

Korea
Software
Technology
Association



UML2.x 기초 다루기

훈련기간: 2010.01.25 ~ 02.05

강사명: 손재현 -넥스트트리소프트
-jhsohn@nexttree.co.kr

□ 교육 목표 & 특징

- UML2.x의 이해
- 유스케이스 작성
- 객체모델링 이해
- UML2.x의 다양한 다이어그램 이해 및 활용
- 모델링 도구 사용법 습득

- 본 강의는 아래 기술에 대한 이해를 필요로 합니다.
 - 객체지향 언어(Java) 기초
 - 개발프로세스 이해

□ 교육은 매 회 4 시간씩 총 5회에 걸쳐 진행합니다.

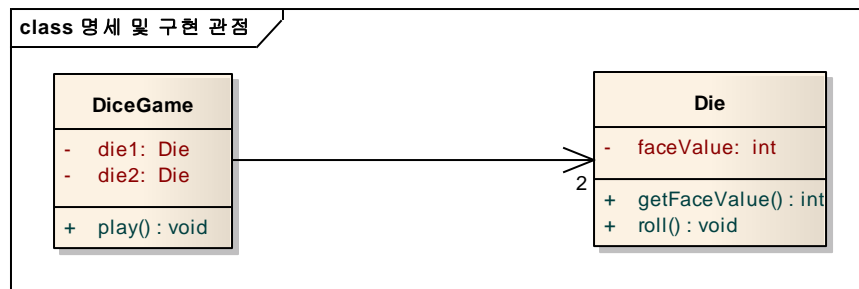
1 일차	2 일차	3 일차	4 일차	5 일차
<ul style="list-style-type: none"> - UML 개요 - UML 소개 - UML 역사 - UML 다이어그램분류 	<ul style="list-style-type: none"> - 구조 다이어그램 - 클래스 - 객체 - 컴포넌트 - 배치 	<ul style="list-style-type: none"> - 행위 다이어그램 - 유스케이스 - 액티비티 - 상태기계 	<ul style="list-style-type: none"> - 상호작용 다이어그램 - 상호작용 Overview - 시퀀스 - 커뮤니케이션 - 타이밍 	<ul style="list-style-type: none"> - 유스케이스 I - 유스케이스 개요 - 유스케이스 내용 - 유스케이스 다이어그램
6 일차	7 일차	8 일차	9 일차	10 일차
<ul style="list-style-type: none"> - 유스케이스 II - 유스케이스 목표수준 - 유스케이스 명세 - 유스케이스 패턴 	<ul style="list-style-type: none"> - 유스케이스 III - 유스케이스 분석기법 - 분석클래스 - 제어클래스 - 실체클래스 	<ul style="list-style-type: none"> - 요구사항 모델실습 I - 유스케이스 - 사용자 시나리오 - 핵심개념 모델 	<ul style="list-style-type: none"> - 요구사항 모델실습 II - 인터페이스 추출 - 유스케이스 분석 - 컴포넌트 식별 	<ul style="list-style-type: none"> - 설계모델 실습 - 컴포넌트 설계 - 유스케이스 설계 - 도메인 모델

- 1 UML 소개
- 2 UML 역사
- 3 13개의 공식적인 다이어그램
- 4 UML 다이어그램 분류
- 5 UML 개발 생명주기
6. 소프트웨어 개발 중요사항

- UML = Unified Modeling Language
- 소프트웨어 개발에 있어 UML의 용도
 - 명세화
 - 가시화: UML은 모델링 언어
 - 아키텍처 설계
 - 구축
 - 테스트 활용
 - 문서화
 - 커뮤니케이션
- 서로 다른 역할을 가진 사람들에게 조금씩 다른 일을 수행
 - 다양한 뷰를 제공
- 설계를 위한 언어
 - 프로그래밍 언어는 설계를 논의하기에는 추상화 레벨이 적절치 않았음
- 공개된 표준(Object Management Group 관할)

- 소프트웨어 개발 시에 서로 다른 목적으로 UML을 사용
- UML을 사용하는 3가지 모드의 특성
 - 스케치(sketch)
 - 시스템의 일부 측면에 대한 설명
 - 의사소통(communication)에 초점
 - 단순한 도구(tool)를 사용
 - 청사진(blueprint)
 - 완전성(completeness)에 초점
 - 코딩을 위한 설계의 세부적인 결정이 끝난 상태
 - 정교한(sophisticated) 도구(tool)를 사용
 - 프로그래밍 언어(programming language)
 - UML 다이어그램이 실행 가능한 코드로 컴파일
 - UML이 소스코드
 - 정교한 도구가 필요

- ❑ 스케치 → 개념적 관점
- ❑ 청사진 → 명세 관점
- ❑ 프로그래밍 언어 → 구현 관점



MDA와 실행 가능한(Executable) UML(1/2)

□ MDA(Model Driven Architecture)의 표준 접근 방법

- UML은 프로그래밍 언어
- OMG 에 의하여 통제

□ MDA의 세 가지 주요 개념

- 컴퓨터 독립적 모델(Computation Independent Model)
 - 도메인 모델 또는 용어사전
- 플랫폼 독립 모델(Platform Independent Model)
 - 특정한 기술과도 독립적인 UML 모델
- 플랫폼 종속 모델(Platform Specific Model)
 - 특정한 실행 환경에 종속적인 UML 모델
 - 모델을 코드로 변환시키기 위하여 PSM을 사용
 - PSM은 UML일 수도 있지만, 항상 그렇지 않음
- PIM -> PSM 변환
 - UML을 프로그래밍 언어로 사용

MDA와 실행 가능한(Executable) UML(2/2)

□ 실행 가능한 UML

- 특징
 - 표준 UML 보다 단순
 - MDA와 거의 비슷한 의미로 사용하지만, 사용하는 용어들은 약간 다름
 - MDA의 PSM이 필요가 없음
- 절차
 - 1단계: MDA의 PIM의미와 동일한 PIM 모델로 시작
 - 2단계: 모델 컴파일러를 사용하여 UML 모델을 배치 가능한 시스템으로 변환
- 모델 컴파일러
 - 특정 프로그래밍 플랫폼으로 변환
- 비 현실적
 - 이를 완전하게 지원하는 도구가 없음
 - UML을 프로그래밍 언어


