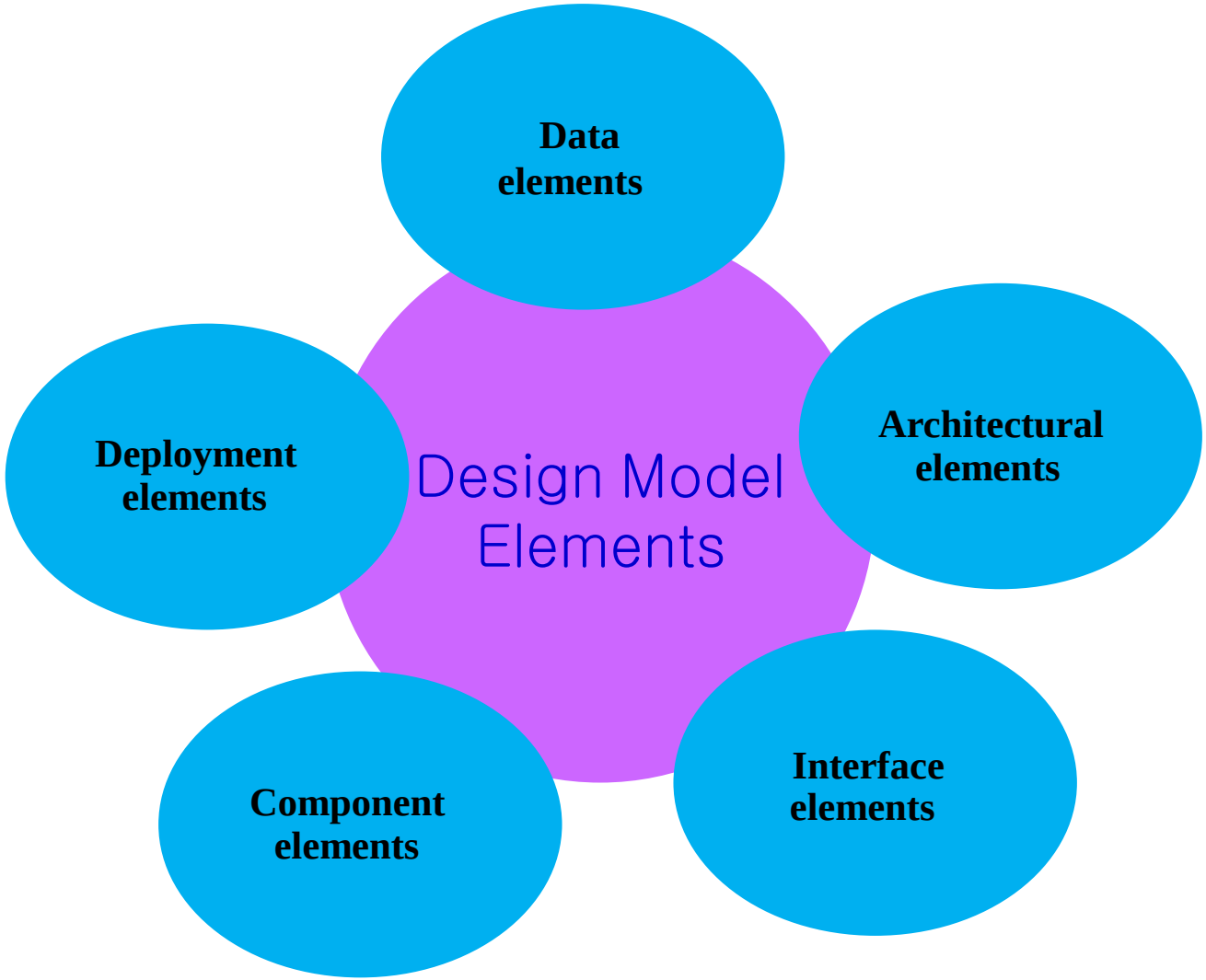
◆ 13. 의존성 역전

- 소프트웨어 아키텍처 구조 : **계층적 -> 의존적**
  - 아키텍처 상부 : **"제어" 컴포넌트**
  - 아키텍처 하위 : "작업자" 컴포넌트가 제공하는 **응집력있는 작업 수행**
- **의존성 역전 원칙 일반화**
  - 상위 모듈 (클래스) : 하위모듈에 [직접] **의존하지 말아야 함**
  - 상위모듈과 하위모듈 모두 **추상화** 를 만족해야 함
  - **상세클래스**가 추상화클래스를 사용하여 의존
    - 추상 클래스가 상세클래스에 의존하는 것이 아님.

