소프트웨어 공학개론

Introduction to Software Engineering

2022-1학기 (Spring)
선문대학교 AI소프트웨어학과

5월 12일 (목)

강의 목차

9.1 모듈설계란?

선문대학교

개요

- 모듈설계
  - 소프트웨어를 모듈로 분할하고 구조화하는 작업

목차

- 9.1.1 모듈이란 무엇인가?
- 9.1.2 좋은구조와 나쁜구조

9.1 모듈설계란?

선문대학교

9.1.1 모듈이란 무엇인가?

- 모듈(Module)
  - 건물, 가구, 자동차, 전기기구, 컴퓨터하드웨어 등 다양한 사물들에 대한 각각의 구성단위를 말함
  - 보다 복잡한 구조를 구축하기 위해 이용가능한 독립적인 기능을 갖는 단위(Unit)
    - 일반적으로 교환 및 탈착이 가능한 구성부품
  - 모듈에 의해 복잡한 시스템을 보다 간결하고 명확한 요소들로 분할할 수 있음

- 여러 개의 모듈로 구성된 시스템 → 모듈러(Moduler) 또는 모듈화되어있다

9.1 모듈설계란?

9.1.1 모듈이란 무엇인가?

- 모듈화된 시스템의 장점 (1/2)
  - (1) 상세하게 파악해야되는 부분들이 분리되어 있으므로, 다른 모듈의 상세를 고려하지 않더라도 특정한 모듈의 상세만을 파악해두면, 해당 모듈의 개발을 시작할 수 있음
    - 개발대상 소프트웨어를 모듈이라는 단위로 추상화하여 파악할 수 있음을 의미
  - (2) 대규모 시스템이라고 하더라도, 모듈을 각각 별도로 설계개발하고, 나중에 모듈을 통합할 수 있음
    - 시스템개발을 수월하게 할 수 있음
    - 여러개의 모듈을 동시에 개발함으로써 개발기간을 단축할 수 있음
    - 그림 예: 모듈화된 시스템에서는 6개의 모듈을 병행적으로 개발할 수 있음

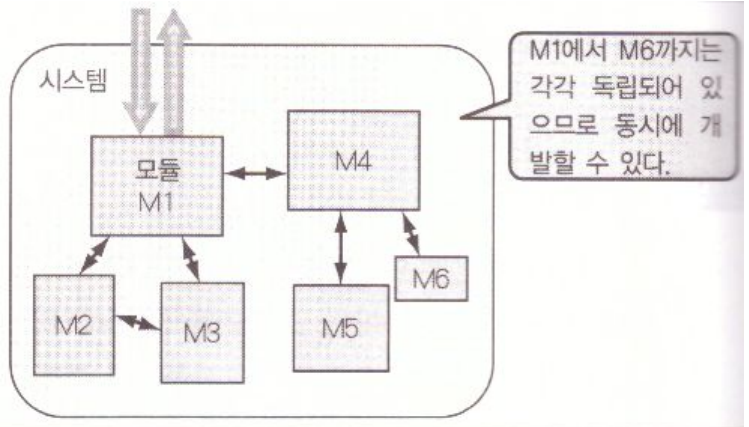

그림 9.1 모듈화된 시스템

9.1 모듈설계란?

9.1.1 모듈이란 무엇인가?

- 모듈화된 시스템의 장점 (2/2)
  - (3) 복잡한 시스템을 개발하는 경우, 기존의 모듈을 재이용하여 구축할 수 있음
  - (4) 대규모 시스템에서도 특정한 소수의 모듈을 변경함으로써 시스템을 변경할 수 있다
    - 그림 예) 시스템에서 모듈M1이 유저인터페이스의 기능에 해당하는 경우
      - 모듈M1을 변경함으로써 같은 서비스를 제공하는 시스템을 상이한 유저인터페이스로 제공하는 다른 시스템으로 변경시킬 수 있음

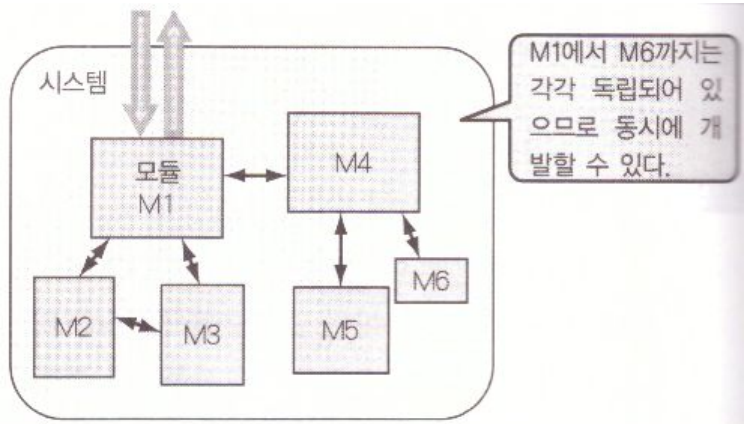

그림 9.1 모듈화된 시스템

9.1 모듈설계란?

9.1.1 모듈이란 무엇인가?

- 서브시스템(SubSystem)
  - 다른 시스템에 의해 제공되는 서비스에 의존하지 않는 독자적인 기능을 가지고 있는 시스템
  - 소프트웨어아키텍처로서 기본구조를 나타내기 위한 구성요소
  - 그림 예) 여러개의 모듈로 구성되며 다른 서브시스템과의 통신을 위한 인터페이스를 갖추고 있음
- 모듈
  - 다른 모듈에 대한 서비스를 제공하며, 다른 모듈의 서비스를 이용함
  - 일반적으로 모듈을 독립된 시스템으로 간주하지 않음
  - 서브시스템의 구성요소

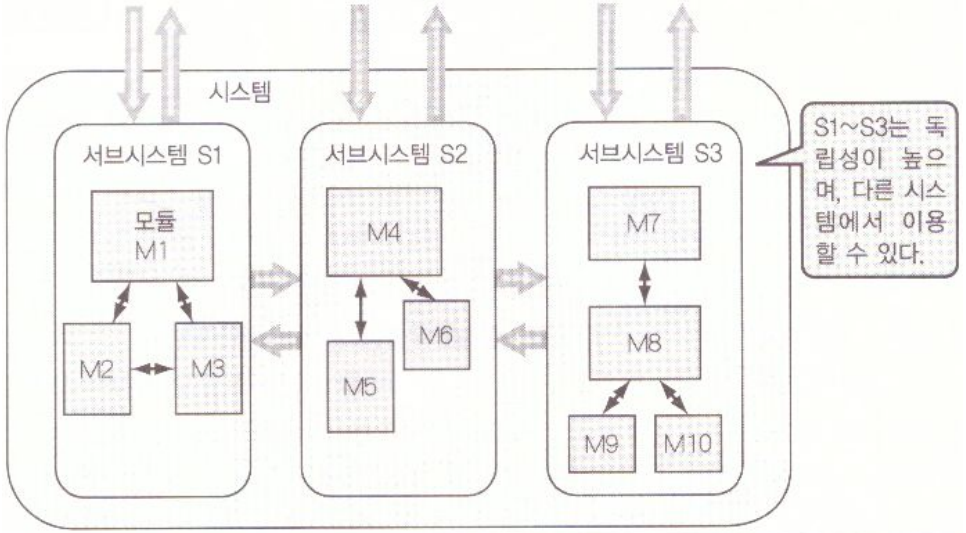

그림 9.2 모듈과 서브시스템

9.1 모듈설계란?

9.1.1 모듈이란 무엇인가?

