



# 애플리케이션 설계 *(Architecting for SW intensive systems)*

v 1.4



# 목 차

---

1. 에피소드
2. EA와 TOGAF
3. IEEE 1471
4. 품질 속성
5. 시스템 모델링
6. 아키텍팅 프로세스
7. 아키텍처 설계
8. 컴포넌트 모델링

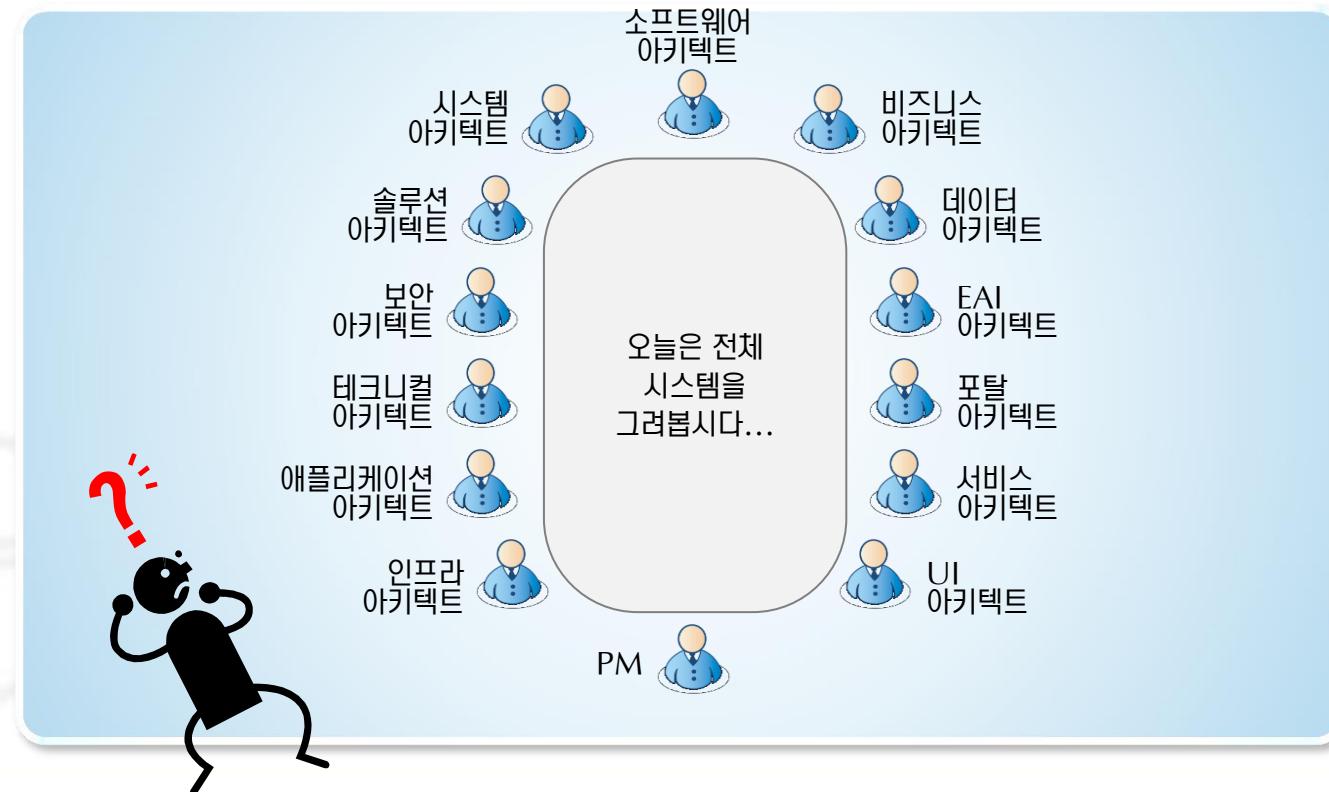
# 1. 에피소드

---

- ✓ Episode #1 – 13 Architects
- ✓ Episode #2 – Nobody's tasks
- ✓ Episode #3 – Not That Architecture
- ✓ Episode #4 – A system has four architectures
- ✓ Episode #5 – Different but same
- ✓ Episode #6 – Unidentified best job
- ✓ 토의

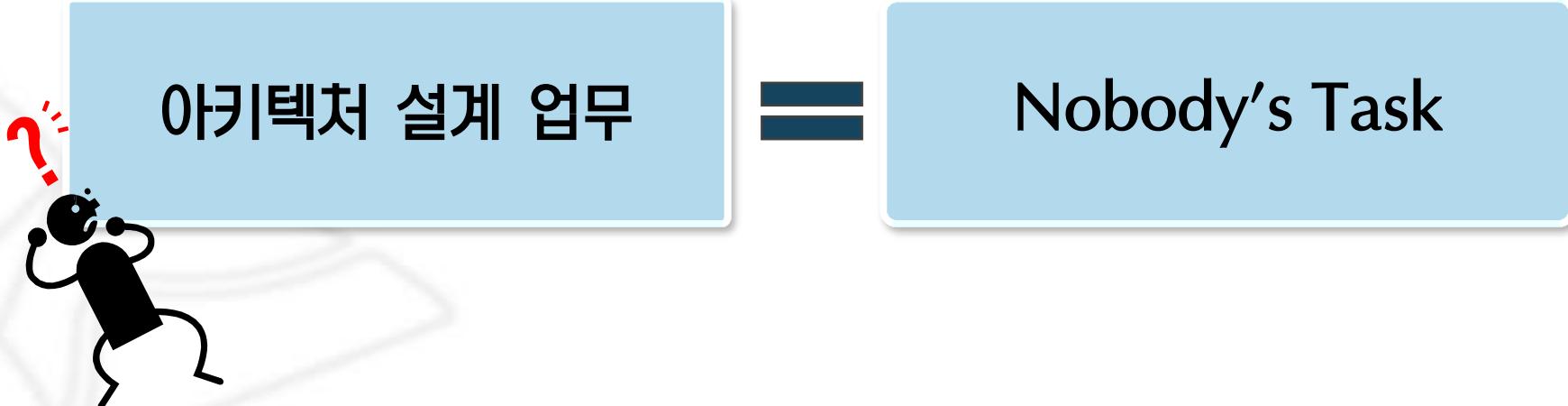
# Episode #1 – 13 Architects

- ✓ 어느 금융 회사의 차세대 프로젝트에서 아키텍트 회의가 열렸습니다. 때는 11월13일 금요일, 13 인의 아키텍트가 모였습니다. 해가 지도록 시스템 Big picture에 대해서 이야기를 합니다.
- ✓ 밤은 깊어 가고 결론은 나지 않습니다...



## Episode #2 – Nobody's tasks

- ✓ 업계에서 이름을 대면 알만한 SW 아키텍트 한 분이 초급 개발자 한 명을 데리고 프로젝트에 참여했습니다. PM은 아키텍트가 왔다고 반기면서, [공통]으로 분류된 여러 태스크를 주었습니다. 차츰 [아키텍처팀 == 공통팀] 등식이 성립되었고, 분쟁의 소지가 있는 일이나 책임소재가 불명확한 일은 아키텍처 팀에게 할당되었습니다. 2명이 참여를 했지만, 10명이 해도 다 못할 분량의 일이 생겼습니다. 아키텍트는 이 일을 PM과 진지하게 논의했습니다. PM은 “공통”일을 있다고 계약했으므로 알아서 끝내어 놓으라고 합니다. 그 SW 아키텍트는 다시는 아키텍트 역할을 하고 싶지 않다고 이야기했습니다.



# Episode #3 – Not That Architecture

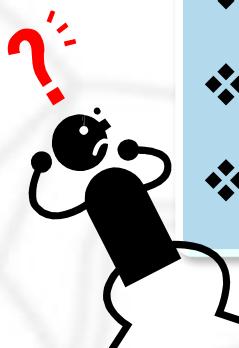
- ✓ 국방 분야 대형 프로젝트가 끝난 후에 사용자들은 시스템 간 연계 부분에 문제가 있다는 사실을 발견했습니다. 시스템 담당자는 현재의 연계방식은 정보를 주고받는 속도도 느리고, 유연성이 부족해 변경 요구를 받아 들이기 힘들다는 이야기를 사용자로부터 들었습니다. 곧바로 시스템 진단을 시작했고, 진단 팀은 연계 방식과 사용 기술, 기술 적용에 문제가 있다는 결론을 내면서, 이것은 전형적인 “아키텍처 설계 부실”이라고 진단을 내렸습니다. 담당자는 이 부분을 개선하기 위한 “아키텍처 개선” 예산을 준비했고, 주변에서 이것은 대규모 시스템의 아키텍처 문제이므로 EA(Enterprise Architecture)로 문제를 해결해야 한다고 조언했습니다. 이듬해 이 EA 프로젝트 공고가 났습니다.

규모가 크거나 정도가 심각한 아키텍처 문제는 모두  
Enterprise Architecture로 풀어야 한다...



# Episode #4 – A system has four architectures

- ✓ 많은 소프트웨어 개발 업체에게 아키텍처 팀을 AA, DA, TA, BA 팀으로 분리하여 운영하고 있습니다. SA 팀을 포함하여 다섯 가지 아키텍처 팀을 운영하는 회사도 있습니다. 아키텍팅 역량 향상은 곧 이 네 가지 영역에서의 역량 향상을 의미합니다. 소프트웨어 개발 업체의 주요 활동은 (Software Intensive) 시스템 구축 프로젝트입니다. 시스템 구축에 AA, DA, TA, BA 네 가지 아키텍팅 역량을 동원하고 있습니다. 이전에 존재하던 시스템 분석가, 비즈니스 분석가, 설계자, 소프트웨어 아키텍트 등의 역할이 눈에 띄게 사라지면서 모두 무슨 무슨 아키텍트라는 역할로 대체되었습니다.



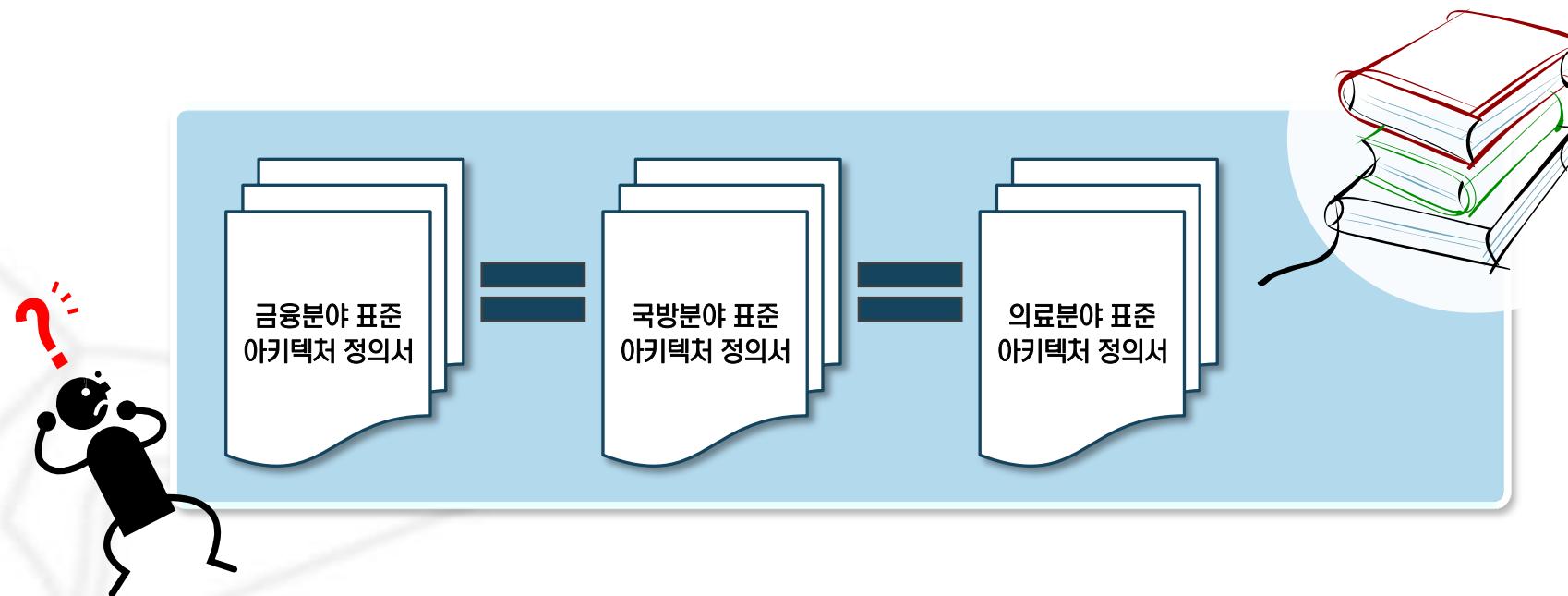
- ❖ AA에서 A가 시스템 구축 영역의 애플리케이션일까요?
- ❖ DA에서 D는 시스템 구축 영역의 데이터일까요?
- ❖ BA에서 B는 비즈니스 프로세스일까요?
- ❖ TA에서 T는 Technical thing일까요?

AA:Application Architecture

DA: Data Architecture

# Episode #5 – Different but same

- ✓ 2000년대 중반, 정통부에서 “산업별 표준 아키텍처 수립” 사업을 발주하였습니다. 사업의 취지는 아키텍처 수립에 비용이 많이 들어가니, 각 산업 분야별로 표준 아키텍처를 수립하고, 그것을 재사용하자는 것이었습니다. 의료, 국방, 교육, 공공 등으로 분류하여 사업자를 선정하였고, 취지대로 각 산업 별로 아키텍처 수립을 한 후 과제를 제출하였습니다. 저마다 두더운 바인더에 분야별 아키텍처를 제출했습니다.
- ✓ 그런데 놀라운 것은 제출한 내용이 대부분 비슷하다는 것이었습니다. 제대로 수행되었다면 5년이 넘게 지난 오늘 그 아키텍처들은 업계의 표준이 되어야 했습니다. 그런데 과제 제출로 모든 것이 끝나버렸습니다.



# Episode #6 – Unidentified best job

- ✓ 잘 아는 후배가 한 명 찾아왔다. 미국에서 SW 아키텍트가 되려면 어떤 커리어패스가 필요한지 물어왔다. 회사에서 일을 잘 알아야 한다고 하고, DA는 데이터 모델링에 능숙한 무언가를 어떻게 해야 할지 모른다고 했다.
- ✓ 기업 인사담당자도 마찬가지이다. 소프트웨어 아키텍트가 없다고들 한다.



## BEST JOBS IN AMERICA

Money/Payscale.com's list of great careers

2010

Full List

High Pay

Job Growth

Quality of Life

Sectors

### 1. Software Architect

[Recommend](#) 5K

1 of 100

Next

Top 100 rank: 1

Sector: Information Technology

**What they do:** Like architects who design buildings, they create the blueprints for software engineers to follow -- and pitch in with programming too. Plus, architects are often called on to work with customers and product managers, and they serve as a link between a company's tech and business staffs.

**What's to like:** The job is creatively challenging, and engineers with good people skills are liberated from their screens. Salaries are generally higher than for programmers, and a typical day has more variety.

"Some days I'll focus on product strategy, and other days I'll be coding down in the guts of the system," says David Chaiken, 46, of Yahoo in Sunnyvale, Calif., whose current projects include helping the web giant customize content for its 600 million users. Even though programming jobs are moving overseas, the face-to-face aspect of this position helps cement local demand.



PHOTO: DAVID LAURIDSEN

Chaiken, a software engineer for more than two decades, relishes the more collaborative work.

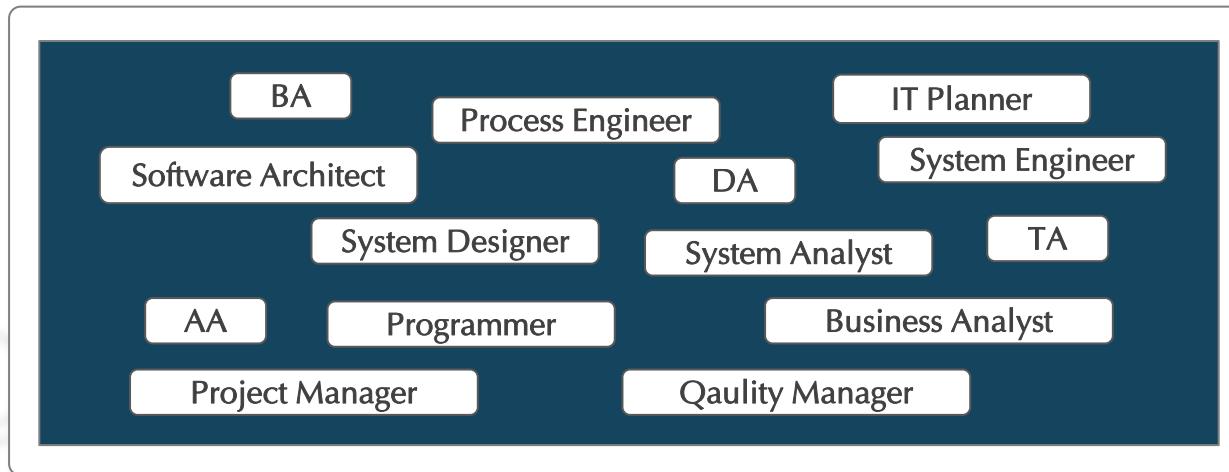
## 2. EA와 TOGAF

---

- ✓ Role mixup
- ✓ IT의 네 영역
- ✓ EA
- ✓ EA 논쟁
- ✓ 토의

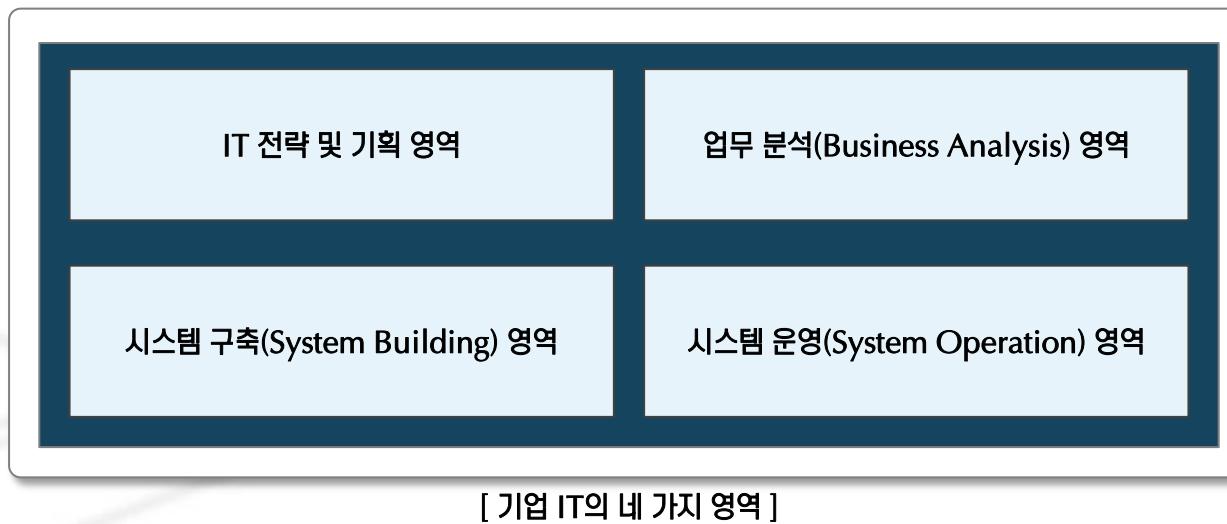
# Role mixup

- ✓ 기업 IT 영역에는 다양한 역할들이 존재합니다.
- ✓ 어떤 역할은 IT의 표준에 정의되어 있고, 어떤 역할은 조직에서 정의한 것들입니다.
- ✓ IT 표준에 준하는 역할을 조직에서 적용하는 과정에서 많은 오류와 시행착오가 발생합니다.
- ✓ 역할에 대한 정확한 이해와 적용이 필요합니다.



