



6장. 소프트웨어 설계



Contents

1. 소프트웨어 설계의 개념
2. 설계 원리
3. 결합력의 이해
4. 응집력의 이해
5. 구조적 설계와 모듈화
6. 모듈화와 기술부채

분석에서 설계로

- 요구 분석 : '무엇을 만들 것인가'를 다루는 작업
- 설계 : '어떻게 실현할 것인가'를 구체적으로 결정하는 활동
 - 기본 구조 설계 : 아키텍처 설계로 각 모듈의 역할과 인터페이스를 정의
 - 상세설계 : 모듈 내부의 알고리즘과 데이터를 명세화

거시적 방법론
 객체지향 - 아키텍처 설계 O
 내부 구조는 클래스에서 → 상세설계 X

1. 설계의 개념 (1/3)

● 설계

- 높은 수준의 의사 결정 과정의 연속
- 설계 원리가 중요

● 전통적 설계 방법

- 분할 정복, 추상화, 합성 등의 원리를 적용

● 최근의 방법 : 아키텍처 기반의 설계 방법

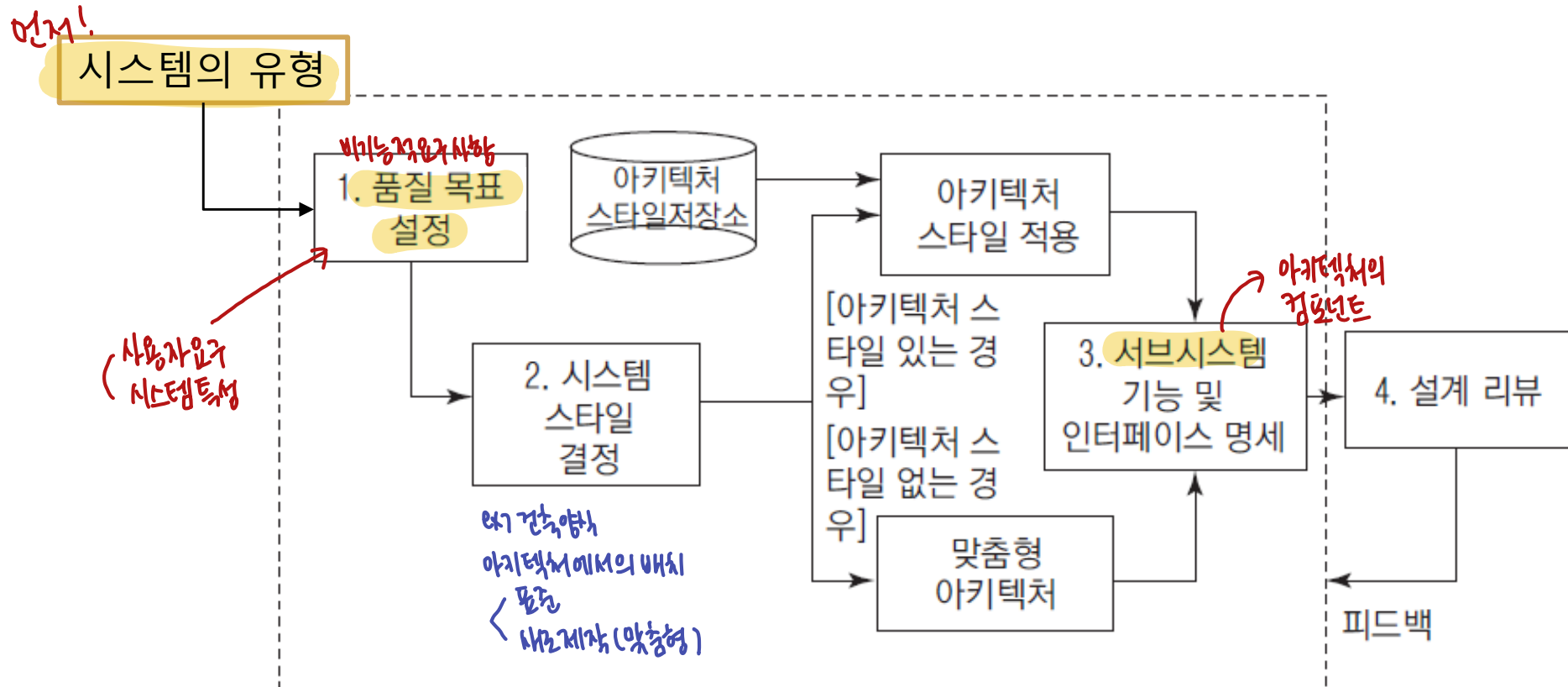
● 소프트웨어 아키텍처

- 시스템을 구성하는 컴포넌트(구성요소)와 컴포넌트 상호작용의 집합

성능 & 안정성
↓
비기능적 요구사항은
문득만족시켜야 됨
→ 우선순위 필요

1. 설계의 개념 (2/3)

- 설계 작업 과정 : 의사결정 과정이면서 동시에 시스템을 알아가는 과정

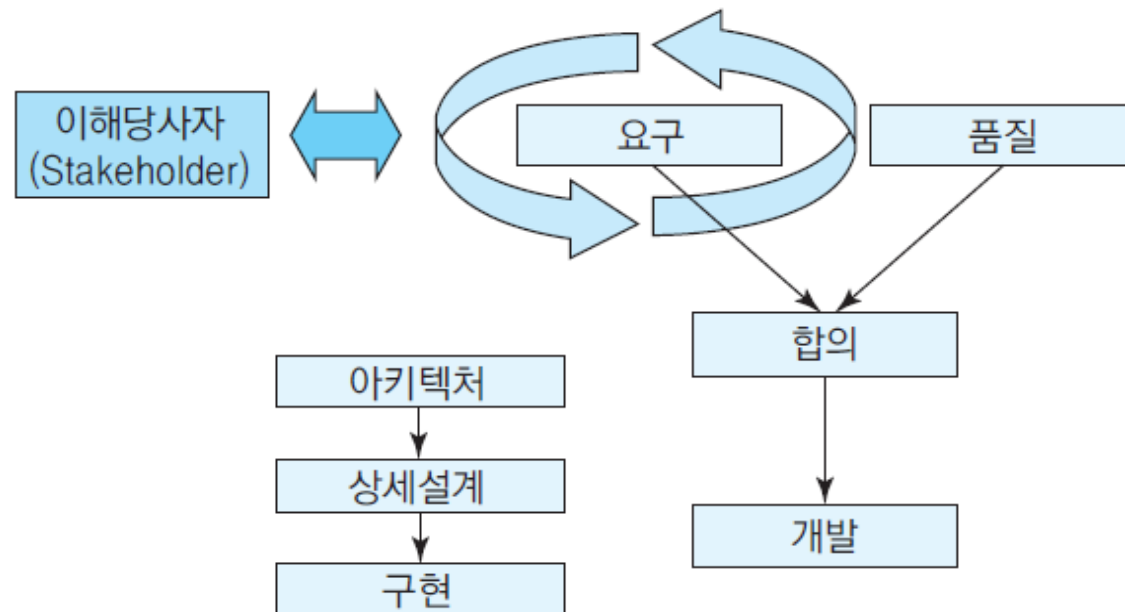


1. 설계의 개념 (3/3)

→ 비기능적 요구사항 (품질목표)

- 품질 제약사항은 설계에 대한 목표가 될 수 있음
 - 비기능적인 요구를 설계 목표로 구체적으로 명시
 - 이를 만족시키기 위하여 설계안을 만들고 그 중에 최적 안을 골라내는 작업

절대적인 방법이
없음.

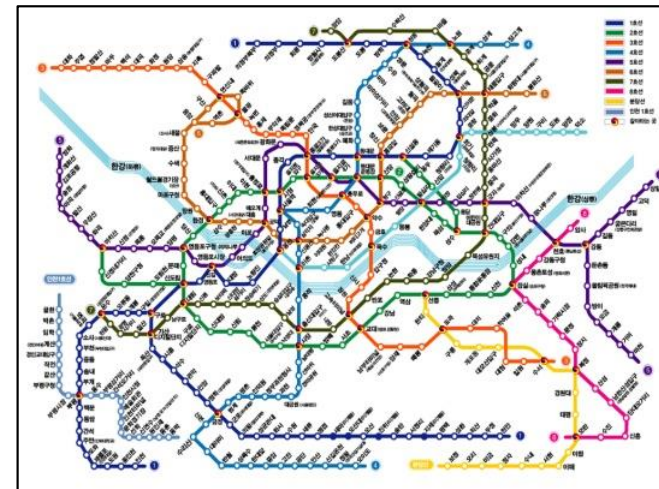
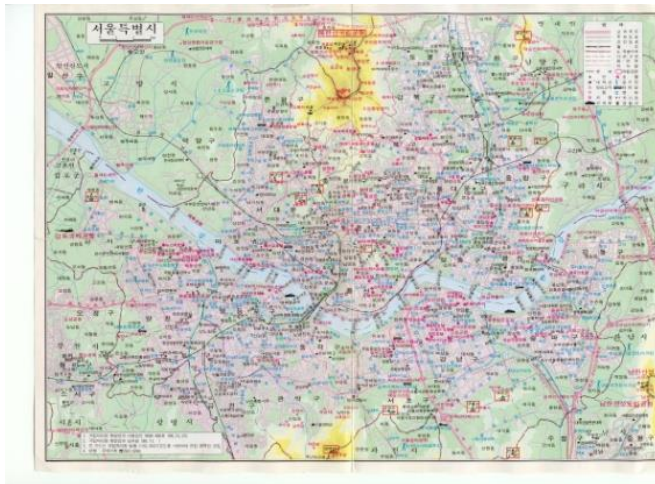