- 소프트웨어에서의 모듈
  - 프로그램의 집합
  - 다양한 크기의 집합
  - 모듈의 예
    - 컴파일러와 컴파일러의 출력을 결합하는 링커의 처리대상이 되는 파일
    - 여러 개의 파일들을 결합하여 만들어진 실행형식 파일과 라이브러리

- 여기에서는, 프로그램의 구조를 생각하는데 있어서 기본이 되는 가장 작은 집합
  - 즉, 분리가능한 최소 프로그램단위
  - 예: 함수(Function), 프로시저(Procedure), 서브루틴(Subroutine), 클래스(Class) 등
    - 모두 프로그램 내부에서 어떤 정해진 기능을 반복적으로 사용할 수 있도록 만들어진 것

9.1 모듈설계란?

선문대학교

9.1.2 좋은 구조와 나쁜 구조

- 모듈은 여러 개의 요소들로 구성됨
  - 예: 프로그램의 함수는 함수내부에 기술되어 있는 변수의 선언문, 대입문, 반복문 등이 모듈의 구성요소들임
- 함수가 다른 함수를 호출하여 계산결과를 반환값으로 받는 것처럼, 모듈은 다른 모듈과 관계를 가짐
- 그림 예: 프로그램을 실행할 때, 어떤 모듈의 실행문을 실행하고 있는지, 다른 모듈을 호출하고 있는지를 다음을 조사하여 도식화
  - 모듈: 사각형
  - 모듈 내 제어흐름: 사각형 내부의 직선
  - 모듈 사이의 호출관계: 사각형을 연결하는 직선

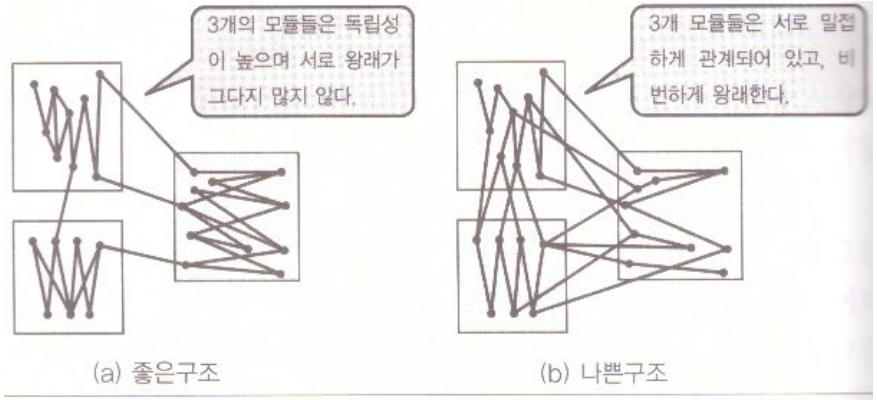

그림 9.3 좋은 구조와 나쁜 구조

9.1 모듈설계란?

9.1.2 좋은 구조와 나쁜 구조

- 그림 (a): 모듈화의 이점을 얻을 수 있을 것으로 기대되는 경우
  - 각 모듈이 대부분의 시간을 자신의 모듈내부에 있는 문장을 수행하는데 소비
  - 다른 모듈을 호출하는 것이 적음

- 그림 (b): 모듈화의 이점을 얻을 수 없는 경우
  - 각 모듈이 빈번하게 다른 모듈을 호출
  - 각 모듈은 밀접한 관계를 갖고 있기 때문에 다른 모듈의 상세사항을 고려하지 않고 해당 모듈을 개발하는 것은 어려움
  - 어떤 모듈을 변경하면 다른 모듈에 크게 영향을 주므로, 일부분의 모듈을 변경하는 것만으로 시스템을 변경할 수 있다고 하기 어려움

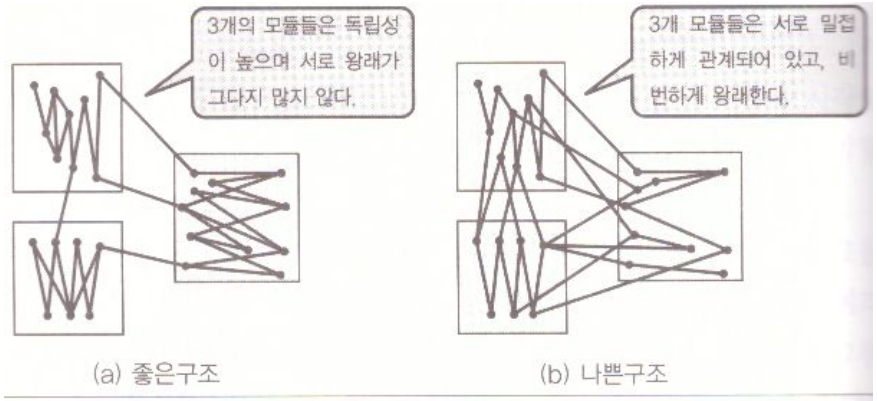

그림 9.3 좋은 구조와 나쁜 구조

9.2 모듈분할의 평가기준

개요

- 기능적 독립성이 높다
  - 어떤 모듈이 어떤 한가지 목적을 위한 기능만을 제공하고 있고, 다른 모듈과의 상호작용이 적은 경우
- 기능적 독립성이 높은 모듈로 구성된 소프트웨어
  - 모듈의 기능이 명확하게 분리되어 있고, 인터페이스도 깔끔하게 되어 있으므로 개발이 용이함
  - 설계의 변경과 프로그램의 수정에 의한 부작용 파급범위가 국소화되므로, 테스트와 유지보수가 수월해짐

목차

9.2 모듈분할의 평가기준

9.2.1 정보은폐(캡슐화)

- David L. Parnas 제안한 정보은폐의 개념
  - 인터페이스와 구현을 명확하게 분리하여, 모듈의 구현에 관한 상세한 설계정보는 다른 모듈로부터 은폐시켜서 참조할 수 없도록 해야한다는 것
    - 인터페이스: 모듈의 외부사양, 해당 모듈의 서비스를 이용하기 위해서 알고 있어야 하는 정보

- 모듈의 절차와 모듈내부의 데이터에 대한 조작에 관한 제약사항을 규정한 것
  - 제약사항에 따라 모듈을 작성한 경우, 모듈내부에 정의된 변수와 하위모듈은 외부로부터 은폐되어 직접 액세스할 수 없음
  - 다른 모듈로 액세스 하는 경우는 인터페이스로서 정의된 절차와 함수를 통해야 함

9.2 모듈분할의 평가기준

9.2.1 정보은폐(캡슐화)

- 그림 (a): 정보은폐가 없는 경우
  - 외부로부터 모듈의 구현에 접근할 수 있으므로 모듈내부의 함수와 데이터를 참조할 수 있음

- 그림 (b): 정보은폐가 적용되는 경우
  - 외부에는 모듈의 인터페이스만을 공개하고 구현은 은폐시켜서 외부로부터 직접 참조할 수 없음

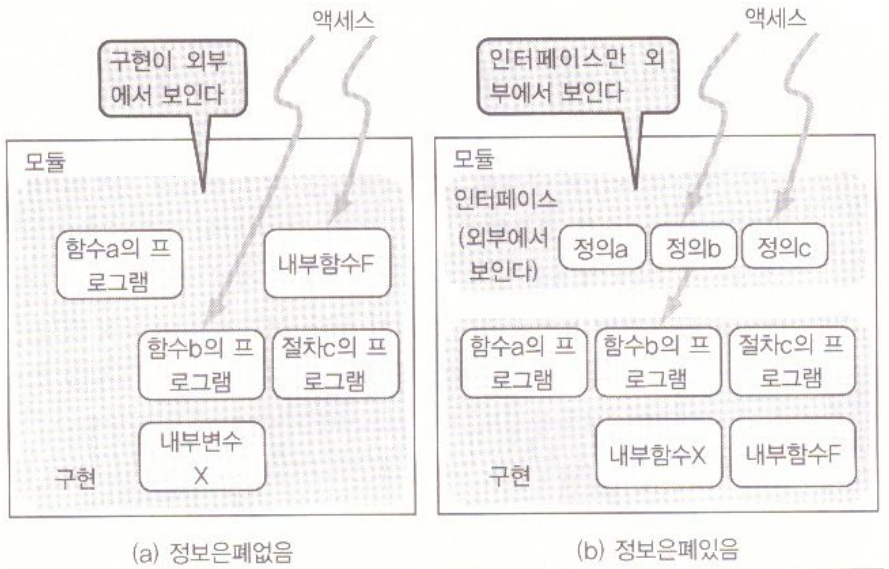

그림 9.4 정보은폐

9.2 모듈분할의 평가기준

9.2.1 정보은폐(캡슐화)