◆ 14. 테스트를 위한 설계

- 테스트 주도 개발자 : 코드를 구현하기 전에 **시험을 먼저 작성**
  - 설계를 먼저 한다면 : 설계 (코드) 와 봉합선과 함께 개발 .
  - **봉합선 (seams)**
    - 상세설계시 "실행 소프트웨어의 상태를 **탐지하는 시험 코드를 삽입** "한 지점과 "실 환경 환경과 분리하여 **테스팅 환경을 통제** 하면서 시험을 진행할 수 있는 환경에서 코드를 분리한 지점"
  - Test Hook[봉합선] : 컴포넌트 수준에서 설계
    - 시험코드가 **실행 시스템을 면밀하게 살피고 통제할 수 있는 설계** 로 바뀌 나갈 수 있게 적절한 테스트 유도.

4. 설계 모델


Data elements
Architectural elements
Interface elements
Component elements
Deployment elements
Design Model Elements

4. 설계 모델

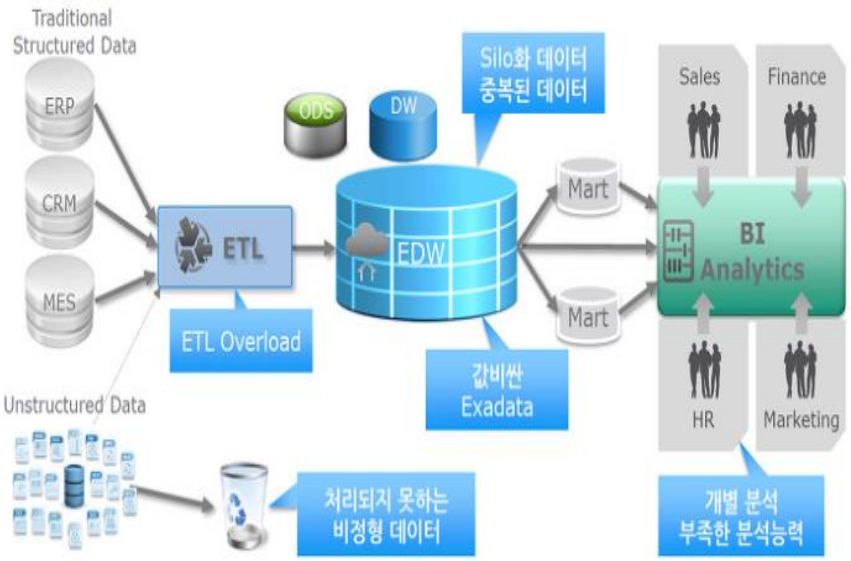
◆ 1. 데이터 설계 요소

- 데이터 설계(데이터 아키텍팅)
  - 상위 추상화 수준 (고객/사용자 관점)에서 표현해야 하는 **데이터 / 정보 모델** 생성
- 데이터 모델 -> **데이터베이스로 전환**
  - 시스템이 갖는 **비즈니스 목적 달성**의 중요 요소
  - 데이터베이스에 저장하고, " **데이터 웨어하우스 (data+warehouse)** "로 재구성 정보 수집

object: automobile

attributes:
- **make**
- **model**
- **body type**
- **price**
- **options code**

[Data Objects and Attribute]

