Object Oriented Software Engineering OOSE

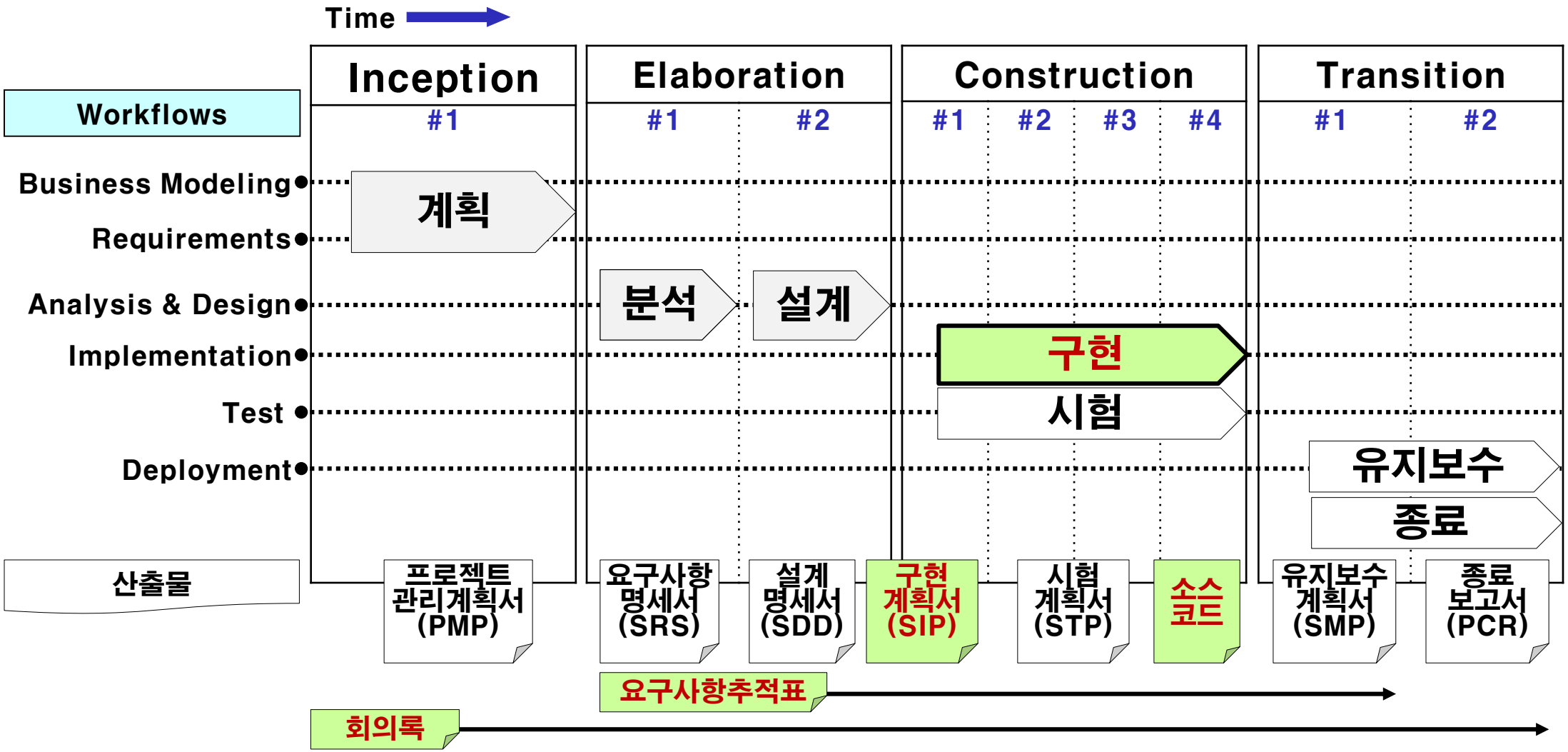
구현(Implementation)

- 구현이란?
- 구현 작업의 산출물(Artifacts)
- 구현 작업의 수행자(Workers)
- 구현 작업의 작업흐름(Workflows)
- 코드 검사(Code Inspection)

Computer & Software Engineering
Lee Jong Yeol
soyeum@kumoh.ac.kr

어디까지 왔나?