- 장점
  - (1) 모듈의 구현내용을 살펴보지 않더라도 모듈의 인터페이스를 이해하고 있으면 모듈을 이용할 수 있음
  - (2) 모듈의 구현을 변경하더라도 인터페이스를 변경하지 않는다면, 해당 모듈을 호출하고 있는 모듈은 그대로 사용할 수 있음
    - 고속화를 위해 구현방식을 개량하는 등 프로그램의 변경이 수월하게 되며 유지보수성을 향상시킬 수 있음

- (Parnas가 제안한 정보은폐의 개념에 의하면,) 정보은폐를 위해서는 아래와 같이 하는것이 중요
  - 모듈응집도를 높게
  - 모듈결합도는 낮게

9.2 모듈분할의 평가기준

9.2.2 모듈응집도

- 모듈내부에 있는 요소들 사이의 연결 강도를 나타내는 기준
  - 높은 응집도: 단일 기능으로 프로그램의 다른 기능과 거의 상호작용하지 않는 모듈
  - 응집도가 높을수록 모듈의 기능적 독립성은 높아짐

- 낮은 응집도부터 순서대로 나열
  - (1) 암호적 응집도(우발적 응집도)
    - 명확한 이유도 없이 요소들을 모아서 독립적인 기능을 구성한 모듈
      - 우연히 같은 문자로 시작되는 절차들을 모아서 작성한 모듈
      - 에디터를 사용하여 보기 좋게 하기 위해서 무리하게 적당한 길이로 분할해서 작성한 모듈

  - (2) 논리적 응집도
    - 겉모습상으로는 같은 기능을 갖지만, 실제로는 다양한 기능을 포함하는 모듈
      - 정수의 덧셈과 행렬의 덧셈을 결합시킨 덧셈모듈
      - 단말기 출력기능과 파일 출력기능을 결합시킨 출력모듈

9.2 모듈분할의 평가기준

9.2.2 모듈응집도

- 낮은 응집도부터 순서대로 나열
  - (3) 시간적 응집도
    - 기능적으로는 유사성이 없으나 실행시키는 시간이 서로 가깝게 연계되어 있어서 한군데 모아서 만든 모듈
      - 초기값설정모듈
  - (4) 순서적 응집도
    - 순서적으로 전후관계에 있는 기능들을 한군데 모아서 만든 모듈
      - 반복문이나 선택문에 의해 한덩어리가 된 기능을 모아서 만든 모듈
  - (5) 연결적 응집도(순차적 응집도)
    - 순서적 응집도를 갖는 모듈 중에서 모듈내부의 기능들 사이에 데이터의 관련성이 있는 모듈
      - 어떤 기능의 출력이 다른 기능의 입력으로 되어 있고, 다시 그 기능의 출력이 또 다른 기능의 입력으로 되는 관계에 있는 모듈

9.2 모듈분할의 평가기준

9.2.2 모듈응집도

- 낮은 응집도부터 순서대로 나열
  - (6) 정보적 응집도
    - 동일한 데이터에 액세스하는 여러 개의 기능들을 한군데 모아서 만든 모듈
      - 명단 데이터에 대한 등록, 변경, 삭제, 검색 등의 처리를 모아서 만든 모듈
  - (7) 기능적 응집도
    - 명확하게 정의할 수 있는 기능들만을 수행하는 모듈
      - 제곱근을 구하는 함수 모듈
- 기능적 독립성에 대한 관점에서, 정보적 응집도 보다 기능적 응집도가 기능적 독립성이 높음
  - 기능을 중시하는 모듈분할법에서는 기능적응집도를 권장
  - 객체지향설계와 같이 데이터를 중시하는 설계기법에서는 정보적 응집도를 권장
    - 정보적 응집도를 갖는 모듈을 여러 개의 기능을 갖고 있지만, 데이터의 구체적인 표현형식을 모듈내부에 은폐하고 있고, 정보은폐의 개념을 따르는 모듈이 됨

9.2.3 모듈의 결합도

- 모듈들 사이의 관련 정도를 나타내는 기준
- 결합도가 낮으면 모듈의 기능적 독립성이 높아지게 되어 바람직

- 높은 결합도부터 나열
  - (1) 내용결합
    - 어떤 모듈이 인터페이스를 통하지 않고 다른 모듈이 관리하는 데이터 또는 제어구조를 직접 이용하여 데이터를 직접 참조/변경하는 경우의 결합도
    - GOTO문 등에 의해 다른 모듈의 내부에 합류하는 경우에 내용결합이 발생함
    - 주로 어셈블러 프로그램의 모듈에서 발생함
    - 모듈의 변경이 다른 모듈에게 영향을 주게 됨

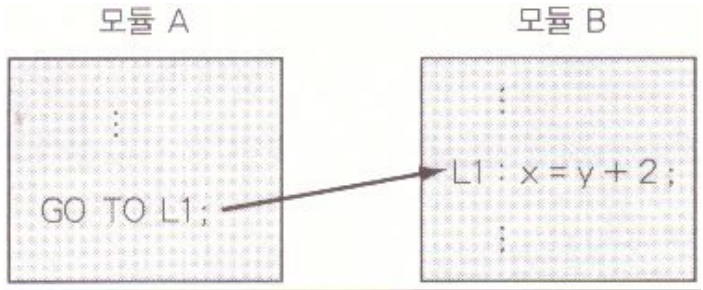

그림 9.5 | 내용결합

9.2.3 모듈의 결합도

- 높은 결합도부터 나열
  - (2) 공통결합
    - 공통데이터영역에 정의한 데이터를 통해서 정보교환을 수행하는 모듈 사이의 결합
    - 공통데이터영역의 데이터구조 등을 변경한 경우에 이를 참조하고 있는 모든 모듈이 영향을 받음
    - 모듈내부의 데이터와 공통데이터영역의 데이터에 대한 참조는 명령으로서 구별되지 않으므로, 어떤 모듈이 공통데이터영역의 데이터를 참조하고 있는지 명시적으로 기술되지 않음
      - 프로그램의 해독을 곤란하게 하고,
        공통데이터영역을 변경했을 경우에 영향을 받는 모듈을 간과해 버리기 쉬운 문제점이 있음

9.2.3 모듈의 결합도

- 높은 결합도부터 나열
  - (2) 공통결합
    - 그림 예: C언어에서 같은 이름의 광역변수를 2개의 모듈에서 초기화하지 않고 선언한 경우와 어느 한쪽 모듈에서만 초기화하여 선언한 경우에는 광역변수의 실체는 1개만 만들어져서 2개의 모듈에서 공유됨 → 공통결합 발생
      - cf. 양쪽 모듈에서 같은 이름의 광역변수를 초기화하여 선언한 경우에는 문법적 오류가 되므로 공통결합이 발생하지 않음

