



## Chapter 3. 정리

---

1. 소프트웨어공학의 프로세스 흐름

2. 소프트웨어공학 프로세스 패턴

# Process





# 제 4장. 프로세스 모델

---

March. 2018

Young-gon, Kim

ykkim@kpu.ac.kr

Department of Computer Engineering

*K*orea *P*olytechnic *U*niversity



# Topics covered

---

## ◆ 규범적 프로세스 모델.

- 폭포수, 점증적 프로세스, 진화적 프로세스, 동시 모델.

## ◆ 특수한 프로세스 모델

- 컴포넌트-기반 개발, 정형 방법론, 관심 지향 소프트웨어 개발

## ◆ 통합 프로세스

- UP(Unified Process)

## ◆ 개인과 팀 프로세스 모델

## ◆ 프로세스 기술



# Process

## ◆ 프로세스 (process)

- 활동 순서의 집합
- 의도한 형태의 결과를 얻기 위한 활동 (activities), 제약조건(constraints), 그리고 자원 (resources)을 포함하는 일련의 과정

## ◆ 라이프사이클 (life cycle)

- [ 생물 ] 한 종의 구성원이 주어진 발생 단계에서 시작해 뒤이은 세대에서 같은 발생 단계의 시작에 이르기까지 겪는 일련의 변화.
- 소프트웨어 개발 프로세스 -> 소프트웨어 라이프사이클 정의

## ◆ 프로세스가 중요한 이유

- 일련의 활동들에 대해서 일관성과 구조를 강요하기 때문
- 활동들의 이해, 통제, 실험, 개선 등을 도움
- 우리의 경험을 획득하여 다른 사람에게 전달 가능.



# Process

## ◆ 프로세스의 특징

- 주요한 프로세스 활동 (major process activity) 을 규정
- 자원을 이용하고, 일정과 같은 제약 조건 을 지킴
- 중간 프로덕트와 최종 프로덕트를 생산 함
- 서로 연결되어 있는 서브프로세스로 구성
  - 각각의 서브프로세스가 고유의 프로세스 모델을 갖도록 프로세스 계층 구조로 정의되거나 조직 될 수 있음
- 진입과 출구 기준 (entry and exit criteria) 을 가짐
  - 각 프로세스의 활동은 활동의 시작과 끝을 알 수 있도록
- 활동은 순차적으로 구성 되어 있음
  - 하나의 활동이 다른 활동들과 관계를 가지고 수행됨을 명확히 하기 위해
- 모든 프로세스는 각 활동의 목표를 설명하는 일련의 지침을 포함함
- 제약 조건 또는 제어는 활동 , 자원 또는 프로덕트에 적용 되기도함.

# Software Process Model

## ◆ 프로세스 모델링을 하는 이유

- 개발 프로세스 기술 시 공통된 이해 형식
  - 소프트웨어 개발에 포함된 활동, 자원, 제약 조건 등
- 불필요한 내용 및 생략된 부분을 찾는데 도움을 줌
  - 개발 팀간에 존재하는 일관적이지 못하거나 불필요한 내용,
  - 프로세스와 프로세스 구성요소에서 생략된 것들을 찾는 경우
- 고품질의 소프트웨어 개발, 오류의 조기 발견 그리고 예산과 일정 제약조건과 같은 개발 목표를 반영함
  - 모델이 완성되면 개발 팀은 목표를 수행하기에 적절한 후보 활동들을 평가함
- 모든 프로세스는 특정한 상황에 따라 적절히 대응 할 수 있음
  - 프로세스 모델의 구축은 유동적인 상황이 일어날 수 있음을 이해하는데 도움이 됨.



# 1. Prescriptive Model

## ◆ 규범적 프로세스 모델

- 소프트웨어 개발에 질서와 구조 필요
  - 프로세스 정의된 가이드라인에 따라 순차적을 활동
- 규범적
  - 프로세스 요소에 대해 규정을 함
    - ✓ 프레임워크 액티비티, 액션, 태스크, 작업산출물, 품질보증, 변경제어기법
- 프로세스 흐름 규정
  - 프로세스 요소들이 상호연관 짓는 방식.

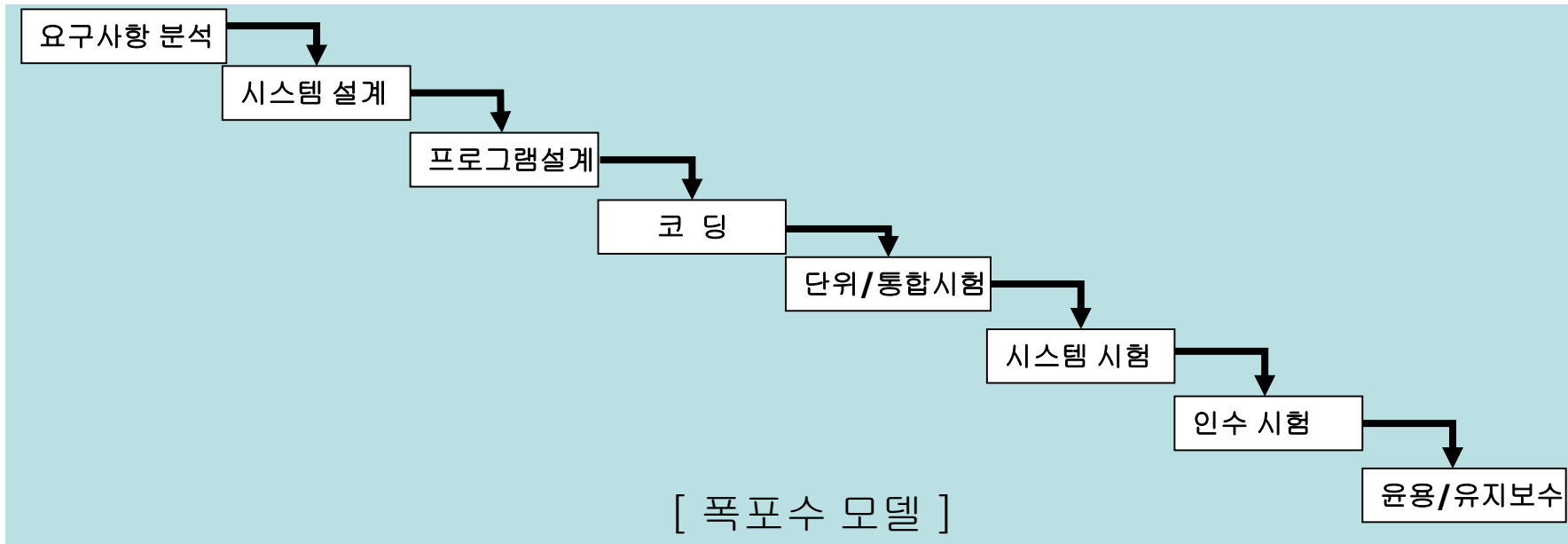


# 1. Prescriptive Model(Waterfall Model)

## ◆ 1.1 폭포수 모델

### ● 고전적 생명주기

- 완전한 소프트웨어를 목표로 고객의 요구사항명세에서 시작하여 계획 수립, 모델링, 구축, 배치순으로 소프트웨어를 개발할 때 유용한 체계적이고 순차적인 방식
- 한 단계에서 다음 단계로 폭포처럼 떨어지는 것과 같이 하나의 개발 단계가 완성되어야 다음 단계가 시작 될 수 있음
- 개발이 진행되는 동안 이루어지는 일들에 대한 상위수준의 관점과 개발자들이 반드시 거쳐야 하는 일련의 과정 을 제안함