- 소프트웨어 개발 프로세스(SDLC, OO, UP, PMBOK)



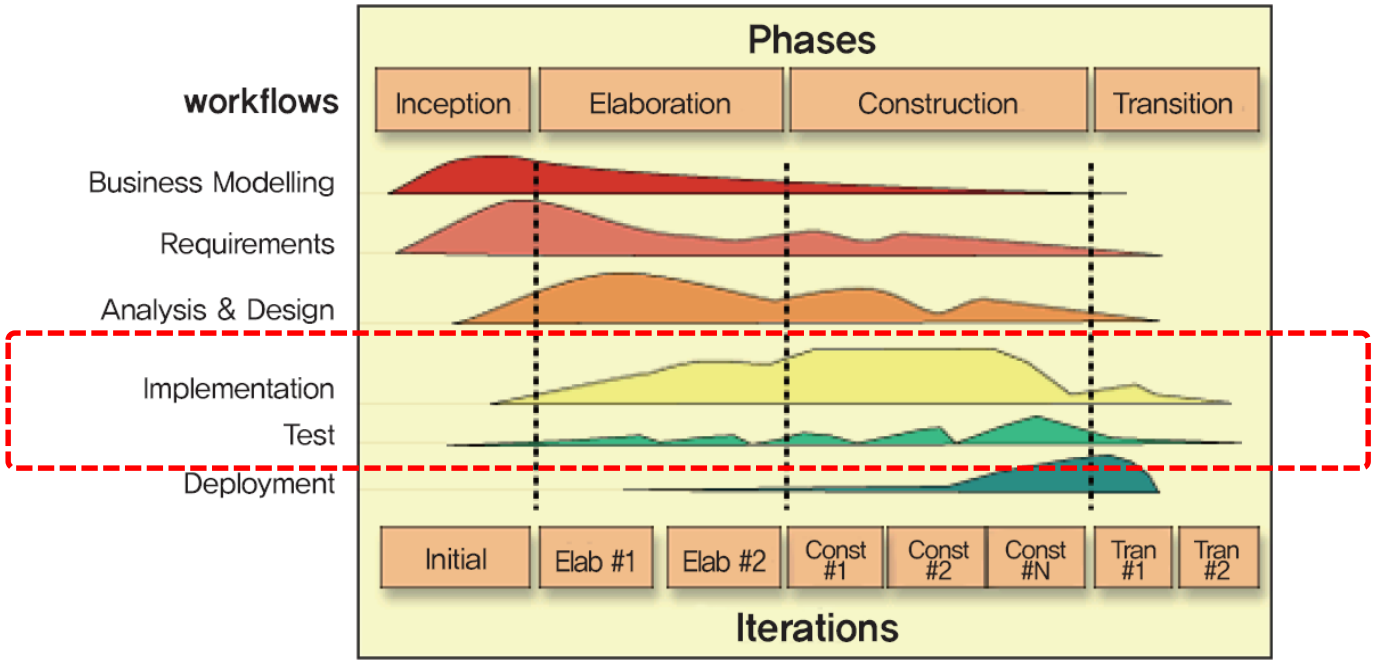
구현이란?

- 정의
  - 설계 문서를 바탕으로 시스템을 컴포넌트의 집합으로 실현
  - 컴포넌트: 소스 코드, 스크립트, 이진 파일, 실행 파일, …

- 구현 작업의 범위
  - 각각의 iteration에서 필요한 시스템의 통합 계획 수립
  - 설계 클래스 및 서브시스템을 독립 코드로 작성
  - 컴포넌트를 구성하는 모듈에 대한 단위 테스트 실시
  - 테스트가 끝난 모듈로부터 실행 파일 생성
  - 실행 가능한 컴포넌트를 배치 모델의 노드에 분산 배치

SDLC에서 구현의 역할

- S/W 개발과정에서 구현의 역할
  - 구축 단계에서는 대부분의 활동이 구현 작업
    - 정제 단계: 시스템 구조의 기준선 설정에 필요한 실행 모듈 구현
    - 전이 단계: 늦게 발견된 오류 수정을 위한 구현 작업
  - 구현 모델은 소스 코드 파일을 포함하고 있으므로, s/w 개발의 전 과정을 통하여 반드시 보존되어야 함



구현 작업의 산출물

- 구현 모델 (Implementation Model)
  - 서브시스템 구현(Implementation Subsystems)
  - 컴포넌트 구현(Implementation Components)
  - 인터페이스 구현(Implementation Interfaces)

- 구조 기술(Architecture Description)
  - 구현 모델 관점(View of Implementation Model)

- 통합 빌드 계획 (Integration Build Plan)

Implementation Model

- 기본 개념
  - 구현 모델은 아래 사항에 대해서 기술
    - 설계 모델의 구성 요소가 어떤 컴포넌트로 구현되는가?
    - 구현 환경 및 프로그래밍 언어가 제공하는 기능을 사용하여, 컴포넌트들을 체계적으로 그룹화하는 방법
    - 컴포넌트들 사이의 종속 관계
  - 구현 모델은 계층적 구조를 형성
    - 구현 모델은 구현 시스템으로 표현
    - 구현 시스템 = 최상위의 서브시스템
  - 서브시스템
    - 하위의 서브시스템을 구성 요소로 가질 수 있음
    - 구성 요소: 구현 서브시스템, 컴포넌트, 인터페이스