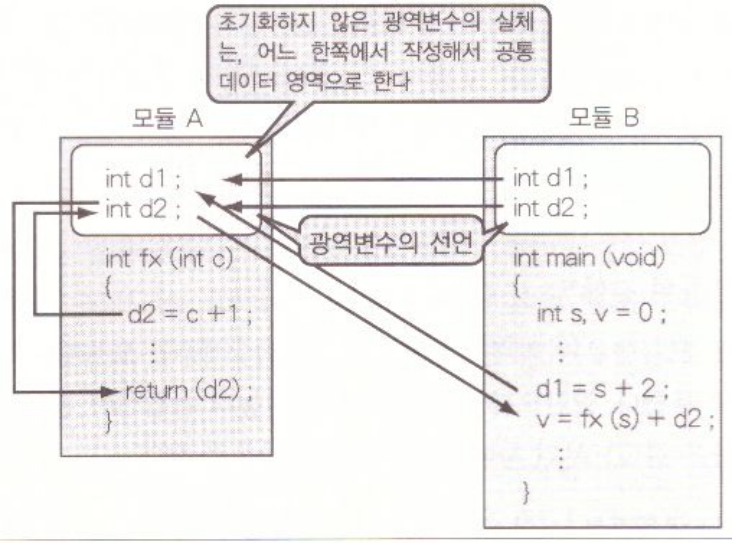

그림 9.6 | 공통결합

9.2.3 모듈의 결합도

- 높은 결합도부터 나열
  - (3) 외부결합
    - 외부선언한 데이터를 공유하는 모듈 사이의 결합
    - 공통결합과 유사하지만, 공유하는 데이터를 외부선언하므로 공통결합에서 지적되었던 문제점은 발생하지 않음

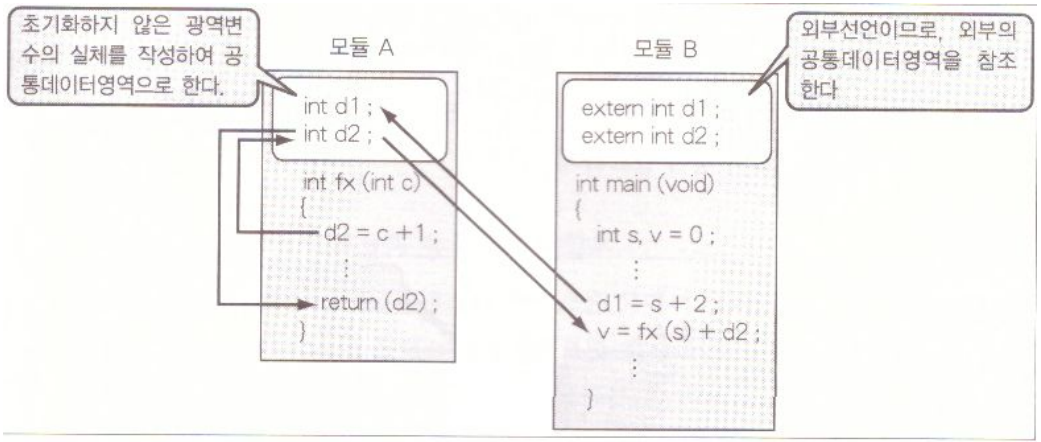

그림 9.7 | 외부결합

- 필요한 데이터만을 외부선언하므로, 불필요한 데이터까지 공유하는 것이 적어지게 되어 결합도가 낮아짐
- 그림 예: C언어의 extern문 등을 사용한 경우에 외부결합이 발생함

9.2.3 모듈의 결합도

- 높은 결합도부터 나열
  - (4) 제어결합
    - 어떤 모듈이 다른 모듈을 호출하는 경우, 호출되는 모듈의 제어를 지시하는 데이터를 매개변수로서 전달하는 모듈 사이의 결합
    - 그림 예: 프로그램에서 모듈A는 모듈B의 함수 FUNC를 호출하고 있음
      - 이때, 함수 FUNC에서는 3번째 매개변수 flag의 값을 분기조건으로 사용하여 제어흐름을 변경하고 있음
      - 따라서, 모듈A에서는 함수 FUNC의 내부에서의 제어방법을 고려하여 매개변수 flag의 값을 정할 필요가 있음

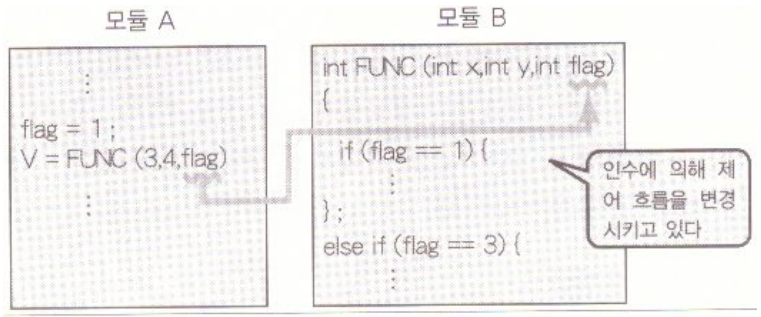

그림 9.8 | 제어결합

    - 호출하는 모듈은 호출되는 모듈의 내부 논리를 파악하고 있어야하고, 어떻게 제어하면 좋을지 이해하고 있을 필요가 있음

9.2.3 모듈의 결합도

- 높은 결합도부터 나열
  - (5) 스탬프결합
    - 공유데이터영역에 없는 구조를 갖는 데이터(구조체)를 전달하고 받는 모듈 사이의 결합
    - 그림 예: 프로그램에서는 모듈A가 모듈B의 함수 FUNC를 호출할 경우, student형 구조체를 매개변수로서 전달하고 있음
      - 이때, 함수 FUNC는 구조체의 일부분만 사용하고 있지만, 함수 FUNC를 작성하기 위해서는 student형 구조체의 정의를 전체적으로 알고 있어야 함

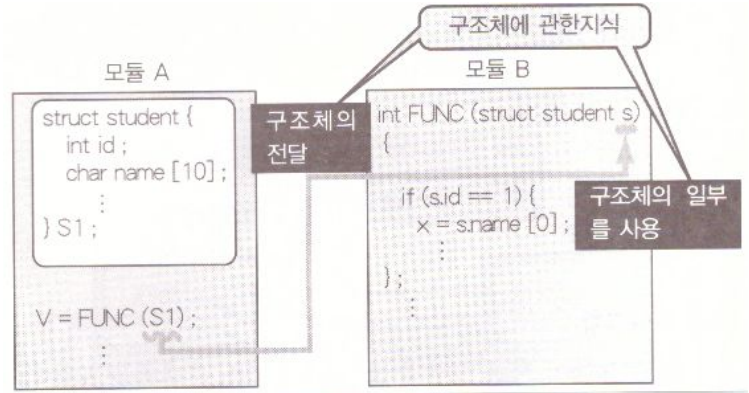

그림 9.9 | 스탬프결합

- 즉, 각 모듈들은 구조체 내부의 데이터 순서와 타입을 알고 있어야하며, 한쪽 모듈에서 구조체의 사양을 변경하게 되면 다른 모듈들도 수정이 필요하게 됨

9.2 모듈분할의 평가기준

선문대학교

9.2.3 모듈의 결합도

- 높은 결합도부터 나열
  - (6) 데이터결합
    - 데이터의 필요한 부분만을 매개변수로 전달하는 방법으로만 정보교환을 하는 모듈 사이의 결합
    - 그림 예: 앞의 그림(9.9)의 프로그램을 함수 FUNC에서 사용하는 정수값과 문자를 전달하도록 변경한 프로그램
    - 호출된 모듈이 구조체 데이터의 일부 요소만을 사용하는 경우에, 그 요소를 기본형 매개변수로하여 전달하도록 하면 호출되는 모듈은 호출하는 모듈의 데이터구조에 관한 전체적인 지식을 필요로 하지 않음
      → 결합도가 낮아짐