2. 설계 원리

- 중요한 SW 설계의 특성: **추상성** & **단계성**
- 소프트웨어 설계의 중심이 되는 원리
 - 추상화(Abstraction)
 - 정보은닉(Information hiding)
 - 단계적 분해(Stepwise refinement)
 - 모듈화(Modularization)

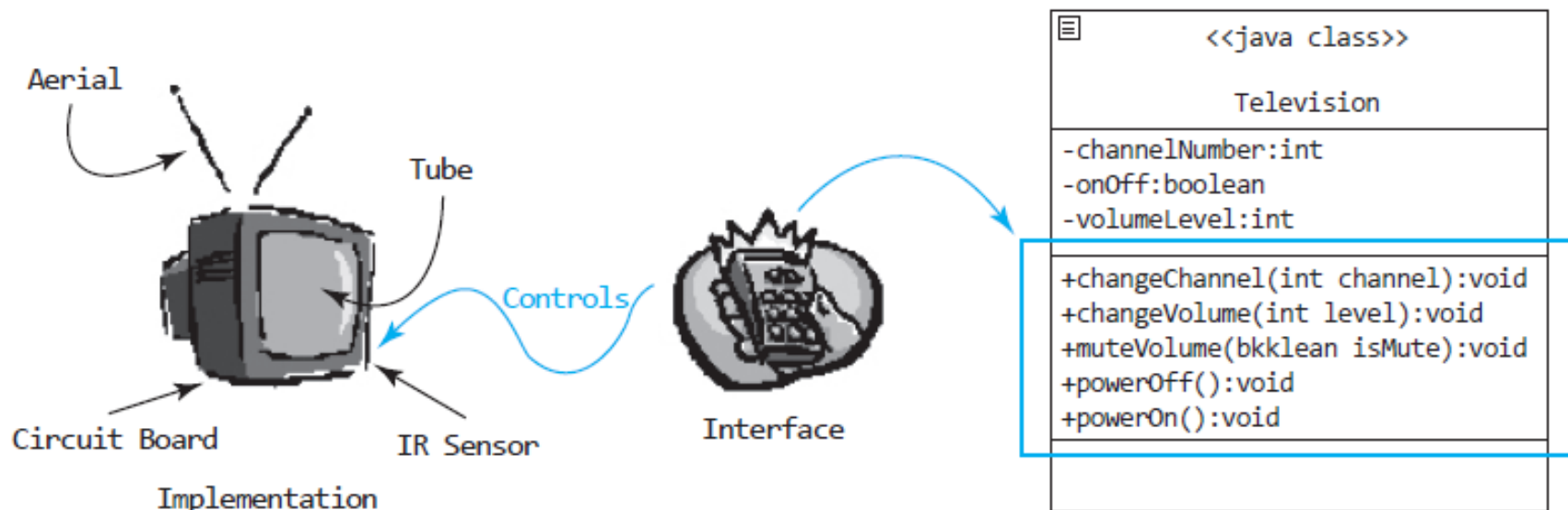
2.1 추상화(abstraction)

- 대상의 특정한 목적에 관련된 정보에 집중하고 나머지 정보는 무시하는 관점
- SW는 데이터나 절차적인 동작 관점으로 정의



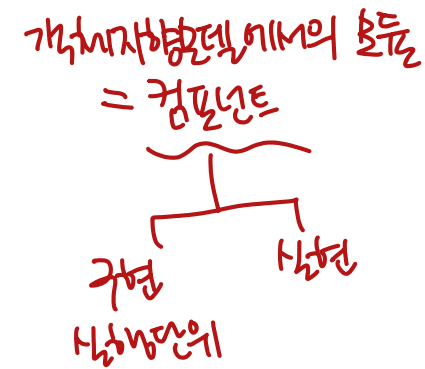
2.2 정보은닉

- 캡슐화 : 추상화된 대상이 제공하는 서비스를 쉽게 접근하게 하는 개념
- 캡슐화를 통해 정보은닉이 가능해짐
- 정보은닉은 좀 더 높은 수준의 추상화를 실현



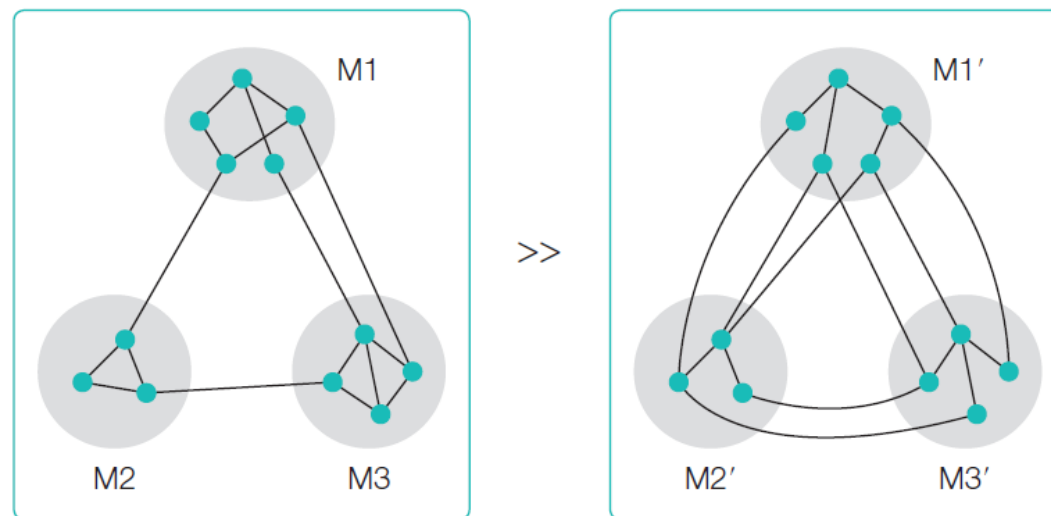
2.3 모듈화 (1/4)

- 모듈(module) :
 - 독립적인 기능이 있는 논리적 묶음 → 코드단위에서 사용X
 - 소프트웨어 구조를 이루는 기본적인 블록
- 모듈화 : 모듈로 소프트웨어 시스템을 구성할 수 있도록 개발한다는 의미
 - 이해하기 쉬움
 - 팀 단위의 개발 작업이 쉬워짐
 - 변경에 의한 수정 사항 반영이 쉬움
 - 재사용 가능성이 높아짐
 - 추적성이 높아짐



2.3 모듈화 (2/4)

- 소프트웨어 구조 : 모듈 및 모듈 간의 관계를 나타냄
- 좋은 모듈화 구조를 갖는 소프트웨어 설계
 - 모듈을 구성하는 내적 요소가 상호 관련성이 있는 것들로 묶여야 함
 - 모듈 간의 상호작용이 가능하면 간단한 형태로 구성되어야 함



2.3 모듈화 (3/4)

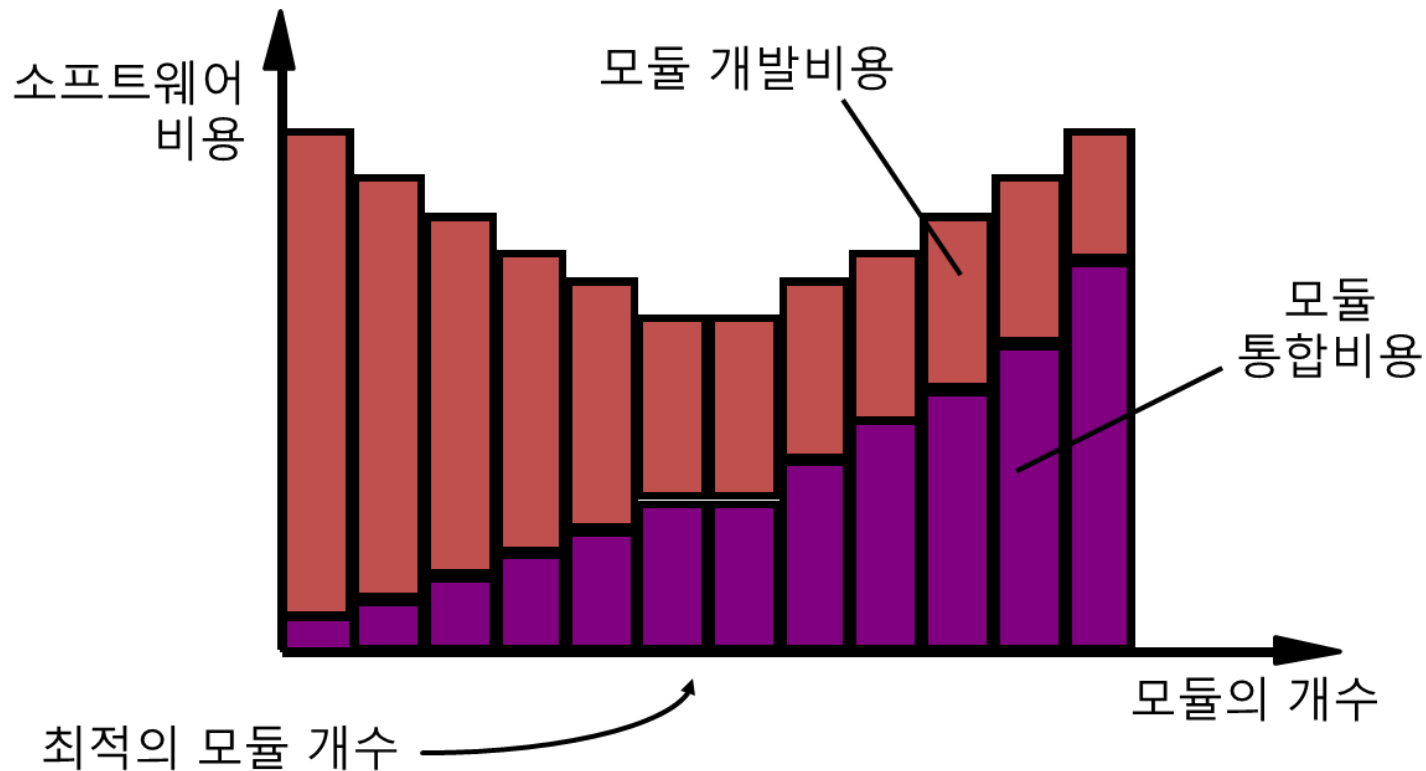
(비용)