□ 1980년대

- 객체 지향 개념이 산업계에 도래하기 시작
 - 많은 사람들이 OO를 위한 도식적인 설계 언어를 생각하기 시작

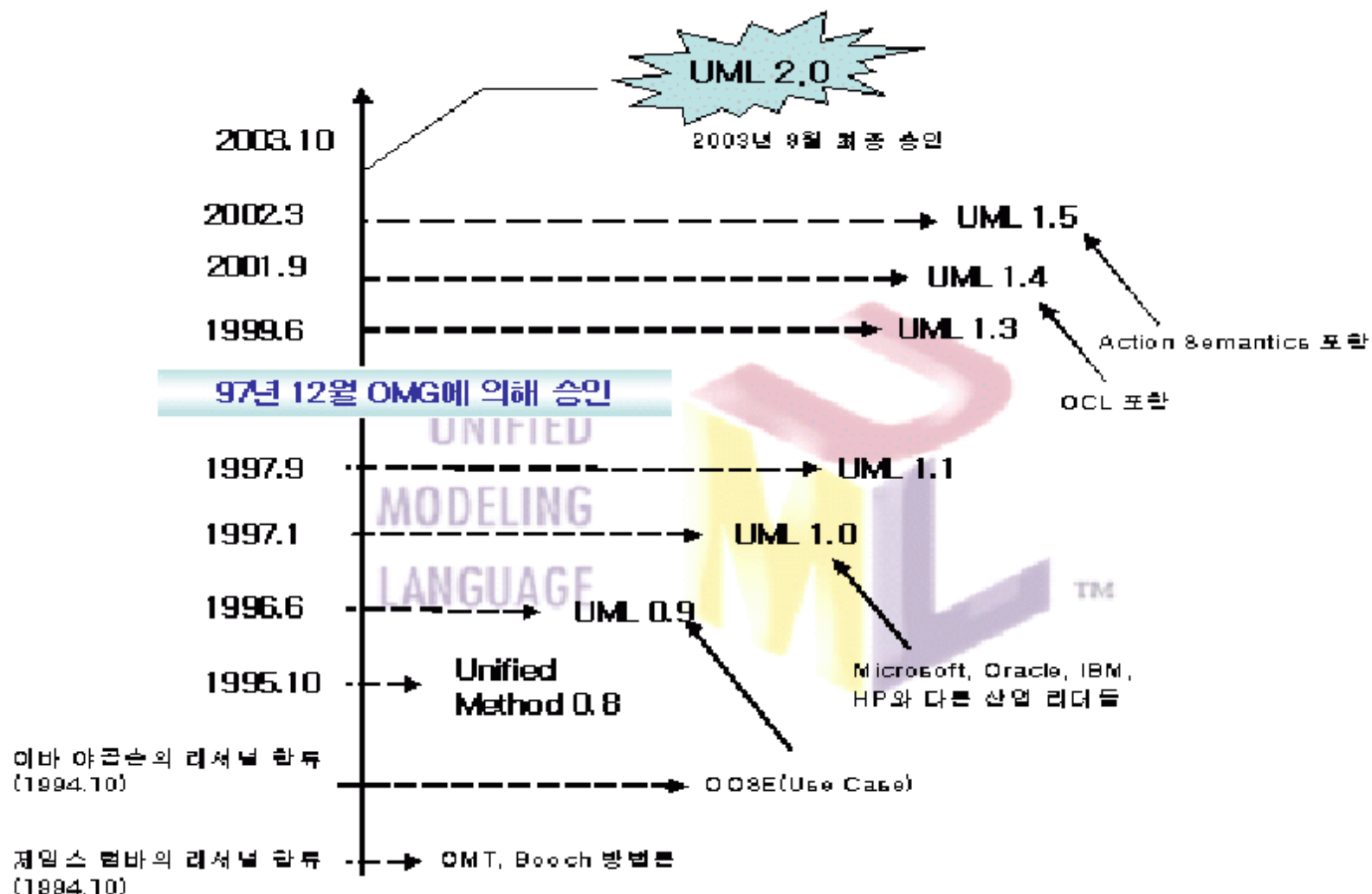
□ 1988년 ~ 1992년

- 같은 개념이 다른 표기법으로 사용되어 혼란을 가중 시킴

	Shlaer & Mellor 방법	Rumbaugh OMT 방법	Booch 방법	Jacobson OOSE 방법
특징	<ul style="list-style-type: none"> • 실시간 시스템 의식 • 치밀한 동적 모델링 지원 	<ul style="list-style-type: none"> • 분석, 설계, 프로그래밍 적용 • 설계 기법이 부족 	<ul style="list-style-type: none"> • Ada용의 방법론이었던 Booch법을 확장 • Hood, OOSD 등 Ada용의 영향이 강함 • 분석을 수행하기에는 부족 	<ul style="list-style-type: none"> • Use Case를 사용하여 시스템에 대한 요구사항을 사용자관점에서 효과적인 모델링 작업 가능 • 분석, 설계의 세부사항이 부족
UML적용	동적 모델링 기법	분석 기법	설계 기법	Use Case 기법



객체지향모델링의 산업계 표준
UML

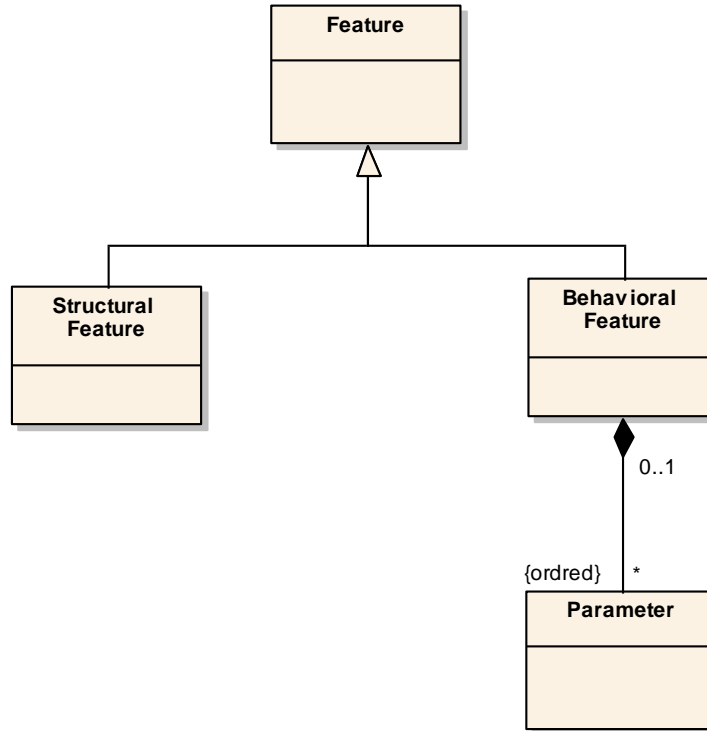


표기법과 메타모델(1/2)

- 현재 UML은 표기법과 메타모델을 정의한다.
- 표기법
 - 모델상에서 볼 수 있는 도식적인 표현을 의미
 - 모델링 언어의 도식적인 문법
 - 예) 클래스 다이어그램 표기법
 - 클래스, 연관, 다중성과 같은 개념들을 표현한다.

□ 메타모델

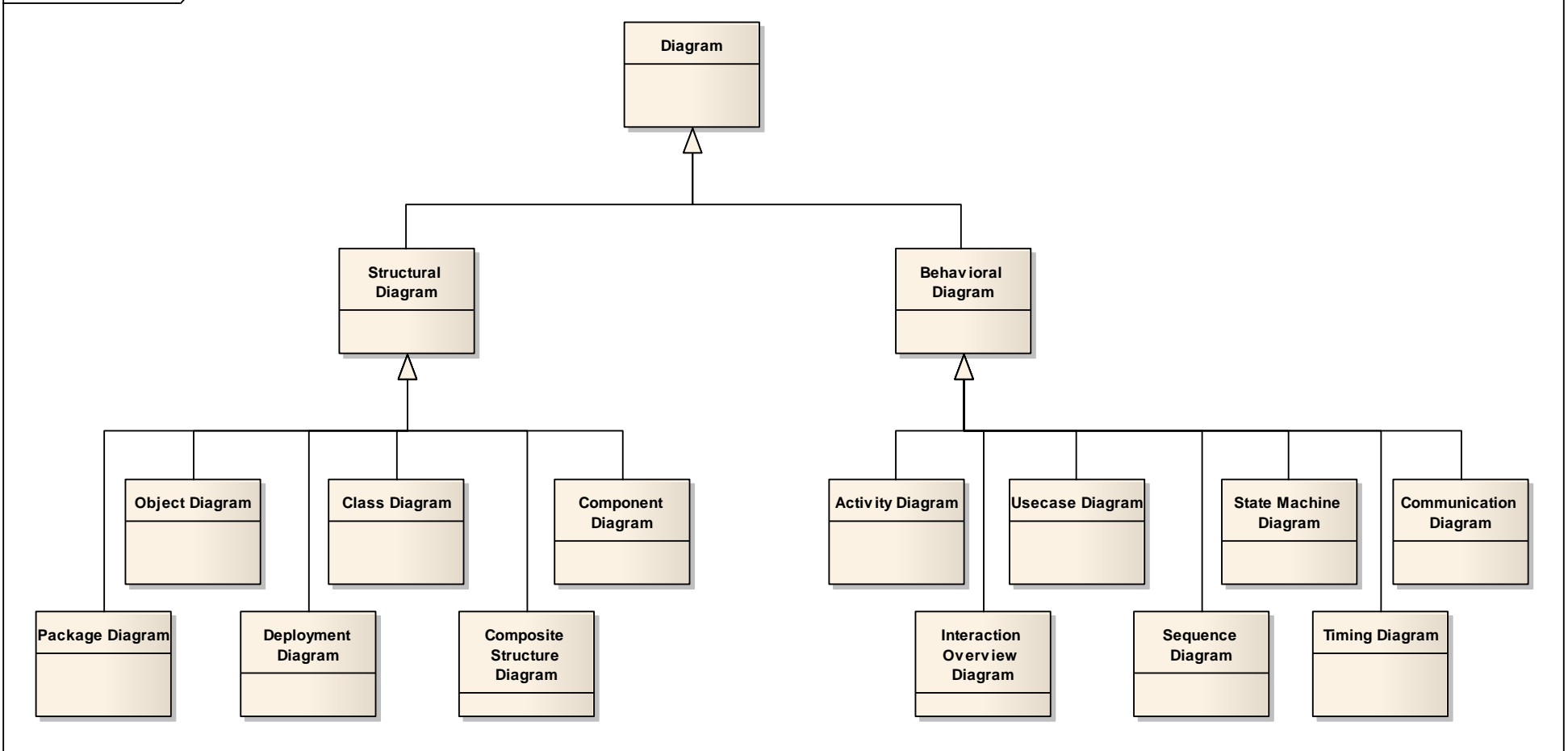
- 모델의 필수 요소와 문법, 구조를 정의
- 보다 엄격한 규칙을 위하여 메타모델을 작성하게 됨
- 주로 클래스 다이어그램을 이용하여 언어의 개념을 정의