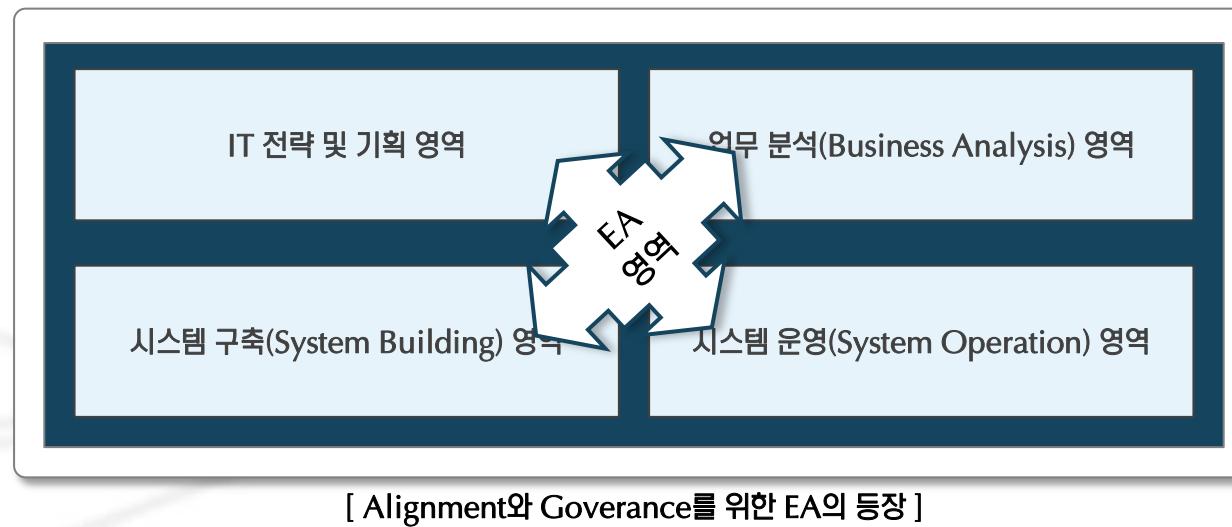
# IT의 네 영역

- ✓ 역할 혼란을 줄이기 위해 어떤 분야든 영역을 정의하고 역할을 영역 안에 둡니다.
- ✓ IT 분야는 정보시스템 라이프사이클을 바탕으로 네 가지 영역으로 나눌 수 있습니다.
  - IT 전략 및 기획(Strategy & Planning) 영역
  - 업무 분석(Business analysis) 영역
  - 정보 시스템 구축(Building) 영역
  - 정보 시스템 운영(Operation) 영역



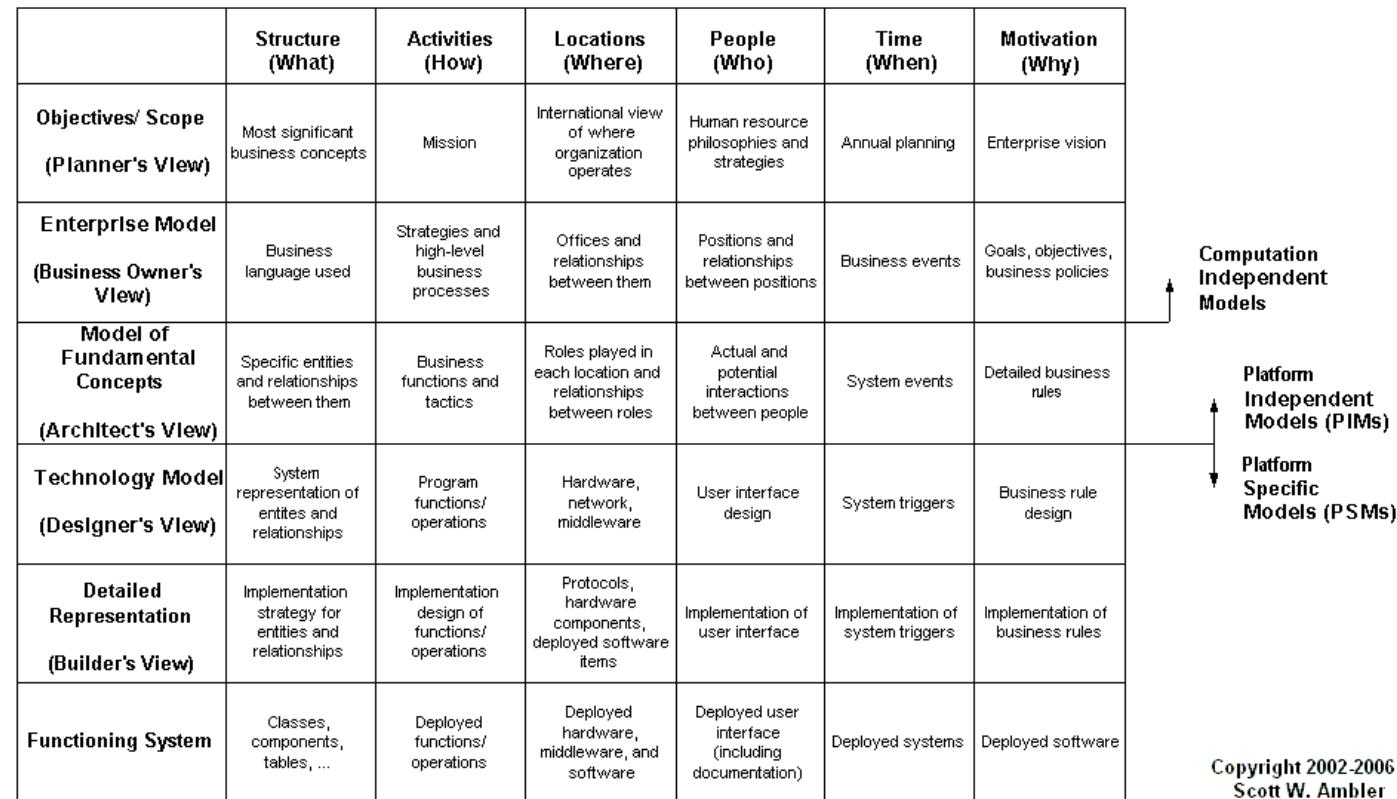
# EA(1/4) – 등장

- ✓ 기업 경영에서 IT 의 비중이 커지면서 표준부재, 중복투자 등으로 인한 비효율성 문제가 드러납니다.
- ✓ 이런 환경에서 조정(alignment)과 거버넌스(governance)를 위해 EA가 등장함
- ✓ 기업의 IT를 높은 수준에서 조망하는 방법을 제시하며 ‘아키텍처’를 유행시킴
- ✓ 관련자료: *Building Bridges between EA and SA(2009, 아키텍트대회, 넥스트리, 송태국)* 참조



# EA(2/4) – Zachman Framework

- ✓ 기업의 복잡한 IT 영역을 바라보는 방법을 제시할 목적으로 다양한 [Viewing] 프레임워크 등장함
- ✓ 1987년 Zachman은 View와 Perspective를 양축으로 하는 프레임워크를 제시함
- ✓ 기업의 IT를 바라보는 일관성있는 관점을 제공해 주었지만,
- ✓ 서로 다른 대상은 서로 다른 관점으로 바라보아야 함을 고려하지 않았으므로 실효성 문제에 부딪힘



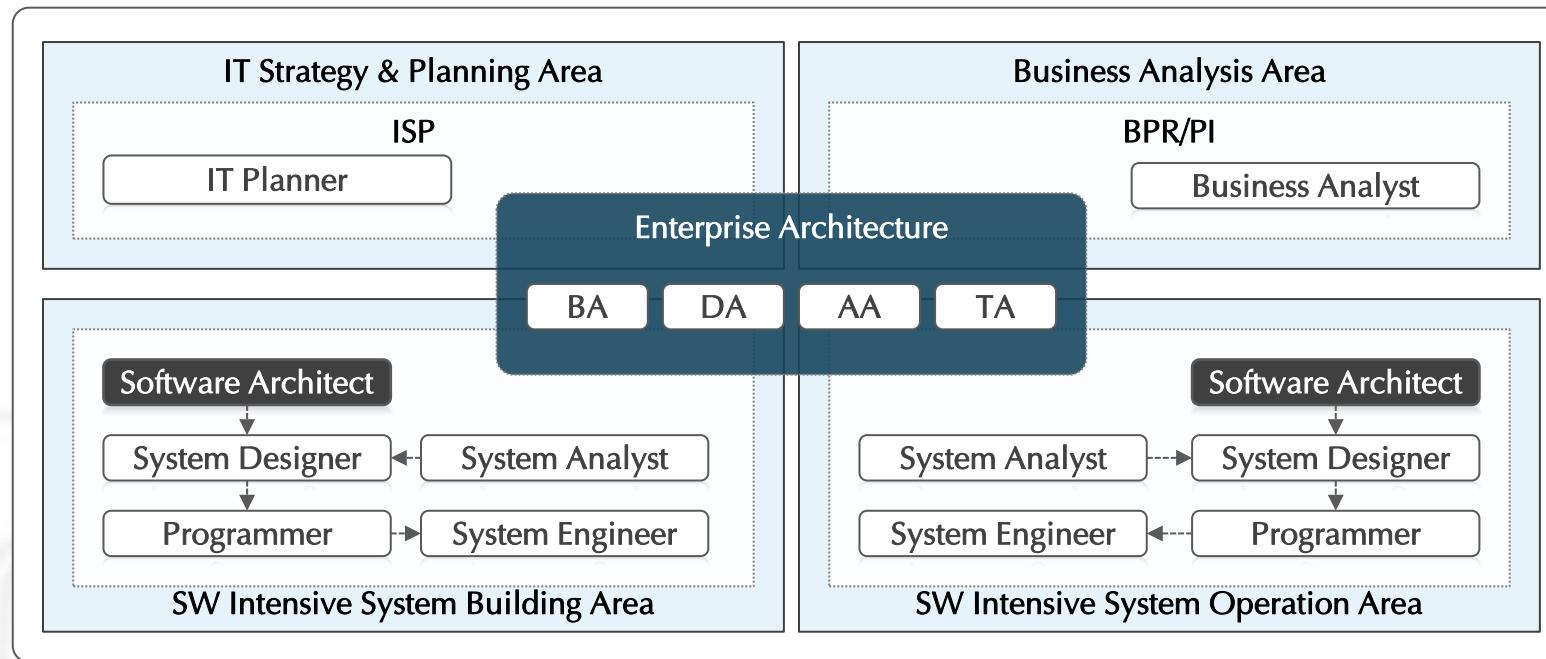
# EA(3/4) – 범정부 EA 프레임워크

- ✓ 범정부 프레임워크의 BA 부분은 조직, 업무 두 가지를 중심으로 전개함
- ✓ 기술은 인프라로 확장함으로써 SW 아키텍처 설계 영역과 중복이 발생함
- ✓ 보안 아키텍처를 EA 영역으로 끌어올림으로써 SW 아키텍처 영역과 중첩

| View<br>Persp<br>pective | BA  | AA   | DA  | Technology/Infra<br>Arch.                                      | Security Arch.  |
|--------------------------|---|--|---|--|---|
| CEO                      | <ul style="list-style-type: none"><li>• 조직구성도/정의서</li><li>• 업무구성도/정의서</li></ul>     | <ul style="list-style-type: none"><li>• 응용 구성도/정의서</li></ul>                       | <ul style="list-style-type: none"><li>• 데이터 구성도/정의서</li></ul>                       | <ul style="list-style-type: none"><li>• 기반구조 구성도/정의서</li></ul> | <ul style="list-style-type: none"><li>• 보안정책</li><li>• 보안구성도/정의서</li></ul>                    |
| 관리자                      | <ul style="list-style-type: none"><li>• 업무기능관계도/기술서</li><li>• 업무기능분할도/기술서</li></ul> | <ul style="list-style-type: none"><li>• 응용 관계도/기술서</li><li>• 응용기능분할도/기술서</li></ul> | <ul style="list-style-type: none"><li>• 개념데이터 관계도/기술서</li><li>• 데이터교환 기술서</li></ul> | <ul style="list-style-type: none"><li>• 기반구조 관계도/기술서</li></ul> | <ul style="list-style-type: none"><li>• 보안관계도/기술서</li></ul>                                   |
| 설계자                      | <ul style="list-style-type: none"><li>• 업무절차 설계도/설계서</li></ul>                      | <ul style="list-style-type: none"><li>• 응용기능설계도/설계서</li></ul>                      | <ul style="list-style-type: none"><li>• 논리데이터 모델</li><li>• 데이터교환 설계서</li></ul>      | <ul style="list-style-type: none"><li>• 기반구조설계도/설계서</li></ul>  | <ul style="list-style-type: none"><li>• 관리보안설계서</li><li>• 물리보안설계서</li><li>• 기술보안설계서</li></ul> |
| 개발자                      |   | <ul style="list-style-type: none"><li>• 응용프로그램 목록</li></ul>                        | <ul style="list-style-type: none"><li>• 물리데이터 모델</li></ul>                          | <ul style="list-style-type: none"><li>• 제품목록</li></ul>         | <ul style="list-style-type: none"><li>• 보안 메뉴얼</li></ul>                                      |

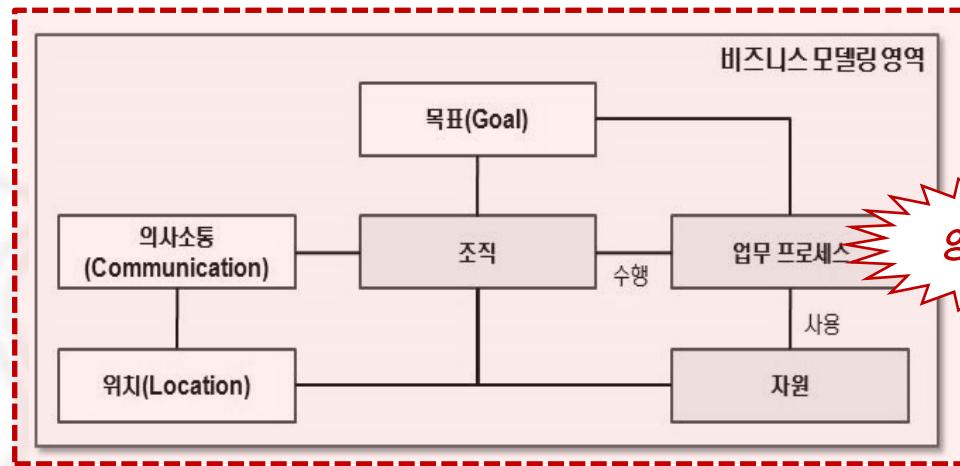
# EA(4/4) – Coverage

- ✓ Alignment와 Governance를 위해 네 가지 영역에 걸쳐져 있음
- ✓ 각 영역 스스로 잘 조정이 되고 관리가 체계적으로 된다면 EA의 역할이 줄어들 수 있음
- ✓ 따라서, 일부 영역과는 Alignment 보다는 오히려 Collision이 발생함



# EA 논쟁(1/10) – BA와 AA 충돌

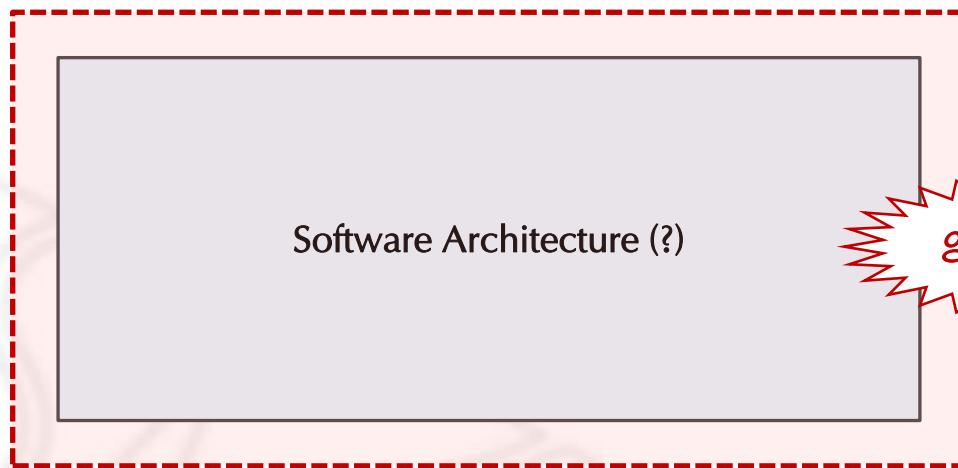
- ✓ 비즈니스 모델링 또는 Business Analysis 영역에서 여섯 가지 분석 또는 모델링 대상이 있음
- ✓ EA의 Business Architecture 정의는 조직과 업무를 중심으로 분석을 수행함
- ✓ 두 영역의 활동 내용에 중복이 발생함
- ✓ EA-BA 활동은 BPR/PI 활동과는 다르다고 하나 활동 내용이 중첩되고 있음



| View<br>Perspective | BA                             | AA                            |
|---------------------|--------------------------------|-------------------------------|
| CEO                 | • 조직구성도/정의서<br>• 업무구성도/정의서     | • 응용 구성도/정의서                  |
| 관리자                 | • 업무기능관계도/기술서<br>• 업무기능분할도/기술서 | • 응용 관계도/기술서<br>• 응용기능분할도/기술서 |
| 설계자                 | • 업무절차 설계도/설계서                 | • 응용기능설계도/설계서                 |
| 개발자                 |                                | • 응용프로그램 목록                   |

# EA 논쟁(2/10) – AA와 SA 충돌

- ✓ EA-AA와 SA 사이에는 활동의 충돌이라기보다는 역할의 충돌이 발생함
- ✓ 일부 조직이나 프로젝트에서는 SA 역할을 AA 역할로 대체하고 있음
- ✓ 이 경우는 충돌이라는 용어보다는 혼란이라는 용어가 더 적절할 것임
- ✓ 충돌과 혼란을 방지하기 위해 EA를 표준기반으로 정확히 이해할 필요가 있음



| View<br>Perspective | BA                             | AA                            |
|---------------------|--------------------------------|-------------------------------|
| CEO                 | • 조직구성도/정의서<br>• 업무구성도/정의서     | • 응용 구성도/정의서                  |
| 관리자                 | • 업무기능관계도/기술서<br>• 업무기능분할도/기술서 | • 응용 관계도/기술서<br>• 응용기능분할도/기술서 |
| 설계자                 | • 업무절차 설계도/설계서                 | • 응용기능설계도/설계서                 |
| 개발자                 |                                | • 응용프로그램 목록                   |

# EA 논쟁(3/10) – TOGAF

- ✓ IT 업계 표준 [Enterprise] 아키텍처 프레임워크임
- ✓ 오픈그룹의 아키텍처 포럼에서 지속적으로 발전시키고 있음
- ✓ 현재 Version 9까지 릴리즈하였음
- ✓ [www.opengroup.org/togaf](http://www.opengroup.org/togaf)



# EA 논쟁(4/10) – EA-AA at TOGAF

- ✓ TOGAF에서는 Application에 대해서 다음과 같이 말하고 있습니다.
- ✓ 조직에서 정의하고 있는 Application Architecture의 의미와 비교하여 봅니다.
- ✓ 조직에서 정의하고 있는 Application Architect의 역할을 검토해 봅니다.