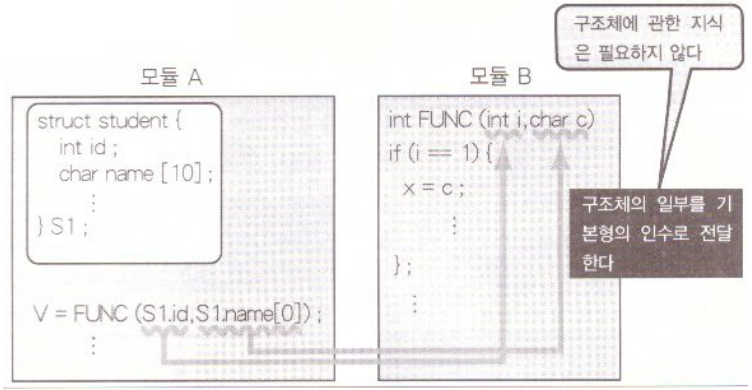

그림 9.10 | 데이터결합

9.2 모듈분할의 평가기준

9.2.4 객체지향설계에서 모듈의 응집도와 결합도

- 객체지향설계에서는 클래스와 클래스를 모아서 정리한 패키지를 모듈로 간주하는 것이 일반적임
- 클래스에 주목해보면, 캡슐화에 의해 동일한 데이터에 액세스하는 기능을 클래스에 모아서 정리하게 됨
  - 모듈의 응집도는 기본적으로 정보적 응집도에 해당함
- 실제 설계에서는 클래스의 이해와 변경을 수월하게 하기 위해서 1개의 클래스에 2가지 데이터를 혼재시키는 것이 더 좋은 경우가 있음
  - 모듈의 응집도는 정보적 응집도보다는 저하됨
- 2개의 클래스가 데이터결합되어 있더라도, 한쪽 클래스가 불필요하게 다른쪽 클래스의 속성(=데이터)을 참조하는 등, 모듈 사이의 결합도 관점에서 바람직하지 않은 설계가 되는 경우도 있음

9.2 모듈분할의 평가기준

9.2.4 객체지향설계에서 모듈의 응집도와 결합도

- 객체지향설계에 대해 기존의 모듈 응집도 및 결합도와는 상이한 모듈화(및 추상화)관련 지침 제안

표 9.1 | 클래스에 관한 설계원칙

원칙	설명
단일책임의 원칙	클래스를 변경하는 이유는 1가지만 있어야 한다.
Open-Close의 원칙	확장에 대해서 오픈하고, 수정에 관해서 클로즈해야 한다.
리스코프의 치환원칙	서브클래스는 슈퍼클래스와 교환가능해야 한다.
의존관계 역전의 원칙	상위의 모듈은 하위의 모듈에 의존하면 않된다. 추상적 모듈은 구현의 상세사항에 의존해서는 않된다.
인터페이스분리의 원칙	강한 관련성을 갖는 인터페이스만을 모아서 그룹화해야 한다.

표 9.2 | 패키지에 관한 설계원칙

원칙	설명
재이용 · 릴리즈 등가 원칙	패키지는 릴리즈의 단위로서 재이용되어야 한다.
전체 재이용의 원칙	패키지내부의 모든 클래스가 재이용되어야 한다.
폐쇄성 공통의 원칙	1개의 변경이유는 단일 패키지에 국한되어야 한다.
안정의존의 원칙	보다 안정되어 있는 패키지에 의존해야 한다.
안정도 · 추상도 등가의 원칙	추상적인 패키지일수록 안정되어 있어야 한다.

9.3 구조화 시스템설계

개요

- 구조화분석에서 모듈설계에 이르기까지의 설계공정에 대해서 소개
- 데이터흐름도로부터 모듈고조도를 작성하는 기법 설명

목차

9.3 구조화 시스템설계

9.3.1 소프트웨어의 설계공정

- 소프트웨어의 설계단계: 각 단계에서 주목해야되는 항목에 관해서 본질적인 사항을 추출하여 추상화한 모델을 작성함
- 그림 예: 구조화시스템설계의 설계개발공정
  - (1) 인터뷰 등에 의해 요구사항 추출하고, 기능적 요구사항과 비기능적 요구사항을 모아 요구사항모델을 작성
  - (2) 구조화분석에 의해 요구사항모델로부터 분석모델을 작성
    - 분석모델: 분석결과를 모아서 정리한 요구사양으로서 다음을 작성
      - 자료흐름도(Data Flow Diagram)
      - 데이터사전(Data Dictionary)
      - 미니사양서(Mini-Specification)
      - ERD(Entity Relationship Diagram)

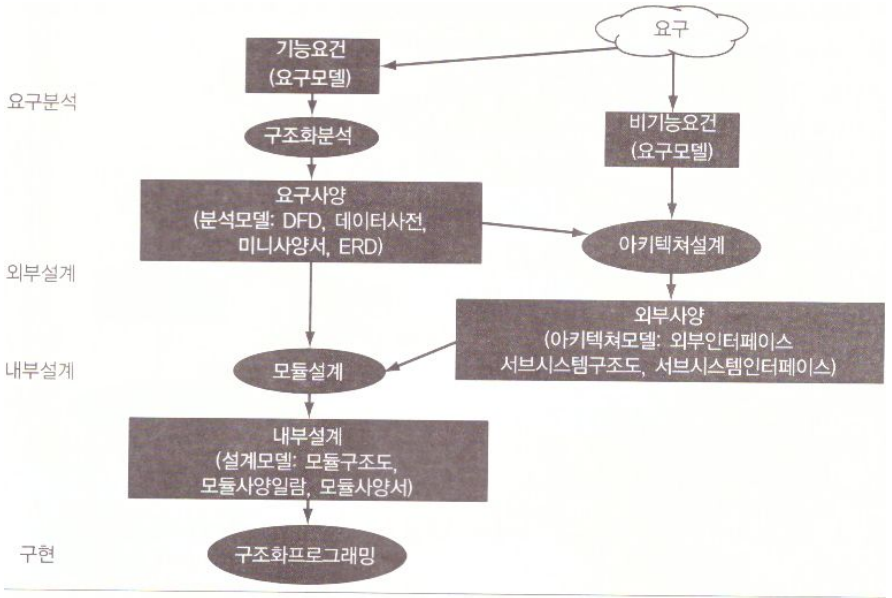

그림 9.11 | 구조화시스템설계의 설계개발공정

9.3 구조화 시스템설계

9.3.1 소프트웨어의 설계공정

- 그림 예: 구조화시스템설계의 설계개발공정
  - (3) 아키텍처설계: 요구사항모델과 분석모델로부터 소프트웨어의 기본구조를 나타내는 아키텍처모델을 작성
    - 외부인터페이스, 서브시스템구조도, 서브시스템인터페이스 등이 기술됨
  - (4) 모듈설계: 분석모델과 아키텍처모델을 토대로 설계모델을 작성
    - 모듈구조도, 모듈사양일람, 모듈사양서가 포함됨
  - (5) 모듈사양서를 토대로 각 모듈의 코딩작업을 수행
- 외부설계와 내부설계 등의 2가지로 나누어 수행할 수도 있음
  - 외부설계: 아키텍처모델을 외부사양서로 제공
  - 내부설계: 설계모델을 내부사양서(상세사양서)로 제공