UML 메타 모델의 일부

13개의 공식적인 다이어그램

class UML 다이어그램



분류	다이어그램	목적	
구조 다이어그램 (Structural Diagram)	Class	클래스, 특징, 그리고 관계	UML 1
	Object	인스턴스의 구성 예제	(Unofficially) UML 1
	Composite structure	실행시간에서의 클래스 분할	UML 2
	Deployment	산출물을 노드로의 배치	UML 1
	Component	컴포넌트의 구조와 커넥션	UML 1
	Package	컴파일 시간에서의 계층적인 구조	(Unofficially) UML 1
행위 다이어그램 (Behavioral Diagram)	Activity	절차적, 병렬적 행위	UML 1
	Use case	사용자와 시스템간의 상호 작용하는 방법	UML 1
	State machine	이벤트에 의한 객체의 생명주기 동안의 상태변환	UML 1
상호작용 다이어그램 (Interaction Diagram)	Interaction overview	액티비티 다이어그램과 시퀀스 다이어그램의 혼합	UML 2
	Sequence	흐름에 중점을 둔 객체간의 상호작용	UML 1
	Communication	링크에 중점을 둔 객체간의 상호작용	UML 1 (Collaboration)
	Timing	타이밍에 중점을 둔 객체간의 상호작용	UML 2

합법적인 UML 이란?

□ 두 가지 측면에서 고려

- 규범적인(prescriptive) 규칙
 - 규범적인 규칙을 사전에 정의
 - 예) 프로그래밍 언어
- 서술적인(descriptive) 규칙
 - 실제로 사용되는 방법을 관찰함으로써 규칙을 이해
 - 예) 자연어(영어)

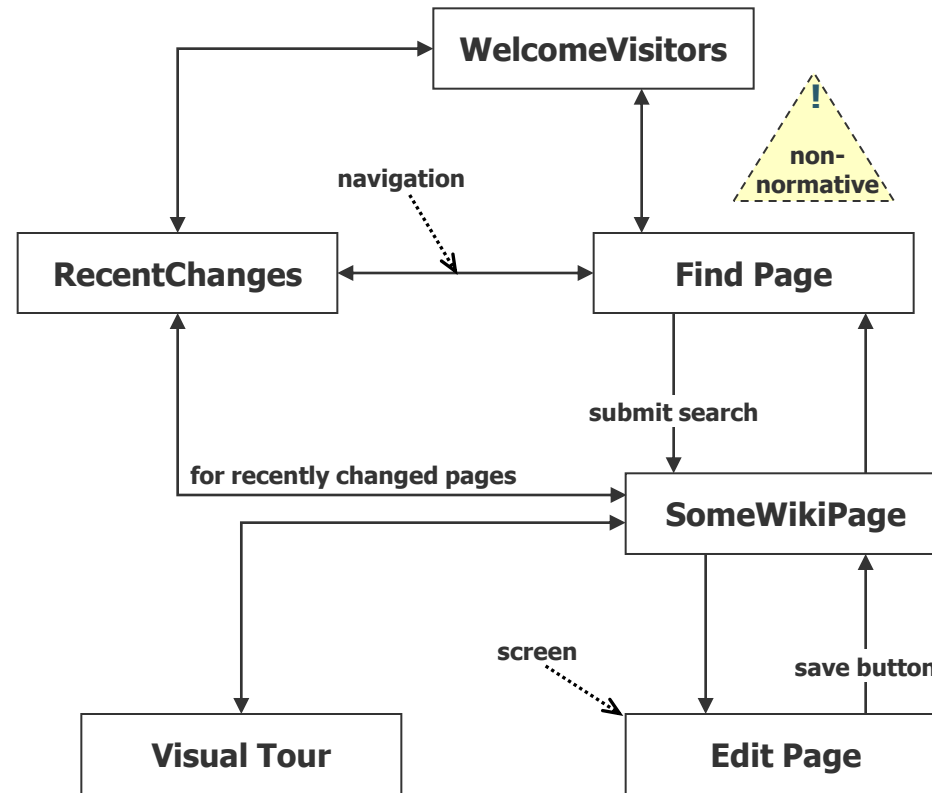
□ UML

- 두 가지 측면을 모두 가짐
- 사람마다 표준을 해석하는 방법이 다를 수도 있음

□ Martin's 의 견해

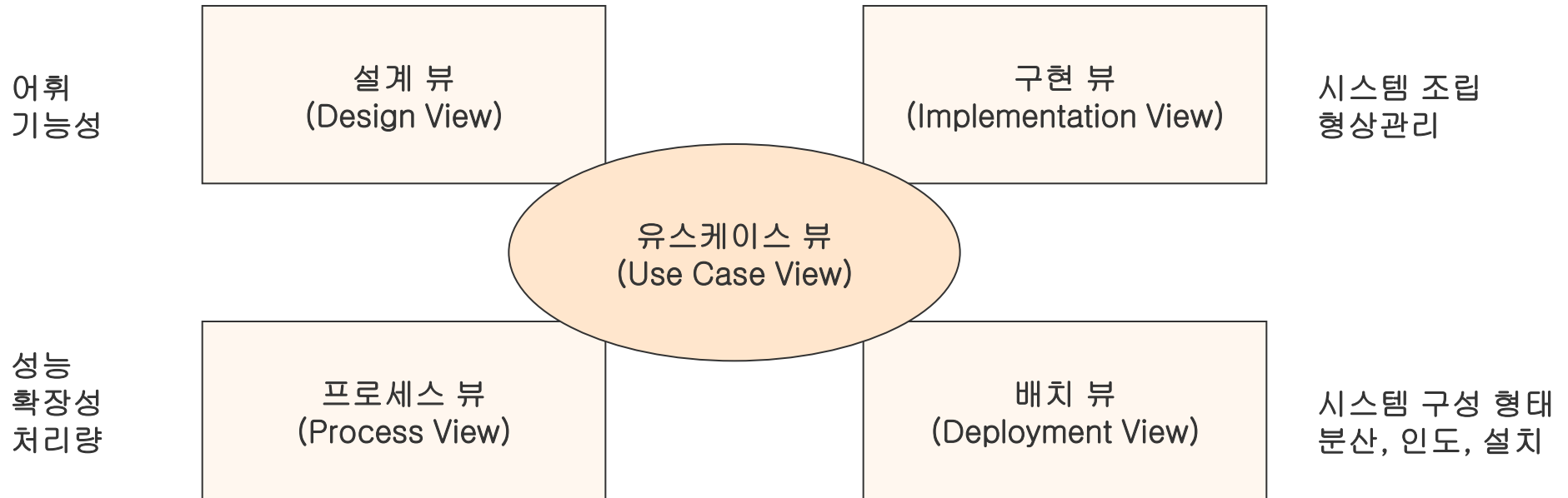
- 대부분의 사람들에게 UML은 서술적인 규칙을 가지고 있음

- UML이 완전하게 모든 유용한 다이어그램을 표현하지는 않음
 - 필요하다고 생각되는 부분에는 목적에 맞는 비-UML 다이어그램을 사용



Wiki의 일부를 표현하는 비정형적인 화면 흐름 다이어그램(<http://c2.com/cgi/wiki>)