Implementation Subsystem(1)

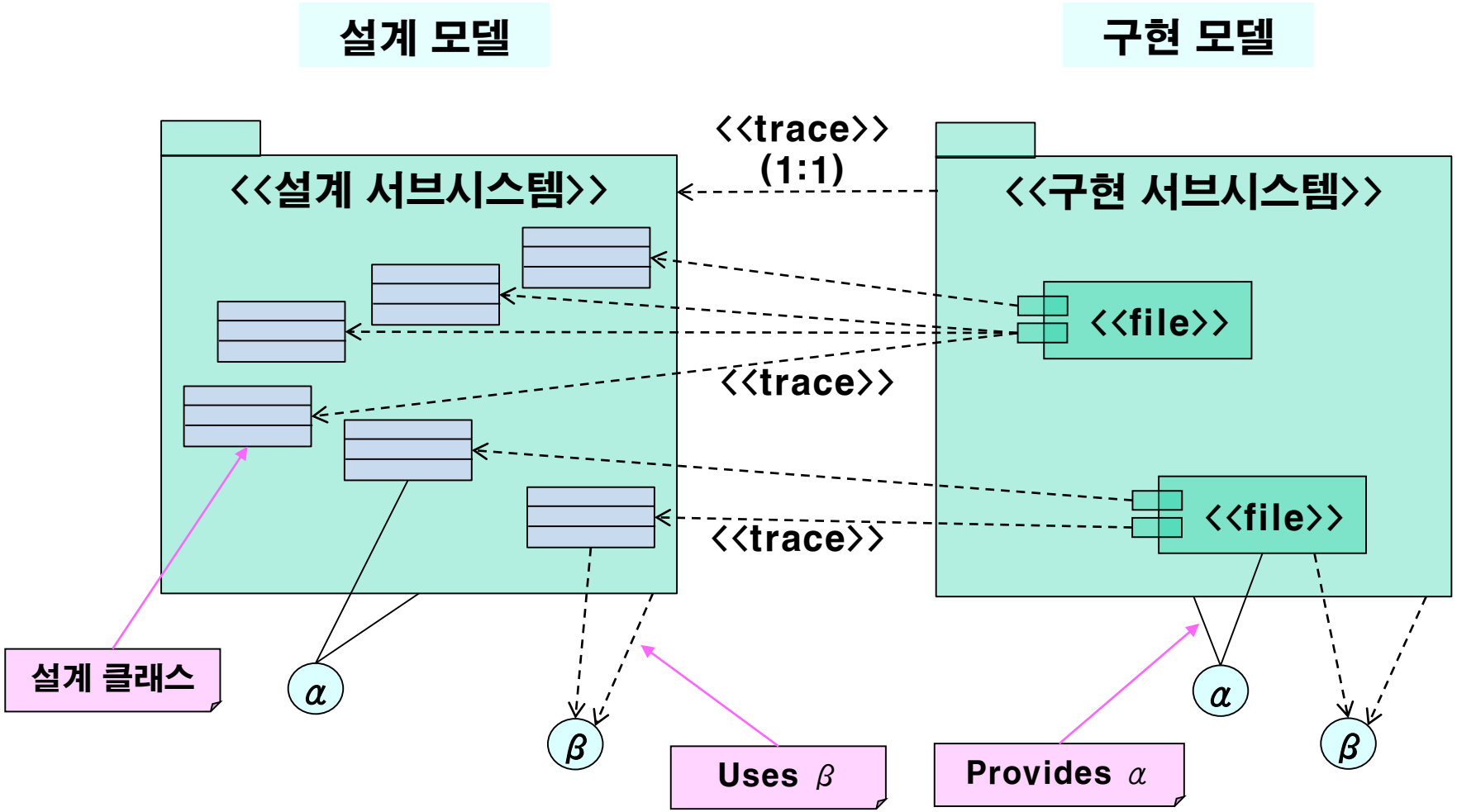
❑ 기본 개념
- 구현 모델의 산출물을 적절한 크기로 분류하는 수단을 제공
- 구성 요소: 하위의 서브시스템, 컴포넌트, 인터페이스
- 오퍼레이션 형태의 인터페이스를 제공할 수 있음

❑ 서브시스템의 구현 수단
- 서브시스템은 프로그래밍 언어가 제공하는 packaging 기능에 의해 실현
- 프로그래밍 언어가 제공하는 packaging 기능의 예
  - Java: package
  - Visual Basic: project
  - C++ : directory of files in project
  - Rational Apex(통합 환경): subsystem
  - Rational Rose(모델링 도구): component view package