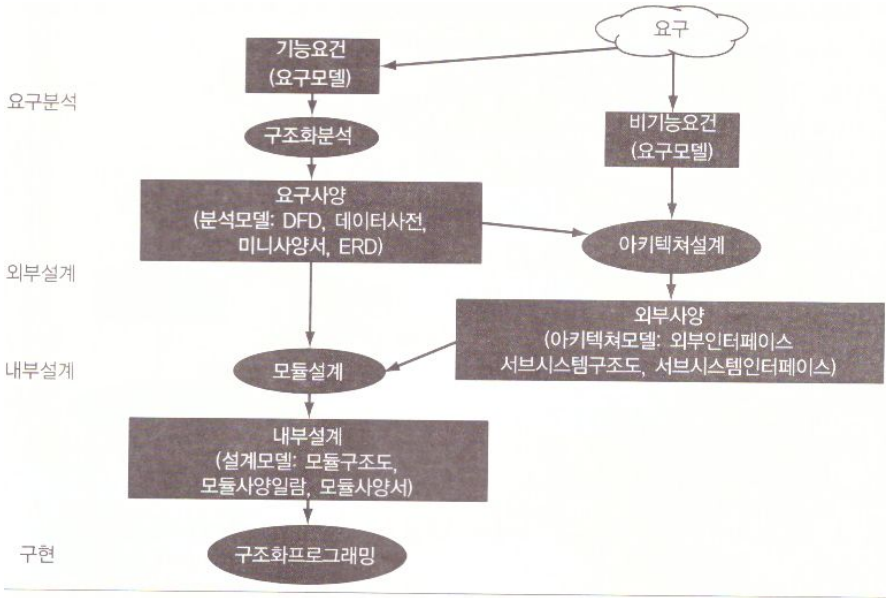

그림 9.11 | 구조화시스템설계의 설계개발공정

9.3 구조화 시스템설계

9.3.1 소프트웨어의 설계공정

- 그림 예: 객체지향시스템설계의 설계개발공정
  - 객체지향 분석/설계
    - 모듈로서 클래스를 정의하고 UML도면을 설계작업의 진척정도에 따라 추가 및 세련화
      - UML 도면의 예: Use Case Diagram, Class Diagram, State Machine Diagram 등
    - 모듈구조도에 상응하는 Class Diagram, Package Diagram을 각각의 설계원칙에 기반하여 작성

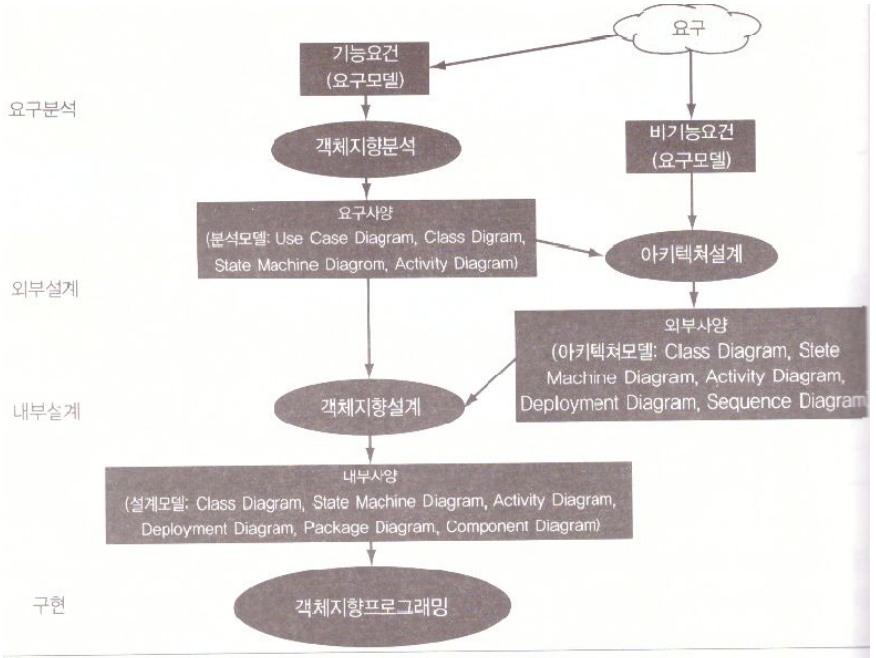

그림 9.12 | 객체지향시스템설계의 설계개발공정

9.3.1 소프트웨어의 설계공정

- 그림 예: 객체지향시스템설계의 설계개발공정
  - 아키텍처설계
    - 아키텍처에 대한 관점(View)에 따라서 도면을 선택하여 아키텍처모델로서 제공
      - Class Diagram에서는 논리View를 표현하고, 클래스를 요소로 하여 클래스 사이의 관계를 기술함 → 시스템 전체를 이해하고 주요 기능들 사이의 논리적 구조를 명확하게 함
      - State Machine Diagram과 Activity Diagram을 사용하여 논리적인 동작을 표현하고, Activity Diagram과 Sequence Diagram을 사용하여실행View를 표현함

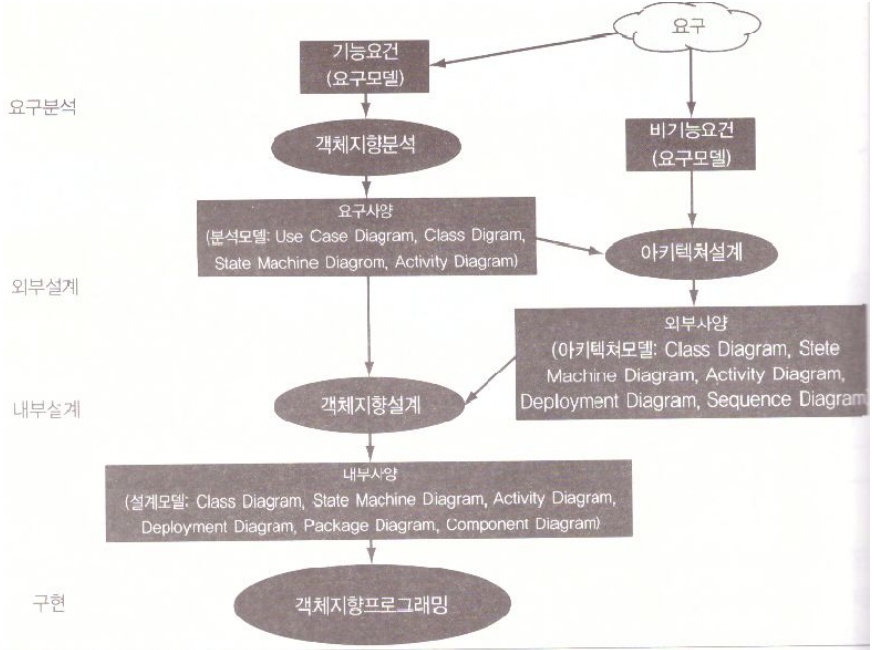

그림 9.12 | 객체지향시스템설계의 설계개발공정

9.3 구조화 시스템설계

9.3.1 소프트웨어의 설계공정

- 그림 예: 객체지향시스템설계의 설계개발공정
  - 구현공정에 가까워지면
    - Package Diagram, Component Diagram, Deployment Diagram을 작성하여 세련화된 Class Diagram과 함께 설계모델로서 제공함
  - 구현단계에서 Java 등과 같은 객체지향프로그래밍 언어를 사용하게 되면
    - 분석에서 구현에 이르기까지 일관되게 객체를 중심으로 생각할 수 있게 되므로, 각 클래스의 구현이 수월하게 됨