AA의 목표는 데이터를 처리하고 업무를 지원하는 애플리케이션이 주로 어떤 종류가 있는지 정의하는 것이다. 이러한 활동은 애플리케이션 시스템 설계와 관련이 없다는 사실에 유의하여야 한다. 목표(goal)는 기업과 관련된 애플리케이션이 어떤 것이 있는가를 정의하고, 데이터를 관리하고 직원들이 정보를 사용하도록 하기 위해서 애플리케이션이 필요로하는 것이 무엇인가를 정의하는 것이다. 애플리케이션은 컴퓨터 시스템이 아니라 DA의 데이터를 관리하고 BA의 업무 기능을 지원하는 역량의 논리적인 그룹이다. 애플리케이션과 애플리케이션의 역량은 특정 기술에 대한 고려 없이도 정의된다.



TOGAF Version 9

# EA 논쟁(5/10) – EA-DA at TOGAF

- ✓ TOGAF에서는 Data Architecture에 대해서 다음과 같이 이야기 하고 있습니다.
- ✓ 데이터는 우리가 수행했던 논리 데이터 모델, 물리 데이터 모델 등과 관련이 없다고 합니다.
- ✓ DA에서 말하는 데이터는 데이터 엔티티 또는 비즈니스 엔티티, 비즈니스 자원 등을 의미합니다.

## 10.1 목표

DA의 목표는 비즈니스를 지원하는데 필요한 데이터의 주요 타입과 소스를 다음과 같은 수준으로 정의하는 것이다.

- ❖ 이해관계자가 이해할 수 있어야 함
- ❖ 완전성과 일관성을 갖추어야 함
- ❖ 안정적이어야 함

이러한 활동은 데이터베이스 설계와 관련이 없음을 이해하는 것이 중요하다. 목표는 기업과 관련된 데이터 엔티티를 정의하는 것이지, 논리 또는 물리 데이터 저장 시스템을 설계하는 것이 아니다.



TOGAF Version 9

# EA 논쟁(6/10) – EA-TA at TOGAF

- ✓ Technology Architecture의 핵심은 AA를 실현할 대상 기술 포트폴리오를 작성하는 것임
- ✓ 보안 아키텍처 설계, 인프라 아키텍처 설계와 같은 [설계 활동]은 TA 범위가 아님
- ✓ “TA는 목표 아키텍처로 가는 로드맵을 상세화하고...”

TA 단계에서는 AA단계에서 정의한 애플리케이션을 기술 컴포넌트 세트로 매핑 할 연결고리를 찾는다. 기술 컴포넌트란 시장에서 얻을 수 있거나 조직의 기술 플랫폼 안으로 들일 수 있는 소프트웨어와 하드웨어 컴포넌트를 말한다. ....[중략]

TA는 **기술 포트폴리오**에 대한 목표 뷰와 기준선을 정의할 것이며, 목표 아키텍처로 가는 로드맵을 상세화한다.... [중략]

TA는 아키텍처 정보를 아우르며 따라서 특정 이전 시나리오를 위한 비용평가 작업을 지원한다.



TOGAF Version 9

# EA 논쟁(7/10) – EA-BA at TOGAF

- ✓ Technology Architecture의 핵심은 AA를 실현할 대상 기술 포트폴리오를 작성하는 것임
- ✓ 보안 아키텍처 설계, 인프라 아키텍처 설계와 같은 [설계 활동]은 TA 범위가 아님
- ✓ “TA는 목표 아키텍처로 가는 로드맵을 상세화하고...”

BA의 목표:

...

다음을 포함하여 목표 비즈니스 아키텍처를 서술함;

비즈니스 원칙과 비즈니스 목표에 근거하여 제품과 서비스 전략, 조직과 업무 프로세스, 정보, 그리고 비즈니스 환경의 지리적인 특성 등을 서술한다.

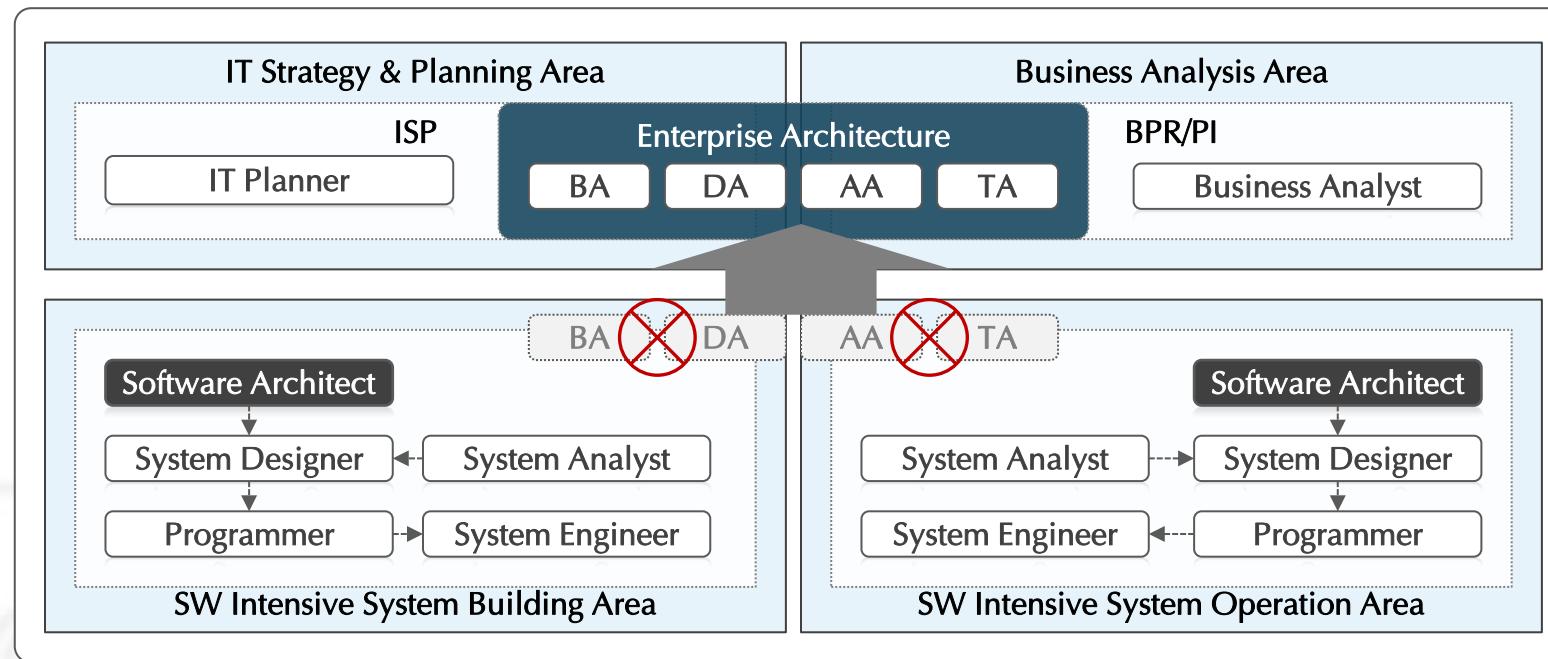
...



TOGAF Version 9

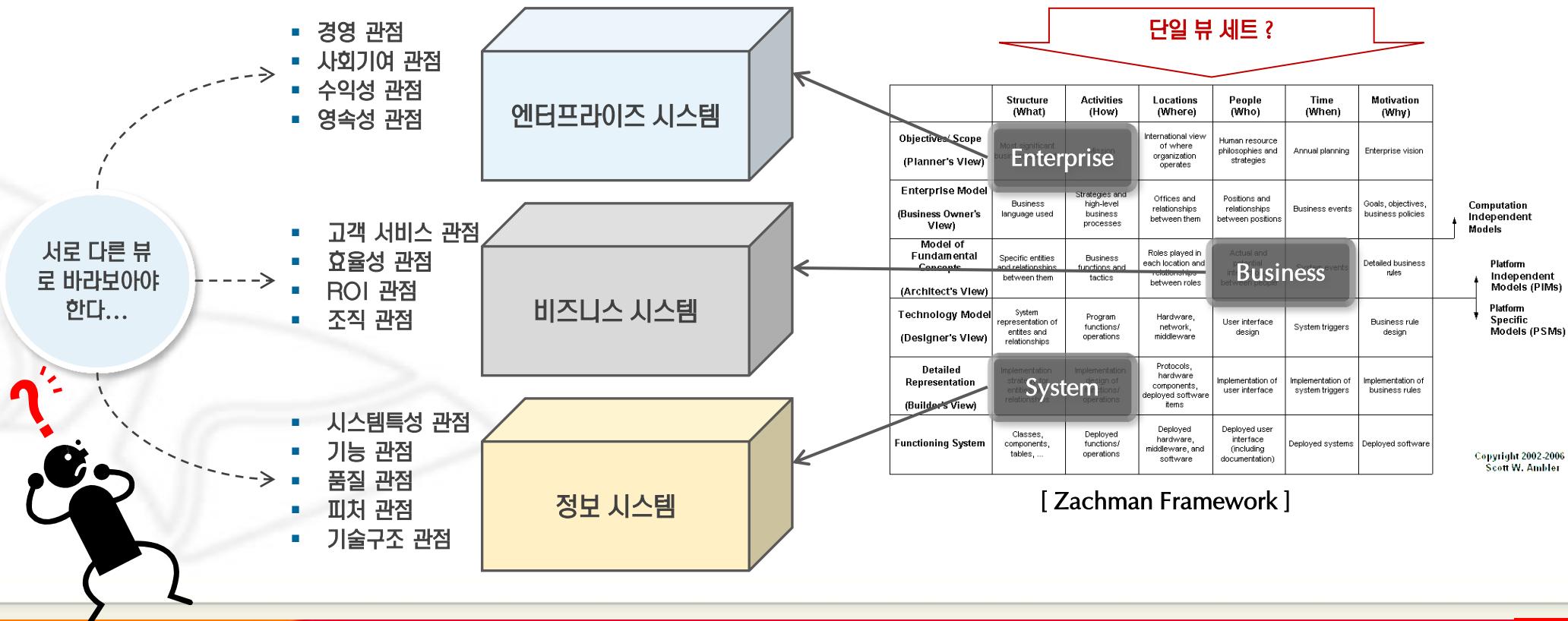
# EA 논쟁(8/10) – EA 포지션

- ✓ 시스템 구축 영역은 다양한 표준화 활동을 통해서 EA에서 이야기하는 Alignment 대상이 거의 없음
- ✓ 시스템 영역의 소프트웨어 아키텍트는 전사 시스템 뷰를 가지고 설계를 수행함
- ✓ “EA는 도시 설계이고 SA는 건축이다”를 주제로 토의합시다.
- ✓ EA를 도시 설계라고 한다면, ISP는 무엇이고, BPR은 무엇이라고 해야 하는가?



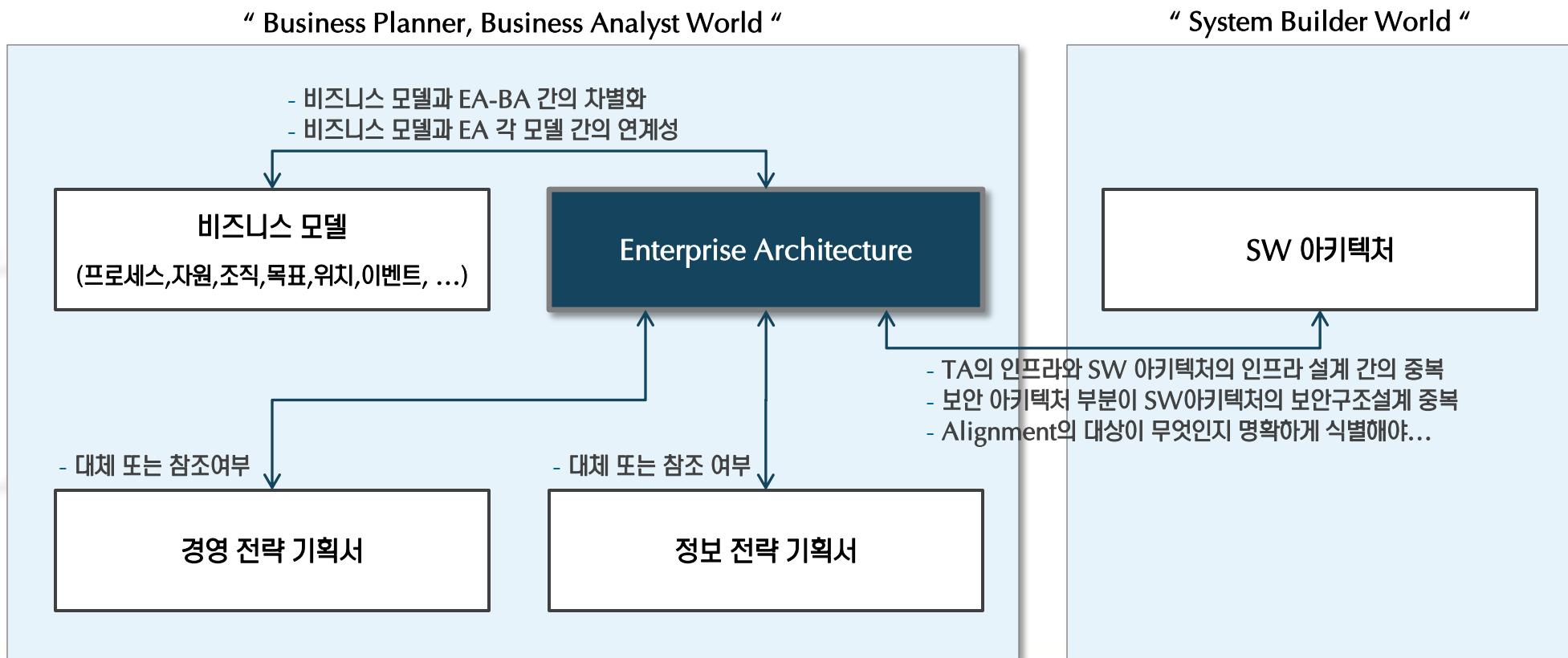
# EA 논쟁(9/10) – Zachman Framework Review

- ✓ 서로 다른 대상은 서로 다른 뷰포인트를 가지고 바라보아야 함
- ✓ 엔터프라이즈 시스템은 경영, 사회기여, 수익의 관점이 필요하고, 비즈니스 시스템은 효율성 관점이 필요하며, 정보 시스템은 기능과 품질, 시스템 특성(aspect) 관점이 필요함
- ✓ 따라서, 세 가지 시스템을 하나의 틀에 놓고 동일한 관점으로 바라보는 것은 타당성이 부족함



# EA 논쟁(10/10) – EA Review

- ✓ 기존에 다양하게 수행하던 Planner 관점의 작업들과 관계를 명확히 해야함
  - PI또는 BPR의 비즈니스 모델, 경영 전략, 정보 전략 등
- ✓ EA는 Planner 뷰를 읽지 말고 Planner 관점의 작업 수행
  - 논리데이터 모델, 물리 데이터 모델, 보안 구조, 인프라 구조 등은 Builder 뷰의 작업임



### 3. IEEE 1471

---

- ❖ 소프트웨어 아키텍처에 대한 표준인 IEEE 1471를 이해합니다.
- ❖ 이를 통해서 표준 관점의 아키텍처 의미를 되새겨 봅니다.
- ❖ 아키텍처 정의의 혼란으로부터 벗어나는 것을 목표로 합니다.

- ✓ **개요**
- ✓ **IEEE 1471 Key Concepts**
- ✓ **IEEE 1471 Close-up**
- ✓ **IEEE 1471 Review**
- ✓ **SA Summary**
- ✓ **토의**

# 개요 – 아키텍처 범람

- ✓ 너무나 많은 아키텍트의 존재로 인해 모두가 혼란스럽군요.
- ✓ 인사 담당자는 아키텍트 직무가 너무 많아 혼란스럽고,
- ✓ 프로젝트 관리자는 아키텍트가 너무 많이 필요해서 어려움이 있고,
- ✓ 엔지니어링 팀은 아키텍처 팀을 어떻게 구성해야 할 지 고민이 있습니다.

❖ 시스템 아키텍트

❖ 솔루션 아키텍트

❖ 보안 아키텍트

❖ 기술 아키텍트

❖ 네트워크 아키텍트

❖ 컴포넌트 아키텍트

❖ 애플리케이션 아키텍트

❖ 비즈니스 아키텍트

❖ 정보 시스템 아키텍트

❖ 데이터 아키텍트

❖ EAI 아키텍트

❖ 포탈 아키텍트

❖ 서비스 아키텍트

❖ 소프트웨어 아키텍트

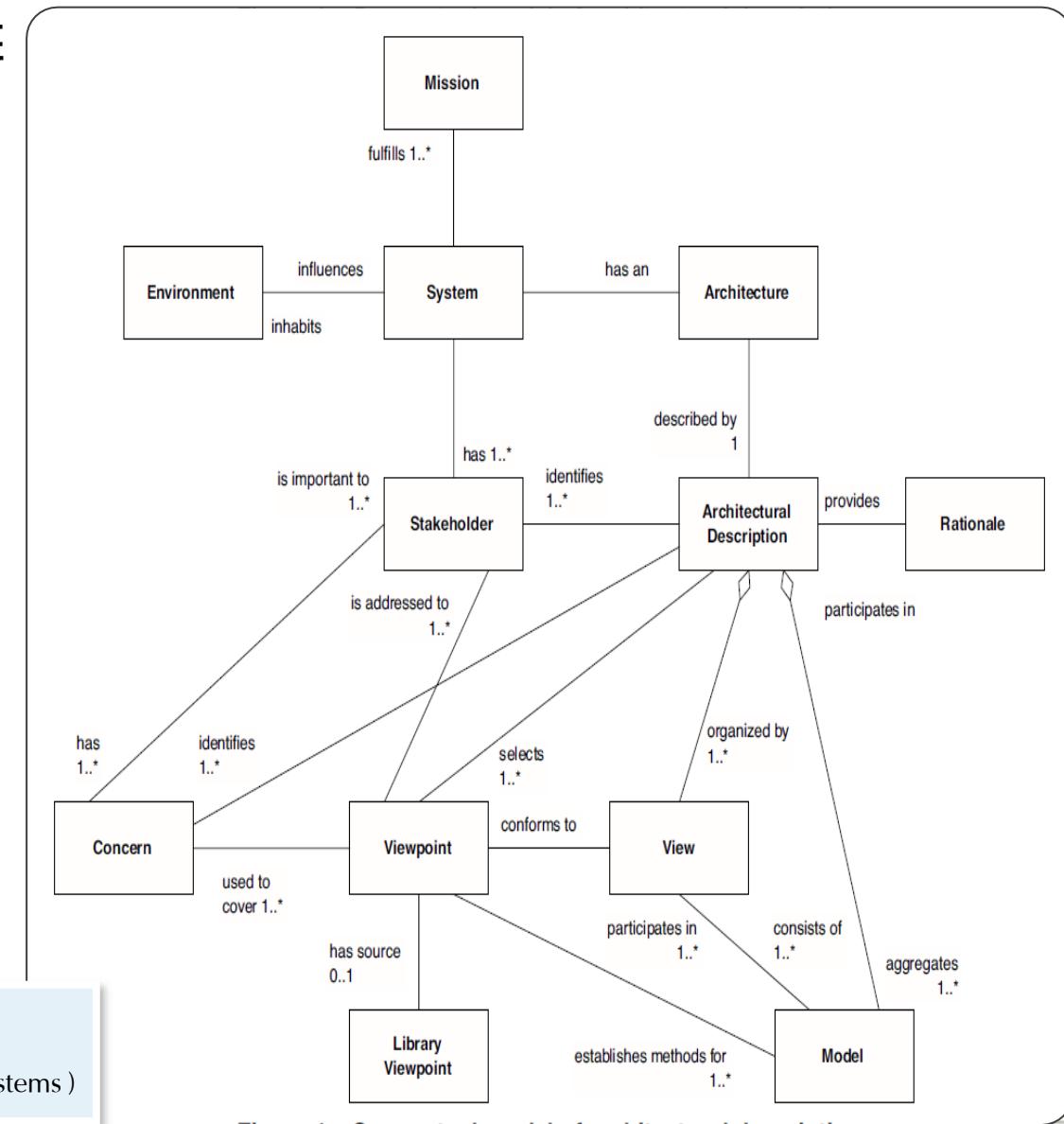
[ 너무나 많은 아키텍트 ]

# 개요 – Why IEEE 1471 ?

- ✓ IEEE 1471는 아키텍처 서술을 위한 권장 실행지침으로 표준에 준하는 위치를 가집니다.
- ✓ 아키텍트의 역할 논쟁을 표준을 근거로 조정하고 정리하는 것이 바람직합니다.
- ✓ IEEE 1471에서 아키텍처를 어떻게 정의하고 있는가로부터 출발합니다.
- ✓ [EA와 TOGAF]에서 정의한 IT 영역 개념도 아키텍처 역할 정리에 필요합니다.