모듈이 크면 → 자체개발비 ↑ 통합 ↓

작으면 → 자체개발비 ↓ 통합 ↑

⇒ 최적화! 적당하게

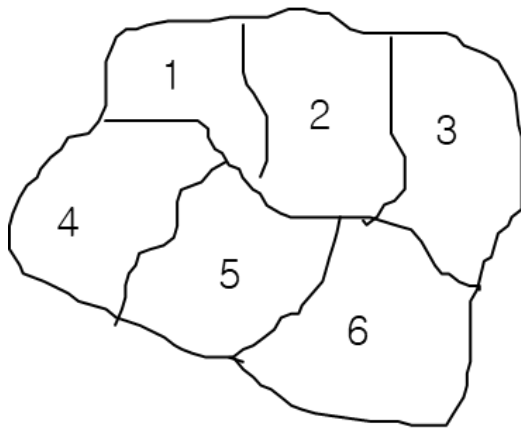
- 모듈화 \approx 시스템의 분해를 어떻게 할 것인가?



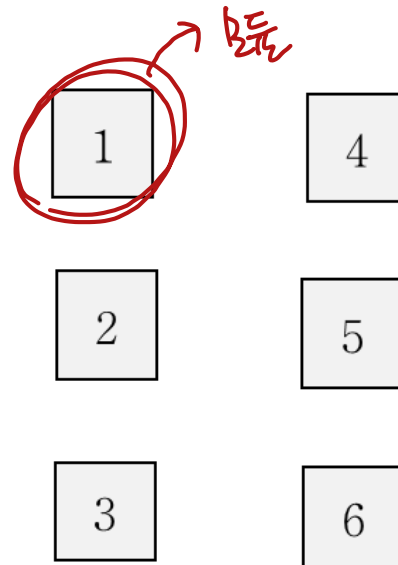
2.3 모듈화 (4/4)

● 모듈화의 기준

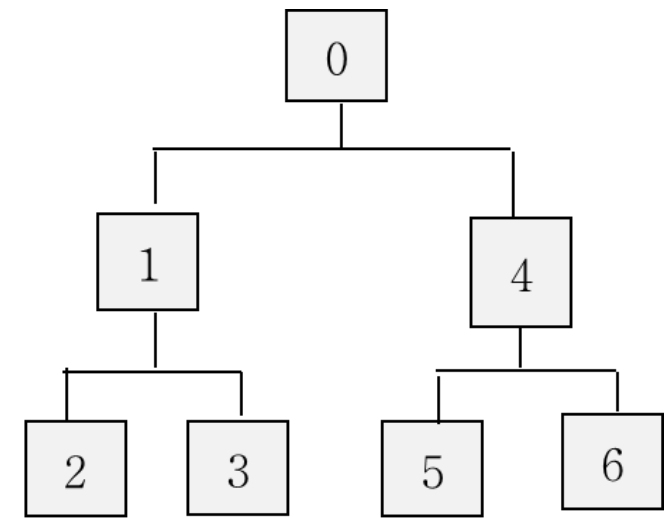
- 모듈의 응집력(cohesion)이 높게
- 모듈 간의 결합력(coupling)은 약하게



문제영역



시스템 분해



시스템 구조

아키텍처 스타일에 따라 11/6

3. 모듈간의 결합 (1/2)

↪ 서로 의존적이다

● 결합도(coupling)

- 모듈 간의 상호 의존하는 정도
- 모듈이 독립적인 기능을 갖도록 설계, 불필요한 중복 제거 필요 ⇒ 독립적인 모듈이 되기 위해서는 다른 모듈과의 결합도가 낮고 의존하는 모듈이 적어야 함

● 모듈 간의 결합력을 최소화 할 때의 장점

- 변경에 의한 파급 효과를 막을 수 있음
- 소프트웨어에 대한 이해도를 높임 한 모델을 이해하려면 다른 모델도 봐야 하니까
- 모듈의 인터페이스가 단순해지고 재사용성이 좋아짐

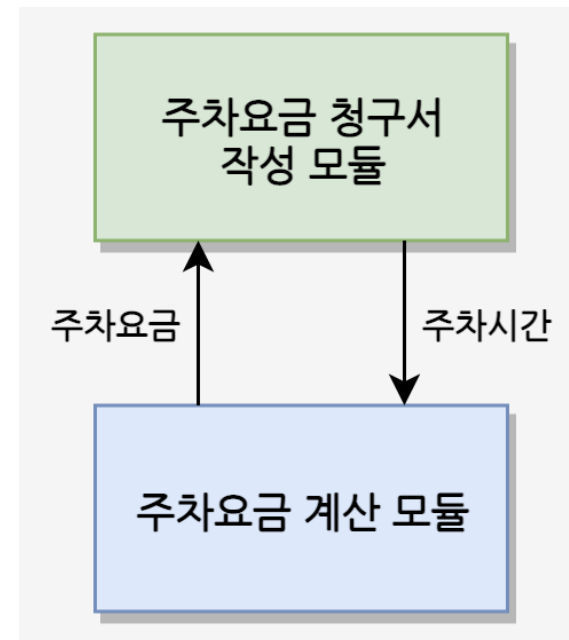
3. 모듈간의 결합 (2/2)

- 결합의 단계



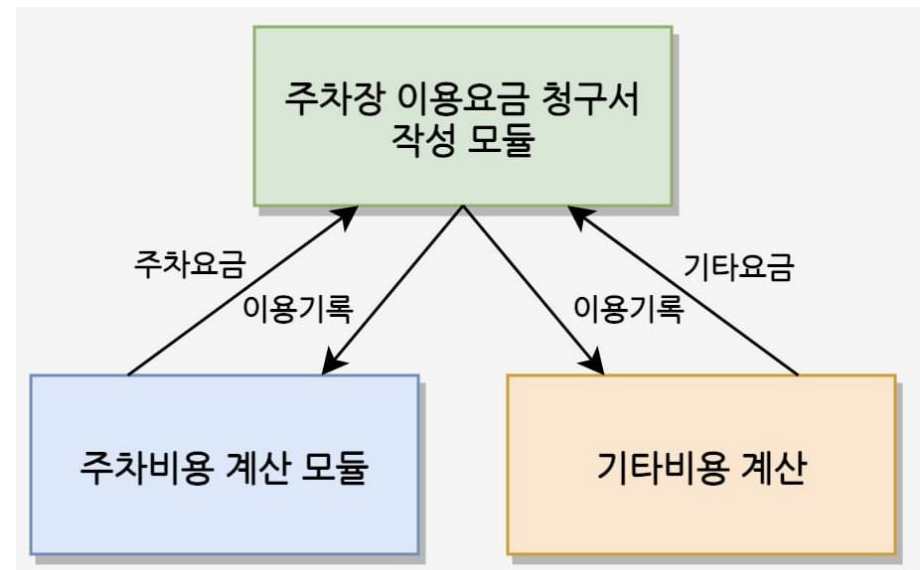
3.1 자료 결합도 (Data coupling)

- 모듈들이 정수형, 문자형 등의 단순한 기본 데이터 타입을 갖는 매개변수(parameters)에 의해 상호작용하는 경우
- 주고 받는 데이터는 모듈의 로직을 제어하지 않는 순수한 자료형 요소
- 한 모듈을 변경해도 다른 모듈에는 영향을 끼치지 않는 결합 상태 (값만 전달함)



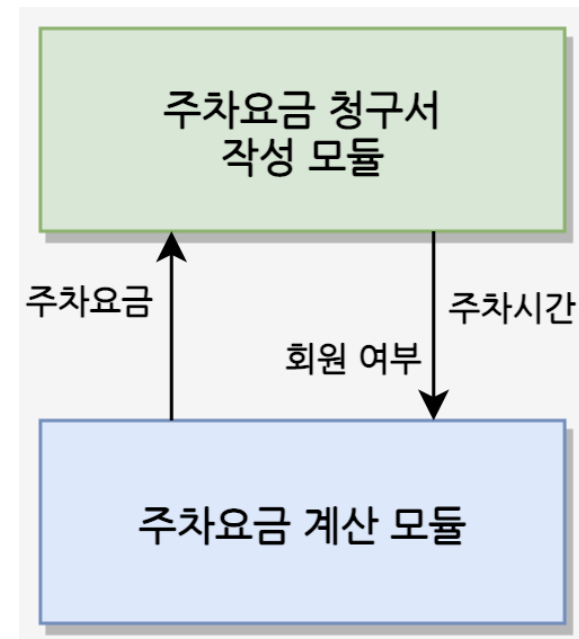
3.2 스탬프 결합도(Stamp coupling)

- 모듈들이 구조체와 같은 non-global 자료구조 (복합 데이터)를 이용하여 상호작용
- 두 모듈이 동일한 자료 구조를 참조하는 형태
- 자료구조의 형태가 변경되면, 그것을 참조하는 모든 모듈에 영향을 주며, 변경되는 필드를 실제로 참조하지 않는 모듈에도 영향을 줄 수 있음