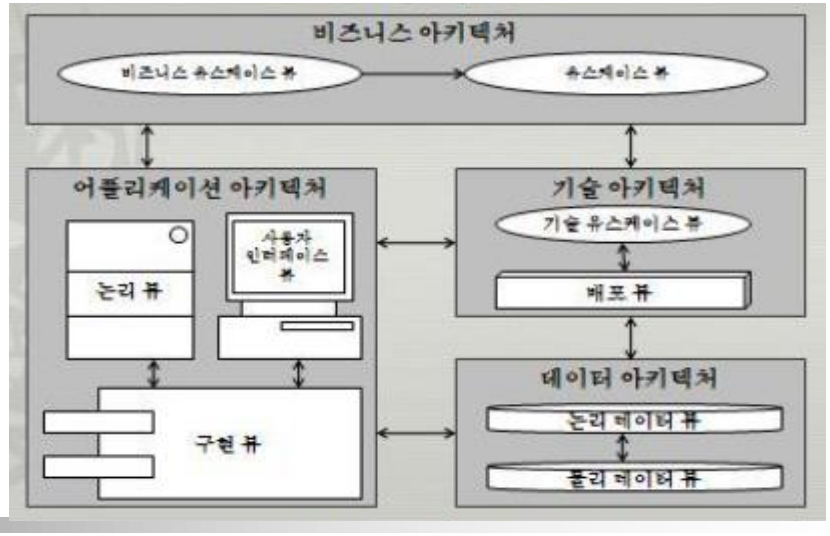


4.설계 모델

◆2. 아키텍처 설계 요소

- 아키텍처 설계 요소
  - 소프트웨어 **전체적인 모습** 제공

    예) 건물의 평면도 : 방의 전체 배치 (크기, 모양, 상호간 관계, 문, 창문)
- 아키텍처 모델 생성을 위한 **참조**
  - 소프트웨어를 구축하는 **응용분야에 관한 정보**
  - **유스케이스 또는 분석 클래스** , 해결 과제가 갖고 있는 **관련성 및 협업**
  - 사용 가능한 **아키텍처 스타일 패턴**
    - 아키텍처 스타일 : 데이터 중심/데이터 흐름/호출및 복귀/객체지향/층위구조 아키텍처.