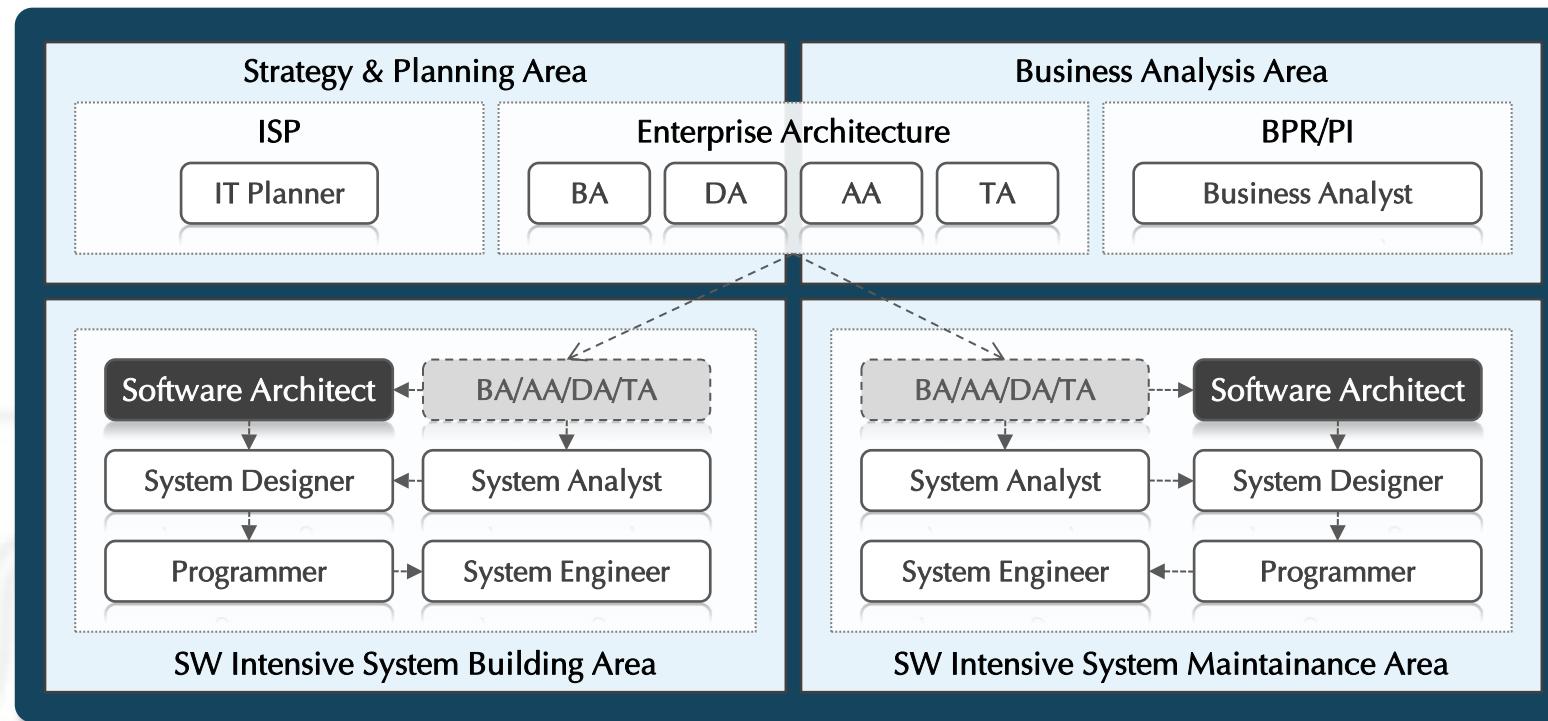
IEEE 1471 - 2000

( IEEE Recommended Practice for Architectural Description of Software-Intensive Systems )



# 개요 - 혼란의 핵심

- ✓ 혼란의 핵심에는 EA에서 정의한 AA, DA, TA, BA가 있습니다.
- ✓ 일부 조직의 경우 Software Architect 는 없고 AA, DA, TA, BA 만 직무로 정의하고 있습니다.
- ✓ 특히 AA와 SA는 역할 중첩 논란이 항상 존재하고 있습니다.
- ✓ EA-AA는 거버넌스를 이유로 시스템 빌딩 영역에서도 지속적으로 활동을 해야 한다고 주장합니다.



# 개요 - 소프트웨어 아키텍처 범위

- ✓ 소프트웨어 아키텍트는 [시스템 구축과 운영 영역]에서 정의하고 사용하며, 전 영역에서 참조하는 역할입니다
- ✓ 소프트웨어 아키텍처는 IEEE 1471에서 정의한 Software Intensive 시스템의 아키텍처를 의미합니다.
- ✓ 소프트웨어 아키텍트는 [Software Intensive System]을 위한 아키텍처를 설계하는 아키텍트입니다.
- ✓ 이러한 범위를 전제로 하고 IEEE 1471를 자세히 분해하여 이해하고자 합니다.



# IEEE 1471 Key Concepts (1/4)

✓ Software Intensive System의 범위(coverage)는?

- 개인용 SW 애플리케이션
- 기업 정보 시스템
- 임베디드 시스템
- SW Product Lines, Product Families
- System-of- [Software intensive] System

What kind of systems does IEEE 1471 cover?

Its focus is software-intensive systems: any system in which software development and/or integration are dominant considerations (i.e., most complex systems these days). This includes computer-based systems ranging from individual software applications, information systems, embedded systems, software product lines and product families and systems-of-systems.

<http://www.iso-architecture.org/ieee-1471/ieee-1471-faq.html>

# IEEE 1471 Key Concepts (2/4)

- ✓ Software Architecture란 무엇인가?
- ✓ CMU 사이트에 가면 아키텍처에 대한 정의를 60여가지 제시합니다.
- ✓ IEEE 1471에서 정의하는 아키텍처를 기준으로 합니다.
- ✓ 컴포넌트 == 아키텍처 요소(architectural element)

아키텍처는 시스템의 근간을 이루는 틀(fundamental organization)로써 시스템을 구성하는 컴포넌트, 컴포넌트 간의 관계, 컴포넌트와 환경 간의 관계, 그리고 설계와 개발 진행을 관리하는 원칙의 형태로 나타난다.

(The fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution)

IEEE 1471-2000

# IEEE 1471 Key Concepts (4/4)

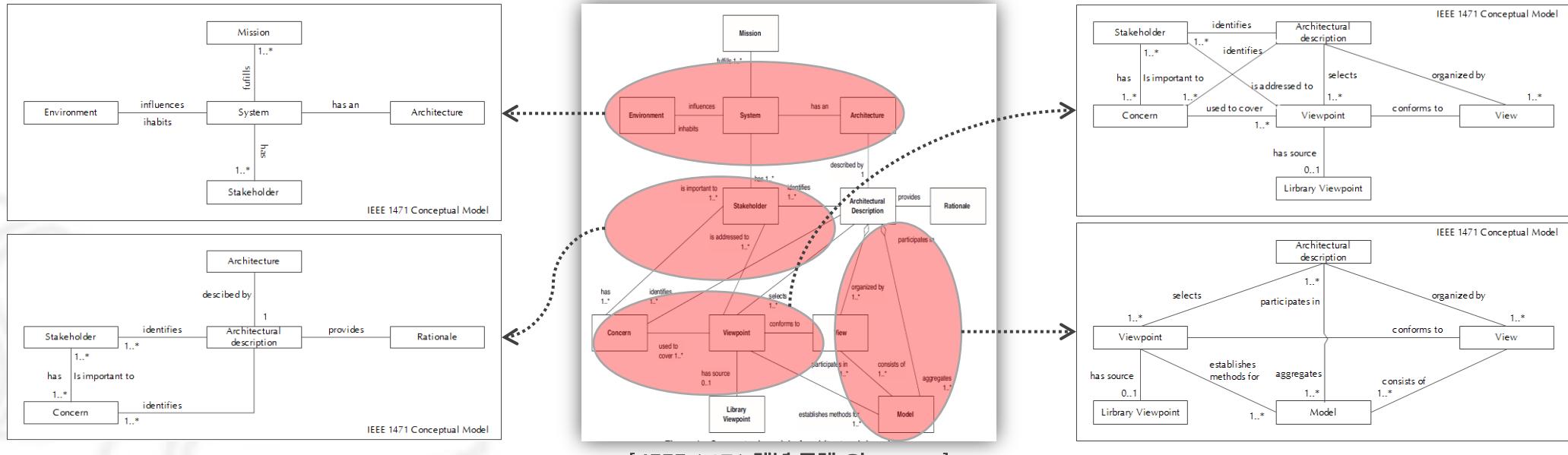
## ✓ 아키텍처 요소(Architectural Element)란 무엇인가?

- 시스템의 틀을 구성하는 요소를 의미합니다.
- 아키텍처 설계 후에 이러한 아키텍처 요소를 획득(개발, 구매, 오픈소스선택)하여 실행 가능한 시스템을 구성합니다.
- 모든 비즈니스 컴포넌트를 아키텍처 요소라고 할 수 있는가?

| 아키텍처 요소 타입 / 획득방법 | 자체개발               | 오픈소스                    | 솔루션 벤더             |
|-------------------|--------------------|-------------------------|--------------------|
| 비즈니스 컴포넌트         | 가입설계, 계약 컴포넌트      | OMG Party컴포넌트           |                    |
| 프레임워크             | 지연처리 프레임워크, 예외처리   | Struts, myBATIS         | T사 프레임워크           |
| 서비스 서버            | Mini-EAI서버, 상품 팩토리 | Artifactory, SVN        | T사의 암호화서버, IM/AM서버 |
| 플랫폼<br>소프트웨어      | 운영체제               | Linux                   | HP-UX, Sun Solaris |
|                   | 개발 언어              | Physon, Groovy          | Java, C, C++       |
|                   | 실행 플랫폼             | JBoss, Spring, JVM      | WebLogic, PaaS     |
|                   | 데이터 플랫폼            | MariaDB, MongoDB, Redis | Oracle RDBMS       |
|                   | 프로세스 플랫폼           | ESBWorkflow, uEngine    | BPM                |
|                   | 비즈니스 플랫폼           | Compiere                | SAP, Oracle ERP    |
|                   | 관리용 소프트웨어 (제외)     | 컴포넌트 관리, 배포관리           | Tivoli, OpenView   |

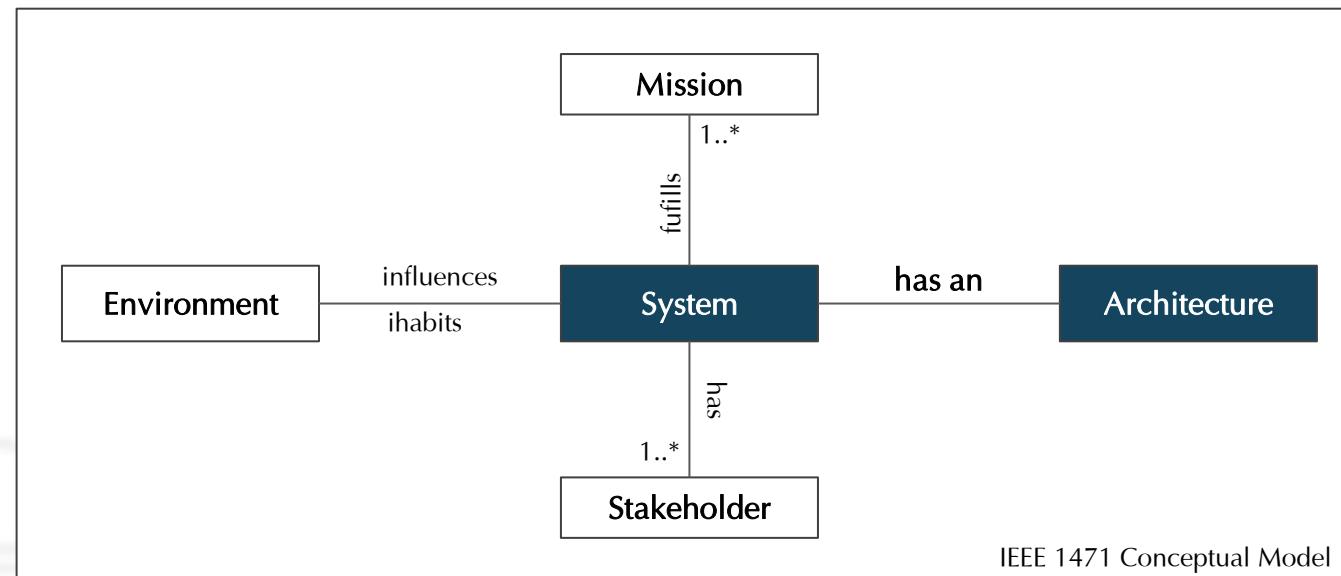
# IEEE 1471 Close-up

- ✓ IEEE 1471 개념 모델은 추상화 수준이 높은 모델로 한 번에 이해하기 어렵습니다.
- ✓ 따라서, 각 부분 별로 확대하여 부분별로 이해한 후 전체를 보는 것이 편리합니다.
- ✓ IEEE 1471 개념 모델을 네 개의 부분으로 나누어서 이해하고자 합니다.



# IEEE 1471 Close-up – System has an architecture.

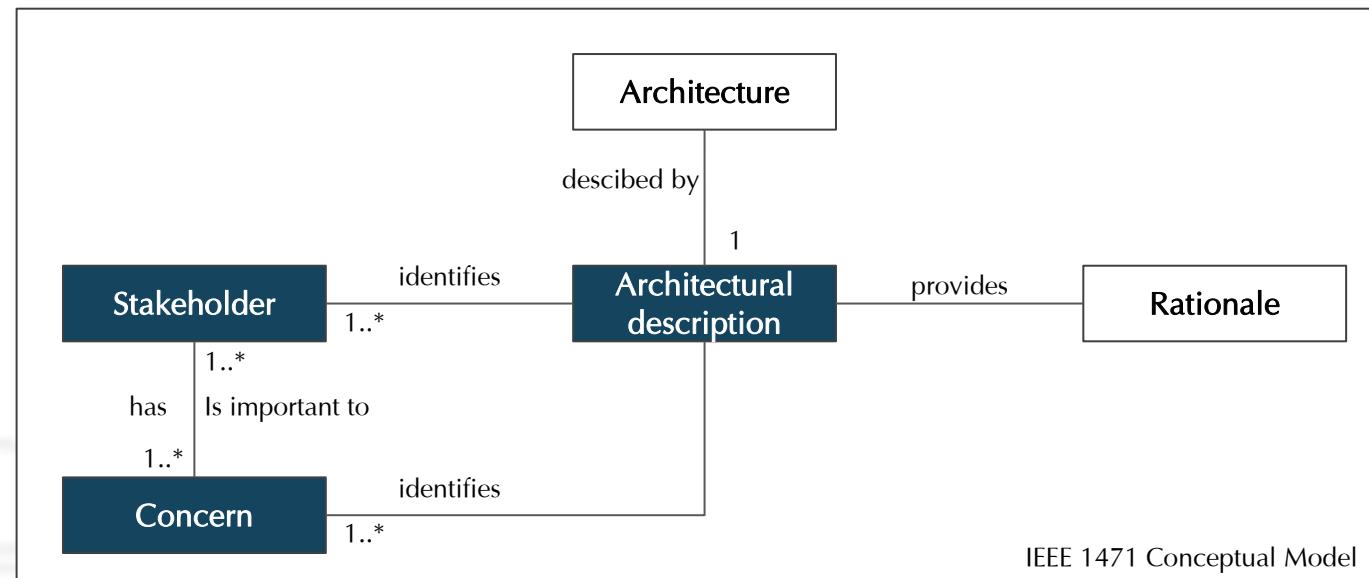
- ✓ 시스템은 [하나의] 아키텍처를 가진다.
- ✓ 시스템은 어떤 환경 속에서 미션을 수행한다. 이 환경은 언제나 제약조건을 가지고 있다.
- ✓ 시스템을 둘러싸고 있는 많은 이해관계자가 존재한다.
- ✓ 시스템은 아키텍처 [하나]를 가진다.



[ IEEE 1471 Close-up : System has an Architecture ]

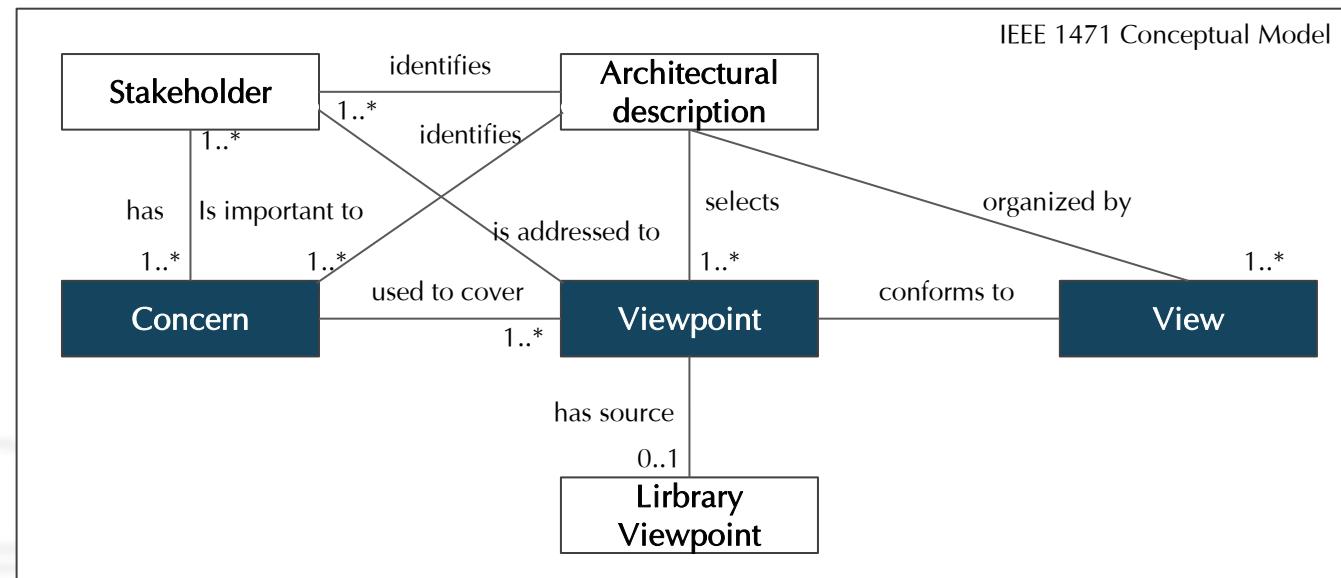
# IEEE 1471 Close-up – Architectural Description

- ✓ 아키텍처는 아키텍트의 머릿속에 존재하는 구상(conception)으로 아키텍처 서술로 표현합니다.
- ✓ 아키텍처 서술은 논리적인 근거 또는 타당성(rationale)을 제공해야 합니다.
- ✓ 아키텍처 서술에서 이해관계자와 그들의 관심사(concern)를 식별하고 명세합니다.
- ✓ 이해관계자는 여러 관심사를 가지고 있으며, 관심사는 아키텍처 설계를 통해서 해결하여야 합니다.