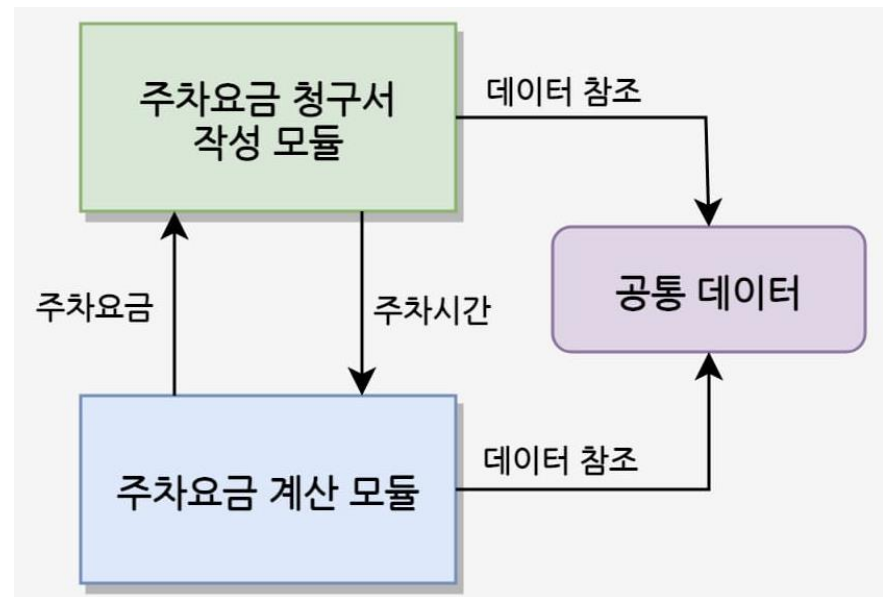
3.3 제어 결합도 (Control coupling)

- 어떤 모듈이 다른 모듈 내부의 논리적인 흐름을 제어하는 요소를 전달하는 경우 (강)
- 파라미터로 전달되는 값에 따라서 모듈 내부 로직의 처리가 달라지는 Flag 값 등으로 결합되는 형태
- 예) 회원 여부에 따라 주차요금 계산 로직이 다른 경우



3.4 공통결합도(Common coupling)

- 여러 모듈이 하나의 데이터 영역(전역 변수, global variable)을 참조하여 사용하는 형태
 - 전역 변수의 변경이 여러 모듈에 영향을 끼치게 됨
 - 모듈의 독립성을 보장하기 위해 꼭 필요한 변수만 광역 변수로 선언해야 함



3.5 내용결합도(Content coupling)

- 어떤 모듈이 사용하려는 다른 모듈의 내부 기능과 데이터를 직접 참조하는 경우
 - 다른 모듈의 로컬 데이터에 접근하는 경우처럼 사용하고자 하는 모듈의 내용(코드)을 알아야 함
 - 모듈이 변경이 발생하는 경우 이를 참조하는 모듈의 변경이 반드시 필요하게 됨 *→ 모듈을 분리하는 의미가 없음. 하나로 합쳐놓게 나옴*
 - 내용 결합력은 현재 거의 발생하지 않음. 기존의 절차적 언어에서 “goto” 명령어가 내용 결합을 유발하는 대표적인 예

4. 응집도 (1/2)

- 응집도(Cohesion)
 - 모듈을 구성하는 내적 요소 간의 기능적 관련성의 강도를 측정하는 척도 모듈내의문제
 - 모듈을 구성하는 내적 요소 : 명령어, 변수 정의, 함수 호출 등
- 소프트웨어 설계 과정 : 독립적인 기능을 갖춘 모듈을 식별하는 것이 중요.
- 모듈의 응집력을 최대화하는 것을 목표로 함
- 한 모듈이 정확히 정의되는 단일의 기능을 갖도록 설계

4. 응집도 (2/2)

- 응집의 단계



4.1 기능적 응집도

- Functional Cohesion

- 모듈을 구성하는 모든 요소가 잘 정의된 하나의 기능을 구현하기 위해 구성된 경우
- 다른 추가 연산이나 기능 수행 없이 하나의 기능이 종료될 수 있음

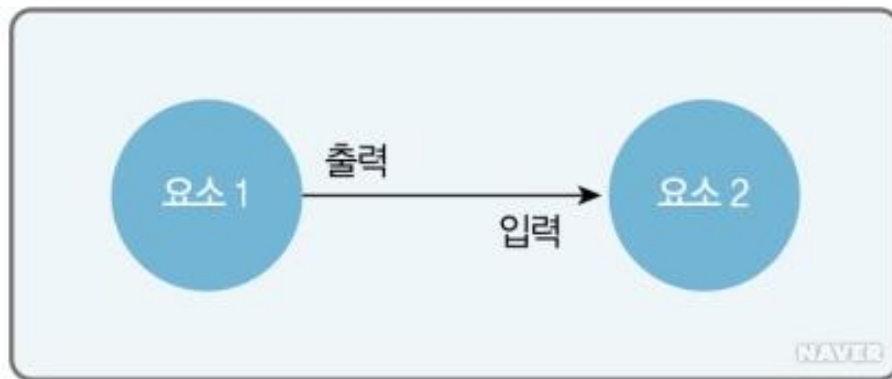
- 기능 응집력이 있는 모듈의 예

- 수학의 코사인각 계산
- 문장의 알파벳 오류를 체크
- 트랜잭션 기록을 읽기
- 미사일의 타격 지점 계산
- 승객의 좌석 배정

4.2 순차적 응집도

- Sequential Cohesion

- 모듈을 구성하는 요소들 사이에서 한 요소의 출력이 다른 요소의 입력으로 사용되는 형태
- 예: 어떤 모듈이 특정 파일을 읽고 처리

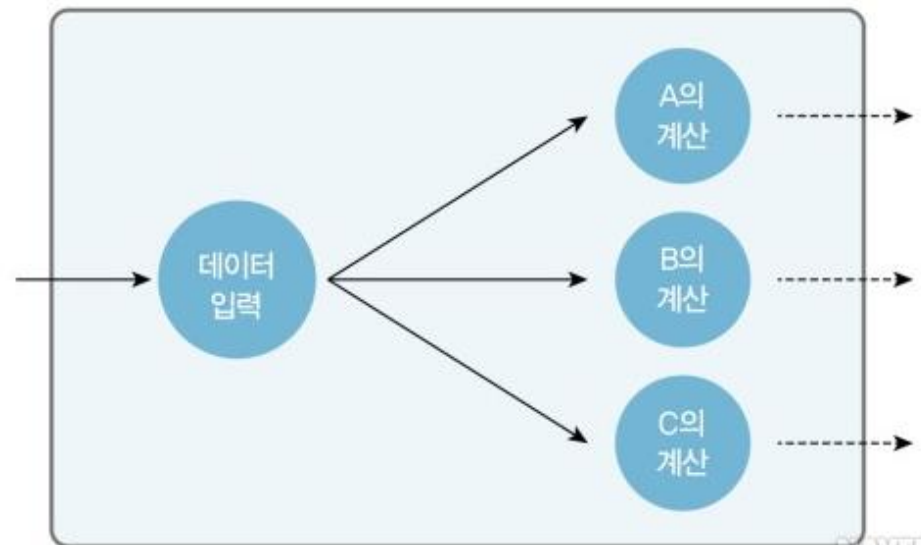


과목별 성적을 읽는다.
성적들의 합(총점)을 구한다.
총점을 이용하여 평균을 구한다.
석차를 부여한다.

4.3 교환적 응집도

● Communicational Cohesion

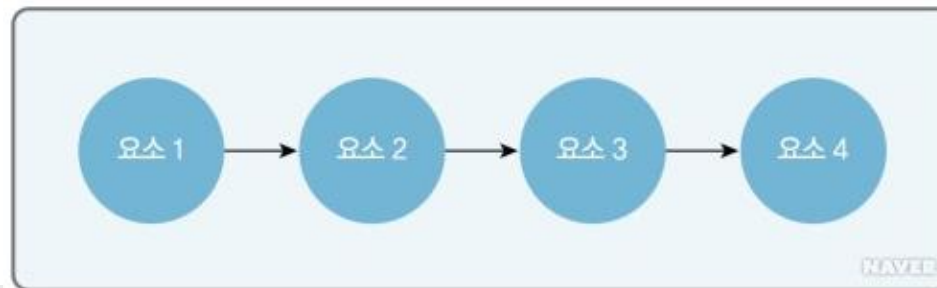
- 모듈을 구성하는 모든 요소가 동일한 입력 또는 출력을 사용하여 서로 다른 기능을 수행하는 경우
- 처리 순서는 중요하지 않기 때문에 순차적 응집도보다 묶인 이유가 약해 짐



4.4 절차적 응집도

● Procedural Cohesion

- 순서가 정해진 몇 개의 구성 요소를 하나의 모듈로 구성하는 경우
- 구성요소들 사이에 의미상 서로 관련은 없으나, 제어흐름의 순서가 있는 경우
- 구성요소의 출력과 입력이 연관되어 있는 순차적 응집도보다 묶인 이유가 훨씬 약해 짐
- 예: 파일을 읽을 때 접근 허가를 확인하고, 파일읽기