Implementation Subsystem(2)

- 구현 서브시스템 vs. 설계 서브시스템
  - 구현 서브시스템과 설계 서브시스템은 1:1 대응 관계
    - 대응 관계는 설계 모델의 서비스 서브시스템에 대해서도 성립
  - 서브시스템이 제공하거나 사용하는 인터페이스에 대한 변경은 설계 작업에서 처리 -> 구현 작업에서는 설계 문서의 내용을 준수
  - 구현 서브시스템과 다른 서브시스템 and/or 인터페이스 사이의 종속 관계는 반드시 정의
  - 구현 서브시스템은 대응하는 설계 시스템과 동일한 인터페이스를 제공해야 함
  - 구현 서브시스템의 내부에서 인터페이스를 제공하는 것이 어떤 구성 요소인가를 분명하게 밝혀야 함

구현 서브시스템 vs. 설계 서브시스템



Implementation Components(1)

❑ 정의
- 컴포넌트: 설계 모델의 구성 요소를 구현한 결과물을 물리적인 단위로 포장한 것

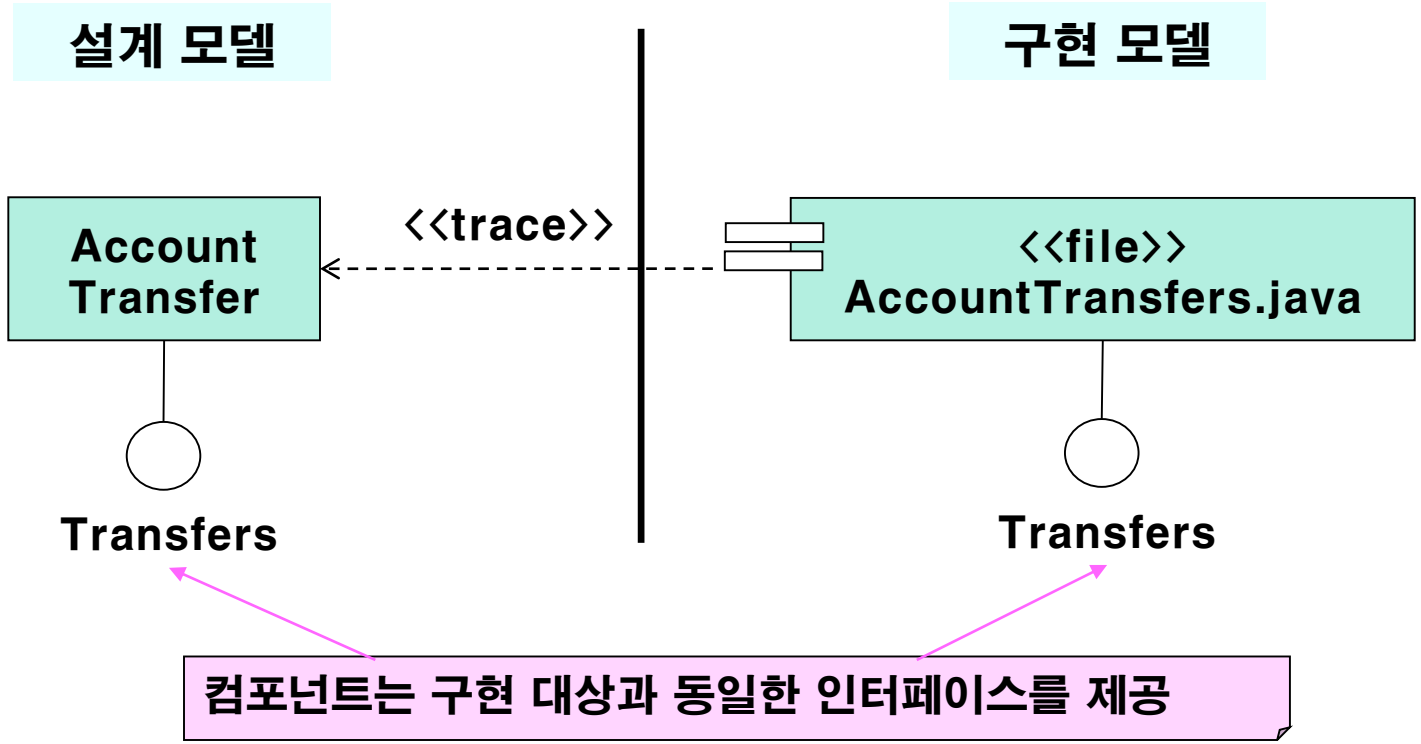
❑ UML에서 정의한 컴포넌트의 종류
- ≪executable≫ : 노드에 탑재하여 실행시킬 수 있는 프로그램
- ≪library≫ : 정적 또는 동적 라이브러리
- ≪table≫ : DB 테이블
- ≪file≫ : 소스 코드 또는 데이터를 갖고 있는 파일
- ≪document≫ : 개발 과정에서 생산된 문서