# 1. Prescriptive Model(Waterfall Model)

## ◆ 1.1 폭포수 모델

### ● 장점

- 다양한 소프트웨어 개발 활동을 기술 하는데 이용됨
- 개발자들에게 필요한 행동이 무엇인지를 계획 하는데 있어 매우 유용함
- 단순하므로 소프트웨어 개발에 능숙하지 못한 고객들에게 쉽게 설명 할 수있도록 해주며 , 다음 개발단계로 진행하기 위해 필요한 중간 프로덕트를 표시해 줌

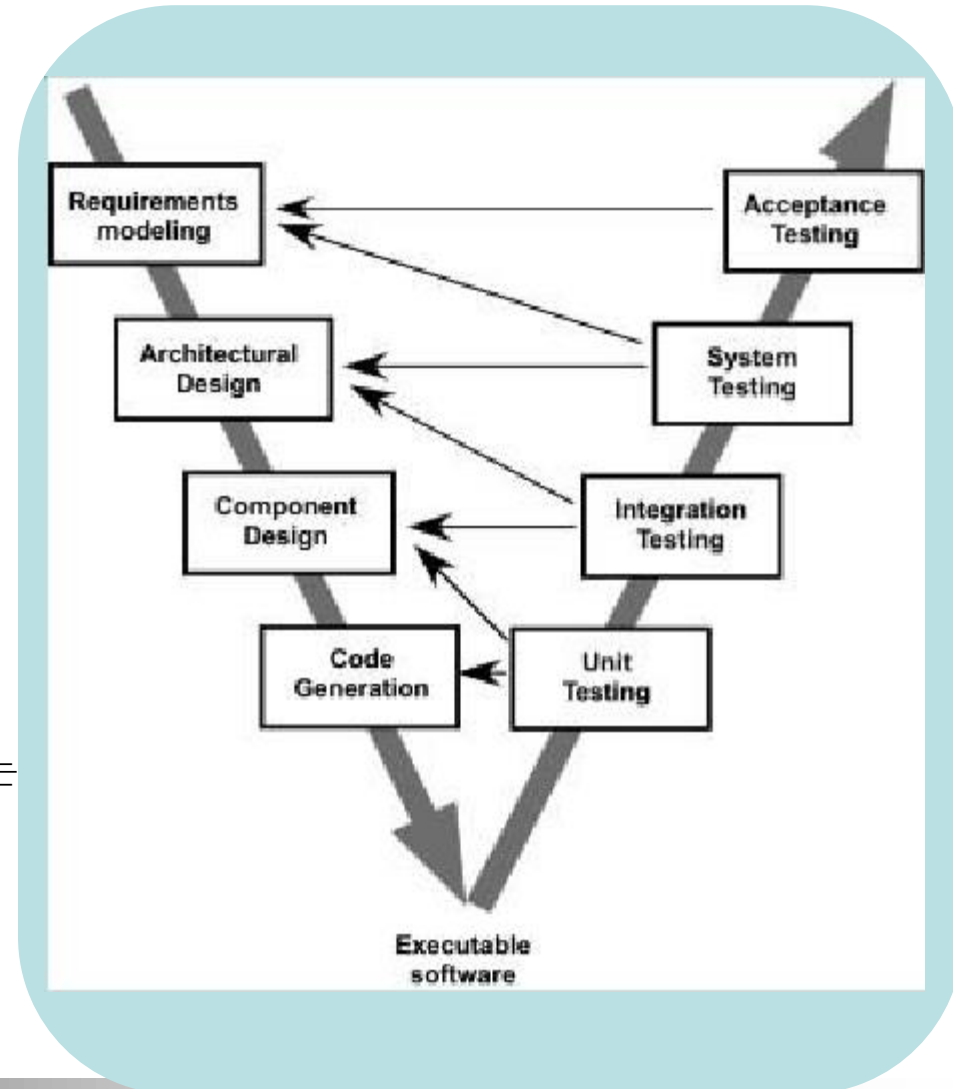
### ● 문제점

- 개발 동안 발생할 수 있는 프로덕트와 활동의 변화에 대처할 방법이 없음
- 소프트웨어 개발을 창조적 과정이 아닌 제조 과정으로 봄
- 최종 프로덕트를 만들어낼 반복적 활동이 없음
- 최종 프로덕트가 나올 때 까지 오래 기다림.

# 1. Prescriptive Model(V-Model)

## ◆ 1.1.1 V-Model

- 폭포수 모델의 변형
- 커뮤니케이션, 모델링, 초기 구축 액티비티와 관련된 액션에 대한 품질보증 액션의 관계
- 방향
  - 왼쪽 : 문제의 기본적인 **요구사항**을 점차적으로 정교화
  - 오른쪽 : 왼쪽을 진행하면서 생성한 각 모델을 **입증**하는 테스트 수행.