# IEEE 1471 Close-up – Viewpoint and concerns.

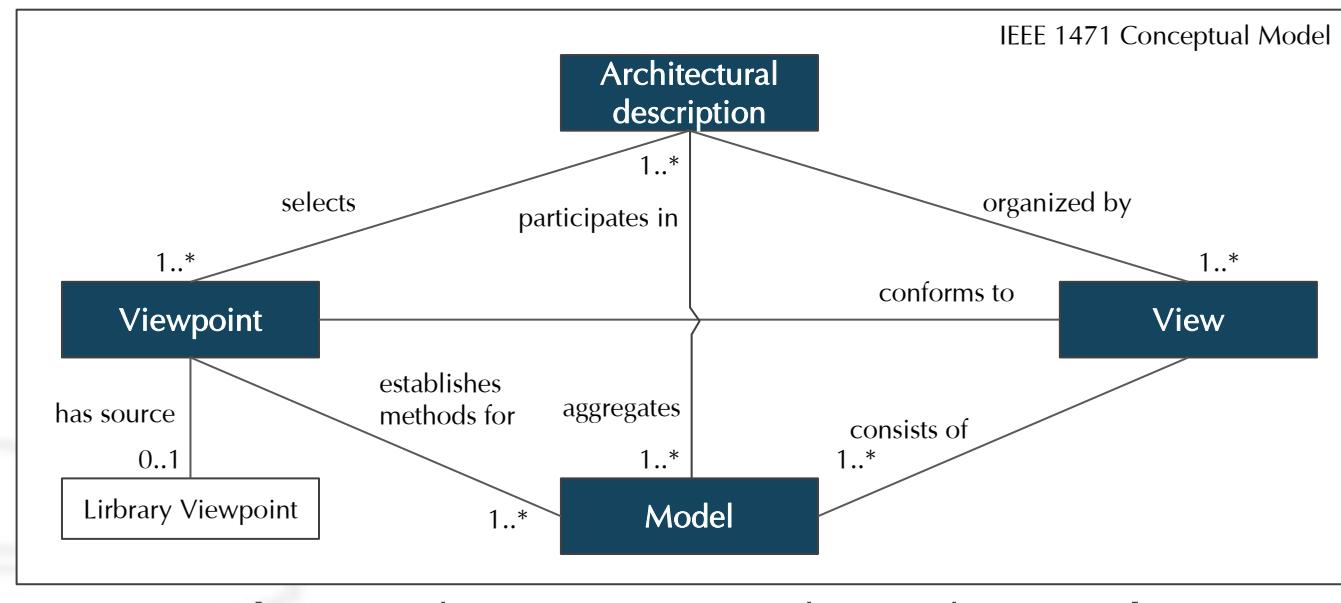
- ✓ AD는 여러 개의 Viewpoint를 가지고, Viewpoint는 관심사를 표현하는데 사용합니다.
- ✓ Viewpoint는 유사한 관심사의 묶음 단위로 설정됩니다.
- ✓ View는 Viewpoint 명세를 준수합니다.
- ✓ 조직에서 미리 정의한 표준 Library Viewpoint로부터 프로젝트를 위한 Viewpoint를 가져올 수 있습니다.



[ IEEE 1471 Close-up : Viewpoint is used to cover the Concerns ]

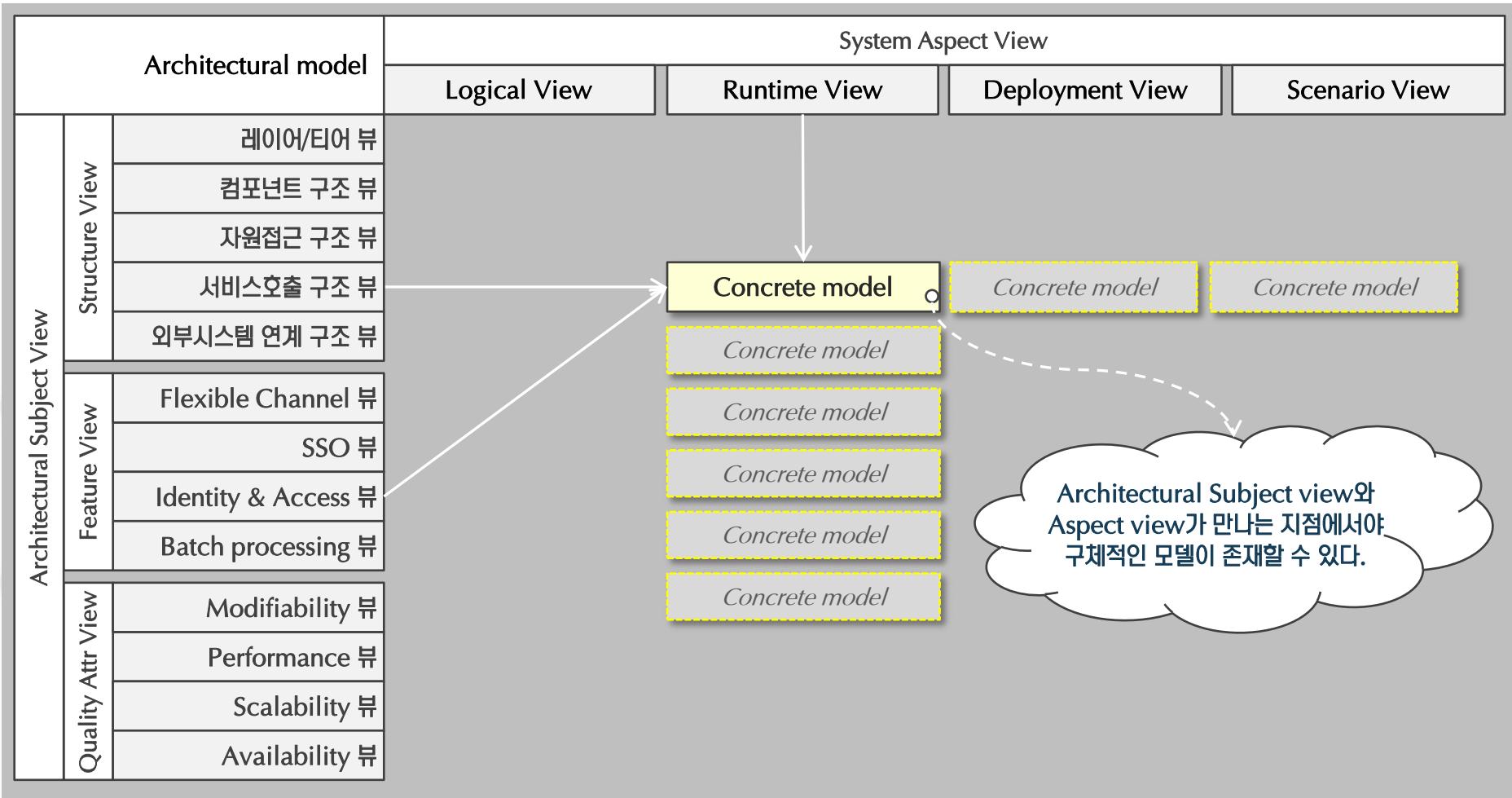
# IEEE 1471 Close-up – View and Model (1/2)

- ✓ AD에는 여러 Model이 필요합니다.
- ✓ Model과 View는 n:n 관계를 가집니다.
- ✓ View와 Model 관계 이해는 IEEE 1471 개념 모델을 이해하는 데 매우 중요합니다.
- ✓ 여기서 Model은 어떤 Model을 의미할까요?



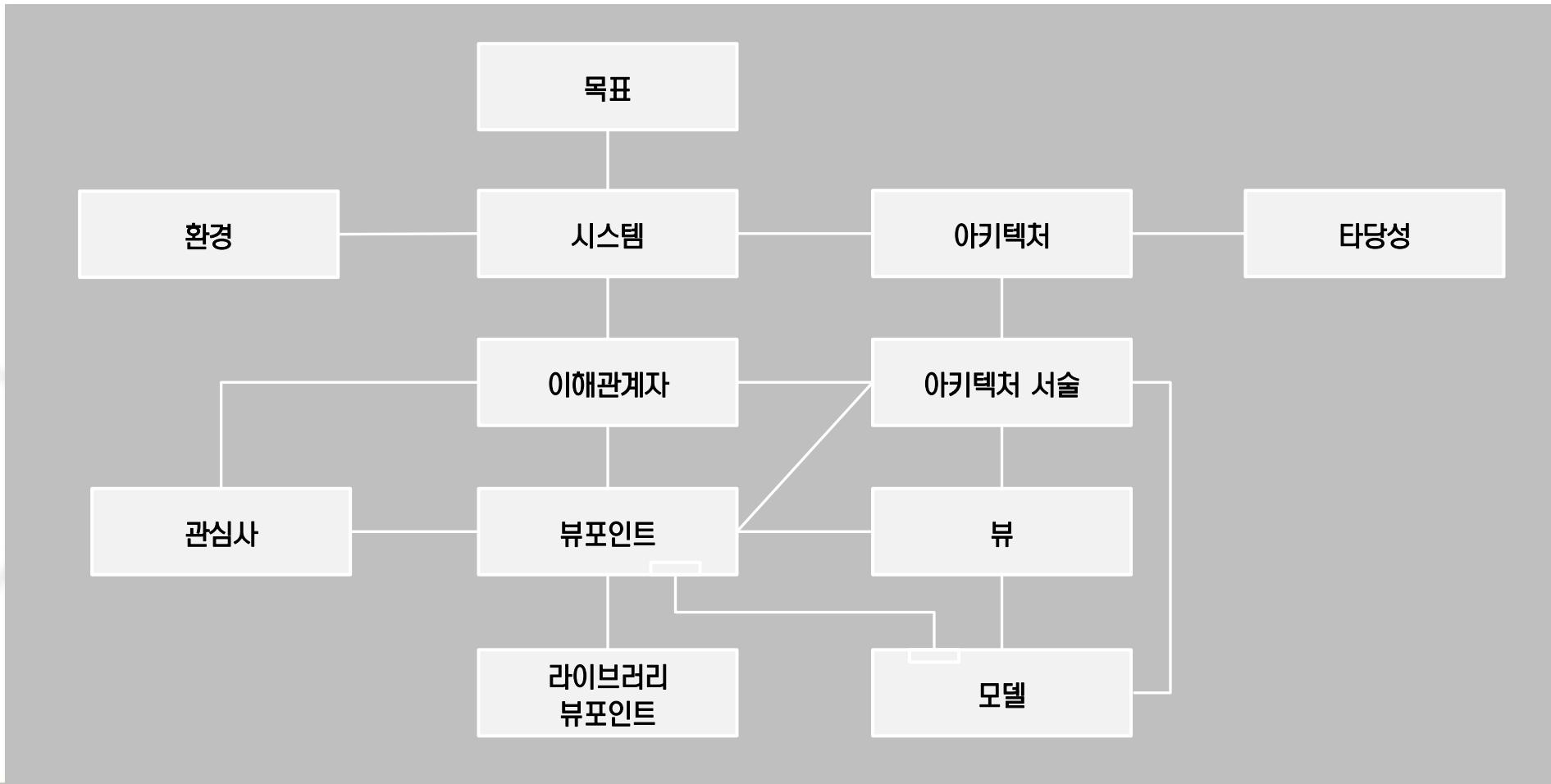
# IEEE 1471 Close-up – View and Model (2/2)

- ✓ View와 View 가 만나는 지점에서 구체적인 Model이 존재합니다.
- ✓ 아래 도표의 의미에 대해 토론합니다.



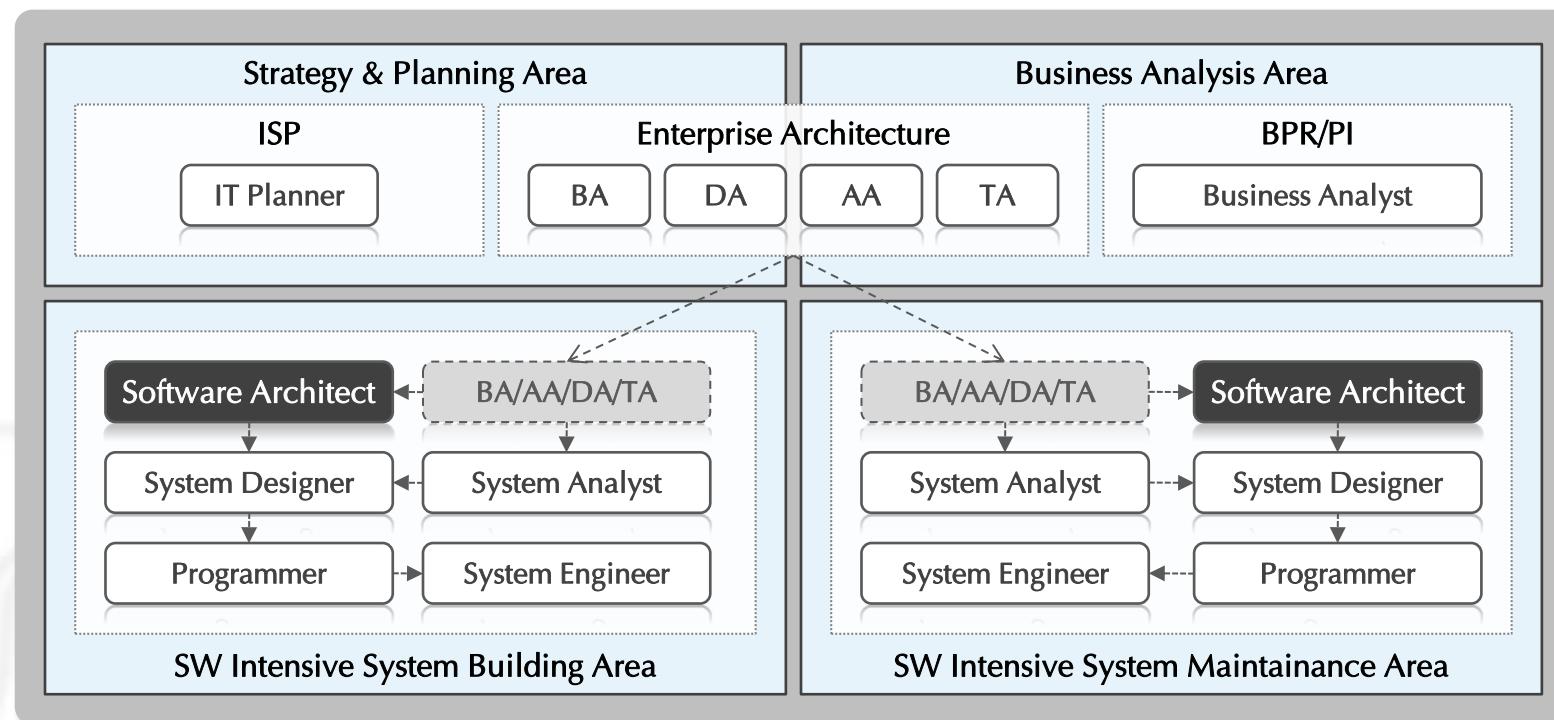
# IEEE 1471 Review

- ✓ 아키텍처 서술을 위한 개념 모델의 의미를 되새겨 봅니다.
- ✓ 이 개념 모델을 확장하여 아키텍처 설계를 위한 구체적인 모델로의 확장에 대해 토론합니다.



# SA Summary

- ✓ AA, DA, TA, BA의 정확한 정의에 대한 정확하게 이해합니다.
- ✓ IEEE 1471 기반으로 Software Architecture를 정확하게 이해합니다.
- ✓ 이들 간의 차이에 대해 토론합니다.
- ✓ SA = [S]oftware intensive system's [A]rchitecture



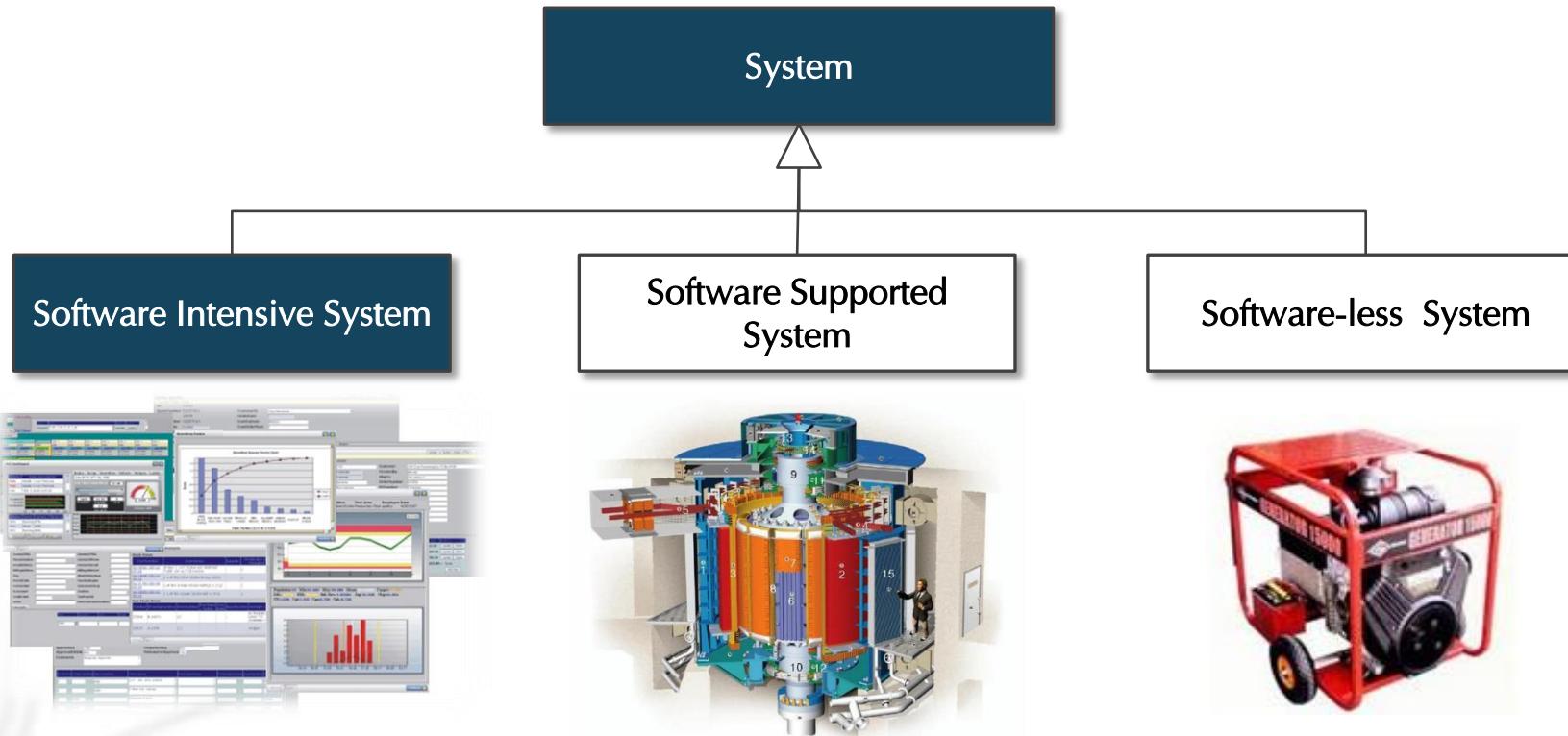
## 4. 품질 속성

### ❖ 목차

- ✓ 시스템
- ✓ 소프트웨어 Intensive 시스템
- ✓ 기능, 품질, 피처
- ✓ 기능과 품질 관계
- ✓ 시스템 품질
- ✓ 품질속성 시나리오
- ✓ 품질속성 시나리오 효율성
- ✓ 품질속성과 프로세스
- ✓ 요약
- ✓ 토의