Implementation Component(2)

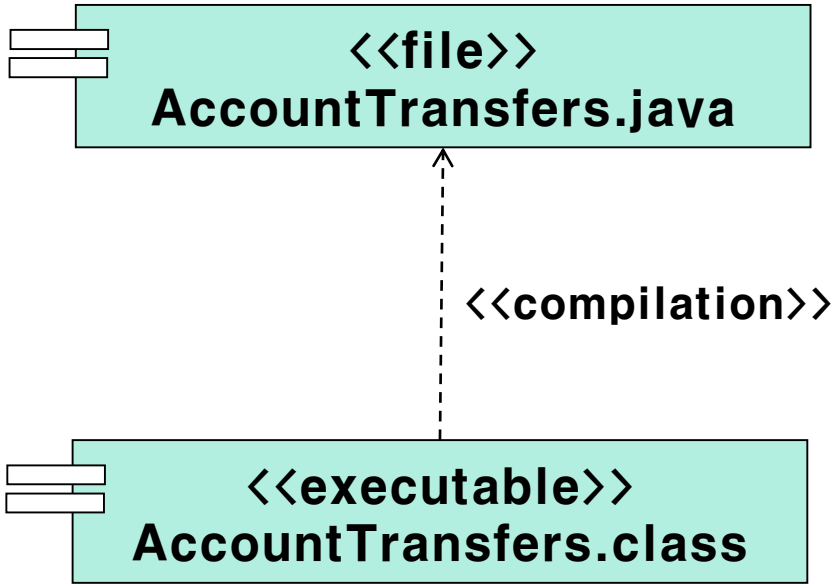
- 컴포넌트의 특성
  - 컴포넌트는 구현 대상과 추적(trace) 관계를 가짐
  - 컴포넌트는 구현 대상이 제공하는 인터페이스를 제공해야 함
  - 통상적으로 하나의 컴포넌트가 여러 개의 설계 구성 요소를 구현함
  - 컴포넌트 사이에 컴파일 종속관계(compilation dependency)가 존재할 수 있음

- Stub & Test Driver
  - 다른 컴포넌트를 테스트하기 위해서 임시로 만들어 사용하는 컴포넌트
  - 시스템의 중간 버전에서 필요한 새로운 컴포넌트의 수를 최소한으로 축소시킬 수 있음
  - 시스템 통합 및 통합 테스트를 단순화

Trace Dependency 예시



Compilation Dependency 예시



Implementation Interface(1)

❑ 기본 개념

- 인터페이스는 컴포넌트 또는 서브시스템이 제공하는 오퍼레이션을 기술하는데 사용
- 다른 컴포넌트와 서브시스템이 인터페이스의 오퍼레이션을 사용
  -> 이들 사이에는 사용 종속 관계가 성립
- 인터페이스를 실현하는 컴포넌트는 반드시 인터페이스에 정의된 모든 오퍼레이션을 올바르게 구현해야 함
- 인터페이스를 제공하는 서브시스템은 반드시 그 내부에 인터페이스를 구현하는 컴포넌트를 포함해야 함

Implementation Interface (2)

[Java Code for **Transfer** Interface]

```
package AccountManagement;
// provided interfaces;

public interface Transfers {
    public Account create(Customer owner, Money balance,
                          AccountNumber account_id);
    public void deposit(Money amount, String reason);
    public void withdraw(Money amount, String reason);
    public bool terminate(AccountNumber account_id, String reason);
}
```