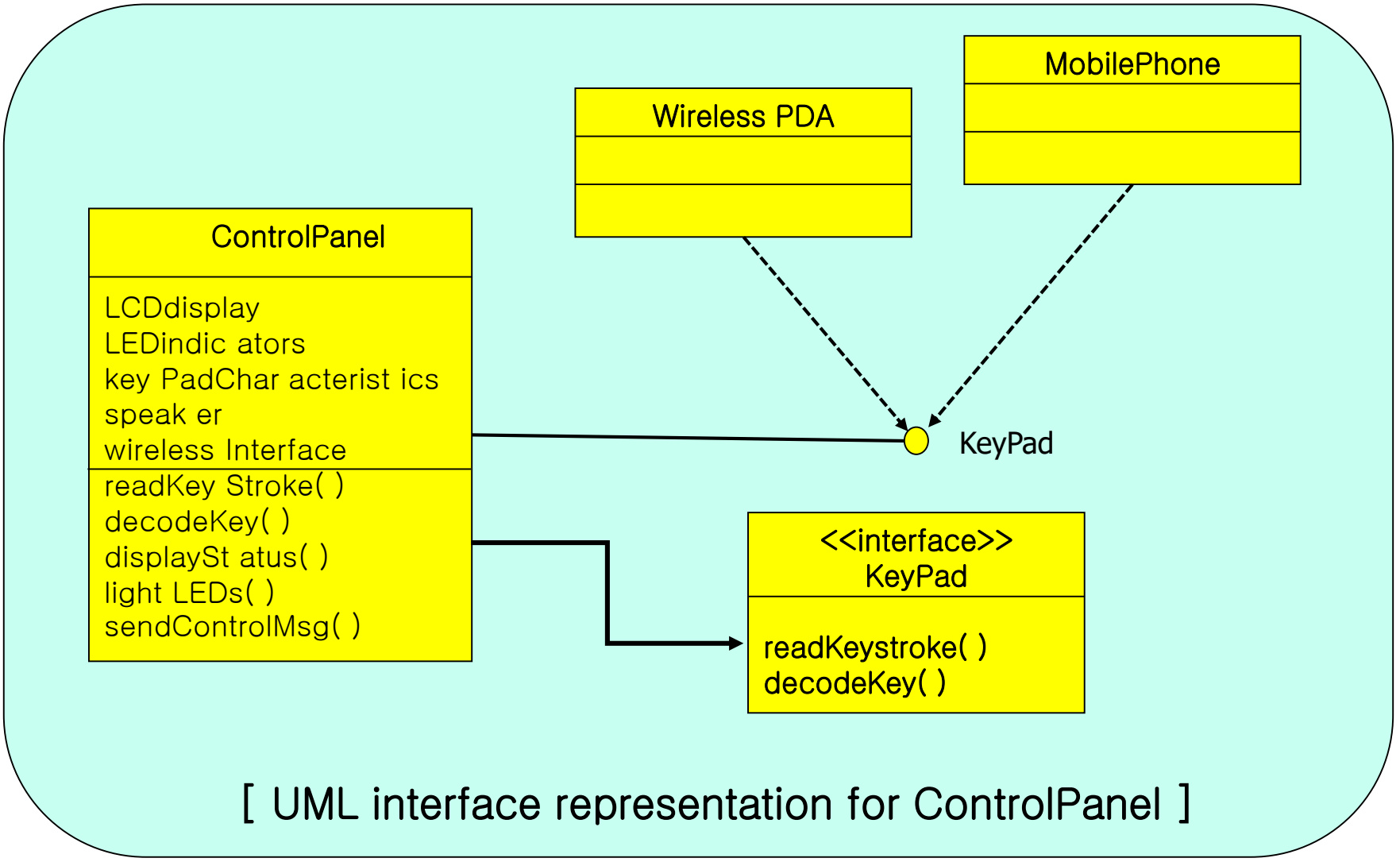
4. 설계 모델

◆3. 인터페이스 설계 요소

- 소프트웨어 인터페이스 **설계 요소**
  - 시스템 안팎으로의 **정보 흐름**을 표현 -> **컴포넌트 사이의 교류**를 보여줌
  - 소프트웨어가 **외부와 교류**하고 , 컴포넌트 사이의 **내부 교류**와 **협업** 가능

- 인터페이스 설계의 **3 가지 중요한 요소**
  - UI 설계 (사용성 설계)
    - **심미적요소** : 배치도, 색깔, 그래픽, 상호작용 메커니즘
    - **인체공학적 요소** : 정보 배치, 탐색
    - **기술적 요소** : UI 패턴, 재사용가능 컴포넌트
  - 외부 인터페이스 설계
    - 정보를 **주고받는 대상에 대한 정보** 필요
    - **오류검사**와 적절한 **보안 기능** : 외부 인터페이스 설계와 통합
  - 컴포넌트 설계
    - **내부인터페이스 설계**를 긴밀하게 조정
    - **연산자와 메시징**(연산자 사이의 교류/협업)을 분석클래스를 설계 표현으로 변환 .