# 1. Prescriptive Model(Increment-Model)

## ◆ 1.2 단계적 개발 점증과 반복

### ● 어떻게 릴리스로 구성할 것인지 결정방법

#### 1) 점증적 개발 ( incremental development )

- 요구사항 문서에 명시된 시스템은 기능에 따라 서브시스템 으로 분할됨
- 릴리스는 소형 이고 기능적인 서브시스템으로 시작되어 새로운 릴리스 마다 기능이 추가됨
  - 작은 기능을 가진 서브시스템으로 시작하여 점차 새로운 버전 릴리즈 추가

#### 2) 반복적 개발 ( iterative development )

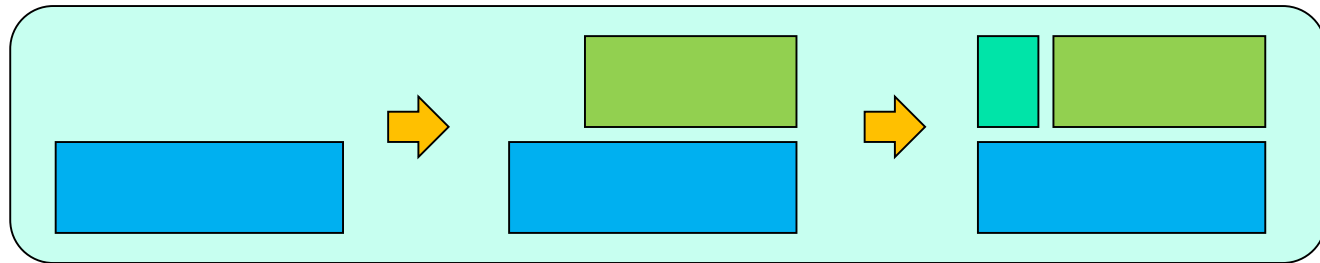
- 초기에 전체 시스템을 인도한 후 각 서브시스템의 기능을 변경 하여 새로운 릴리스를 만듦
  - 전체 시스템으로 시작 , 변경은 각 서브시스템을 새로운 버전으로 릴리스.

# 1. Prescriptive Model(Increment-Model)

## ◆ 1.2 단계적 개발 점증과 반복

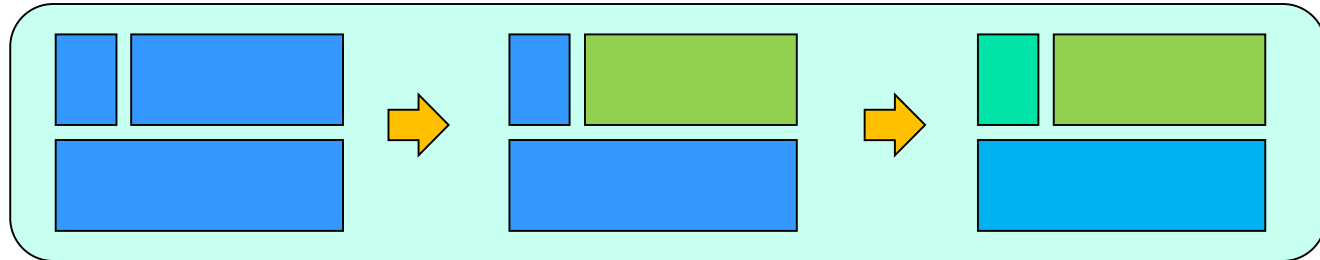
### ● 점증적 개발

- 새로운 릴리스마다 기능을 추가하여 전체 기능을 갖춘 시스템으로 작성되는 과정



### ● 반복적 개발

- 3 개의 릴리스를 보여주고 있음

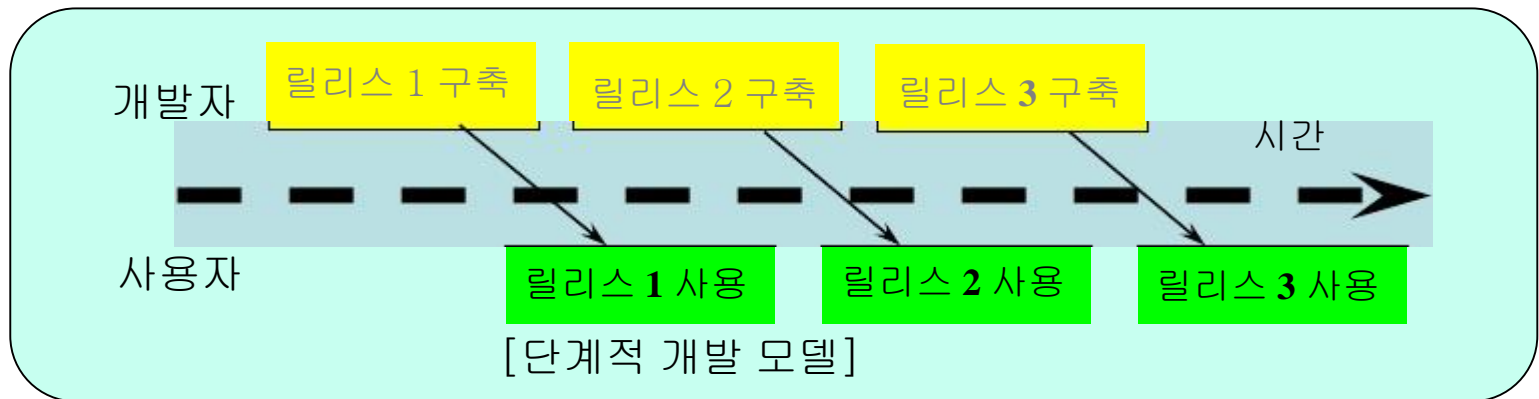


- 실제 많은 조직들은 진화적 개발과 점증적 개발 방법을 함께 사용 함
- 새로운 릴리스에는 새로운 기능이 추가될 뿐만 아니라 기존의 기능이 향상됨.