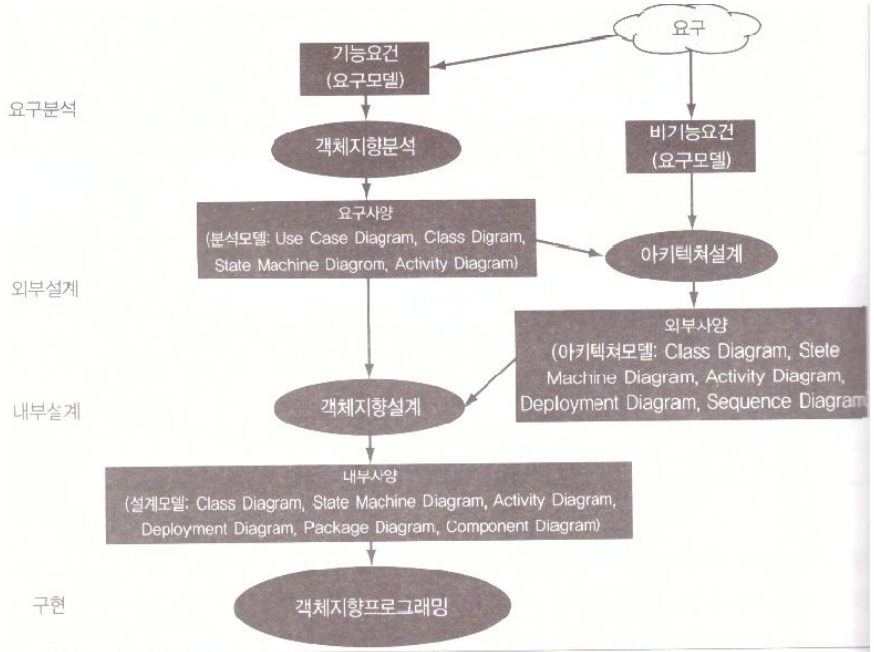

그림 9.12 객체지향시스템설계의 설계개발공정

9.3.2 설계모델

- 모듈구조도, 모듈사양일람, 모듈사양서를 설계모델로서 작성함
  - (1) 모듈구조도
    - 모듈과 데이터, 상호호출관계, 데이터의 상호작용 등을 나타내는 도면
    - 소프트웨어설계의 전체모습과 모듈의 의존관계 등을 파악가능
  - (2) 모듈사양일람
    - 모듈이름, 모듈인터페이스, 모듈의 목적 등과 같은 항목들을 정리하여, 개발대상이 되는 모듈전체를 파악하기 수월하도록 일람표형식으로 관리함
  - (3) 모듈사양서
    - 각 모듈마다 인터페이스사양과 기능사양을 기술함
      - 인터페이스사양
        - 모듈이름, 매개변수, 반환값 등을 기술하여 해당 모듈이 무엇을 해야 하는지를 정의
      - 기능사양
        - 어떻게 구현할 것인지를 표현함
        - 미니사양서(프로세스 사양서) 이용가능

9.3 구조화 시스템설계

9.3.3 모듈분할기법

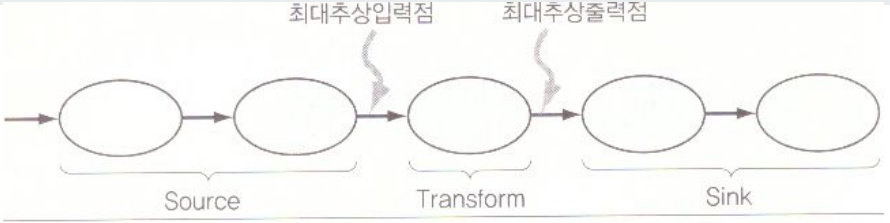

그림 9.13 | STS분할기법에서의 모듈 분석

- 자료흐름도(Data Flow Diagram)로부터 모듈구조도를 작성하는 기법
- (1) STS분할기법
  - 데이터가 변환되는 지점(**최대추상점**)을 찾아내서 자료흐름도를 Source, Transform, Sink로 분할하고, 각각을 제어하는 모듈을 메인 모듈로 매치함
  - 구체적인 순서
    - ① 자료흐름도에서 입력으로부터 출력에 이르기까지의 주요 경로를 찾아냄
    - ② 위에서 찾아낸 경로에 대해 최대추상입력점과 최대추상출력점을 찾아내고, 입력부분(Source), 입력에서 출력으로의 변환부분(Transform), 출력부분(Sink)으로 분할함
      - 최대추상입력점 : 입력데이터를 왼쪽에서 오른쪽으로 추적하면서, 처리된 데이터가 입력데이터로 간주할 수 없게 된 지점
      - 최대추상출력점 : 출력데이터를 오른쪽에서 왼쪽으로 추적하면서 처리된 데이터가 출력데이터로 간주할 수 없데 된 지점
    - ③ Source, Transform, Sink를 종속모델들로 하여, 이들을 제어하는 모듈을 메인모듈로 하여 배치
    - ④ 종속모듈들에 대해서 응집도와 결합도의 관점에서 모듈을 모아서 정리하거나 분할

9.3.3 모듈분할기법

- (1) STS분할기법
  - 앞의 순서에 의해, 메인모듈(제어모듈)을 최상위로 하는 계층적인 모듈구조도가 작성됨

  - 그림 예: STS분할기법에 의함 모듈구조도의 예

그림 9.14 | STS분할기법에 의한 모듈구조도의 예

9.3.3 모듈분할기법

- (2) TR분할기법
  - 트랜잭션(Transaction) 처리
    - 입력데이터가 주어지면 해당 데이터에 관련된 일련의 처리를 수행하는 것
    - 데이터베이스의 일관성을 유지하기 위해
      - 일련의 처리를 모아서 수행
      - 다른 입력데이터의 처리와 분리
  - 그림 예: TR분할기법에서의 모듈분석
    - **Transaction Center**
      - 트랜잭션 처리에 있어서 입력데이터의 종류에 따라, 처리분기를 판단하는 지점
    - 종류에 따라 분기하는 모듈과 분기된 곳에서 개별데이터를 처리하는 모듈을 분리함
      - 데이터 처리의 독립성을 높일 수 있으므로, 효율적인 처리가 가능함

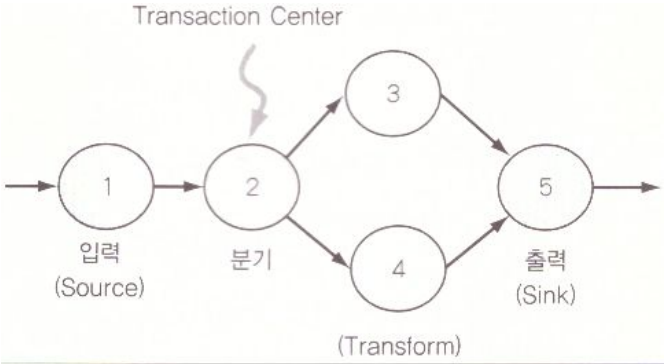

그림 9.15 | TR분할기법에서의 모듈분석

9.3.3 모듈분할기법

- (2) TR분할기법
  - 그림 예(위): TR분할기법에서의 모듈분석 순서
    - ① 자료흐름도에서 트랜잭션센터(분기지점)를 찾아냄
    - ② 분기지점의 앞부분을 입력부분(Source)으로 하고, 분기 이후의 부분을 입력에서 출력으로의 변환부분(Transform)과 출력부분(Sink)으로 분할
    - ③ 입력, 분기, 출력을 종속모듈로 하고, 이러한 모듈을 제어하는 모듈을 메인모듈로 하여 배치함
      - 변환부분을 분기모듈의 종속모듈로 배치
    - ④ 종속모듈에 대해 응집도와 결합도의 관점에서 모듈을 모아서 정리하거나 분할
    - 위 순서에 의해 메인모듈(제어모듈)을 최상위모듈로 하는 계층적인 모듈구조도(그림 예(아래))가 완성됨
      - 그림(아래)의 숫자는 그림(위)의 자료흐름도의 프로세스 번호

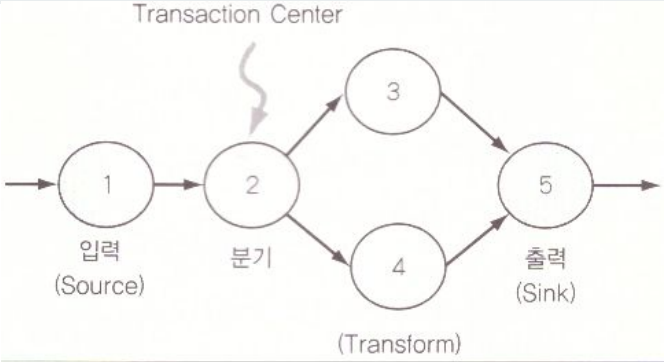

그림 9.15 | TR분할기법에서의 모듈분석

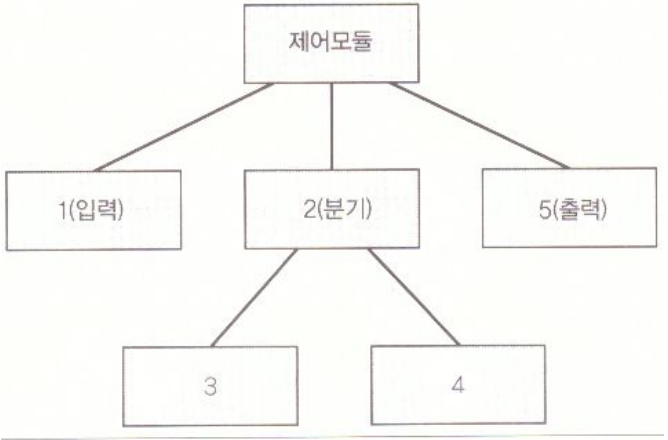

그림 9.16 | TR분할기법에 의한 모듈구조도의 예

연습문제 (객관식)

선문대학교

각 질문에 대한 알맞은 답의 번호(숫자)를 괄호안에 표시하시오.