# 시스템

- ✓ 다양한 유형의 시스템이 있으며, 그 다양성으로 인해 분류체계를 제시하기 어려움
- ✓ 소프트웨어를 중심으로, 전혀 없거나, 또는 지원 역할을 하거나, 주(main)가 되는 시스템으로 분류할 수 있음
- ✓ 소프트웨어 아키텍처 설계 대상이 되는 시스템은 소프트웨어가 주가 되는 시스템임



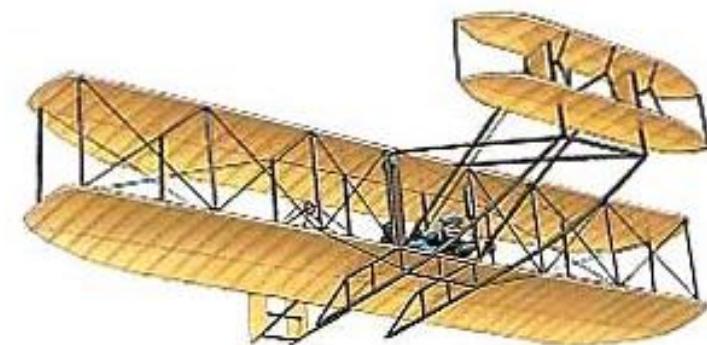
[ 돌극형 발전기, 출처:두산중공업홈페이지 ]

# 소프트웨어 Intensive 시스템

- ✓ 라이트 형제가 만든 인류 최초 비행기인 Flyer 1은 Software-less 시스템이었음
- ✓ 현재 생산되는 보잉기 원가의 40%는 소프트웨어 비용이며, 기능의 80%는 소프트웨어가 담당하고 있음
- ✓ 시스템에서 소프트웨어가 차지하는 비용과 역할은 점차 증가하고 있음

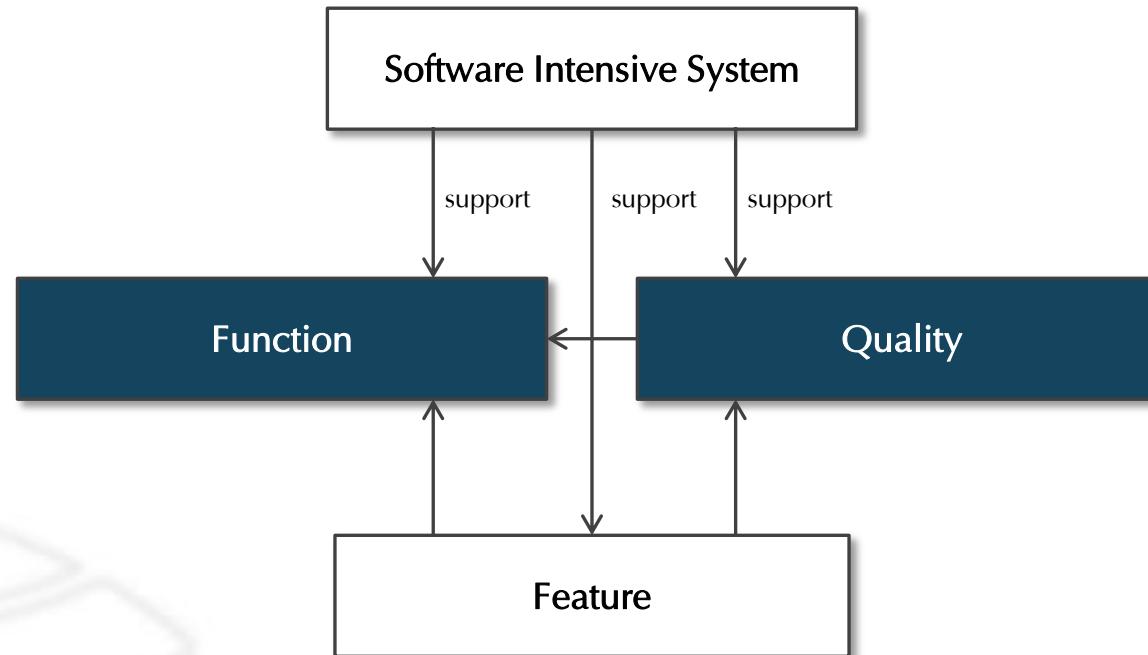
Software Intensive System

Software-less System



# 기능, 품질, 피처

- ✓ 시스템은 사용자에게 기능(function)을 제공하고, 기능은 사용자가 요구하는 수준의 품질 범위 안에서 제공되어야 함
- ✓ 품질은 그 자체로써 의미가 없으며, 항상 기능을 통해서만 가치를 보여줄 수 있음
- ✓ Feature는 시스템을 제공하는, 강조할 만한 것으로 기능과 품질을 모두 포함하는 개념임



# 기능과 품질 관계

- ✓ 기능과 품질은 직교(orthogonal) 관계임
- ✓ 다리를 하나의 시스템으로 보았을 때, 다리의 상판은 기능을, 교각은 품질을 결정함
- ✓ 교각을 먼저 건설하듯이, 시스템 설계에서 품질에 대한 설계가 우선되어야 함, 물론 기능에 대한 이해가 선행되어야 함



프랑스 남부 미요에 위치한 다리로 산악지대를 흐르는 터른 강 계곡을 가로지름, 2004년에 공식 개통함

# 시스템 품질 (1/5) – 종류

- ✓ 소프트웨어 Intensive 시스템의 품질에는 어떤 것들이 있는가?
- ✓ 조직마다 서로 용어로 품질을 표현하고 있으며, 각 품질 항목을 시스템의 품질 속성(Quality Attribute)라고 함
- ✓ Embedded 시스템에서는 SEI에서 제시하는 품질 속성을 사용하며, 엔터프라이즈 시스템에서는 TOGAF를 참조함

## 시스템의 품질 속성(From SEI)

- 가용성(availability)
- 변경 용이성(modifiability)
- 수행성능(performance)
- 보안성(security)
- 테스트 용이성(testability)
- 사용성(usability)
- 상호운영성(interoperability)
- 이식성(Portability)
- 범위성(Scalability)

조직에서  
정의한  
품질속성은 ?

## 비즈니스의 품질속성 (From SEI)

- 시장 출하 시기(time to market)
- 비용과 장점(cost and benefit)
- 시스템의 예정 생명주기(projected lifetime of the system)
- 목표 시장(targeted market)
- 최초 공개 일정(rollout schedule)
- 기존 시스템과의 통합(integration with legacy systems)

## 시스템의 품질 속성(From TOGAF)

### Availability

- manageability
- serviceability
- performance
- reliability
- recoverability
- locatability

### Assurance:

- security
- integrity
- credibility

### Usability

- International operation

### Adaptability

- interoperability
- scalability
- portability
- extensibility

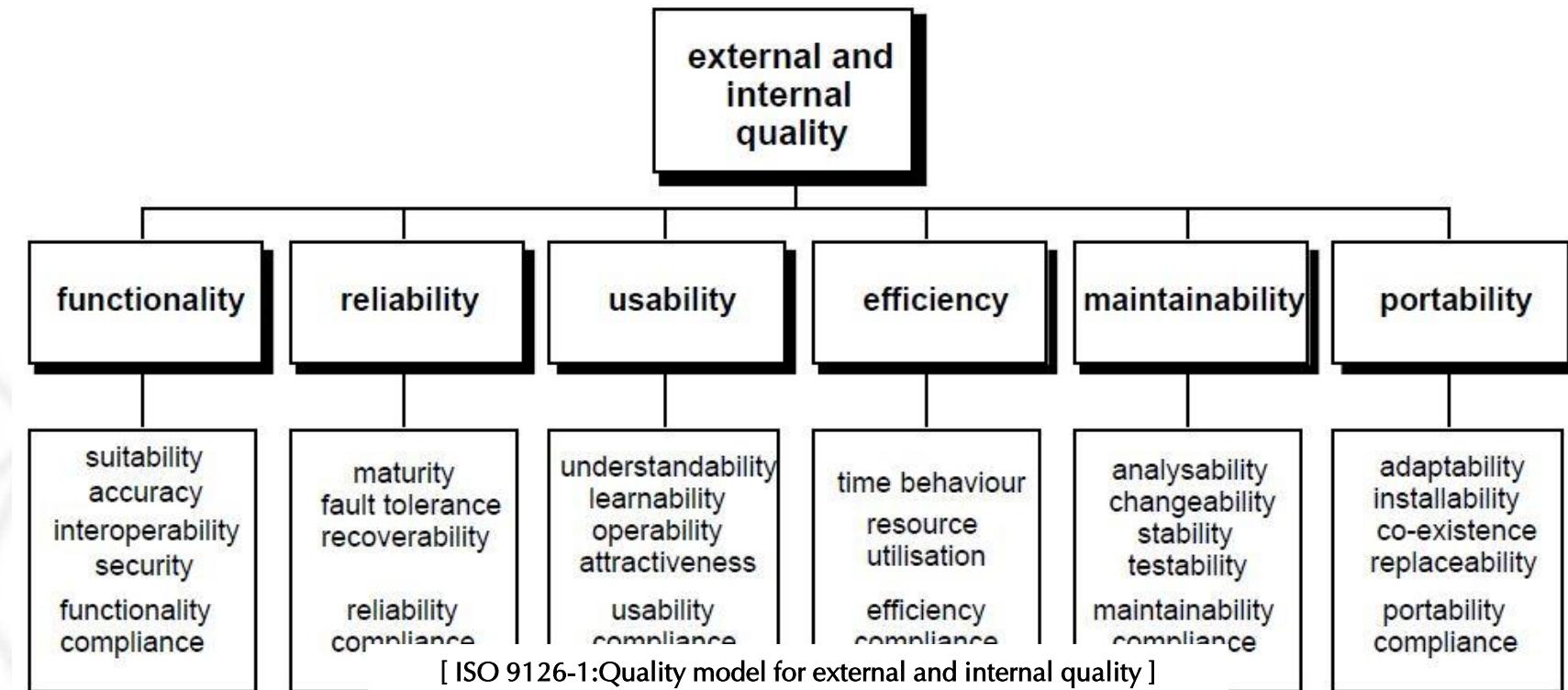
## 시스템 품질 (2/5) – 표현

- ✓ 품질은 기능을 통해 확인되고 검증될 뿐만 아니라 기능 별로 서로 다른 수준의 품질을 요구할 수 있음
- ✓ 품질수준을 표현하는 단위를 표현하기 위한 ISO-9126의 기준도 있지만, 기업용 소프트웨어에서 적용사례는 찾기 어려움
- ✓ 품질 요구수준은 조직이나 프로젝트 별로 적절한 수준의 표준을 정의한다면 매우 유용할 것임

| 구분   |      |           | 경매등록             | 경매물품조회 | 입찰    | 낙찰금액지불    | 경매마감   |
|------|------|-----------|------------------|--------|-------|-----------|--------|
| 품질속성 | 명세내용 | 단위        |                  |        |       |           |        |
| 가용성  | 중요도  | 상중하       | 중                | 상      | 상     | 중         | 상      |
|      | 허용범위 | 다운시간/년    | 30H              | 10H    | 20H   | 30H       | 1H     |
| 성능   | 중요도  | 상중하       | 하                | 상      | 상     | 하         | 상      |
|      | 허용범위 | 초(sec)    | 3                | 1      | 1     | 5         | 1      |
| 범위성  | 중요도  | 상중하       | 하                | 상      | 중     | 하         | 하      |
|      | 허용범위 | 동시 사용자수   | 5천               | 10만명   | 1만명   | 5천명       | 5천명    |
| 보안   | 중요도  | 상중하       | 중                | 하      | 중     | 상         | 하      |
|      | 허용범위 | 인증/인가/암호화 | 인증/인가            | 인증     | 인증/인가 | 인증/인가/암호화 | 해당사항없음 |
| 확장성  | 중요도  | 상중하       | 중                | 상      | 중     | 중         | 하      |
|      | 허용범위 | 작업일       | 3                | 3      | 3     | 5         | 3      |
| 관리성  | 중요도  |           | Cross-cutting 이슈 |        |       |           |        |
|      | 허용범위 |           |                  |        |       |           |        |
| 사용성  | 중요도  | 상중하       | 중                | 상      | 상     | 중         | 하      |
|      | 허용범위 | 사용자만족도    | 80%              | 95%    | 90%   | 80%       | 해당사항없음 |

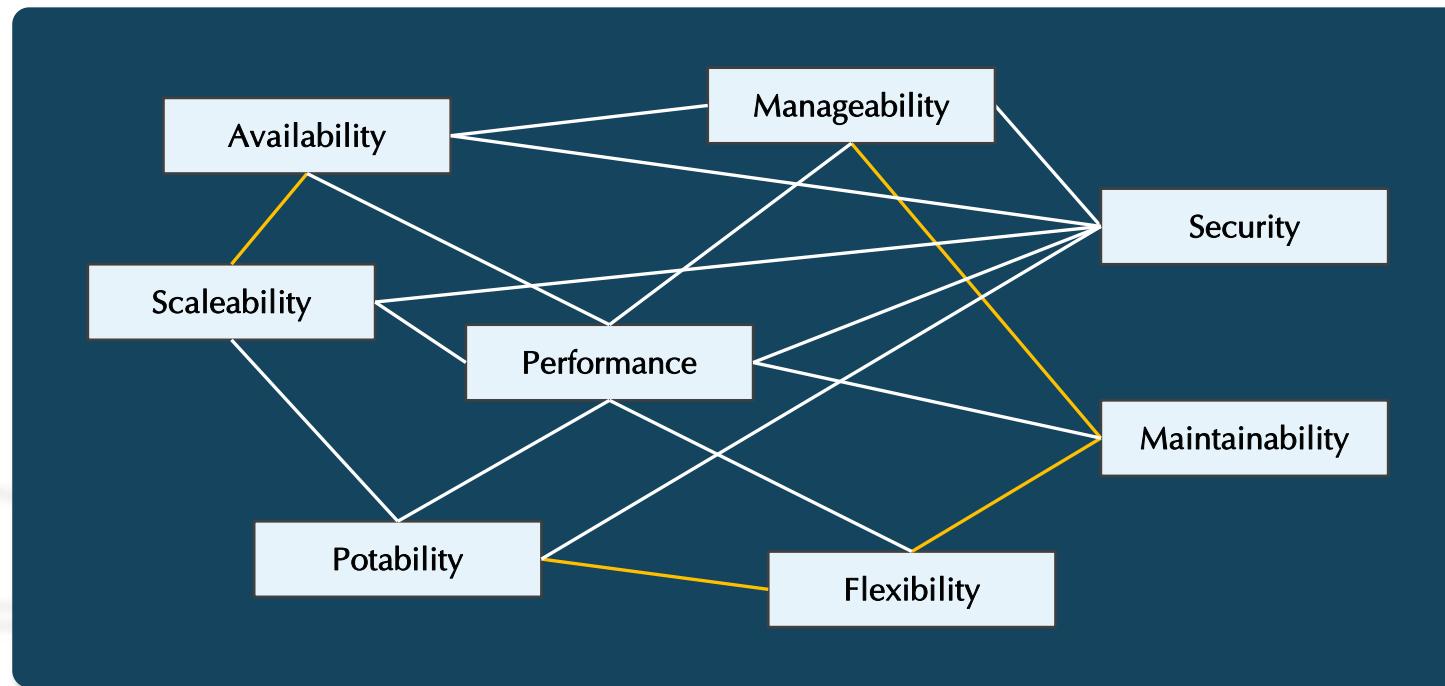
# 시스템 품질 (3/5) – 표준

- ✓ ISO-9126에서 소프트웨어의 품질 유형과 특성을 정의하였음
- ✓ 소프트웨어의 규모가 작지만 매우 중요한 시스템(mission-critical)에서 ISO-9126을 준수하기가 더 쉬움
- ✓ 표준 기반의 품질 목표를 달성하기 위해서는 많은 시간과 비용이 수반되므로, 시스템에 맞는 기준을 찾아서 적용해야 함



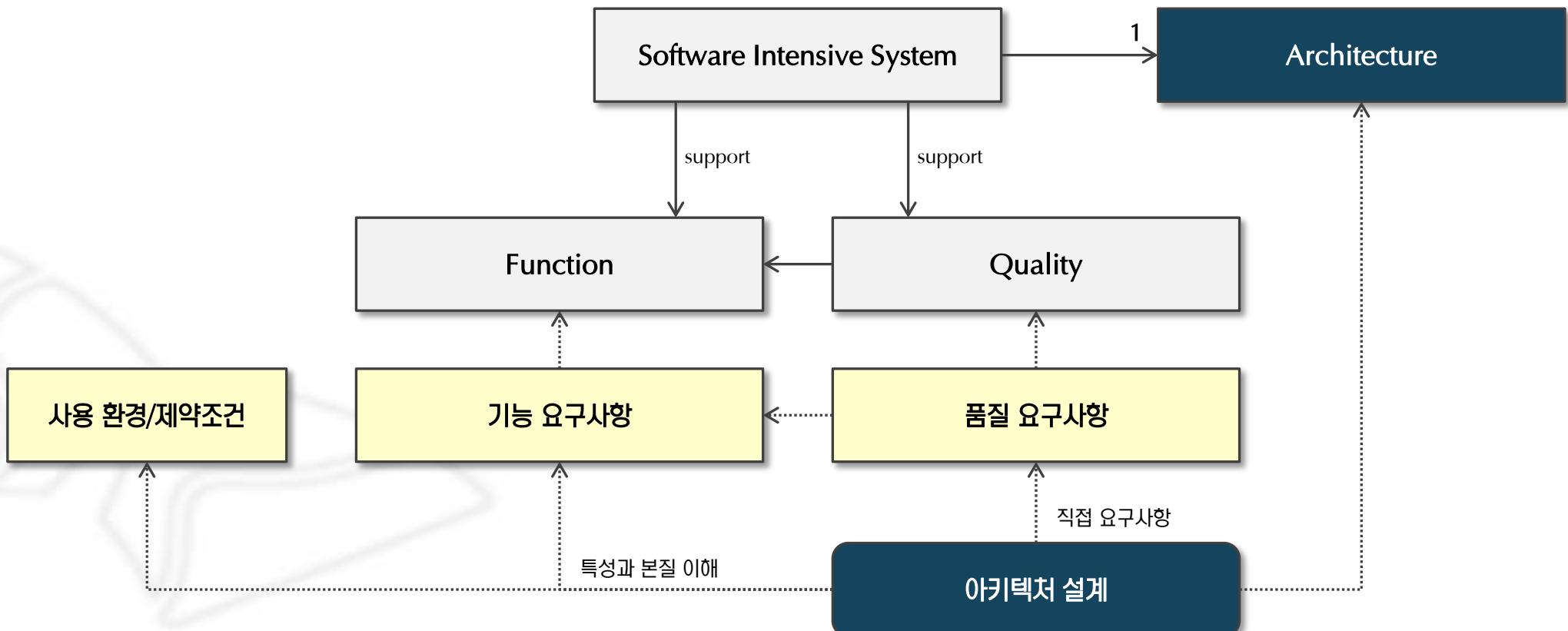
## 시스템 품질 (4/5) – 상관관계

- ✓ 시스템의 품질 속성 상호 간에는 대체로 부정(negative) 관계가 형성됨, 노란색은 긍정(positive)관계를 표현함
- ✓ 예를 들면, 가용성(availability)을 높이기 위해 서버를 늘리면, 성능(performance)과 보안(security)에 문제가 발생함
- ✓ 품질 수준을 동시에 모두 높이려 하면 시간과 비용의 상승을 피할 수 없음