□ 4+1 뷰



□ 뷰 설명

뷰의 종류	내 용
유스케이스 뷰 (Use Case View)	시스템 행동을 설명 최종사용자, 분석가, 설계자, 테스트 담당자에게 제공되는 뷰 시스템 아키텍처를 구체화하는 요인들을 명세화
설계 뷰 (Design View)	시스템이 최종사용자에게 제공해야 할 서비스를 표현 문제 영역과 해법의 어휘를 형성하고 있는 Class, Interface, Collaboration으로 구성
프로세스 뷰 (Process View)	시스템의 성능, 신축성, 처리 능력을 표현 시스템의 동시성과 동기화 메커니즘을 형성하고 있는 Thread와 Process로 구성
구현 뷰 (Implementation View)	시스템 배포의 형상관리 표현 물리적인 시스템을 조립하고 배포하는데 사용되는 Component와 File 들로 구성
배치 뷰 (Deployment View)	시스템을 구성하는 물리적 부분의 분산, 인도, 설치 표현 H/W 형태를 형성하는 Node로 구성

□ 프로세스

- OOAD(Object Oriented Analysis and Design) 시각적인 모델링 언어(UML)과 더불어 소프트웨어 개발을 주도
- UML과는 달리 합의에 도달하지 못함
- 모델링 기법을 프로세스와 분리하여 고려할 경우 의미가 없음

반복적인 프로세스와 폭포수 형태의 프로세스

□ 반복적인(iterative) 프로세스 vs 폭포수 형태(waterfall)의 프로세스

- 잘못 사용되는 개념
 - 반복적인 스타일-> 최신 유행
 - 폭포수 형태의 스타일-> 고전
- 프로젝트 크기를 나누는 차이 (N vs 1)

□ 폭포수 형태의 프로세스

- 활동을 기반으로 하여 프로젝트를 분할
 - 요구사항 분석, 설계, 코딩, 테스트

□ 반복적인 프로세스

- 기능의 하위집합으로 프로젝트를 분할
 - 몇 개의 반복(iteration)으로 분할

예측 계획법과 적응 계획법

□ 예측에 대한 욕구

- 무엇보다 비용에 대한 예측이 절실함
- 폭포수 형태의 프로세스가 존재하고 있는 이유

□ 소프트웨어 프로젝트가 예측 가능한가?

- 요구사항 분석 문제가 핵심
- 요구사항 변경에 대한 관리가 중요

□ 적응 가능한 계획

- 요구사항의 변경은 피할 수 없다고 생각
- 예측 가능한 계획은 무의미
- 반복적인 프로세스 필요

□ 기민한 프로세스의 예제

- XP(eXtreme Programming), Scrum, FDD(Feature Driven Development), Crystal, DSDM(Dynamic System Development Method)

□ 특징

- 강한 적응성
- 사람 중심(people oriented)의 프로세스
- 프로젝트의 성공여부는 프로젝트 팀원의 능력과 팀워크에 좌우
 - 사용하는 프로세스의 종류나 도구의 종류는 부차적인 문제
- 아주 짧은, 타임 박스화된 반복(iteration)을 사용
- Time Box 사용시
 - 시간을 늘리지 않고, 기능 축소
 - 규칙적인 개발흐름을 지킬 수 있고, 기능의 우선순위 파악이 용이

RUP(Rational Unified Process)

□ RUP 특징

- 프로세스를 설명하는 어휘와 느슨한 구조를 제공하는 프로세스 프레임워크
- 반복적으로 개발
- 요구 사항 관리
- 컴포넌트 아키텍처 사용
- 시각적 모델링
- 품질의 지속적으로 관리
- 변경 사항 관리

□ RUP 단계

- 도입(inception): 프로젝트에 대한 초기 평가를 실시
- 발단(elaboration): 주요 유스케이스를 식별하고, 시스템의 아키텍처를 안정화
- 구축(construction): 시스템 구축을 수행
- 전이(transition): 배치, 사용자 교육과 같은 활동들을 수행

프로젝트에 적당한 프로세스 맞추기

- 모든 프로젝트에 적합한 하나의 프로세스는 존재하지 않음
 - 프로젝트 개발은 다양한 요인에 의존
 - 예; 구축 시스템 종류, 사용하는 기술, 위험의 특성 등
 - 프로세스를 사용하면서, 이를 조정하여 점차적으로 프로젝트에 적용
 - 적게 가지고 시작
- 반복주기 소급(iteration retrospective)
 - 유지(Keep): 계속적으로 수행하고 싶은 것들
 - 문제점(Problems): 제대로 동작하지 않았던 것들
 - 시도(Try): 개선 시키고 싶은 것들

UML을 프로세스에 맞추기(1/2)

□ 요구사항 분석

- UML을 사용하는 가장 중요한 목적은 사용자 및 고객과 의사 소통
 - 유스케이스: 사용자와 시스템의 상호작용
 - 클래스 다이어그램: 개념적인 관점을 기술
 - 액티비티 다이어그램: 조직의 작업흐름
 - 상태 다이어그램: 중요한 생명주기를 갖는 개념을 표현

□ 설계

- 보다 기술(technical)적인 내용들을 보다 상세하게 다이어그램 상에 표현
 - 클래스 다이어그램: 클래스와 이들의 관계를 소프트웨어 관점에서 기술
 - 시퀀스 다이어그램: 유스케이스의 주요 시나리오를 기술
 - 패키지 다이어그램: 큰 스케일의 소프트웨어의 구성
 - 배치 다이어그램: 소프트웨어의 물리적인 레이아웃
 - 상태 다이어그램: 복잡한 상태를 가진 클래스의 상태

UML을 프로세스에 맞추기(2/2)

□ 문서화

- UML은 시스템의 전반적인 문서화에 적합
- 전반적인 시스템의 상세한 문서화는 지양
- 상세한 문서화는 코드로부터 생성
- 전반적으로 시스템 개발에 유용한 정보를 제공하는 것을 위주로 문서화 수행
 - 예) 패키지 다이어그램: 시스템의 논리적인 로드맵
 - 예) 배치 다이어그램: 높은 수준의 물리적인 그림