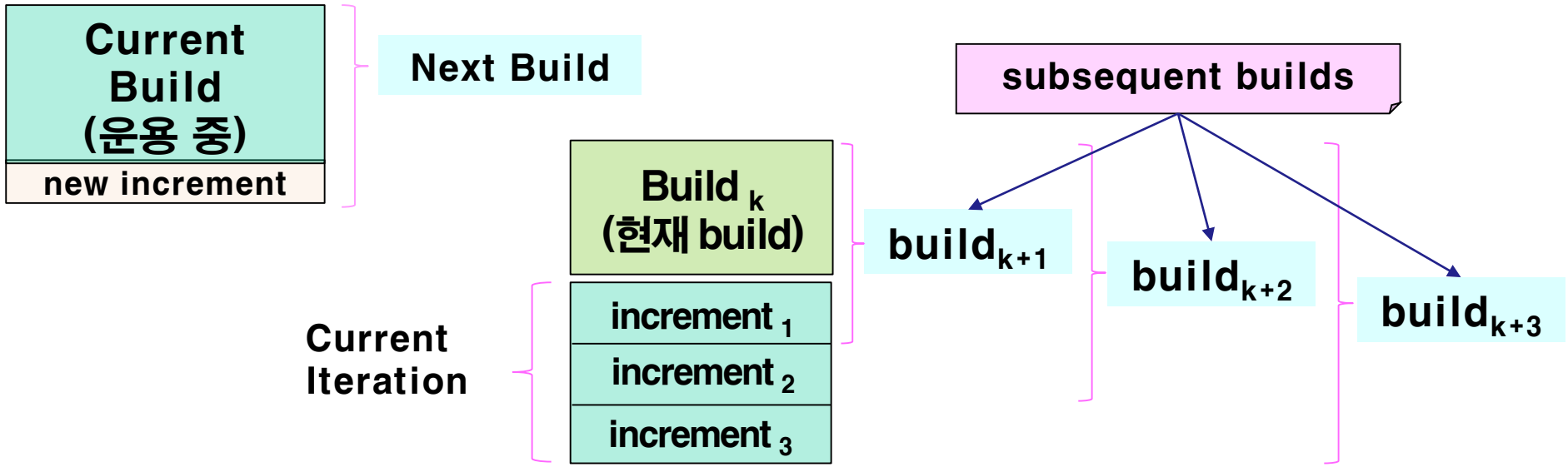
Architecture Description

- 정의
  - 구조적으로 중요한 산출물을 표현하는 구현 모델에 대하여 구조적 관점을 기술한 것

- 구조적으로 중요한 산출물
  - 서브시스템 구성
    - 서브시스템 설정 내역
    - 서브시스템의 인터페이스
    - 서브시스템 사이의 종속 관계
  - 중요한 컴포넌트
    - 중요한 설계 클래스를 구현한 컴포넌트
    - 범용 설계 메커니즘을 구현한 컴포넌트

Integration Build Plan(1)

- Build란?
  - 시스템의 일부로서 실행 가능한 버전(version)
    - Cycle 수행 결과 = s/w product
    - Iteration 수행 결과 = 1개 이상의 increments -> 1개 이상의 builds
  - Build는 통합 테스트를 거쳐서 완성되며, 버전 관리가 필요



Integration Build Plan(2)

- 점진적 통합(Incremental Integration)
  - 정의: 반복적 개발절차에서의 시스템 통합을 지칭
  - 관리하기 쉬운 규모의 increment를 대상으로 통합
    - 하나의 iteration에서 구현할 기능이 너무 복잡하면, 일련의 builds를 설정하여 통합 대상의 크기를 작게 조정

- 점진적 접근방법의 장점
  - 시스템을 부분적으로 완성하여 가동
    -> 시스템이 작동하는 모습을 이해 당사자에게 보여줄 수 있음
  - 기존의 build에 조금씩 추가 -> 오류 발견이 용이
  - 보다 완전한 통합테스트가 가능

Workers for Implementation(1)

- 구조 설계자
  - 다음 사항에 대한 책임
    - 구현 모델의 무결성
    - 올바르고, 모순이 없으며, 읽기 쉬운 구현 모델을 보증
    - 최상위 서브시스템의 구성
      ※ 대형 시스템에서는 별도의 책임자를 둘 수 있음
    - 구현 모델의 구조
    - 컴포넌트의 노드 배치
  - 구현 모델의 구성 요소에 대한 세부 내역 작성은 구조 설계자의 책임이 아님

Workers for Implementation(2)

- 컴포넌트 엔지니어
  - 컴포넌트의 구현 -> 코딩 및 소스 파일의 유지
  - 개별 서브시스템의 무결성 유지
    - 서브시스템의 구성 요소가 올바른가?
    - 서브시스템 사이의 종속관계는 잘 유지되고 있는가?
    - 서브시스템이 제공하는 인터페이스는 제대로 구현되었는가?
  - 설계 클래스와 설계 서브시스템의 담당자가 이에 대응하는 컴포넌트 및 서브시스템을 담당하는 것이 적절

- 시스템 통합자
  - 시스템 통합에 대한 책임
  - Integration Build Plan 수립
  - 통합 계획에 따른 시스템 통합 작업

Workflow in Implementation(1)

1. Architectural Implementation
    1.1 Identifying Architecturally Significant Components
        1.1.1 Identifying Executable Components and Mapping them onto Nodes

2. Integrate System
    2.1 Planning a Subsequent Build
    2.2 Integrating a Build

3. Implement a Subsystem
    3.1 Maintaining the Subsequent Contents

Workflow in Implementation(2)