# 시스템 품질 (5/5) – 아키텍처

- ✓ 제약조건이 있는 환경에서 원활한 운영을 위해서 시스템은 품질목표를 가짐
- ✓ 시스템의 품질 목표 달성을 위해 아키텍처 설계를 하고, 아키텍처 설계는 다양한 제약조건을 극복하여야 함
- ✓ 품질은 기능을 통해 표현되므로, 아키텍처 설계 시 기능요구사항과 사용환경에 대한 이해가 선행되어야 함



## 품질속성 시나리오 (1/5) – 개요

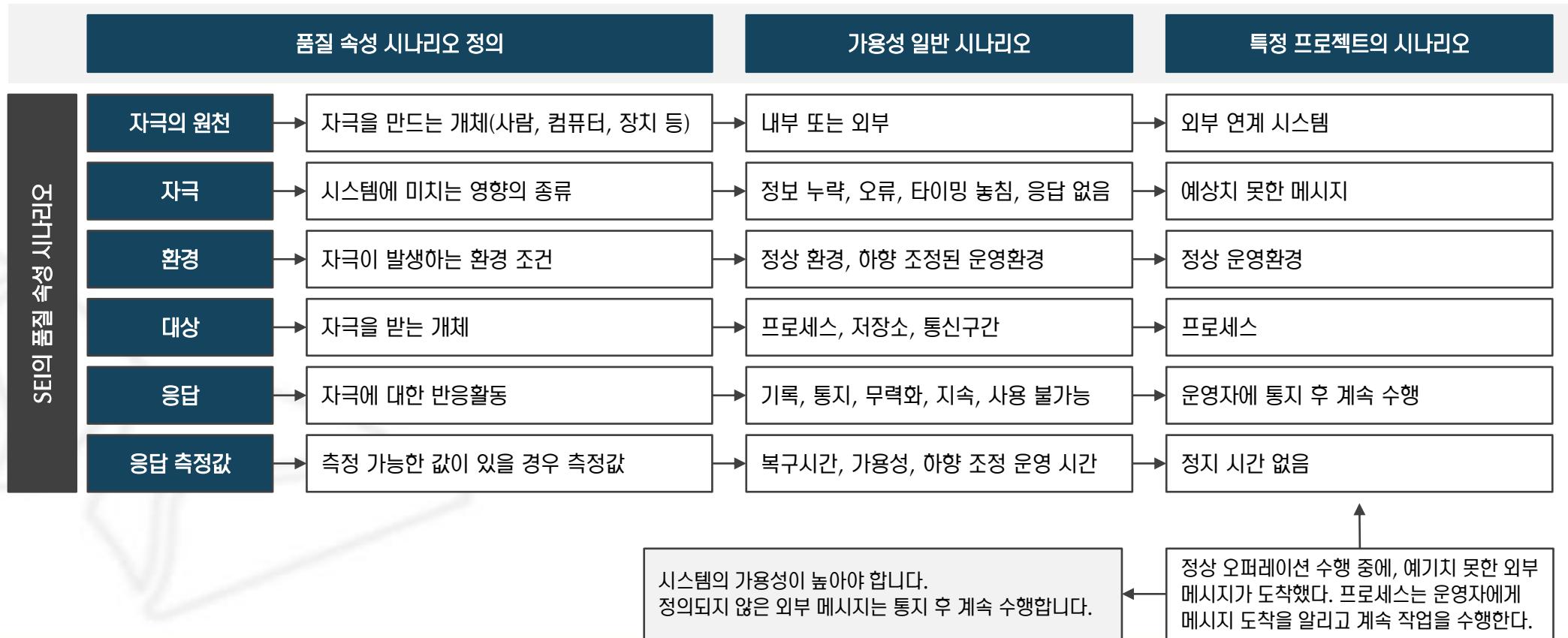
- ✓ 우리의 품질 표현방식은 이해관계자들과 소통하기에 충분한 정보를 담고 있는가?
- ✓ 품질이 중요하다면 보다 명확한 기준이 필요하지 않은가?
- ✓ 품질의 특성상 상세함의 표현에는 다양한 정도의 차이가 있을 수 있음



- ✓ 경고등은 0.1초 내에 동작하여야 한다.
- ✓ SMS 메시지는 반드시 전송되어야 한다.
- ✓ 고객등록은 24시간 365일 가능해야 한다.
- ✓ 비밀번호 확인은 신속히 이루어져야 한다.

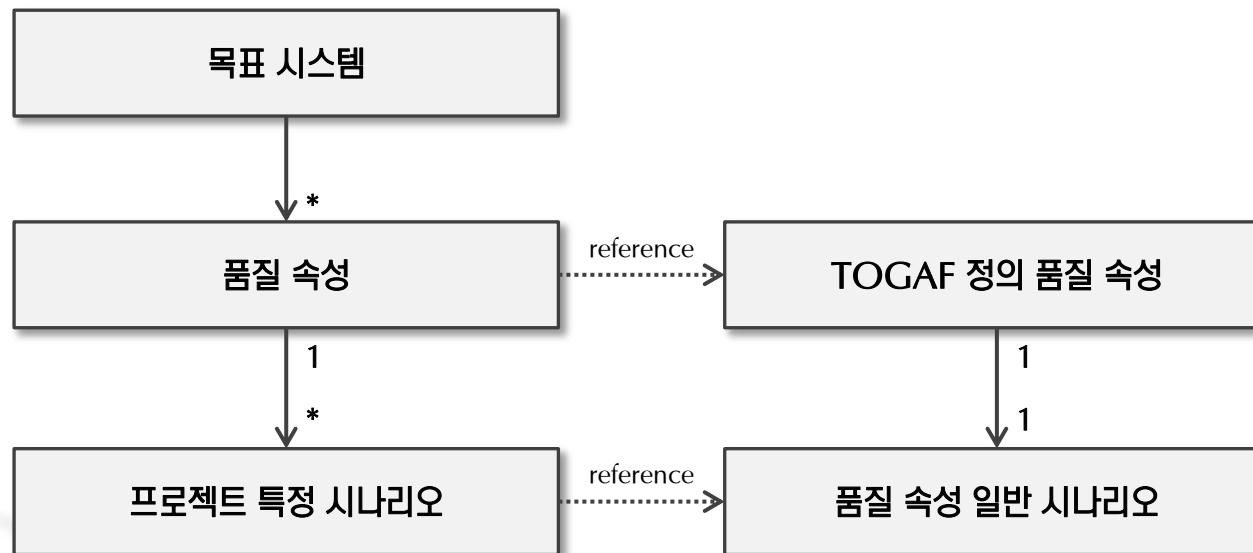
# 품질속성 시나리오 (2/5) – 개요

- ✓ 품질을 어떻게 구체적으로 표현하는가에 대한 문제에 대한 대답이 필요함
- ✓ 품질을 여섯 가지 요소를 기준으로 표현하는 품질 속성 시나리오가 있음
- ✓ 품질 속성 시나리오는 품질 요구사항을 보다 명확하게 표현하는 틀을 제공함



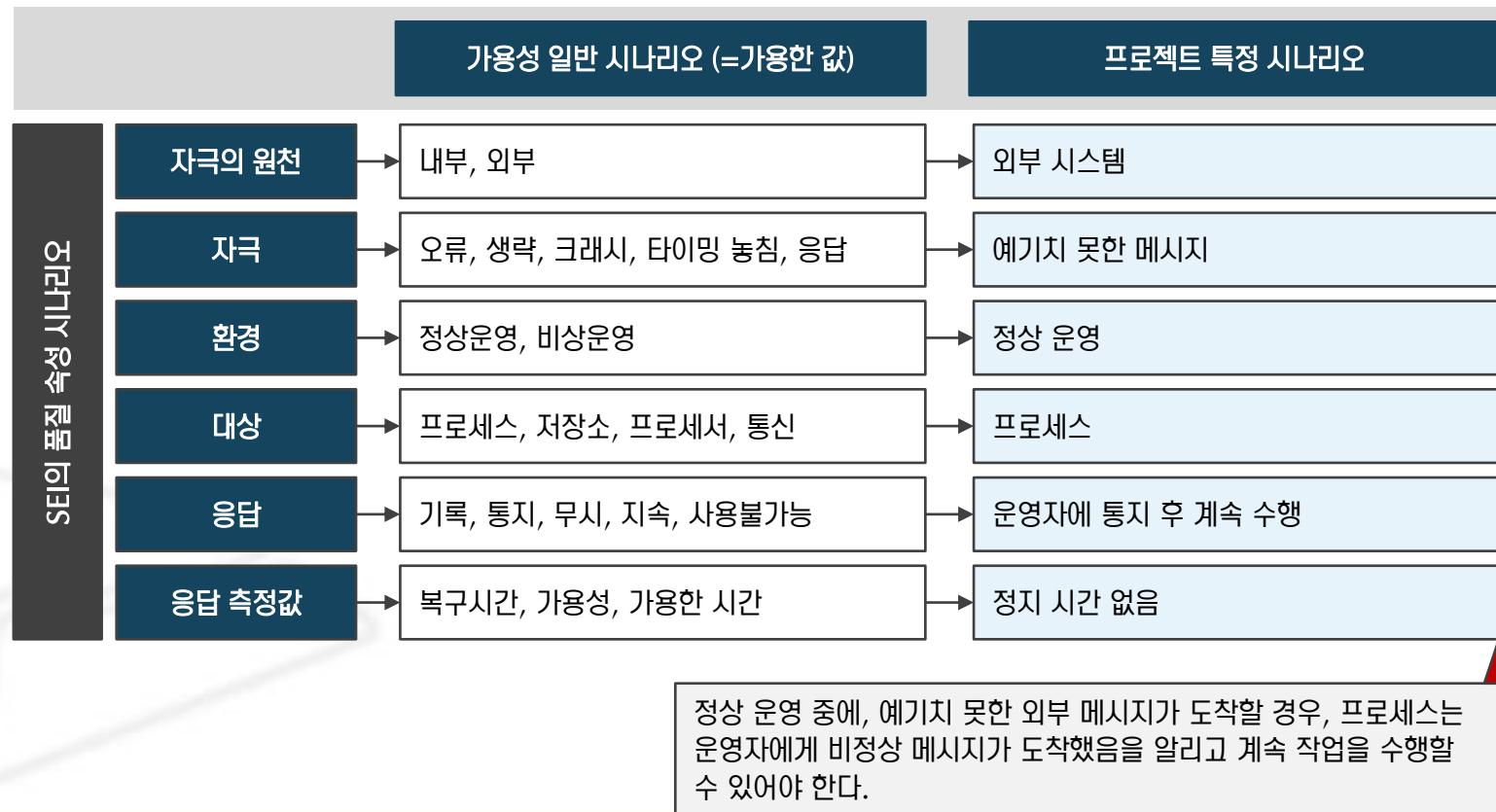
## 품질속성 시나리오 (3/5) – 개념모델

- ✓ 목표 시스템은 조직에서 참조하거나 미리 정의해 놓은 품질 속성 여러 개를 가질 수 있음
- ✓ 품질 속성 일반 시나리오는 프로젝트 특정 시나리오를 구성할 때, 선택 가능한 값들을 모아 놓은 것임
- ✓ 품질속성에 대한 프로젝트 특정 시나리오의 개수에는 제한이 없지만, 반드시 필요한 시나리오로 필요한 최소한 구성함



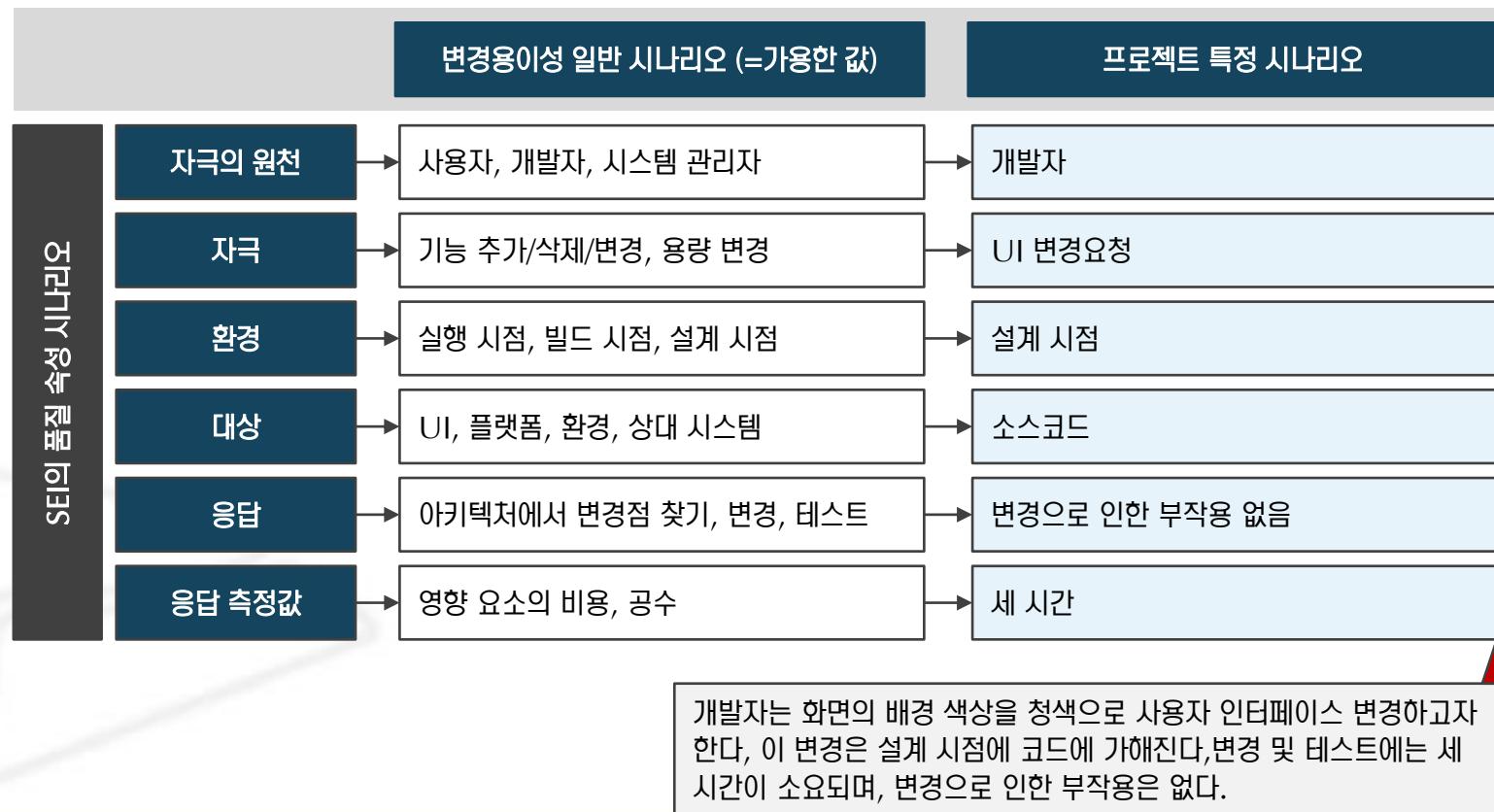
# 품질속성 시나리오 (4/5) – 예제 1

- ✓ 가용성은 시스템의 실패, 그리고 실패와 관련된 결과를 다룸
- ✓ 가용성(availability)은 필요할 때 운영 가능한 확률임, 따라서 가용성은  $(\text{실패평균시간}/(\text{실패평균시간}+\text{복구평균시간}))$ 임
- ✓ 관련 분야는 실패감지, 실패 빈도, 실패 결과, 실패 허용, 실패 방지, 실패 시 통지 등이 있음



# 품질속성 시나리오 (4/5) – 예제 2

- ✓ 변경 용이성 일반 시나리오는 프로젝트 특정 시나리오를 구성할 때 선택 가능한 값들을 가지고 있음
- ✓ 프로젝트 특정 시나리오는 문장으로 표현할 수도 있지만, 6가지 정보요소로 표현하면 보다 정확한 소통이 가능함
- ✓ 여섯 가지 항목이 항상 모두 필요한 것은 아니며, 때로는 당연하거나 중복이 될 수도 있는 정보로 채울 수 있음



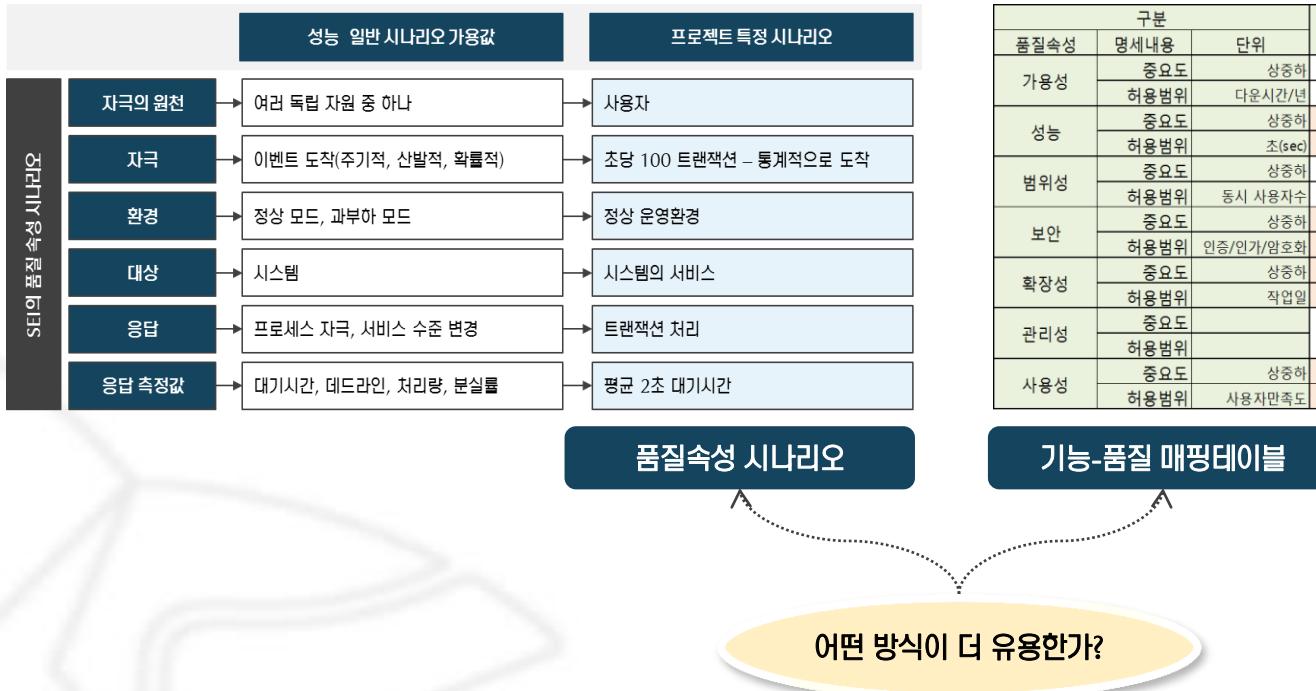
# 품질속성 시나리오 (5/5) – 예제 3

- ✓ 성능 품질속성에 대한 일반 시나리오 마찬가지 선택 가능한 값들을 포함하고 있음
- ✓ “사용자가 정상 운영 시에, 확률적으로 분당 1,000 트랜잭션을 초기화했다. 이 트랜잭션은 평균 대기 시간 2초를 가지고 처리된다.” 라는 문장을 여섯 가지 정보요소로 분리하여 표현을 했음