# 1. Prescriptive Model(Increment-Model)

## ◆ 1.2 단계적 개발 점증과 반복

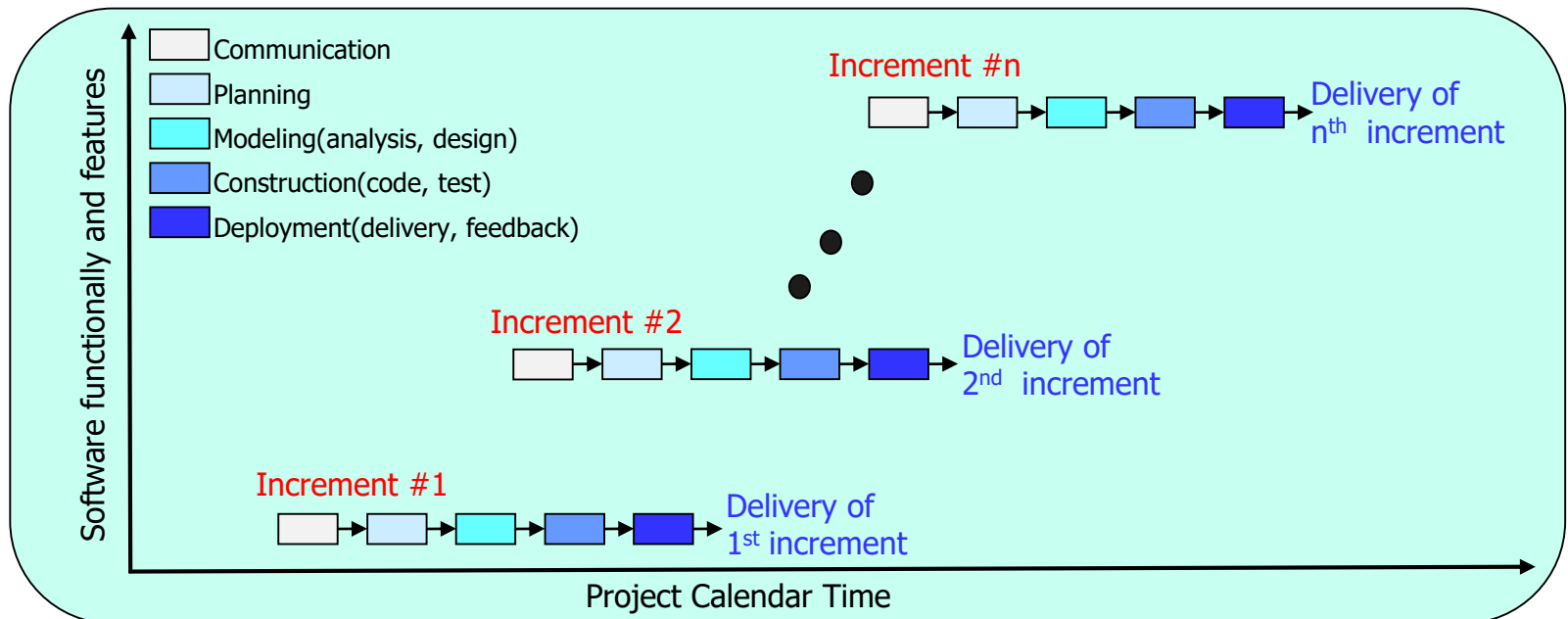
- 사이클 시간을 감소시키는 한가지 방법
  - 단계적 개발 방법을 이용하는 것
- 시스템은 사용자가 나머지 부분이 개발되는 동안 **특정 기능을 사용자가 사용** 할 수 있도록 분리되어 인도되도록 설계
- 생산 시스템 (production system)과 개발 시스템 (development system)의 두 가지로 나누어 볼 수 있음
  - **운영 또는 생산 시스템** : 고객과 사용자가 현재 사용중인 시스템임
  - **개발시스템** : 현재 생산 시스템을 대신할 다음 버전임.



# 1. Prescriptive Model(Increment-Model)

## ◆ 1.2 단계적 개발 점증과 반복

- 점증 (increment) : 실행 가능한 소프트웨어 생성
  - 제한적인 기능을 **빨리 제공**할 필요가 있는 경우
  - **정교화** 시키고 나중에 소프트웨어를 릴리스할 때 **기능 확장**
- 점증 순서
  - 핵심 프로덕트 우선 개발 -> 사용 -> 평가 -> 점증 계획 수립



# 1. Prescriptive Model(Increment-Model)

## ◆ 1.2 단계적 개발 점증과 반복

### ● 단계적 개발 방법이 바람직한 이유

- 사용자 교육이 초기 에 이루어짐
  - 몇 가지 기능이 누락되더라도 훈련 (training) 은 초기 릴리스 에서 시작됨
  - 훈련 프로세스는 개발자로 하여금 특정 기능이 어떻게 수행되는지 관찰 하도록 하고 다음 릴리스에서 보강하도록 함
  - 개발자들은 사용자에게 책임 을 다할 수 있음
- 지금까지 제공된 적이 없는 기능 만이 초기 시장을 개척 가능
- 빈번한 릴리스는 운영중인 시스템에서 보고되는 문제점을 개발자가 신속하고 전체적으로 고칠 수 있도록 함
- 개발 팀은 릴리스별로 다른 영역의 전문성에 중점 을 둘 수 있음
  - 예 ) 릴리스 1: 사용자 인터페이스 전문성에 초점
  - 릴리스 2 : 시스템 성능의 향상에 초점.

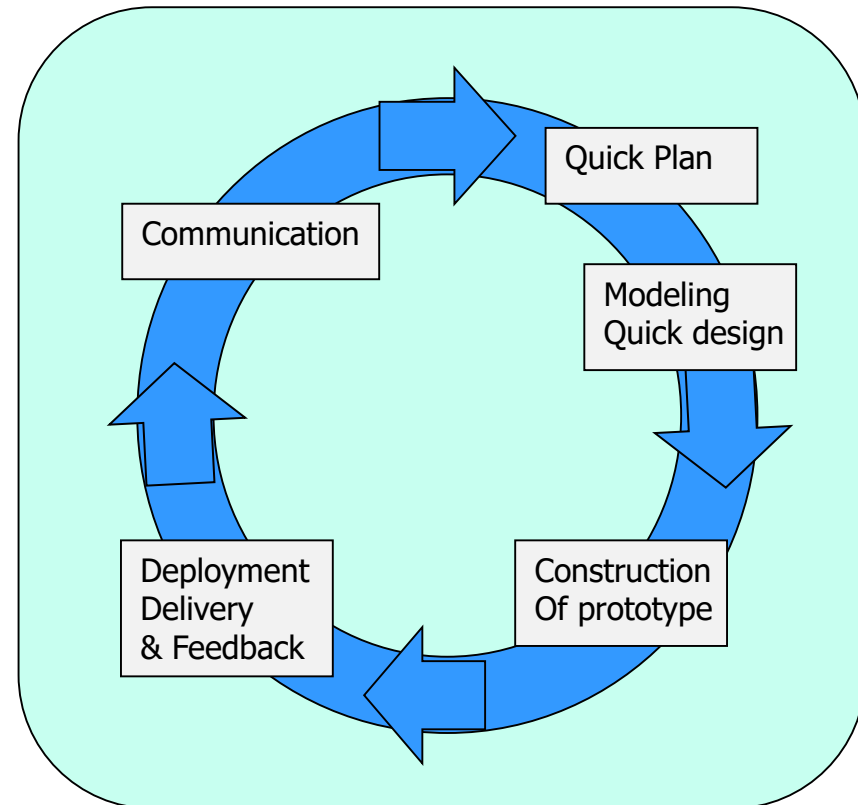


# 1. Prescriptive Model(Evolutionary-Model)

## ◆ 1.3 Evolutionary Model

### ● 1.3.1 Prototyping Model

- 기능, 특징에 대한 상세한 요구사항에 대해 잘 모를 경우
- 알고리즘의 효율성 , OS 의 적응성 , 인간 - 기계 상호작용의 형태 에 대해 미확신
- 패러다임 : **요구사항이 미확실한 경우** 이해당사자 이해 증진.