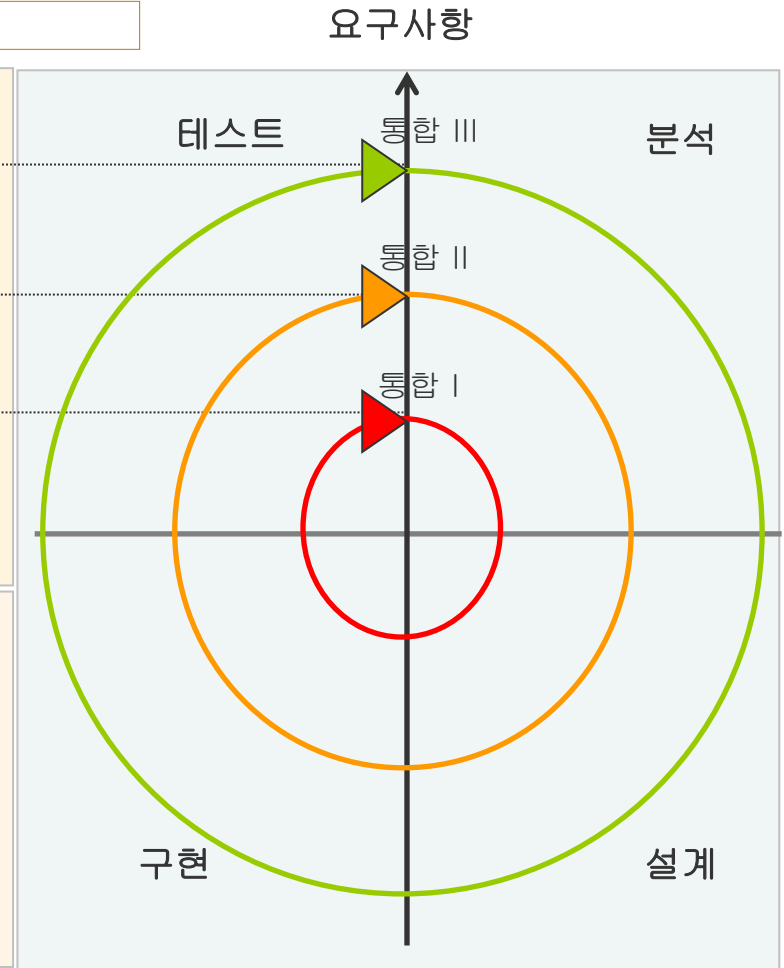
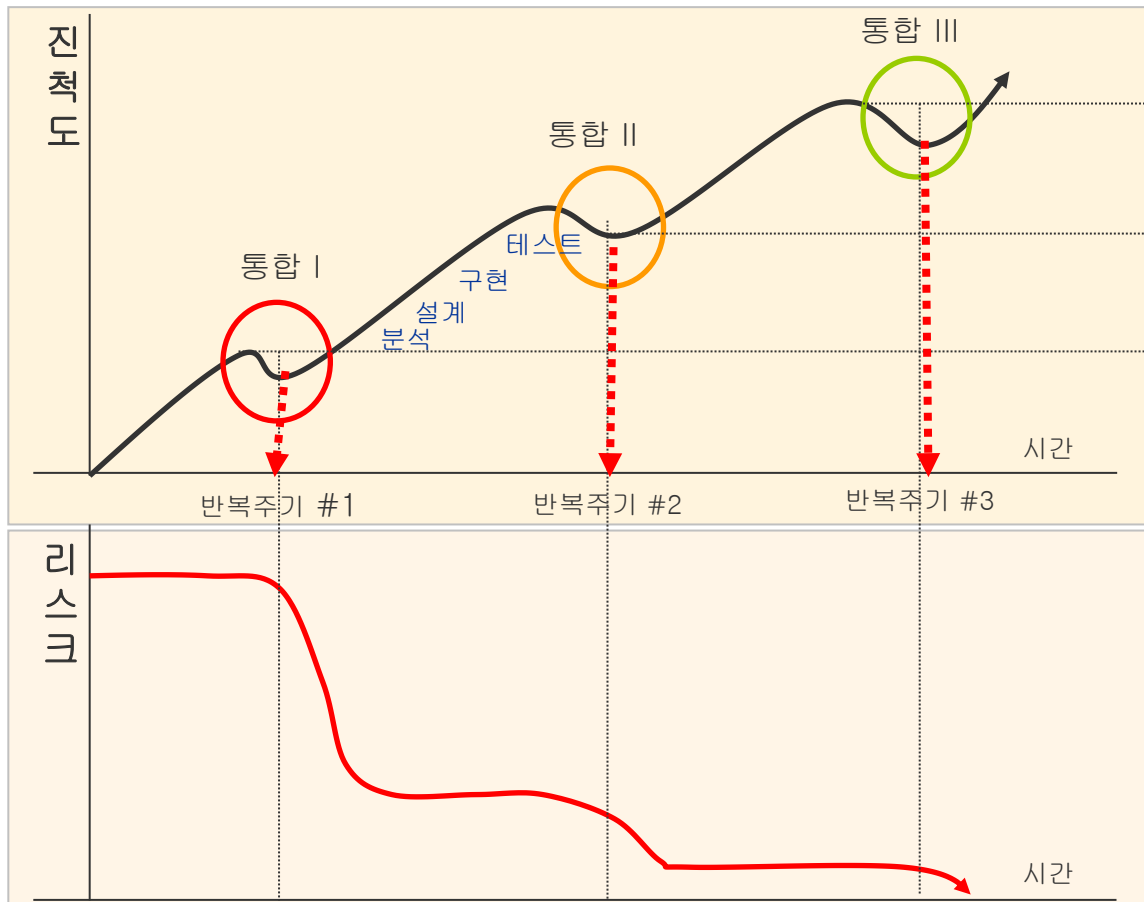
□ 레거시(legacy) 코드를 이해하기

- 익숙하지 않은 코드에 대하여 이해할 수 있도록 도와줌
- 도구를 사용하여 시스템의 상세한 다이어그램을 생성

□ Martin의 선택

- 프로젝트의 성공을 원한다면 반복적인 형태의 개발 프로세스를 채택
 - 위험 요소를 미리 발견
- XP에 대하여 매우 긍정적

개발 프로세스 사상



□ 개발 Process 고려 사항

프로세스	설 명
유스케이스 주도	System에 요구되는 행동을 파악 System Architecture 검증, 확인 및 Test Project 관련자의 의사소통 (Use Case 관련 주요 산출물 활용)
아키텍처 중심	개발중인 System의 개념화, 구축, 관리 진화(변화) 내용을 파악하고 수행 (System Architecture 관련 주요 산출물 활용)
반복/점진적 프로세스 중심	반복 프로세스는 실행 배포판을 관리 점진적 프로세스는 System Architecture를 지속적으로 통합하고 개정 배포판을 작성