❏ 1. 다음 중 모듈화된 시스템의 설명으로 옳지 않은 것은? ( )
(1) 특정한 모듈의 상세만을 파악해두면, 해당 모듈의 개발을 시작할 수 있다.
(2) 모듈을 각각 별도로 설계개발하고, 나중에 모듈을 통합할 수 있다.
(3) 다른 시스템에 의해 제공되는 서비스에 의존하지 않는 독자적인 기능을 가지고 있는 시스템이다.
(4) 기존의 모듈을 재이용하여 구축할 수 있다.
(5) 특정한 소수의 모듈을 변경함으로써 시스템을 변경할 수 있다.

❏ 2. 다음 중 모듈화의 이점을 얻을 수 있을 것으로 기대되는 경우에 해당하는 것은 무엇인가? ( )
(1) 각 모듈이 빈번하게 다른 모듈을 호출한다.
(2) 다른 모듈의 상세사항을 고려하지 않고 해당 모듈을 개발하는 것은 어렵다.
(3) 어떤 모듈을 변경하면 다른 모듈에 크게 영향을 준다.
(4) 각 모듈이 대부분의 시간을 자신의 모듈내부에 있는 문장을 수행하는데 소비하고, 다른 모듈을 호출하는 것은 아주 적다.

객관식 문제 (답안지)

❏ 1. (3) → 서브시스템에 대한 설명이다.

모듈은 다른 모듈에 대한 서비스를 제공하며 다른 모듈의 서비스를 이용하므로, 일반적으로 독립된 시스템으로 간주하지 않는다.

❏ 2. (4) → 모듈의 독립성이 높으며 서로 왕래가 많지 않아 모듈화의 이점을 얻을 수 있을 것으로 기대되는 경우이다.

연습문제 (객관식)

각 질문에 대한 알맞은 답의 번호(숫자)를 괄호안에 표시하시오.

- 3. 모듈의 응집도 중 동일한 데이터에 액세스하는 여러 개의 기능들을 한군데 모아서 만든 모듈로, 객체지향설계와 같이 데이터를 중시하는 설계기법에서 권장하는 것은 무엇인가? ( )
  (1) 정보적 응집도 (2) 연결적 응집도 (3) 시간적 응집도 (4) 기능적 응집도

- 4. 모듈의 응집도 중 명확하게 정의할 수 있는 기능들만을 수행하는 모듈로, 기능을 중시하는 모듈분할법에서 권장하는 것은 무엇인가? ( )
  (1) 시간적 응집도 (2) 연결적 응집도 (3) 정보적 응집도 (4) 기능적 응집도

- 5. 모듈의 결합도를 높은 결합도부터 나열한 것은 무엇인가? ( )
  (1) 데이터결합-스탬프결합-제어결합-외부결합-공통결합-내용결합
  (2) 내용결합-공통결합-외부결합-제어결합-스탬프결합-데이터결합
  (3) 데이터결합-공통결합-외부결합-내용결합-스탬프결합-제어결합
  (4) 내용결합-공통결합-외부결합-스탬프결합-제어결합-데이터결합

객관식 문제 (답안지)

- 3. (1) → 정보적 응집도
- 4. (4) → 기능적 응집도
- 5. (2) → 내용결합-공통결합-외부결합-제어결합-스탬프결합-데이터결합

연습문제 (주관식)

선문대학교

주관식 문제 (괄호안에 알맞은 단어를 입력하시오.)

- 1. 건물, 가구, 자동차, 전기기구, 컴퓨터하드웨어 등 다양한 사물들에 대한 각각의 구성단위인 ( )은 보다 복잡한 구조를 구축하기 위해 이용가능한 독립적인 기능을 갖는 단위(Unit)이며, 일반적으로 교환 및 탈착이 가능한 구성부품이다.

- 2. David L. Parnas가 제안한 정보은폐의 개념에 의하면,정보은폐를 위해서는 모듈의 ([1])를 높게하고, 모듈의 ([2])는 낮게 하는 것이 중요하다. 모듈내부에 있는 요소들 사이의 연결 강도를 나타내는 기준을 모듈의 ([1])라고 부르고, 모듈들 사이의 관련 정도를 나타내는 기준을 모둘의 ([2])라고 부른다.

- 3. 모듈의 결합도 중 ([1])결합은 공유데이터영역에 없는 구조를 갖는 데이터(구조체)를 전달하고 받는 모듈 사이의 결합이고, ([2])결합은 데이터의 필요한 부분만을 매개변수로 전달하는 방법으로만 정보교환을 하는 모듈 사이의 결합이다.

주관식 문제 (답안지)

- 1. 모듈(Module)

- 2.
  (1) 응집도
  (2) 결합도

- 3.
  (1) 스탬프
  (2) 데이터

연습문제 (주관식)

주관식 문제 (괄호안에 알맞은 단어를 입력하시오.)

- 4. ( )는 모듈과 데이터, 상호호출관계, 데이터의 상호작용 등을 나타내는 도면이며, 소프트웨어설계의 전체모습과 모듈의 의존관계 등을 파악할 수 있다.
- 5. 모듈 분할 기법 중 ( )은 데이터가 변환되는 지점을 찾아내서 자료흐름도를 Source, Transform, Sink로 분할하고, 각각을 제어하는 모듈을 메인 모듈로 매치한다.
- 6. 모듈 분할 기법 중 STS분할기법에서는 데이터가 변환되는 지점을 찾아내서 자료흐름도를 Source, Transform, Sink로 분할한다. 데이터가 변환되는 지점을 무엇이라고 하는가? ( )
- 7. 모듈 분할 기법 중 TR분할기법에서, 트랜잭션 처리에 있어서 입력데이터의 종류에 따라 처리분기를 판단하는 지점을 무엇이라고 하는가?
  ( )

주관식 문제 (답안지)

- 4. 모듈구조도
- 5. STS분할기법
- 6. 최대추상점
- 7. Transaction Center (또는 트랜잭션 센터)

연습문제 (주관식)

선문대학교

주관식 문제 (괄호안에 알맞은 단어를 입력하시오.)

- 8. 모듈 분할 기법 중 TR분할기법에서는 자료흐름도에서 트랜잭션센터를 찾아낸 후,
  트랜잭션센터의 앞부분을 입력부분([1])으로 하고,
  분기 이후의 부분을 입력에서 출력으로의 변환부분([2])과
  출력부분([3])으로 분할한다.

- 9. 모듈 분할 기법 중 TR분할기법에서, Transaction Center는 트랜잭션 처리에 있어 입력데이터의 종류에 따라, 처리분기를 판단하는 지점을 의미한다.
  이때, 종류에 따라 '분기하는 모듈'과 '분기된 곳에서 개별데이터를 처리하는 모듈'을 서로 분리하는 이유는 무엇인가?
  ( )

주관식 문제 (답안지)

- 8.
  (1) Source
  (2) Transform
  (3) Sink

- 9. 데이터 처리의 독립성을 높일 수 있으므로, 효율적인 처리가 가능함

Any questions?