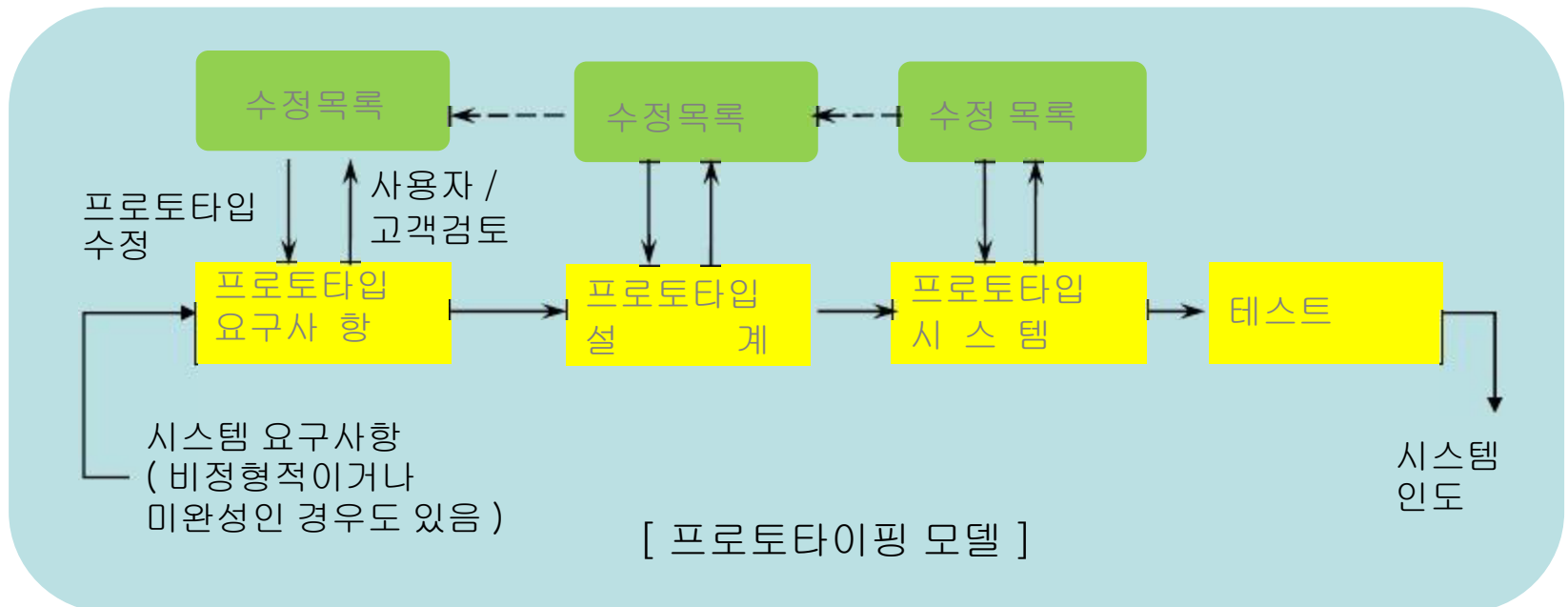


# 1. Prescriptive Model(Evolutionary-Model)

## ◆ 1.3 Evolutionary Model

### ● 1.3.1 Prototyping Model

- 개발에 따르는 위험과 불확실성을 감소 시키고자 하는 목표
- 요구사항과 설계의 반복적 검사를 허용
  - 요구사항이나 설계는 개발자와 사용자 그리고 고객이 필요한 것과 제안된 것 모두에 대한 공통적인 이해를 위해 반복적인 심사를 필요로 하는 공학 프로토타입과 동일한 목적을 가짐





# 1. Prescriptive Model(Evolutionary-Model)

## ◆ 1.3 Evolutionary Model

### ● 1.3.1 Prototyping Model

- 프로토타입은 확인과 검증에 유용함
  - 확인(validation)은 각 시스템 기능이 요구 명세서에 따라 시스템  
요구사항에 맞게 구현 되었음을 보증하는 것
    - 시스템 시험은 요구사항을 확인하는 것
    - 확인은 개발자가 정확한 프로덕트를 만들었음을 증명함
  - 검증(verification)은 각 기능이 올바르게 작동함을 보증 함
    - 검증은 구현된 시스템의 품질을 점검함.

# 1. Prescriptive Model(Evolutionary-Model)

## ◆ 1.3 Evolutionary Model

### ● 1.3.2 Spiral Model

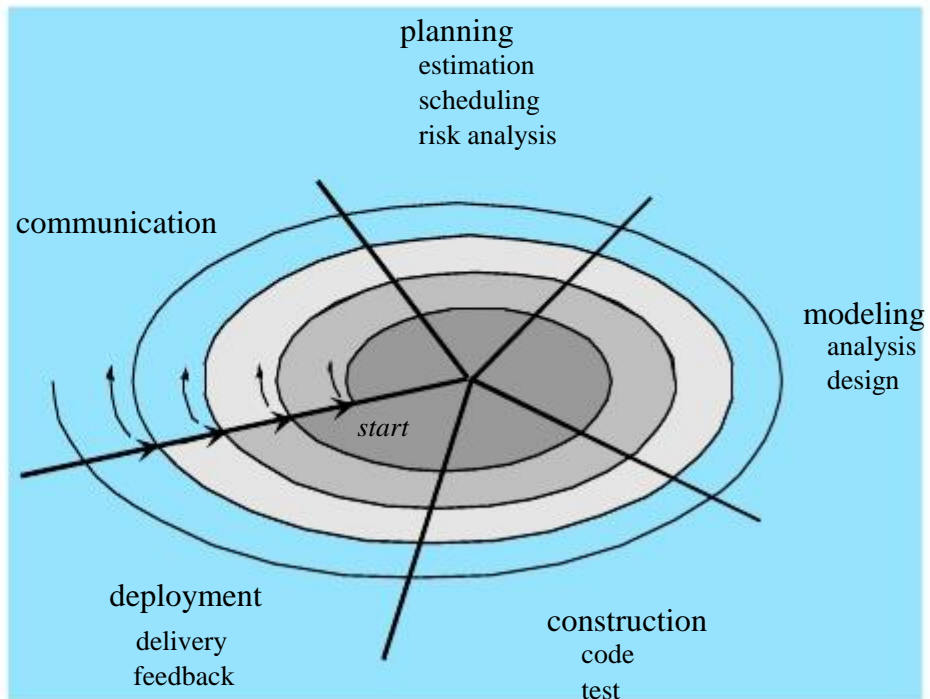
- 프로토타입 ( 반복적인 특성 ) + 폭포수모델 ( 통제적 , 체계적인 특성 )
- 특성