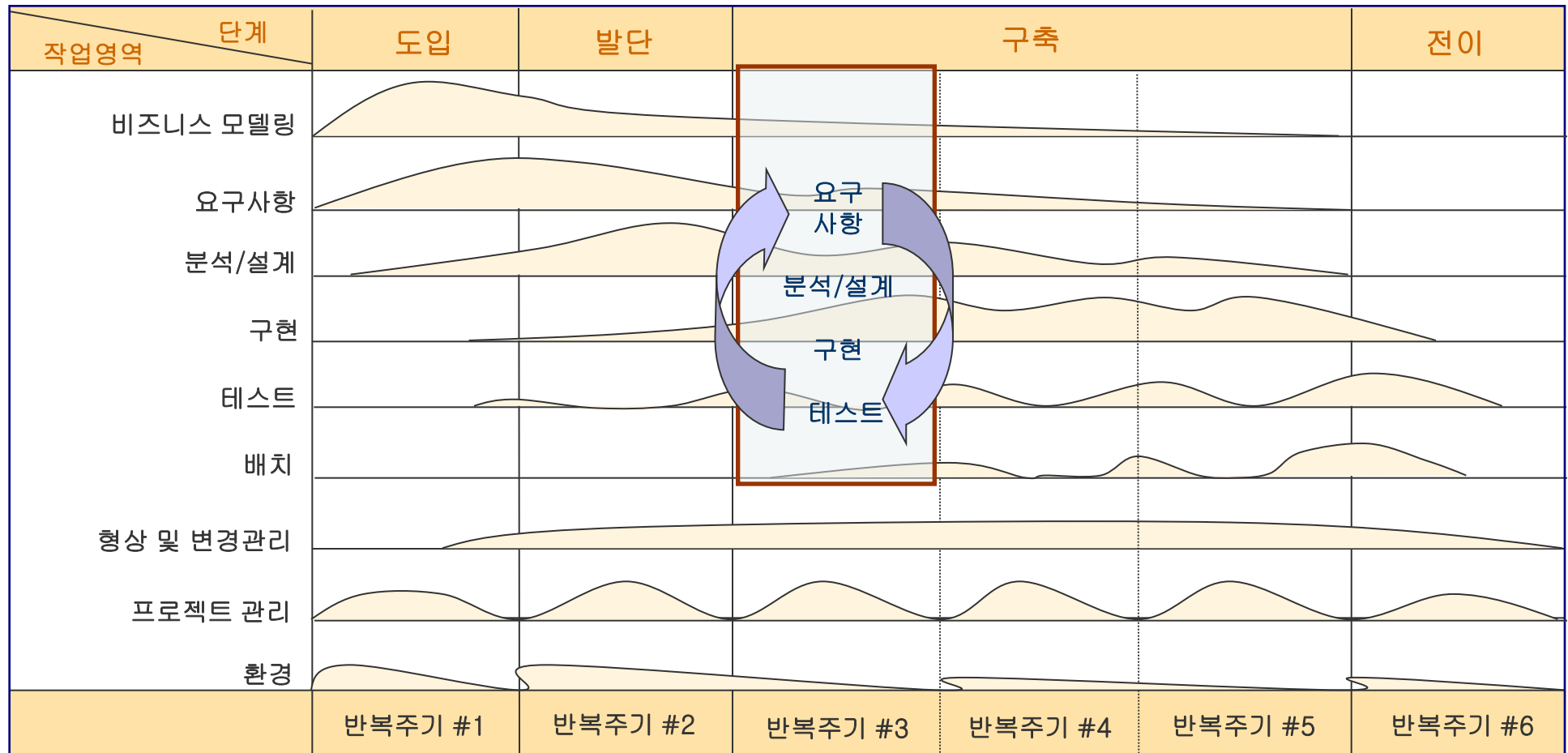
□ UML은 개발 Process에 독립적

□ S/W 개발 생명 주기 단계

- 각 단계는 반복적으로 수행

단계	설 명
도입 (Inception)	개발의 시작점으로써 대상 요소들을 정의 발단 단계로 진입할 수 있는 충분한 근거 파악
발단 (Elaboration)	제품 Vision과 Architecture를 정의 System의 요구 사항의 명료화, 우선 순위 결정, 기준선 설정 및 Test 기준 설정 요구 사항의 기능적 행동과 비 기능적 행동을 명세화
구축 (Construction)	S/W의 작성 및 실행 Architecture 기준선으로부터 전이의 준비 단계 Project에 대한 요구 사항과 평가 기준의 재 검사 위험 요소들을 제거하기위한 자원의 할당
전이 (Transition)	S/W의 사용자 전달 System의 지속적인 개선, 결함 제거 배포판에 새로운 특성 추가