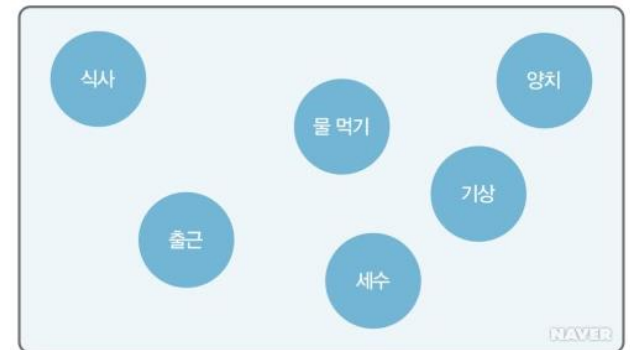
4.5 시간적 응집도

- Temporal Cohesion

- 같은 시간대 처리되어야 하는 활동들의 모듈로 구성
- 시간상으로 동일하게 진행되기 때문에 입출력이 서로 영향을 주는 것은 아님

- 시간 응집력을 발생시키는 예

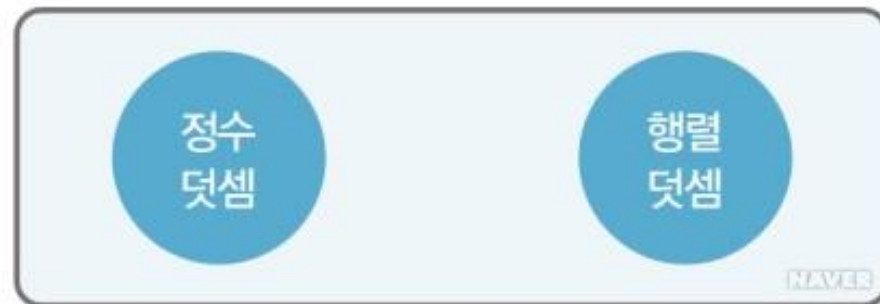
- 데이터를 사용하기 전에 모든 변수를 처음에 초기화
- 예외상황이 발생했을 때 오류 로그를 전송하는 기능
- 사용자에게 알림 정보를 제공하기 위한 모든 메시지 처리



4.6 논리적 응집도

● Logical Cohesion

- 유사한 성격을 갖거나 특정 형태로 분류되는 요소들을 하나의 모듈로 묶은 경우
- 순서와 무관하며, 서로 입력과 출력 간의 상호의존도 없는 경우가 많고, 단지 기능적으로 비슷해 보이는 요소 간의 묶음
- 논리적으로 비슷한 기능을 수행하지만 서로의 관계는 밀접하지 않은 형태



4.7 유연적 응집도

● Coincidental Cohesion

- 가장 나쁜 형태로, 절대 작성되어서는 안되는 모듈 구성
- 모듈을 구성하는 모든 요소가 아무런 관련성이 없는 것으로 묶인 경우
- 유사한 성격이나 형태가 없으며, 모듈 수정이 side effect를 유발시킬 가능성이 있음



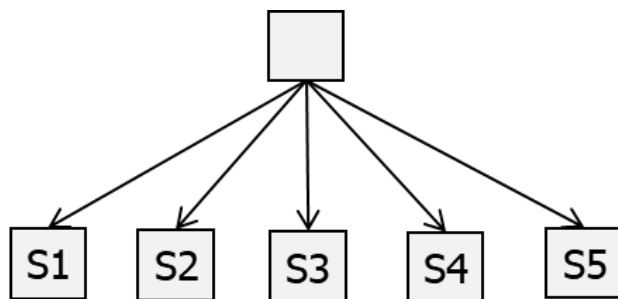
5. 구조적 설계 (1/2)

구조적 분석 방법론

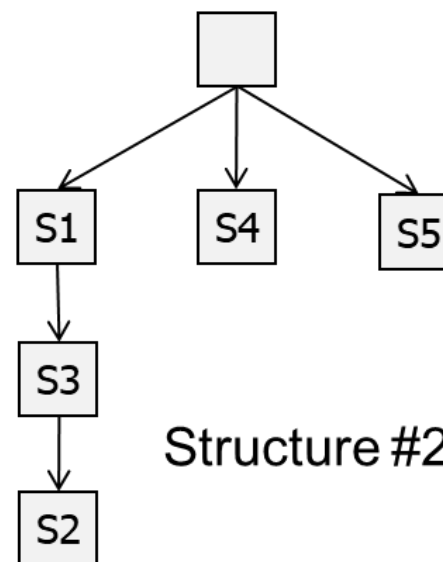
- 시스템을 이루는 모듈의 구조를 파악하는 방법
 - 설계 결과는 구조도(structure chart)로 나타냄
 - DFD를 이용하여 프로그램구조(call-and-return구조)를 도출
- 구조적 설계기법은 데이터의 흐름 형식에 중점
 - 변환흐름(Transform flow)에 대한 변환 분석
 - 처리흐름(Transaction flow)에 대한 처리 분석

5. 구조적 설계 (2/2)

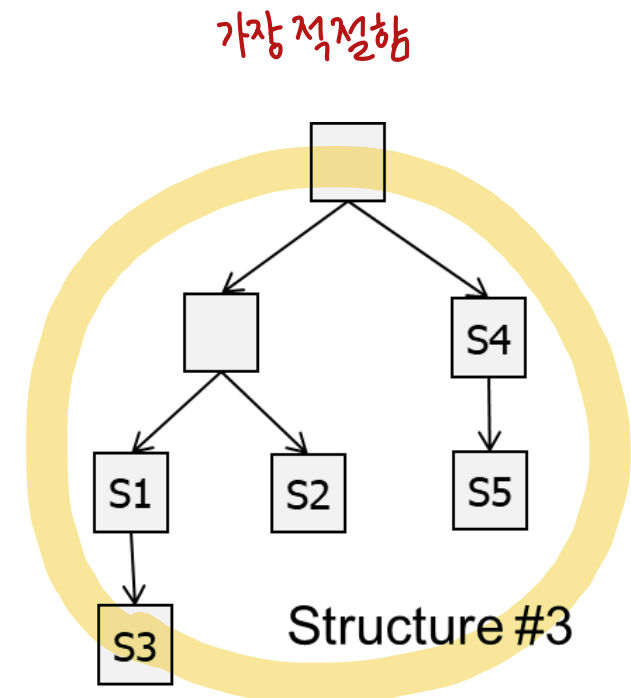
- 시스템 구조도(structure chart)의 도출
 - 시스템을 모듈 단위로 분할
 - 모듈의 계층적 구성
 - 모듈 사이의 입출력 인터페이스
 - 모듈의 이름과 기능



Structure #1







Structure #2



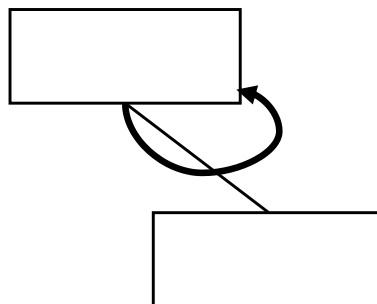
Structure #3

5.1 시스템 구조도 (1/2)

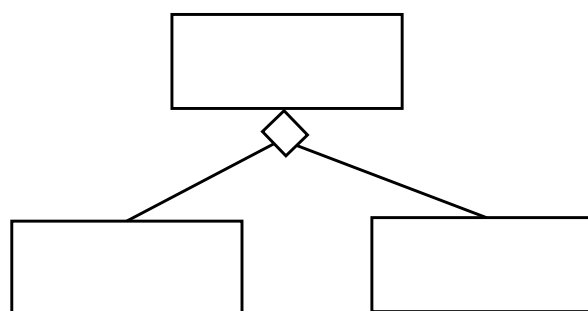
표준 기호

	한 모듈이 모듈을 호출
	자료 흐름 (변수나 자료구조)
	제어 흐름 (플래그)
	모듈

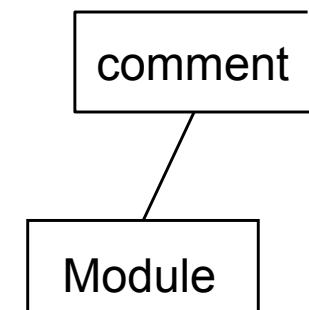
반 복



선 택



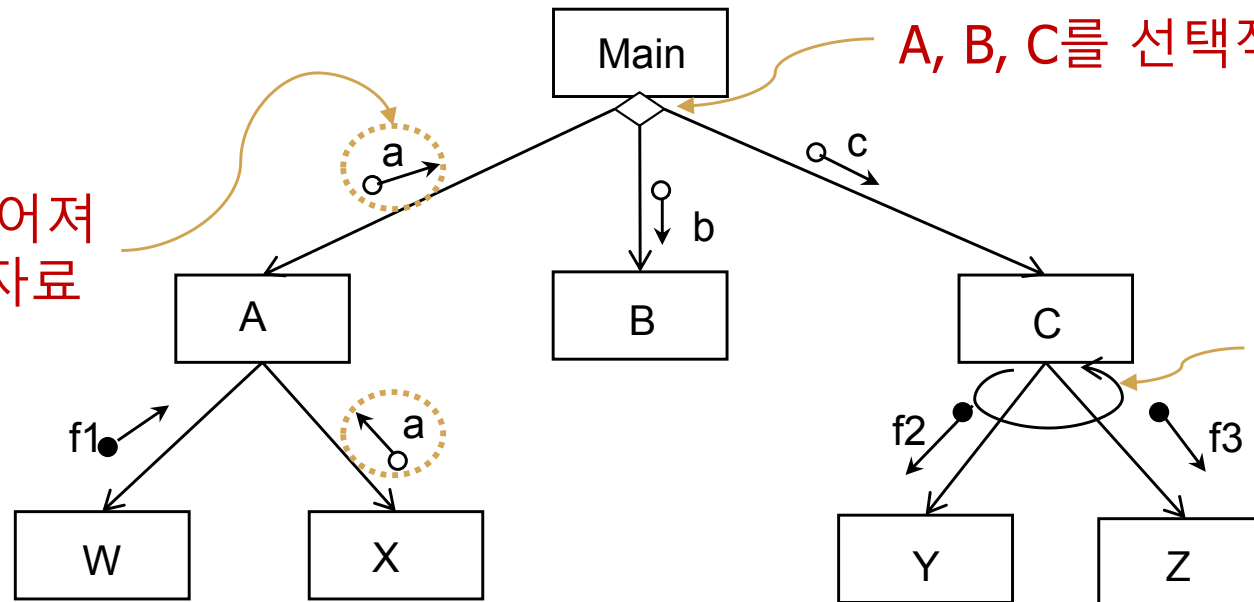
주석달기



5.1 시스템 구조도 (2/2)