- ✓ 리스크를 감소 하면서 시스템의 정의와 구현 정도를 증가시키는 순환
- ✓ 적합하고 상호 만족스러운 해결방안을 이해관계자에게 보장하기

#### 앵커 포인트 이정표

### ➢ 진화적 릴리스의 형태

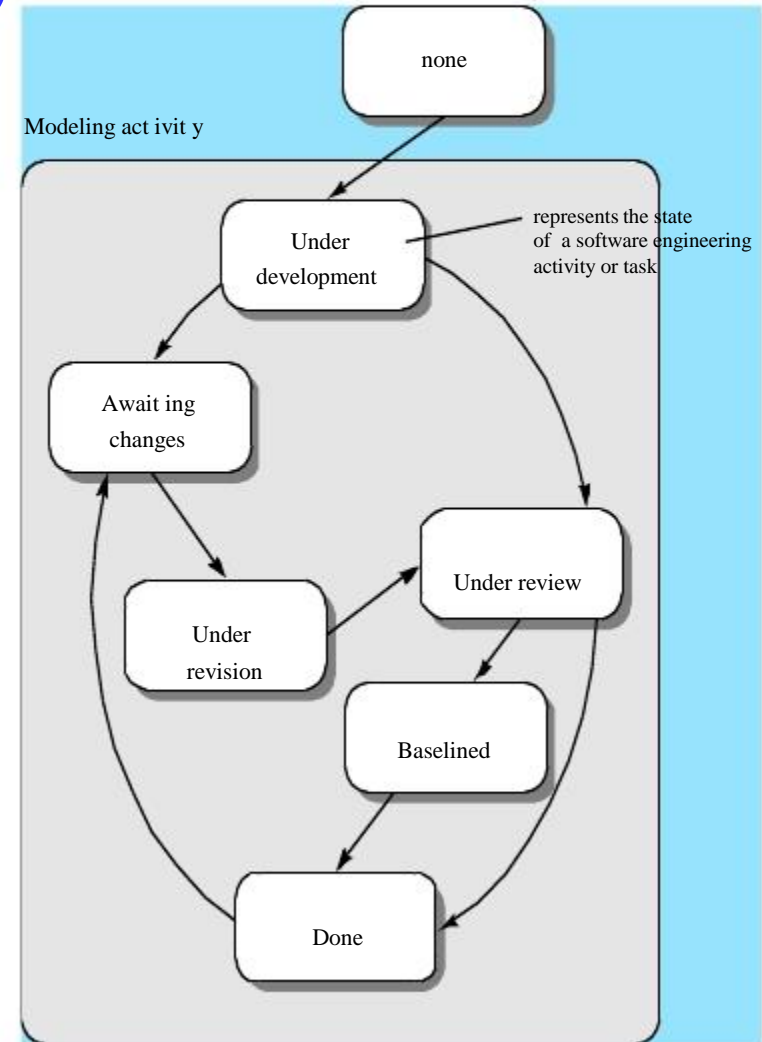
- ✓ 초반 반복 과정에서 얻는 릴리스  
모델이나 프로토 타입
- ✓ 후반 반복 과정에서 얻는 릴리스  
- 점증적으로 완전한 버전  
공학적인 프로덕트.

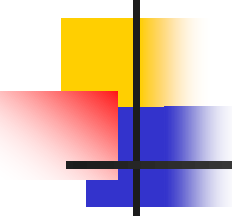


# 1. Prescriptive Model(Evolutionary-Model)

## ◆ 1.4 Concurrent Model(동시 모델)

- 프로세스 모델의 반복과 동시적인 요소
- 모든 액티비티는 동시에 존재하지만 , 서로 다른 상태
  - 상태 변환 : 이벤트
- 액티비티 , 액션 , 태스크를 연속적으로 한정하기보다
  - 프로세스와 네트워크로 정의.





## 2. 특수한 프로세스 모델

### ◆ 2.1 components based development model

- 미리 만들어진 패키지화된 **소프트웨어 컴포넌트** 를 사용
  - 프로덕트를 구축하기 위해 필요한 **불확실한 회수의 순환** 때문
  - **COTS** (Commercial off-the shelf) : 통합 용이 , 인터페이스 정의
- **유연성** 과 **확장성** 에 집중
  - 상대적인 높은 품질 측면보다
- **컴포넌트 기반 모델** 모델 단계
  - 1. 가능한 **컴포넌트 - 기반 프로덕트** 를 조사하고 , 논의가 되고 있는 애플리케이션 도메인을 위해서 평가
  - 2. **컴포넌트 통합** 이슈 고려
  - 3. 컴포넌트 수용할 수 있도록 **소프트웨어 구조를 설계**
  - 4. **컴포넌트 구조에 통합**
  - 5. 적합한 기능을 확보하기 위해 **전체적으로 테스트** 을 수행.

## 2. 특수한 프로세스 모델

### ◆ 2.2 Formal Methods model

- 컴퓨터 소프트웨어에 대한 정형적인 수학적 명세를 유도하는 일련의 액티비티들로 구성
  - 엄밀한 수학적 표기법을 적용하여 시스템을 확실하게 표현, 개발, 검증 가능.
  - 수학적 분석과정을 적용 : 모호성, 불완전성, 불일치성 쉽게 수정 가능
  - 설계과정 정형방법론 사용 : 프로그램 검증 근거로 에러를 쉽게 발견 / 수정
  - 클린룸 소프트웨어공학 : 결점없는 소프트웨어 개발
- 기업환경에서 정형방법론 적용시 고려 사항
  - 현재 정형방법론 사용하여 개발 : 시간과 비용 증대
  - 많은 훈련과정 필요 : 현재 개발자 정형방법론 기본지식 부족
  - 의사소통의 어려움 : 기본지식 없는 고객.

```
DCM
current : Block
direction : Direction
pma : Block → PMA
speed : Block ↔ Speed
obstacle :  $\mathbb{P}$  Block
redBlock :  $\mathbb{P}$  Block
yellowBlock :  $\mathbb{P}$  Block
greenBlock :  $\mathbb{P}$  Block
connection : ConnectionStatus

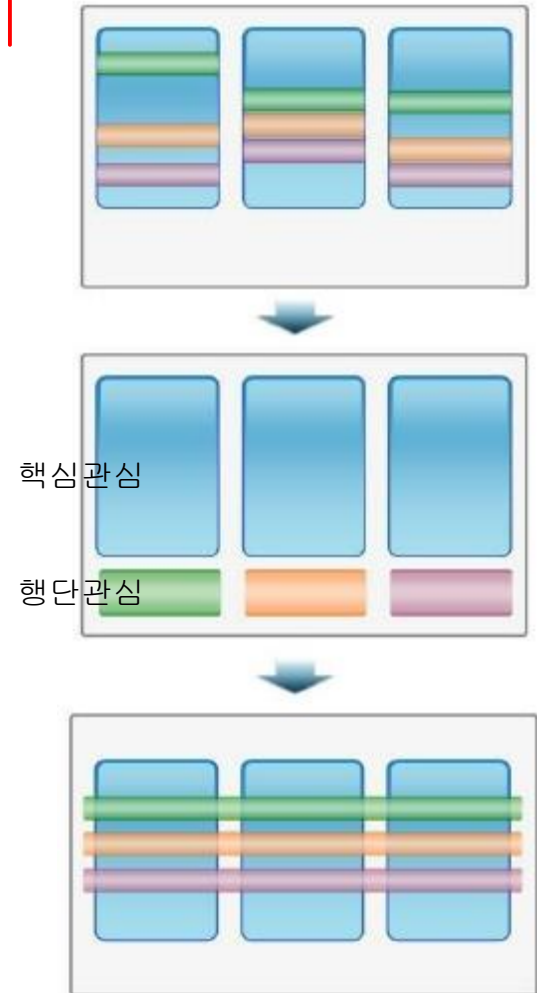
⟨redBlock, yellowBlock, greenBlock⟩ partition Block
behindOf (redBlock, direction)  $\cap$  greenBlock =  $\emptyset$ 
 $\forall b : \text{redBlock} \bullet \text{pma}(b) = \text{Red}$ 
 $\forall b : \text{yellowBlock} \bullet \text{pma}(b) = \text{Yellow}$ 
 $\forall b : \text{greenBlock} \bullet \text{pma}(b) = \text{Green}$ 
```