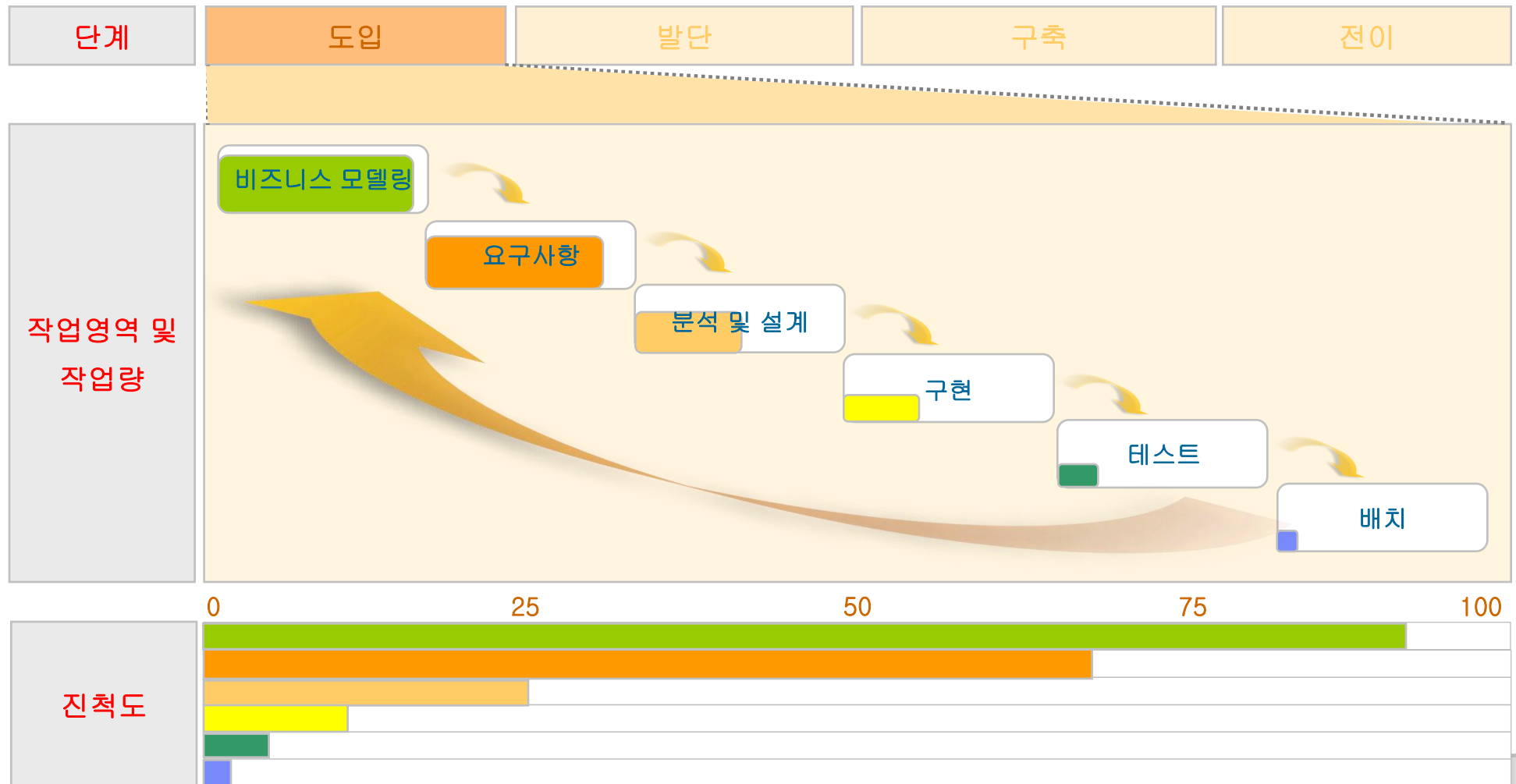
□ 개발 프로세스



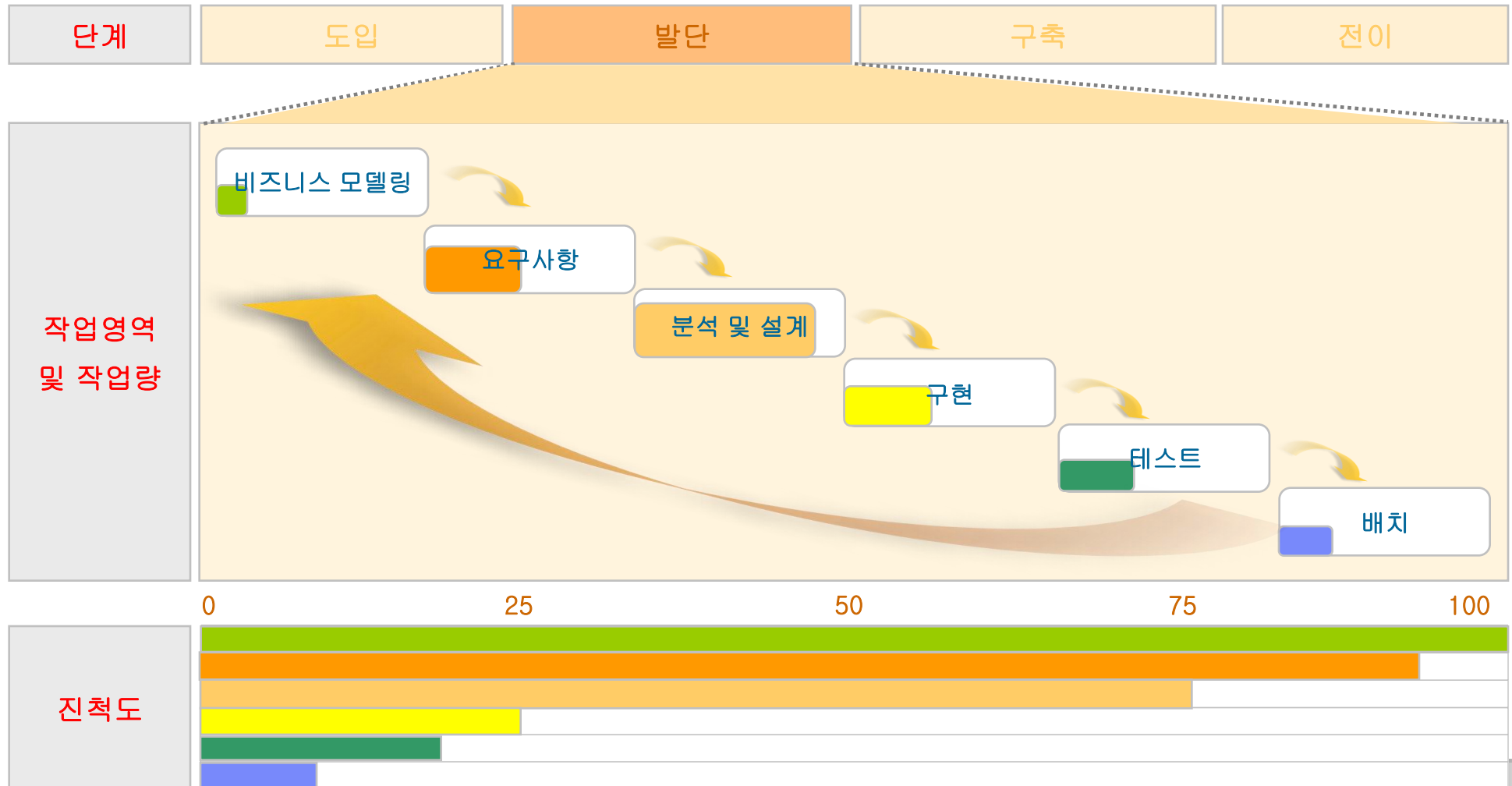
□ 단계별 작업수행

단계(기간)	도입(10%)	발단(30%)	구축(50%)	전이(10%)
	반복 #1	반복 #2,#3	반복 #4,#5,#6	반복 #7
수행영역	<div>비즈니스 모델링</div> <div>요구사항</div>	<div>요구사항</div> <div>분석 및 설계</div> <div>구현</div> <div>테스트</div> <div>배치</div>	<div>요구사항</div> <div>분석 및 설계</div> <div>구현</div> <div>테스트</div> <div>배치</div>	<div>테스트</div> <div>배치</div>
관리영역	<div>형상 및 변경관리</div> <div>프로젝트 관리</div> <div>환경</div>			