예

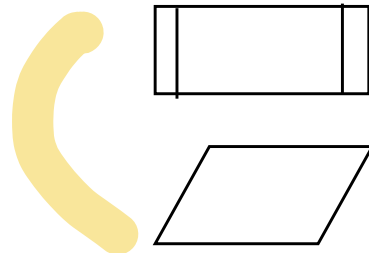
X에서 만들어져
전달되는 자료



A, B, C를 선택적으로 호출

Y, Z를 반복적으로
호출

기타 사용되는 기호

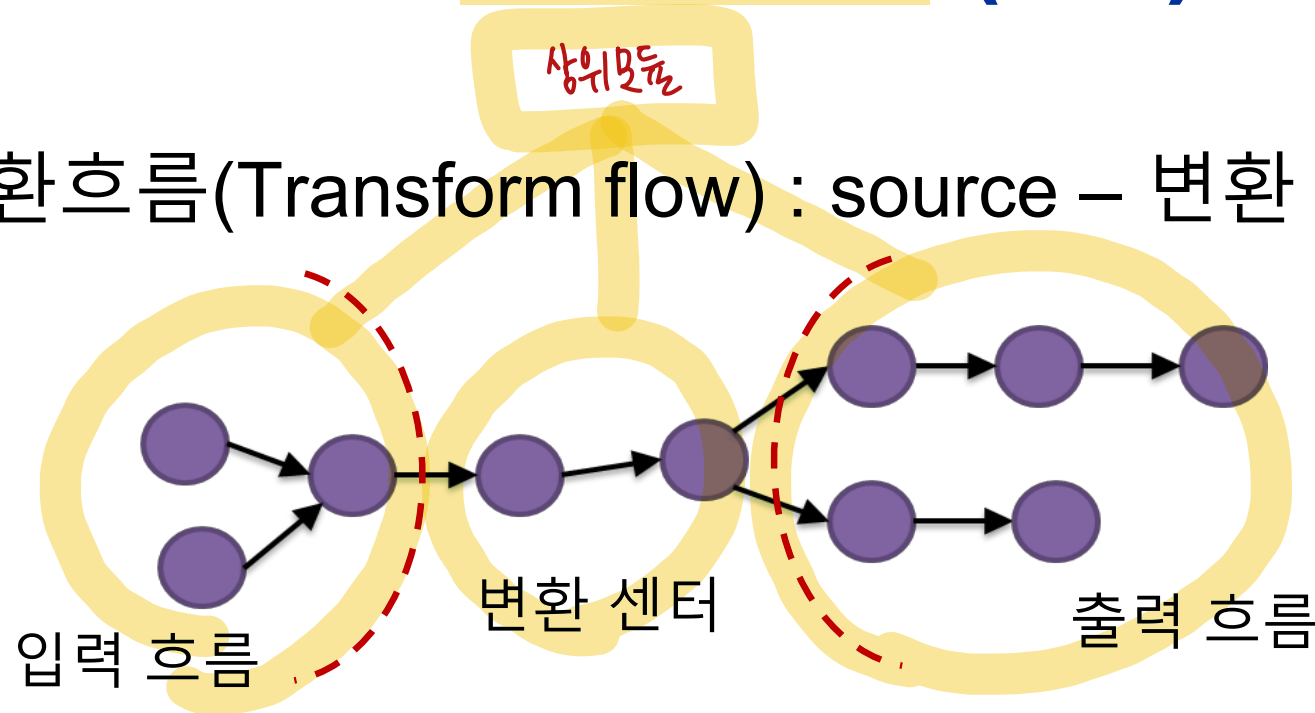


미리 정의된 모듈(라이브러리)

입출력 모듈

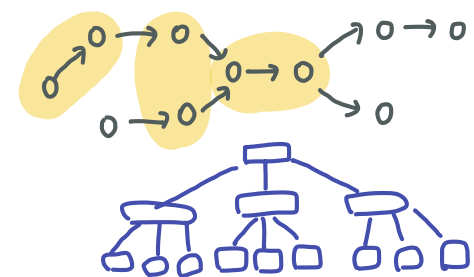
5.2 변환분석 (1/5)

- 변환흐름(Transform flow) : source – 변환 – sink



- 변환 분석은 DFD를 입력 흐름, 변환 센터, 출력 흐름으로 분할하는 과정
 - 입력 흐름: 입력을 준비하는 단계(입력, 검증...)
 - 변환 센터: 실제 자료가 변환
 - 출력 흐름: 변환된 자료가 출력되는 단계

5.2 변환분석 (2/5)



● 변환분석 방법

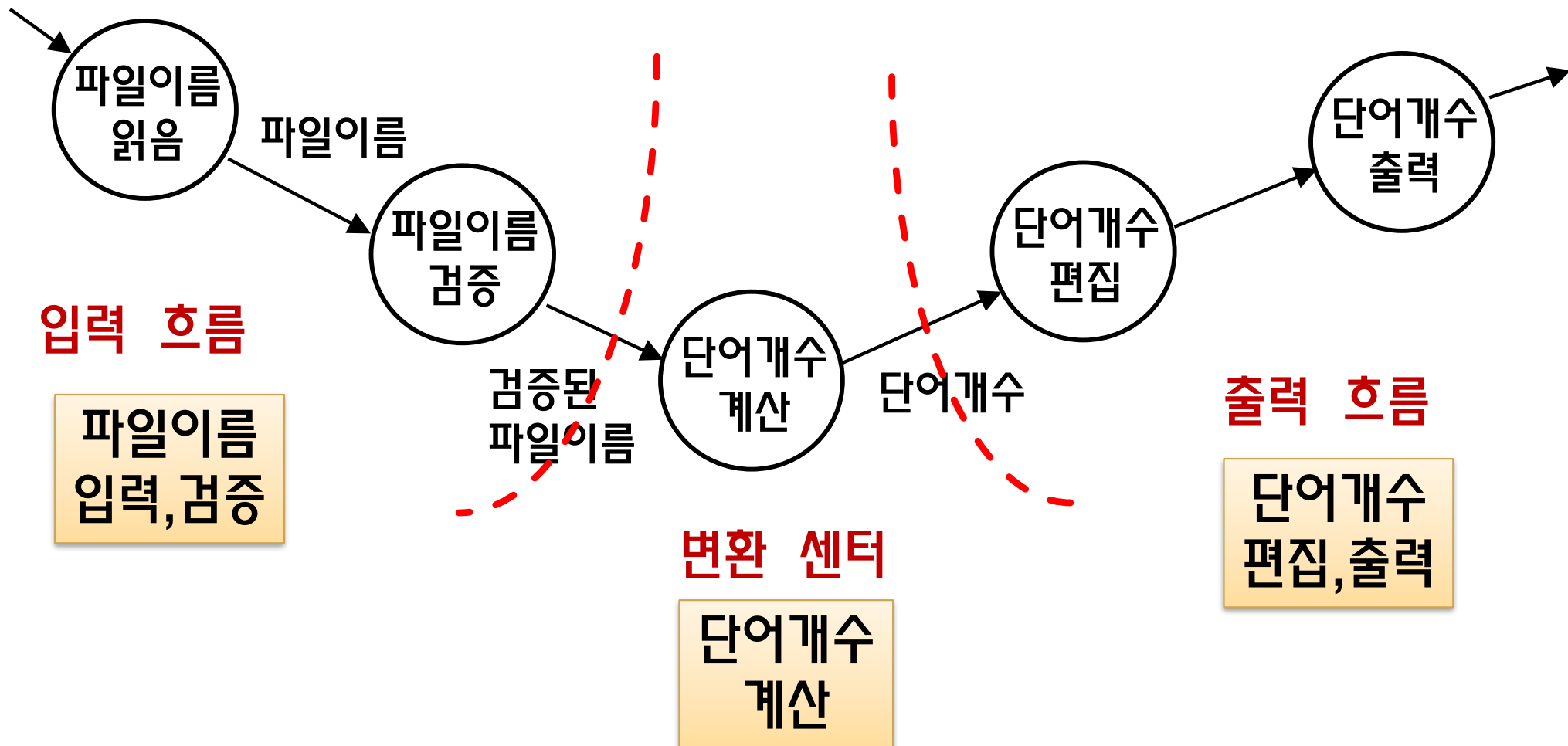
- DFD에서 입력 자료 흐름과 출력 자료 흐름을 파악
- 중앙 변환 부분을 식별
- 변환 중심부를 축으로 상위 구조(first-cut) 작성
- 각 모듈의 하위 구조도 같은 방법으로 분석 또는.가운데.싱크
- 설계 기준을 적용하여 수정, 최적화 (Fan-in/Fan-out)

깊이 (depth)	제어 단계의 수
넓이 (width)	제어의 전체 폭
팬-아웃 (fan-out)	한 모듈이 직접 불러 제어하는 하위 계층 모듈수
팬-인 (fan-in)	주어진 모듈을 직접 불러 제어하는 상위 조정 모듈수