## 2. 특수한 프로세스 모델

### ◆ 2.3 관점-지향 소프트웨어 개발 model

#### ● 관점

- 교차 관심사를 표현하기 위한 **서브루틴이나 상속개념이상의 매커니즘**
- 부분적인 소프트웨어 컴포넌트와는 독립적으로 생성
- 관점상의 요구사항
  - 소프트웨어 구조에 영향을 미치는 **교차 관심사**
  - **핵심 관심** : 시스템의 **핵심가치와 목적**이 그대로 드러난 관심영역
  - **횡단 관심** : 다른 핵심 관심을 구현한 모듈과 긴밀히 **결합되어 분리가 어려운 영역** ( 보안 , 예외처리 )
- 프로세스 : **관점 (aspect)** 정의 - 명세화 - 설계 - 구축
- 모듈 전역에 넓게 퍼져 있는 **횡단부를 분리하여 중복을 제거** 하고 **필요한 부분에서 결합할 수 있는 방법** 제공.

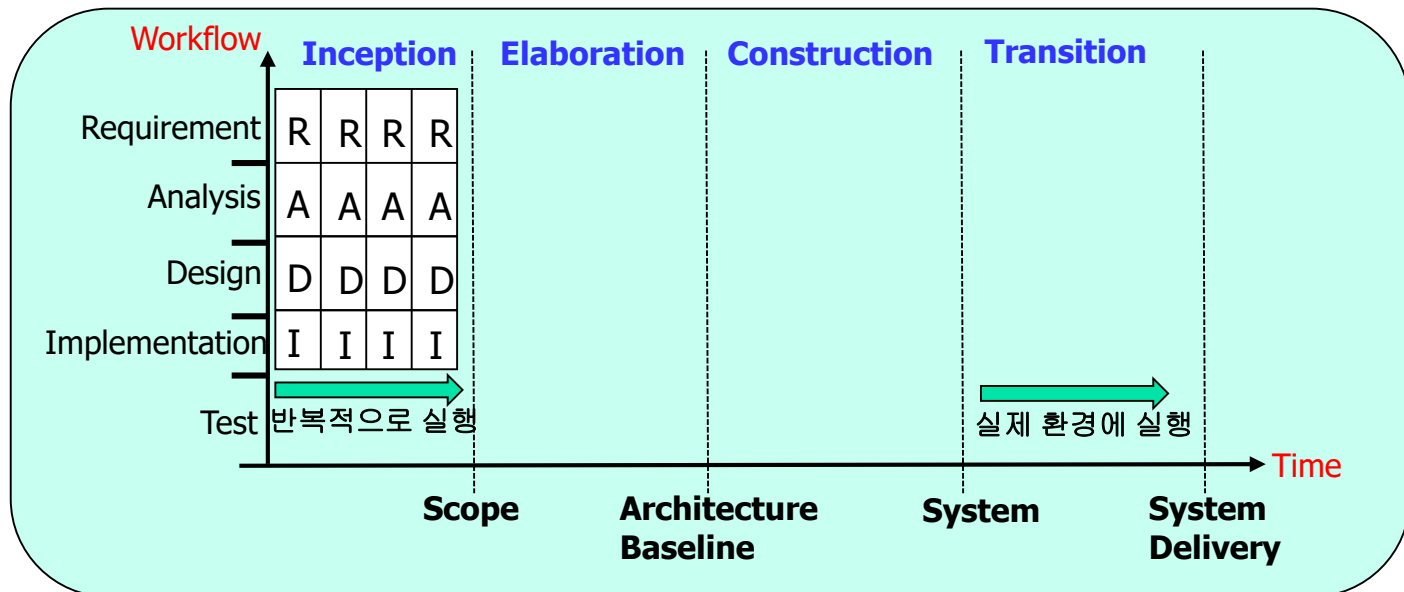




### 3. UP(Unified Process)

#### ◆ 통합 프로세스 모델

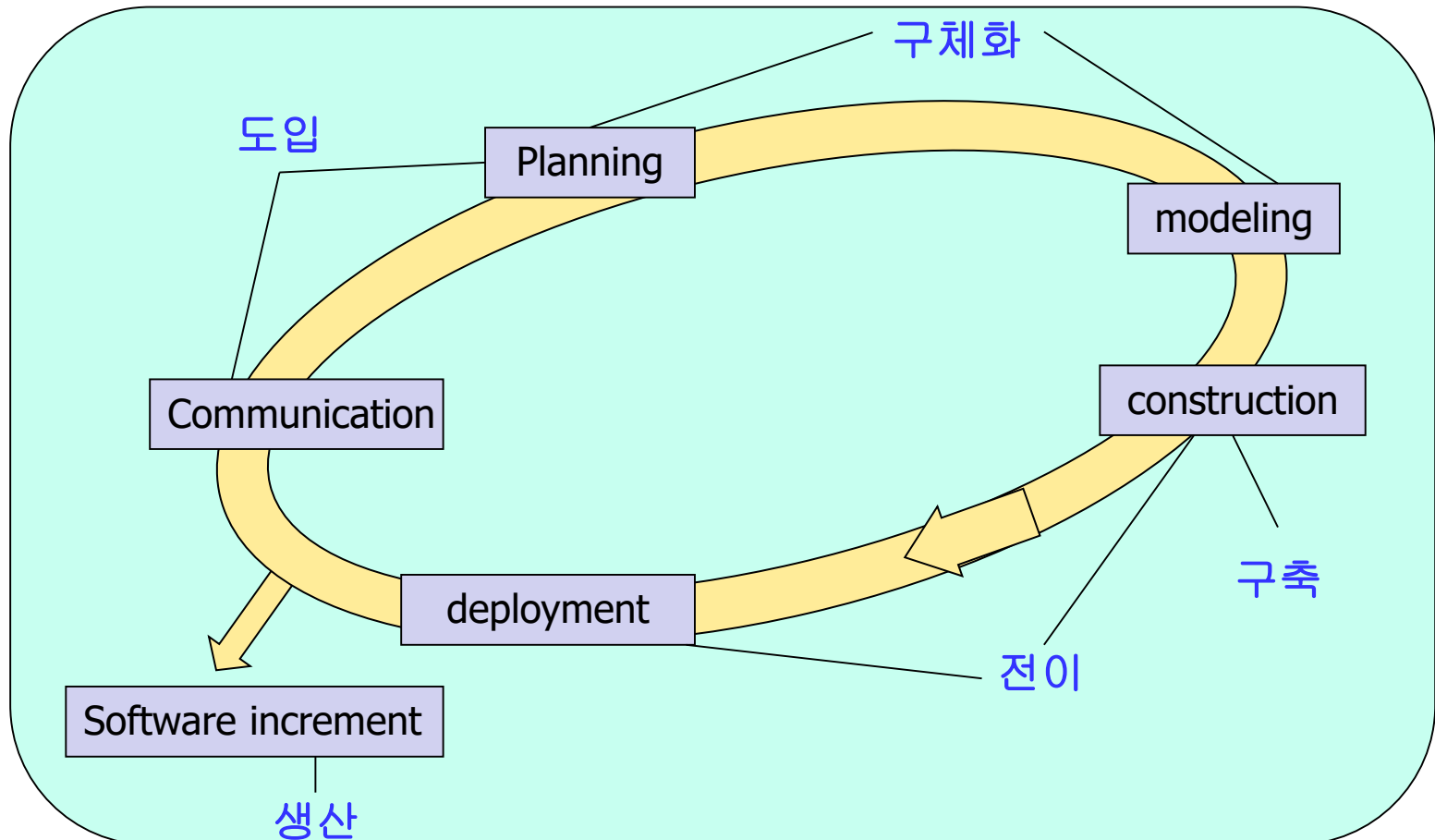
- 유스케이스 기반 , 구조중심 , 반복적이고 검증 적인 프로세스
  - 고객과 의사소통 , 시스템을 바라보는 **고객의 시각을 묘사** 하는 합리적 방법
  - 소프트웨어 구조의 중요한 역할 강조
    - 구조설계가 이해가능성 , 변경에 대한 믿음 , 재사용 목표에 집중 에 도움
- 통합 방법론 : UML(Unified Model Language)
- UP(Unified Process) 단계



### 3. UP(Unified Process)

#### ◆ 통합 프로세스 모델

##### ● 3.1 UP(Unified Process)



## 4. 개인 소프트웨어 프로세스

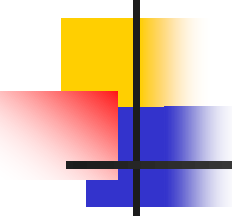
### ◆ 4.1. PSP(Personal Software Process)

- 개인 측정 강조 : 개발되는 작업 산출물 , 품질
- 실무자가 프로젝트 계획 ( 예측과 스케줄링 ) 책임 , 품질 제어
- PSP 모델 프레임워크 액티비티
  - 1. 계획수립
    - 요구사항 따로 고려 , SW 크기 / 자원 / 결함 예측 , 태스크 확인 , 스케줄 수립
  - 2. 상위레벨 설계
    - 컴포넌트 ( 외부명세 구축 ) , 컴포넌트 설계 , 프로토타입 개발 ( 불확신 경우 )