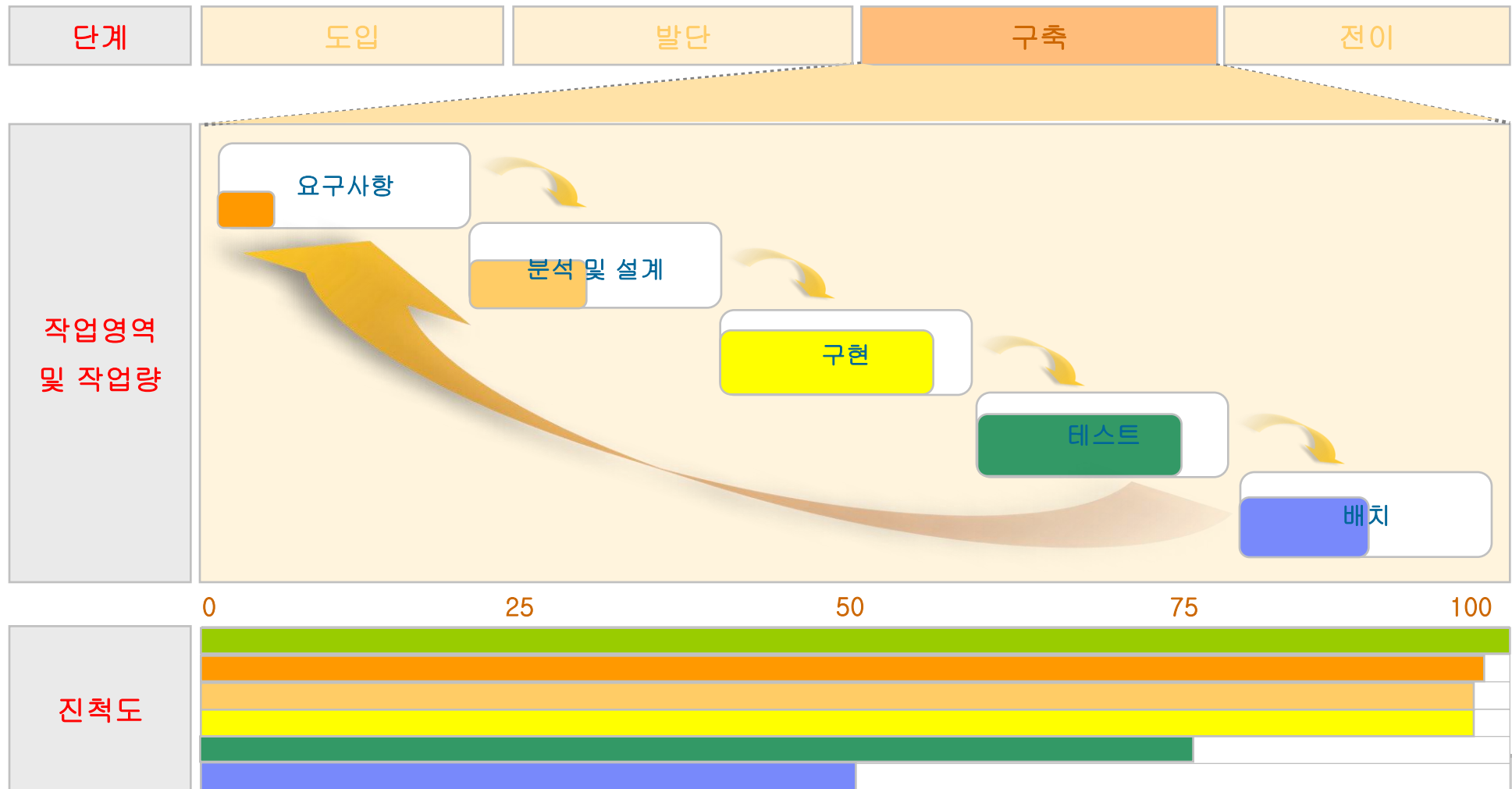
□ 도입(Inception) 단계



□ 발단(Elaboration) 단계



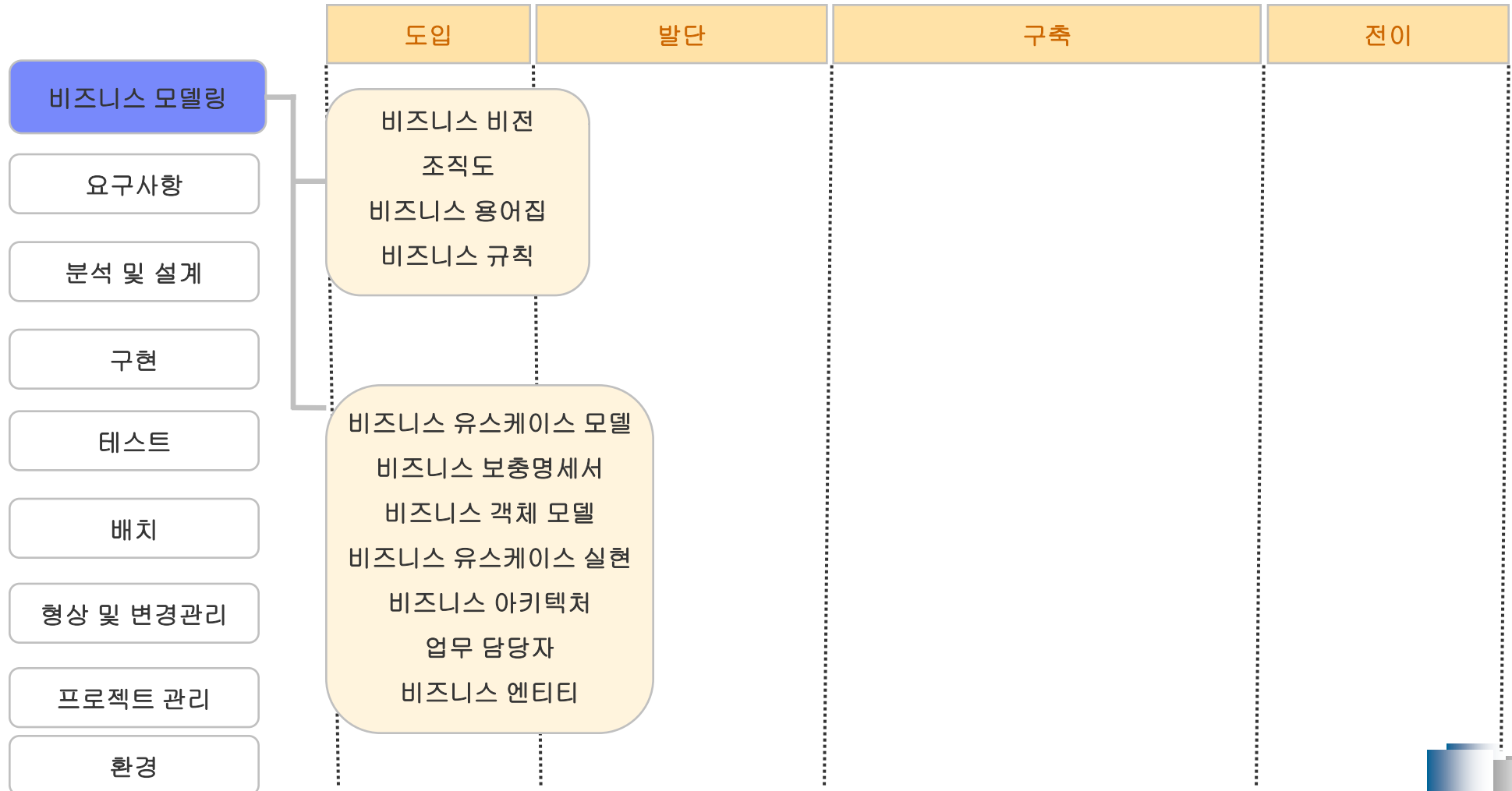
□ 구축(Construction) 단계



□ 전이(Transition) 단계



□ 비즈니스 모델링



□ 요구사항



□ 분석 및 설계



□ 구현



□ 테스트

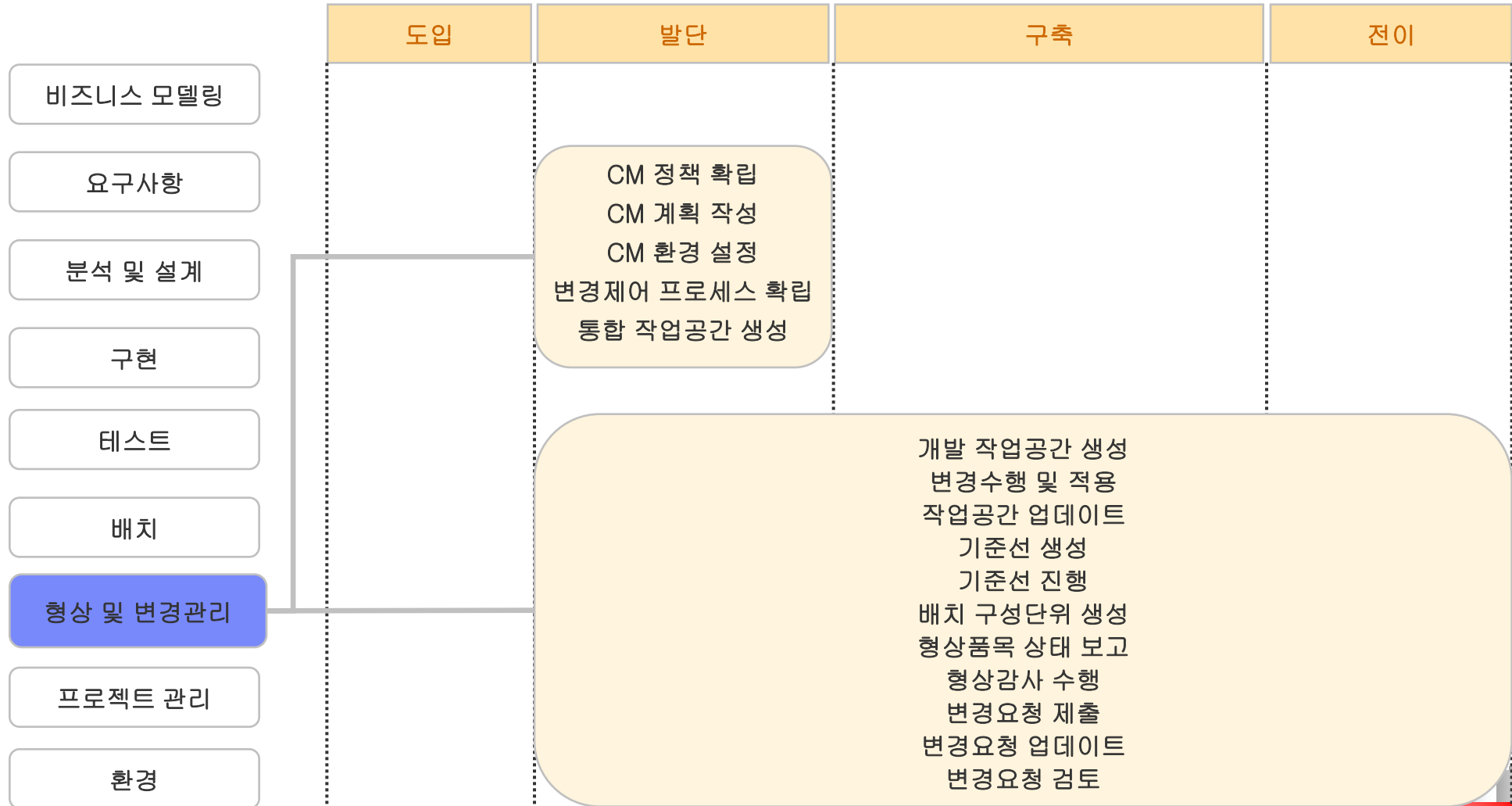


□ 배치



형상/변경관리 산출물

□ 형상 및 변경관리



□ 프로젝트 관리

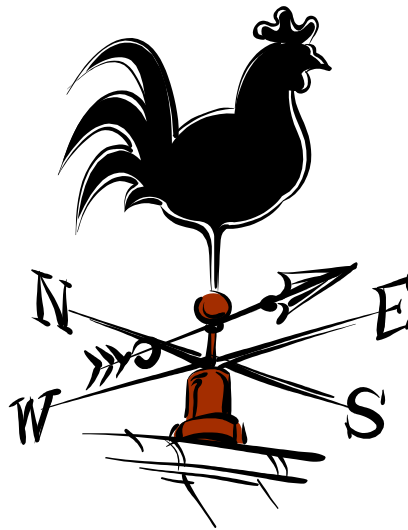


□ 환경



□ 중심적인 4가지 중요사항

- 의사소통 *Communication*
- 단순성 *Simplicity*
- 피드백 *Feedback*
- 자신감 *Courage*



- 질책을 두려워하지 않고 프로젝트에 대해 자유롭게 논의할 수 있는 장을 마련하는 것이 중요
 - 작업 견해 논의
 - 짝 프로그래밍 *Pair programming*
 - 반복적인 계획을 수월하게 수행할 수 있는가?

- ❑ 현재 프로젝트들이 매우 복잡한 것이 일반적이기 때문에 직관적이지 못하는 경우가 발생
- ❑ 고객이 원하는 사항을 잘 전달하는 것이 중요
 - 가능한 단순한 설계, 기법, 알고리즘 등

□ 프로젝트의 진행과정에서 발생하는 매우 중요한 과정

- 코드 테스트
- 고객의 요구사항
- 부분적인 반복작업 및 결과물 인도 *Delivery*
- 짝 프로그래밍 *Pair programming* / 지속적인 코드 검토

모든 코드에 대한
단위테스트 매일 수행

시스템 품질 상태 및
가치에 대한 피드백

시스템 인도 전날 밤
중대한 오류 발견 감소

- 문제에 대해 바른 길이 무엇인지 적극적으로 판단하는 것에 관한 사항
- 수준이하의 코드를 가진 프로젝트에 대한 경험이 있는가?
 - 관리자나 개발자가 코드를 변경하면 시스템에 어떤 영향을 미치는지 걱정하는 상황이라면?
 - 단위 회기 테스트 *Unit Regression Testing*을 통해 코드에 대한 믿음을 확보
 - 리팩토링 *Refactoring*을 통한 시스템의 재구조화
 - 테스트 작업은 코드에 대한 자신감을 갖게 해줌

Martin Fowler – Refactoring

코드에 좋지 않은 부분이 보이면, 리팩토링해라!

좋지 않은 코드를 버릴 수 있는 **자신감***Courage*을 가지고 있어야 한다. 문제 있는 코드는 재작성하는 것이 좋다.