프로그램 구조를 측정하고 표현하기 위한 용어

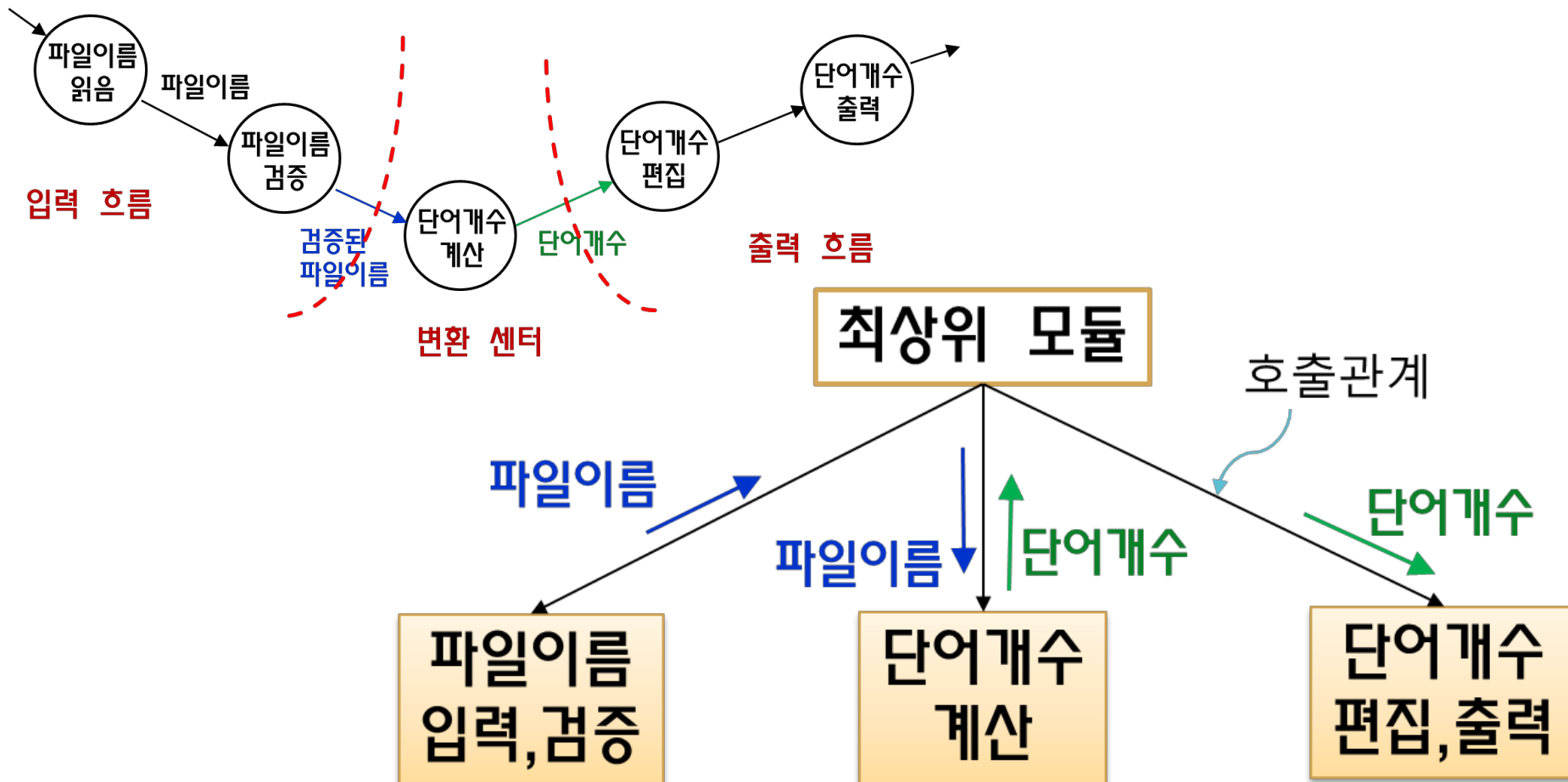
5.2 변환분석 (3/5)

- 예: 파일 안에 포함된 단어의 개수를 계산



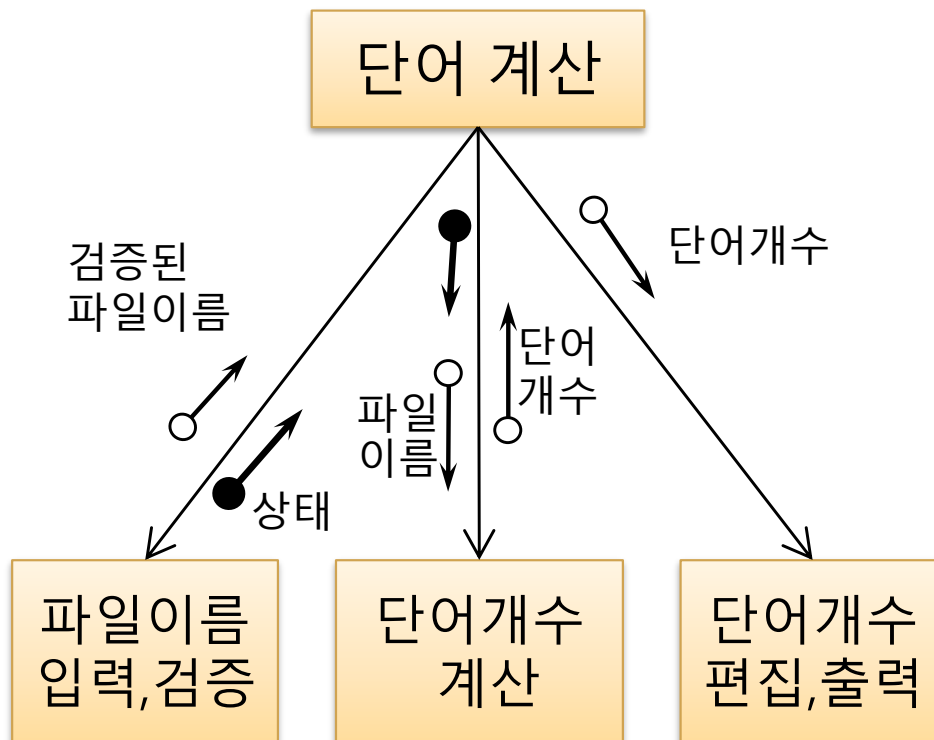
5.2 변환분석 (4/5)

- 변환 중심부를 축으로 상위 구조(first-cut) 작성



5.2 변환분석 (5/5)

시스템 구조도



```

main() {
    ...
    read_file(file_name, status);
    count_word(file_name, &word_count);
    display(word_count);
}

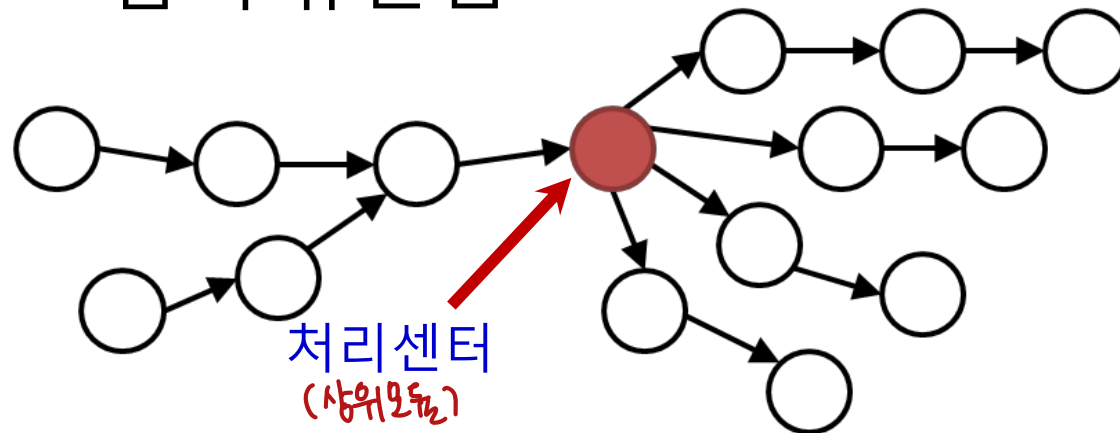
read_file(char* file_name, boolean status){
    ...
}

count_word(char* file_name, boolean status) {
    ...
}

display(int word_count) {
    ...
}
  