4. 설계 모델

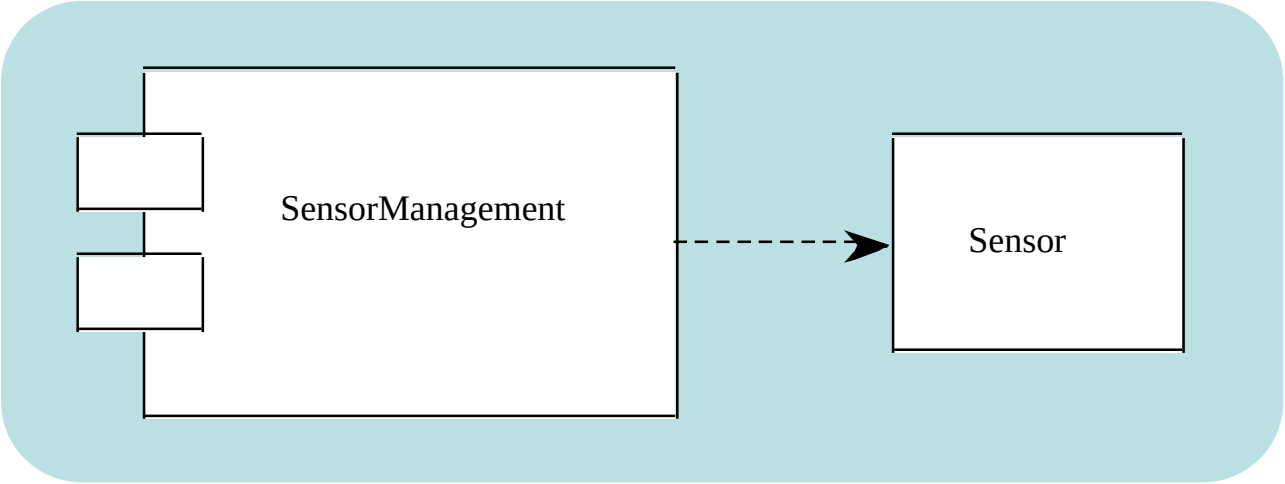


[UML interface representation for ControlPanel]

설계 모델

◆ 4. 컴포넌트 수준 설계 요소

- 소프트웨어의 컴포넌트 수준 설계
  - 소프트웨어 **컴포넌트 내부를 아주 상세** 하게 기술
  - 정의할 사항
    - **자료 구조** : 모든 내부 데이터 객체에 대한 자료구조
    - **알고리즘** : 컴포넌트 내 모든 처리와 연산자 접근 허용하는 인터페이스
  - 예) 각방의 상세도 : 도면 (배선, 배관, 전기소켓, 스위치, 수도꼭지, 샤워기, 욕조 , 벽장)