4. Implement a Class
   - 4.1 Outlining the File Components
   - 4.2 Generating Code from a Design Class
   - 4.3 Implementing Operations
   - 4.4 Making the Component Provide the Right Interface

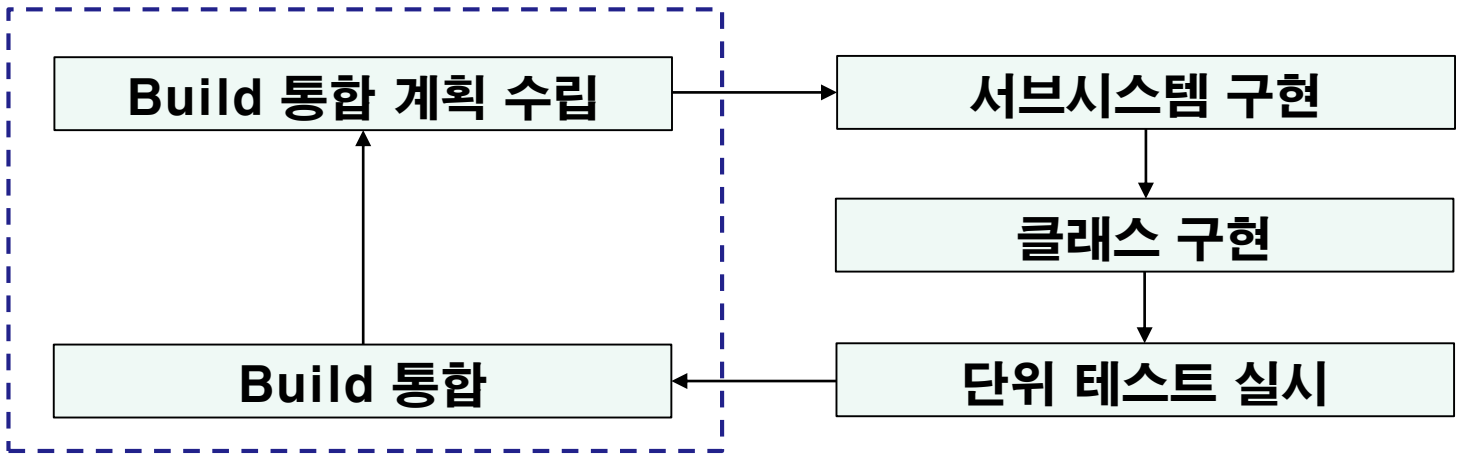
5. Perform Unit test
   - 5.1 Performing Specification Tests
   - 5.2 Performing Structure Tests

Integrate System

❑ 입출력 자료

- 입력 자료: 보충 요구서, 유즈 케이스 모델, 설계 모델, 직전까지의 builds에 대한 구현 모델
- 출력 자료: Integration Build Plan, 후속 builds에 대한 구현 모델

❑ 작업 개요



Generating Code from a Design Class

- 작업 개요
  - 오퍼레이션에 대한 signatures를 코딩
    - 반환 값의 자료 형
    - 함수 이름
    - 매개변수 목록
  - 연관 관계 및 aggregation 관계의 구현
    - 매우 미묘한 문제
    - 프로그래밍 언어에 따라 구현 방법이 상이
    - 일반 원칙: 연관된 객체를 찾아갈 수 있도록 구현해야 함
    - 속성 또는 연관 클래스를 사용하여 구현

Implementing Operations

- Method란?
  - 정의: 오퍼레이션을 구현한 것
  - 예: methods in Java, methods in Visual Basic, member functions in C++, …

- 작업 개요
  - 오퍼레이션의 구현에 필요한 작업
    - 적절한 알고리즘의 선택
    - 필요한 자료 구조를 채택
    - 알고리즘을 바탕으로 코딩
  - 설계 작업의 산출물에 포함된 methods는 반드시 활용할 것!

Perform Unit Test

- 입출력 자료
  - 입력 자료: 구현한 컴포넌트 및 인터페이스
  - 출력 자료: 단위 테스트가 끝난 컴포넌트

- 단위 테스트의 종류
  - Specification Testing
    - Black box Testing
    - 메소드가 주어진 임무를 올바르게 수행하는가를 검증
  - Structure Testing
    - White box Testing
    - 메소드 내부가 제대로 구현되었는가를 확인
  - Other Testing: 성능, 메모리 사용량 등에 대한 테스트