```

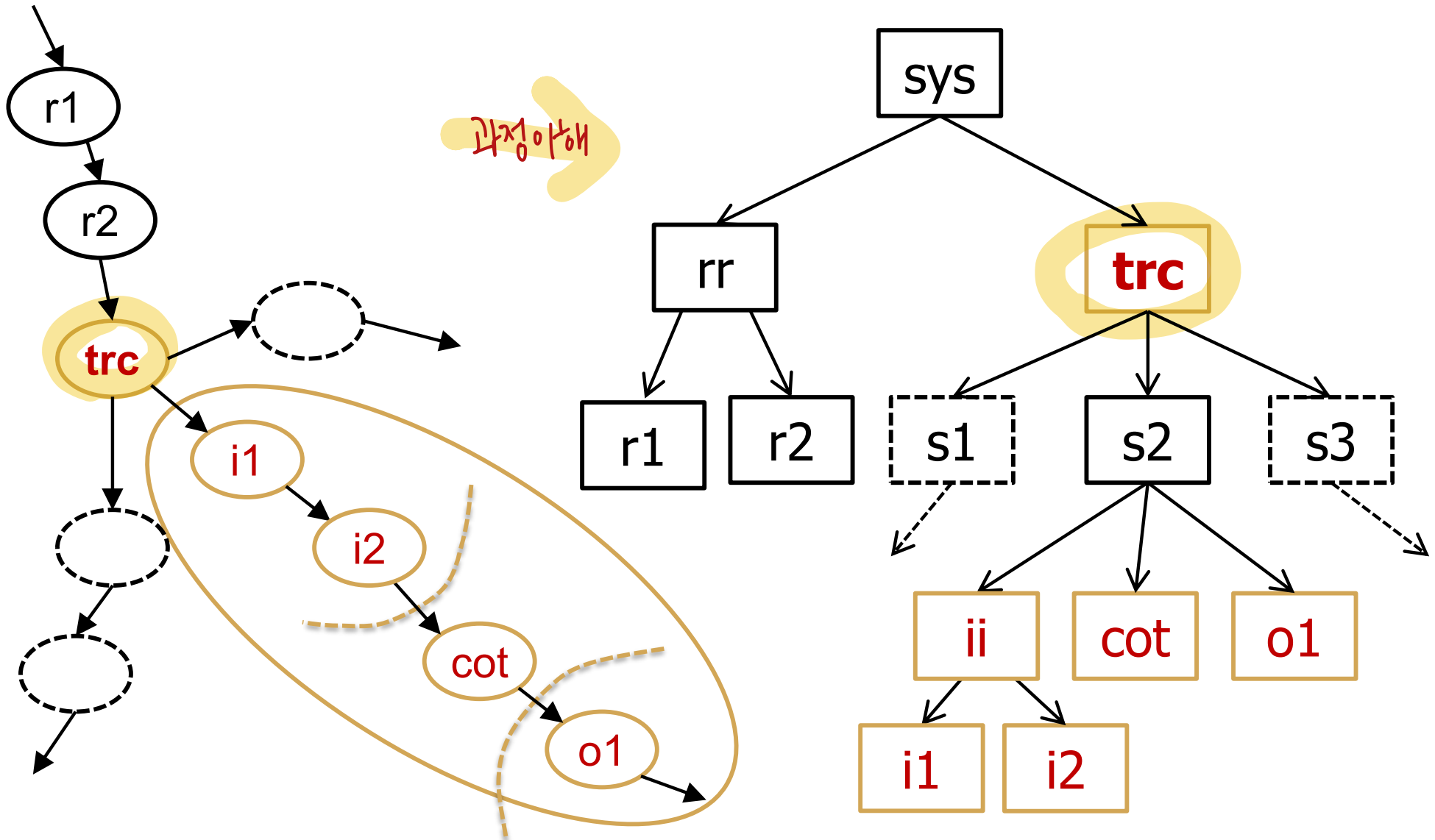
5.3 처리분석 (1/2)

- 처리흐름(Transaction flow): 한 프로세스에서 여러 개의 자료 흐름이 유출됨



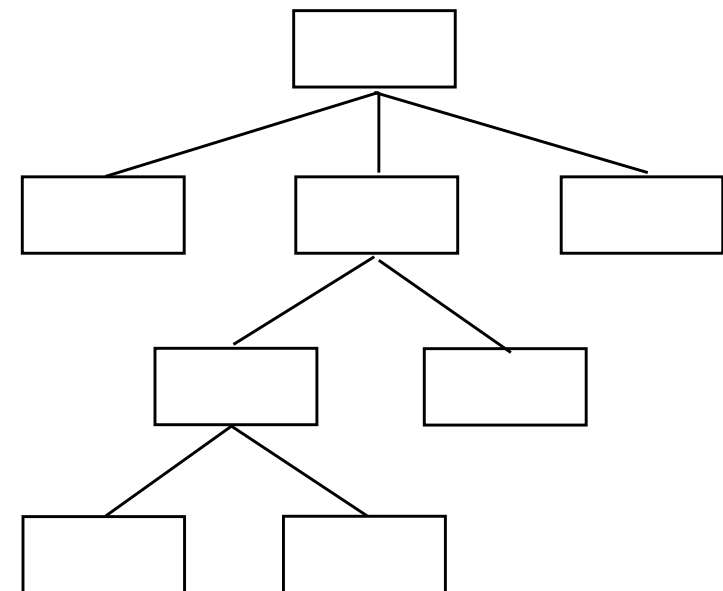
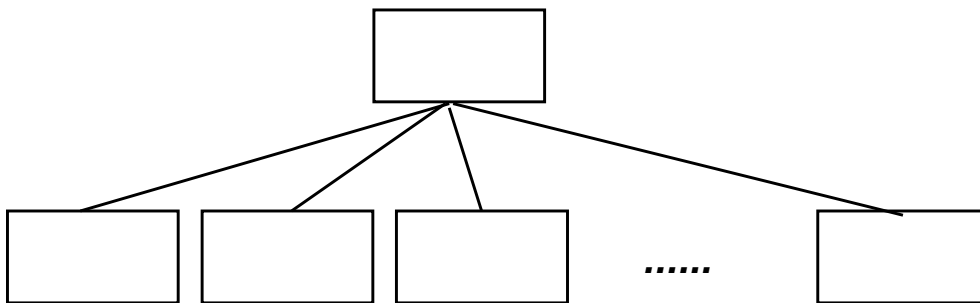
- 변환 방법
 - DFD에서 처리 센터를 식별
 - 처리 센터를 중심으로 구조도 작성
 - 구조도를 상세화 \Rightarrow 하위구조도를 작성

5.3 처리분석 (2/2)



5.4 구조적 설계 원칙(1/2)

- 모듈의 결합은 줄이고 응집은 높이도록 최대한 노력
- 전체적인 균형을 이루도록 함
 - 입력편중, 처리편중, 출력편중 되지 않도록
 - 구조도의 깊이 *한쪽으로 깊어지면 안됨*
 - fan-in, fan-out 고려

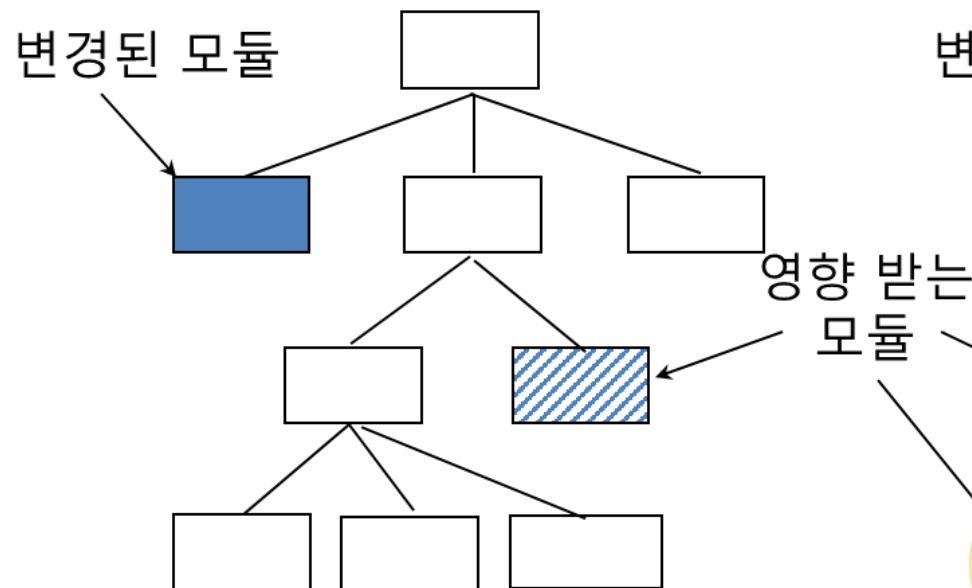


5.4 구조적 설계 원칙(2/2)

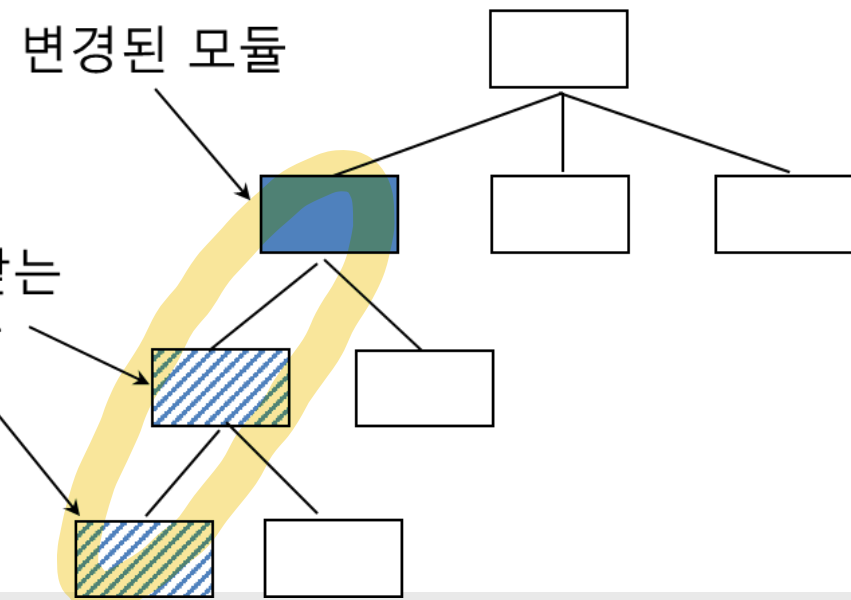
● 모듈의 배치 요령

- 복잡한 모듈의 연결은 피함
- 과다한 깊이를 가진 구조도 피함
- 모듈의 영향권은 그 모듈의 하위에 둔다

<잘못된 예>



<잘된 예>



6. 모듈화와 기술 부채

빛. 감지 안 돌아오는 것

- 기술부채(Technical debt)
 - SW를 재구조화하는 데 왜 예산이 필요한가를 설명하기 위한 은유적 표현
 - SW 시스템을 개발하는 과정에서 제대로 작업하지 않으면 추후 이자까지 붙은 빚이 되어 돌아오기 때문에 더 많은 작업을 통해 이전의 올바르게 못한 결과를 바로잡아야 한다는 의미
 - 예) 개발단계에서 문서화가 제대로 이루어지지 않은 경우 ⇒ 기능 개선이나 오류 수정이 필요한 경우, 분석/설계 문서가 없기 때문에, 문서가 있을 때보다 더 많은 시간과 노력이 필요함