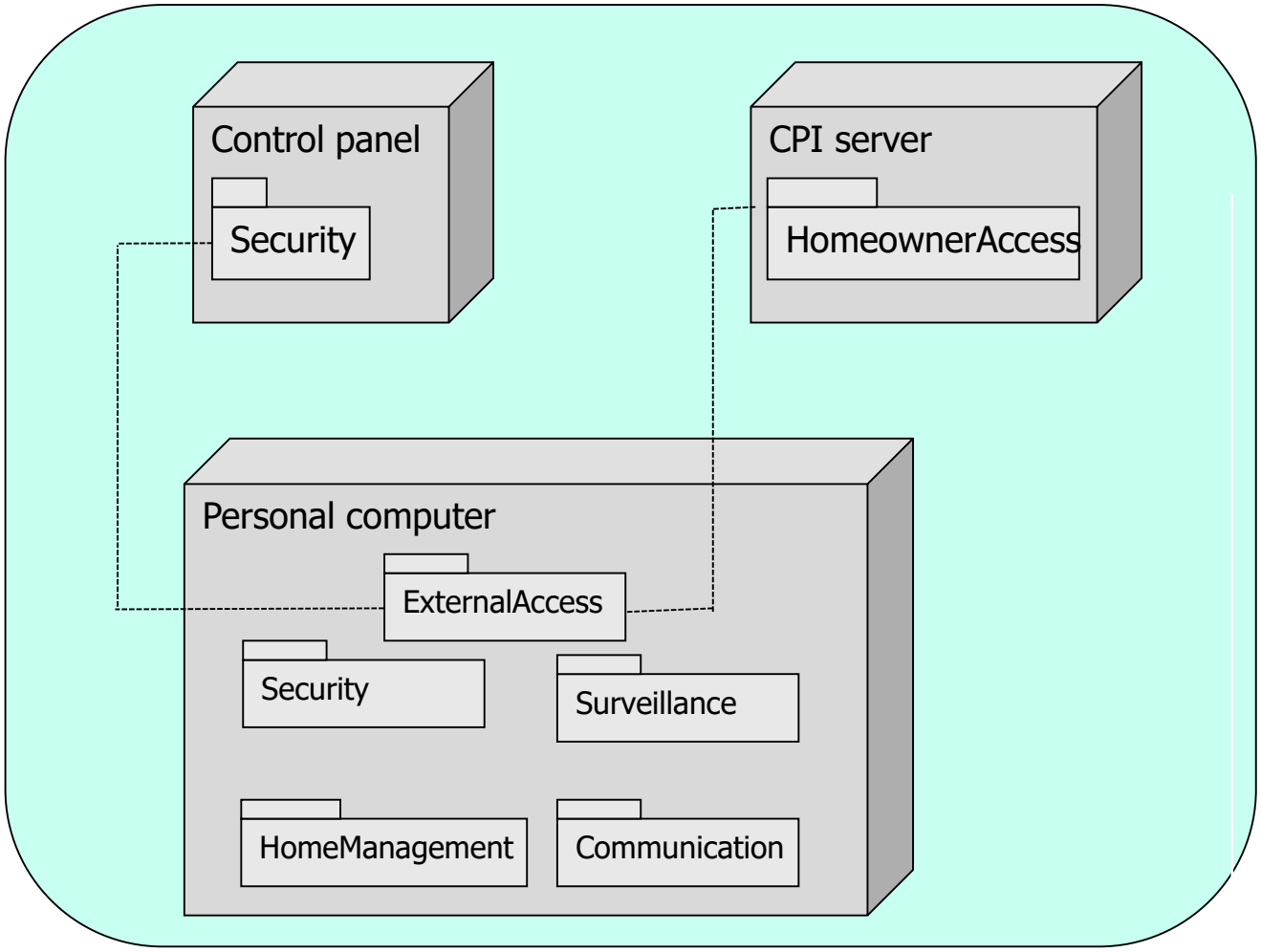
4. 설계 모델

◆5. 배치 수준 설계 요소

- 배치 수준 설계
  - 소프트웨어 **기능성과 하부시스템**
  - 소프트웨어를 지원하는 **물리적인 전산 환경** 안에서 어떻게 할당하는지 나타냄
  - UML 배치다이어그램
    - **서술자 형식** : **전산환경** 은 보여주지만 ,
      구성 내용은 명백하게 나타내지 않음
    - **인스턴스 형식** : 설계 최종 단계에서
      특정한 명칭의 하드웨어 구성 식별 .

4. 설계 모델

◆5. 배치 수준 설계 요소



정리 및 Homework

1) 설계의 목표
2) 설계 완성에 필요한 설계모델의 종류및 특징
3) 분석 모델에서 설계 모델의 변화 관계
4) 좋은 설계 평가 와 품질 가이드라인
5) 설계 작업 프로세스
6) 설계 개념과 특징

Project

1장. 프로젝트 개요
- 1.1 프로젝트 제목
- 1.2 선정 이유
- 1.3 팀 운영 방법

2장 시스템 정의
- 2.1 시스템 간략한 설명
- 2.2 유사 사례 간략한 설명

3장 프로세스 모델
- 3.1 규범적인 프로세스 모델 선정 및 이유
- 3.2 특수한 프로세스 모델 선정 및 이유

4장. 실무 가이드 원칙
- 4.1 각 프레임워크 원칙에서 중요한 3 개 정의
- 4.2 프로젝트 계획 보고서

5장. 요구사항 획득
- 5.1 기능 요구사항과 비기능 요구사항 정의
- 5.2 표준 양식을 사용한 시스템 요구사항 명세 3개 작성
- 5.3 정형적인 형식에 따른 유스케이스 작성

Chapter 6. 시스템 설계

6.1 설계 개념의 중요한 개념을 적용

(14개 중에서 : 개인별로 6개씩 선정하여 => 팀별 6개 결정)

6.2 설계 모델에 따른 요소별 설계

(4개 요소 간략하게 설계: 단, 아키텍쳐 요소는 제외)