Performing Specification tests

❑ 작업 개요

- 메소드가 어떻게 구현되었는가는 관심의 대상이 아님
- 선정한 테스트 케이스에 대해서 적절한 결과를 보여주는가에 초점
- 적절한 테스트 케이스의 선정이 매우 중요함
  - 허용 범위 안에 들어가는 정상적인 값
  - 허용 범위의 경계 값
  - 허용 범위 밖에 있는 값
  - 적법하지 않는 자료

Performing Structure Tests

❑ 작업 개요

- 메소드 내부가 제대로 구현되었는가를 검증
- 모든 문장이 적어도 1회 이상 테스트될 수 있도록 테스트 케이스를 선정
- 메소드 내부에서 중요한 경로는 반드시 확인할 것!
  - 가장 보편적으로 사용되는 경로
  - Critical Path(핵심 경로)
  - 알고리즘에서 특이한 경우에 대한 경로
  - 위험도가 높은 작업을 수행하는 경로
- 가능한 한 높은 수준의 Coverage를 달성하도록 노력

Structure Test 예시

```
public class Account {
        private Money balance = new Money(0);
        public Money withdraw(Money amount) {
                if balance >= amount
                then {
                        if amount >= 0
                        then {
                                try {
                                        balance = balance – amount;
                                        return amount;
                                }
                                catch (Exception exc) {
                                        // Deal with failures reducing
                                        // the balance
                                }
                        }
                        else { return 0; }
                else { return 0; }
        }
}
```

테스트 케이스 예시

case	인출금액	계좌잔고
1	50원	100원
2	-50원	10원
3	50원	10원
4	예외를 발생시키는 테스트 자료	

코드 검사(Code Inspection)

❑ 작업 개요

- 소프트웨어의 오류를 발견하여 제거함으로써 높은 품질의 소프트웨어를 얻기 위한 활동 중의 하나
- 코드 구현 후, 단위 시험 이전 혹은 이후에 동료 프로그래머들에 의해 수행
- 모든 코드에 대해 검사하는 것이 아니라 핵심 코드를 기준으로 실시
- 오류를 지적하는 것이지 해결 방법을 찾으려는 것이 아님
- 품질 보증 활동(SQA, Software Quality Assurance)의 일부
- 검사팀은 4~7명이 적당하며, 아래와 같은 역할 분담
  저자(author), 사회자(moderator), 낭독자(reader),
  검사자(inspector), 기록자(recorder)
- 코드 검사의 결과는 '결함 통계(Fault Statistics)'로 유지

코드 검사(Code Inspection)

❑ 코드의 오류 유형(Error Type) (1)

- 데이터 오류(DA: data error)
  : 데이터가 다루어지는데 발생하는 오류로 데이터 유형 정의, 변수 선언, 매개 변수에서 나타나는 오류
- 문서 오류(DC: documentation error)
  : 프로그램 구성요소인 선언 부분, 서브루틴, 모듈에 대한 적합하지 않은 주석, 잘못되거나 불필요한 주석을 의미
- 기능 오류(FN: function error)
  : 서브루틴이나 블록이 잘못된 것(what)을 수행하는 오류
- 논리 오류(LO: logic error)
  : 서브루틴이나 블록이 수행하는 방법(how)이 잘못되어 있는 오류

코드 검사(Code Inspection)

❑ 코드의 오류 유형(Error Type) (2)

- 성능 오류(PF: performance error)
  : 프로그램을 수행하며 요구되는 효율성(성능)을 만족시키지 못하는 오류
- 표준 오류(ST: standard error)
  : 과정이나 표현이 표준에 의해 이루어지지 않은 경우
- 기타(OT: error)
  : 문법적인 오류, 사람의 개성이 포함된 경우 등 앞의 유형으로 설명하기 힘든 불투명하고 애매모호한 오류

코드 검사(Code Inspection)

❑ 각 오류에 대한 구분(Error Class)

- 실종(M: Missing)
  : 구성 요소 안에 있어야 할 것들이 없음으로써 생기는 오류
- 실수(W: Wrong)
  : 구성 요소 안에 있으나 잘못이 발견된 오류
- 불필요(E: Extra)
  : 요구되는 것 이상으로 필요 없는 것이 들어간 오류

코드 검사(Code Inspection)

❑ 코드 검사 오류 목록 기록 예

Page 1 of 1

PDM-INSPECTION ERROR LIST

Project: PDM System: PC Date : 2/24/98
Unit: XCOpcCubToSpec.c, XCOpcCubToSpec.h

Page and Line #:	Error Description	Ty	Cl	#
209-238	Extra unnecessary code	FN	E	1
441, 447, 449	Second & Third arguments in	DA	W	1
509, 547, 553				
561, 567, 716	memset function are reversed			
757,1210,1222,1267				
855, 875	if buff != Y, #388-#909 is	LO	M	1
	executed without proper data			