  - 3. 상위레벨 설계 검토
    - 정형 검증방법 적용 ( 설계에러 발견위해 ) , 척도유지 ( 주요 태스크 , 업무 결과 )
  - 4. 개발
    - 정교 / 검토 ( 컴포넌트 수준 설계 ) , 코드 ( 생성 , 검토 , 테스트 ) , 척도유지
  - 5. 사후 검토
    - 프로세스 효과 결정 ( 측정치와 척도 사용 )
    - 측정치와 척도 효과 개선 : 프로세스 수정시 길잡이 역할.

## 4. 개인 소프트웨어 프로세스

### ◆ 4.2. TSP(Team Software Process)

- TSP 목표 : 고품질의 SW 를 생산하기위한 스스로 조직하는 “자체적으로 유도되는” 프로젝트 팀을 생성하는것
  - 자체적으로 유도되는 팀
    - 전체적으로 목표를 일관적으로 이해
    - 팀의 구성원 : 역할과 책임을 정의
- TSP 목표 정의
  - 1. 자신들의 작업에 대해 계획 , 추적 , 목표 설정 , 프로세스 / 계획을 가지고 자체적으로 유도되는 팀 생성
    - 순수 SW 팀 , 3~20 명 엔지니어로 구성된 통합 프로젝트 팀 (IPT)
  - 2. 매니저에게 팀을 어떻게 지도하고 자극하며 , 팀이 최고의 성취를 유지할 수 있도록 도와줄 것인지를 알려줌
  - 3. CMM 레벨 5 행동을 정상과 기대 상태로 만들어 줌으로써 소프트웨어 프로세스 개선 촉진
  - 4. 많이 성숙된 조직에 개선을 위한 안내를 제공
  - 5. 산업체급 팀 기술에 대한 내용을 대학에 가르치는것 용이하게 해 줌.



## 5. 실용적인 프로세스 모델링

### ◆ 프로세스 모델링 도구와 기법의 바람직한 속성

- 바람직한 속성을 5 가지로 분류함
  - 인간의 이해와 의사소통을 용이 하게 하여야 함
  - 프로세스 성능향상 을 지원하여야 함
  - 프로세스 관리 를 지원하여야 함
  - 프로세스 수행 시 자동화된 지침 을 제공하여야 함
  - 자동화된 프로세스 실행 을 지원하여야 함.



# Homework

---

## ◆ Chapter4. 프로세스 모델

4.1 프로세스 정의

4.2 프로세스가 중요한 이유

4.3 프로세스 특징

4.4 프로세스 모델링을 하는 이유

4.5 각 프로세스 모델의 장단점 및 활용



# 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 모델중 1개를 선정 및 이유

### 3.2 특수한 프로세스 모델중 1개를 선정 및 이유