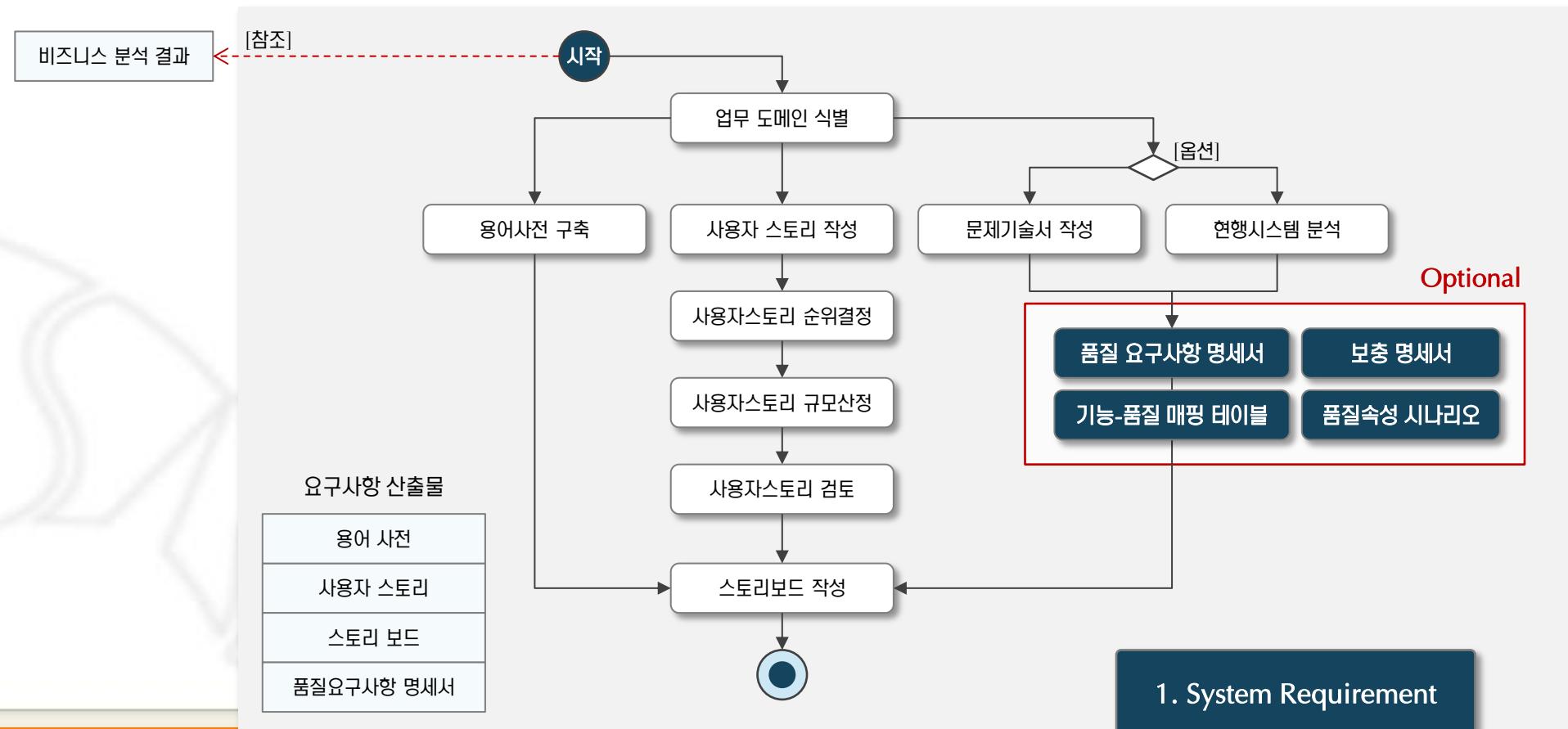
# 품질속성 시나리오 효율성

- ✓ 품질속성 시나리오를 이용한 품질 목표를 표현하는 방법은 상대적으로 많은 작업을 요구함
- ✓ 품질 요구수준이 매우 높으며, 프로젝트 기간과 인력이 충분한 경우에 품질 속성 시나리오를 고려해볼 수 있음
- ✓ 대부분의 경우, 기능-품질 매핑 테이블을 이용하여 주요 기능별 품질속성 요구수준을 표현함



# 품질속성과 프로세스

- ✓ 품질속성을 식별하는 활동은 시스템 요구사항 식별(trawling) 단계에서 수행함
- ✓ 표현하는 방법은 품질요구사항 명세서, 보충 명세서, 기능-품질 매핑 테이블 등의 형태로 다양함
- ✓ 어떤 형태로 표현이 되든 시스템의 비-기능(품질) 기대 수준을 정확히 표현하여야 하고 공유하여야 함



# 요약

- ✓ 시스템 구축 시 요구사항에는 기능 요구사항과 비기능 (품질) 요구사항 두 가지가 있음
- ✓ 품질 요구사항은 아키텍처 설계의 직접 input이 되는 요구사항임, 즉, 아키텍처 설계를 통해서 품질목표를 달성해야 함
- ✓ 아키텍처 설계 시, 품질 요구사항 외에도 기능요구나 환경에 대한 이해가 선행되어야 함

- ✓ 시스템 요구사항을 기능과 품질로 명시적으로 나누어 정리할 준비가 되었습니까?
- ✓ 조직의 표준 품질속성은 정의되어 있습니까? 또는 참조 정의가 있습니까?
- ✓ 아키텍트 역할 담당자는 지정되었고, 요구사항 식별 활동에 참여하고 있습니까?
- ✓ 아키텍팅 프로세스는 준비되어 있습니까?

## 5. 시스템 모델링

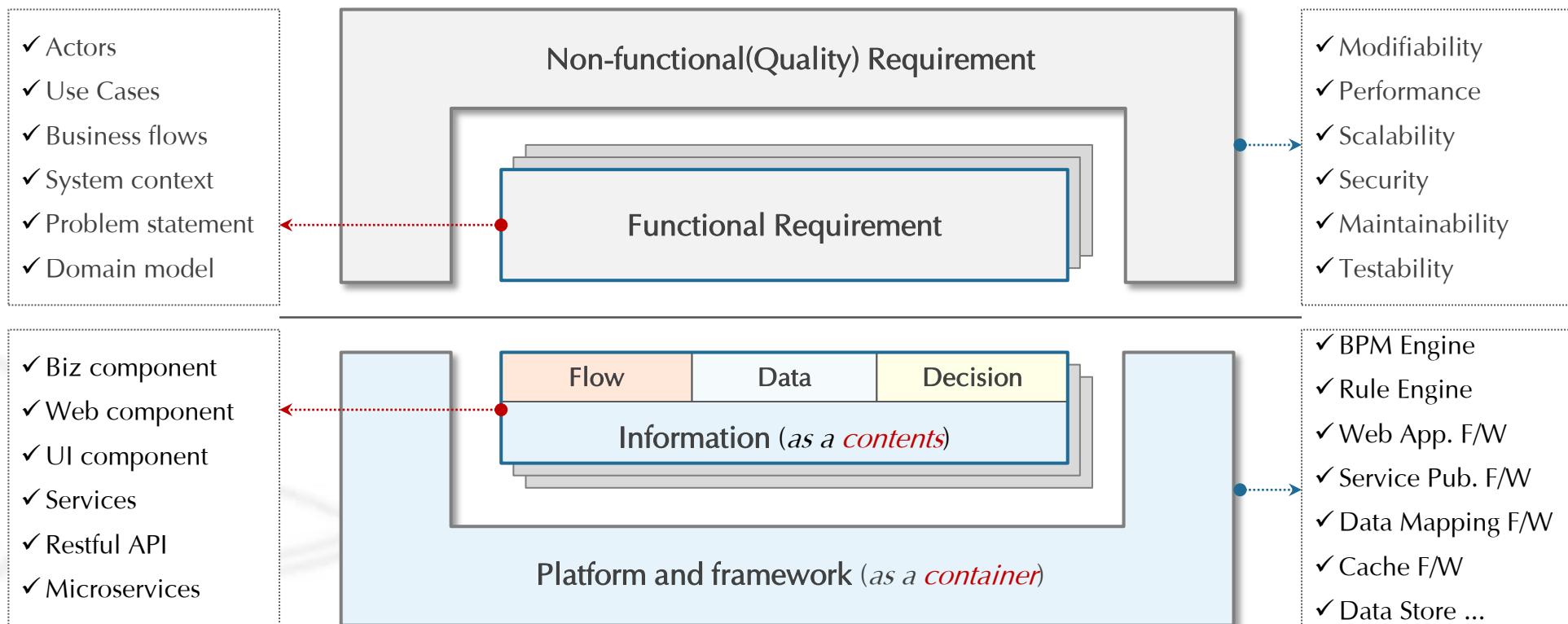
---

- ❖ 모델링을 해야 하는 이유를 생각해 봅니다.
- ❖ 모델링이 Thinking Process 인 이유를 이해합니다.
- ❖ 모델링을 해야 한다면, 올바른 절차는 무엇인지 알아 봅니다.

- ✓ 시스템 이해
- ✓ 모델링 이해
- ✓ 도메인 모델링
- ✓ 모델링 핵심 요소
- ✓ 모델링과 의사 결정

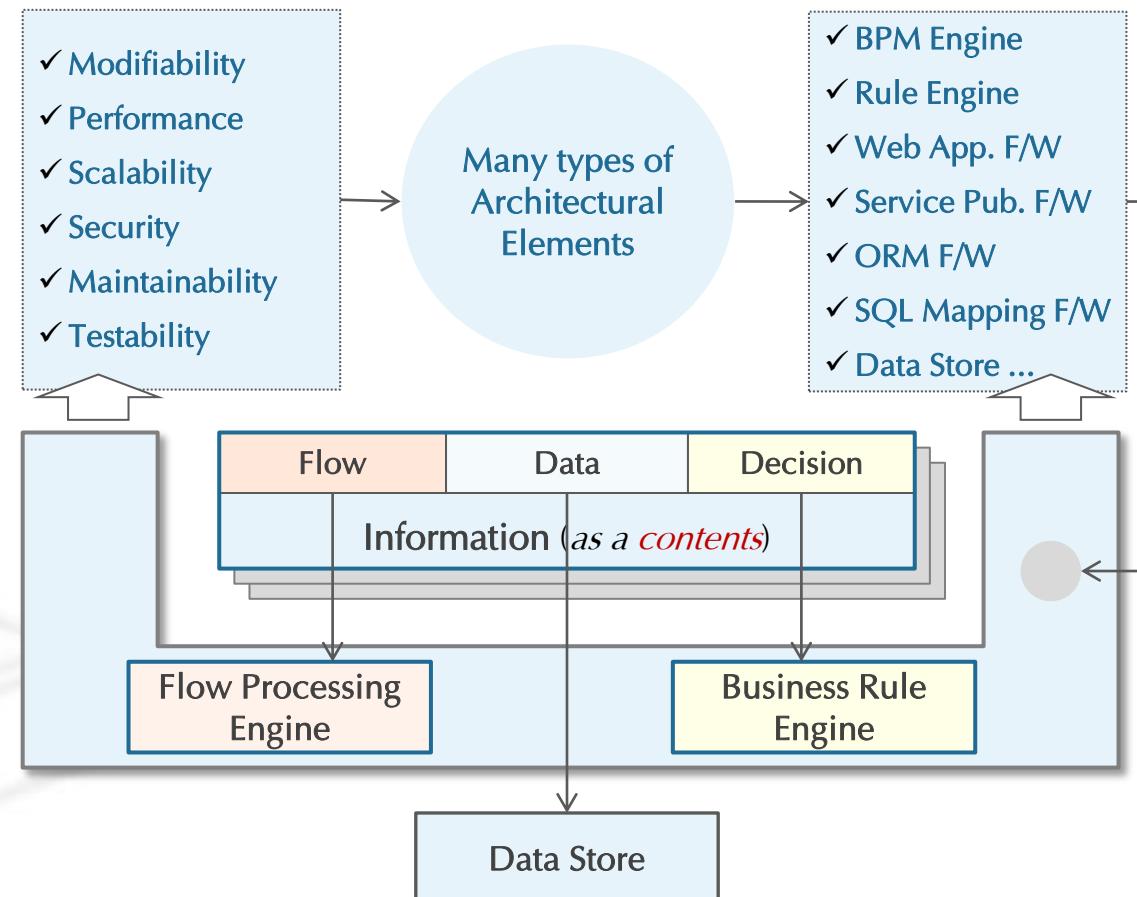
# 시스템 이해(1/3)

- ✓ 시스템의 요구사항은 기능 요구와 비기능 요구로 나누며, 두 가지를 반영하여 시스템을 개발합니다.
- ✓ 시스템은 비기능 요구를 반영한 결과를 컨테이너라고 표현할 수 있습니다. 그 속에는 많은 요소들이 있습니다.
- ✓ 컨텐츠는 기능 요구를 반영하며, 그 속에는 데이터, 흐름, 판단 등의 형태로 존재하는 정보들이 있습니다.



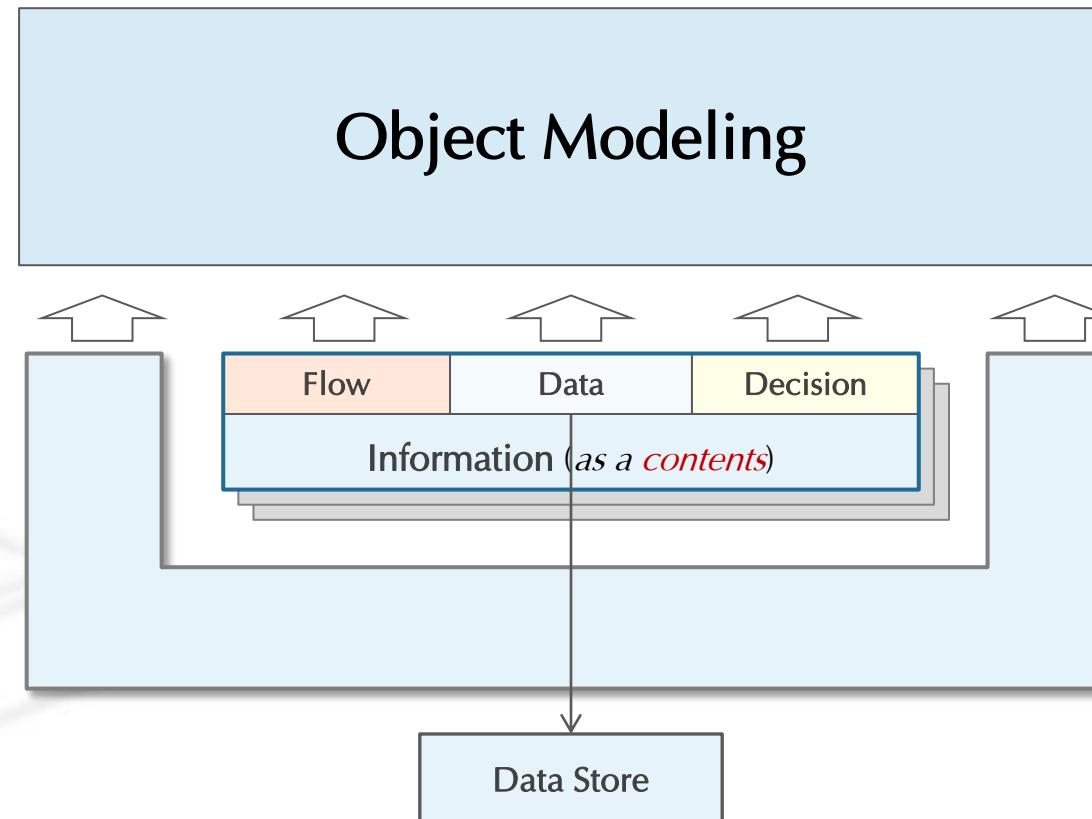
## 시스템 이해(2/3)

- ✓ 컨테이너를 구성하는 많은 아키텍처 요소들이 있습니다. → 프레임워크, 플랫폼, 라이브러리 등
- ✓ 컨테이너 구성 요소를 개발할 때도 모델링이 필요합니다. 이것은 아키텍처 모델링 영역입니다.
- ✓ 컨텐츠를 구성하는 컴포넌트, 서비스 등을 개발할 때 수행하는 모델링을 도메인 모델링이라고 합니다.



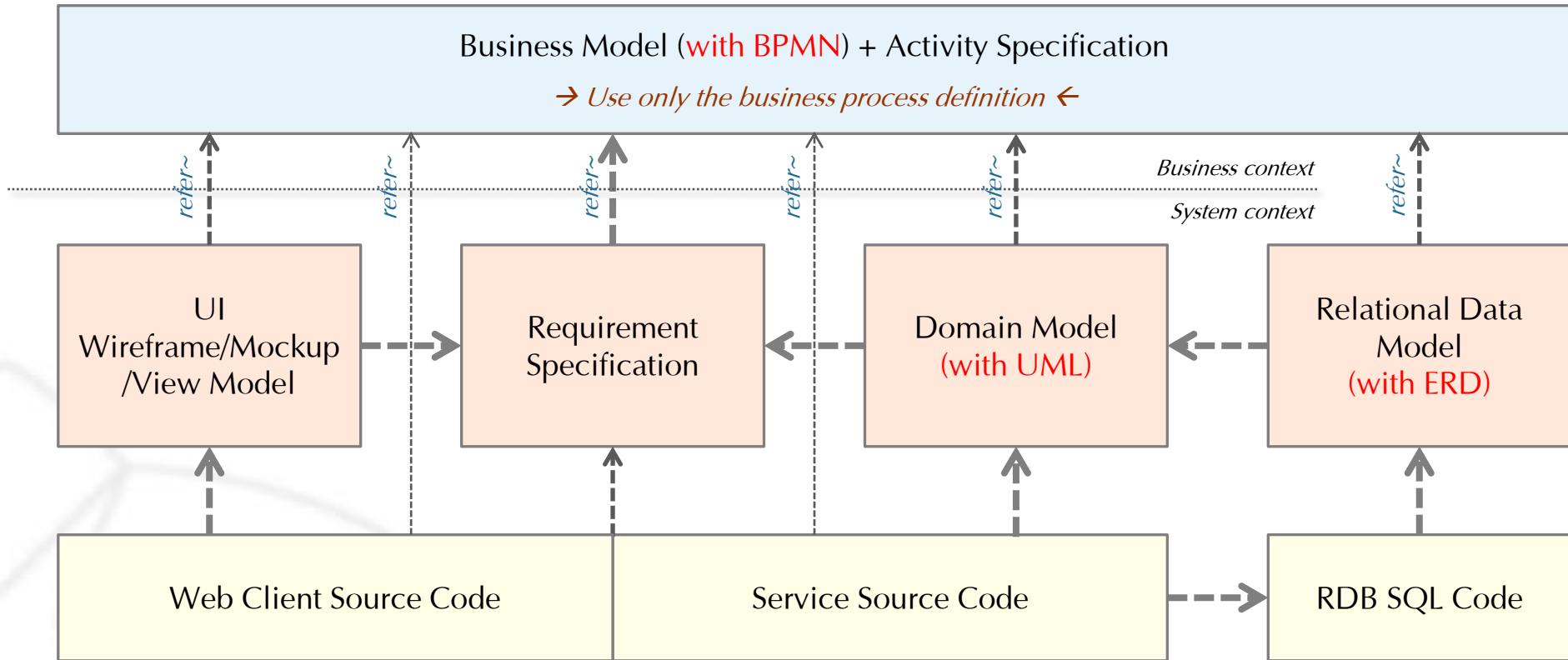
## 시스템 이해(3/3)

- ✓ 컨테이너를 구성하는 요소를 설계하는 컨텐츠를 구성하는 요소를 설계하는 객체 모델링이 기반이 됩니다.
- ✓ 비즈니스 컴포넌트를 설계해야 한다면, 당연히 객체 모델링 과정을 거쳐야 합니다.
- ✓ 데이터 접근 프레임워크나 연계 브로커를 설계해야 한다면, 이 활동 또한 객체 모델링 과정을 거쳐야 합니다.



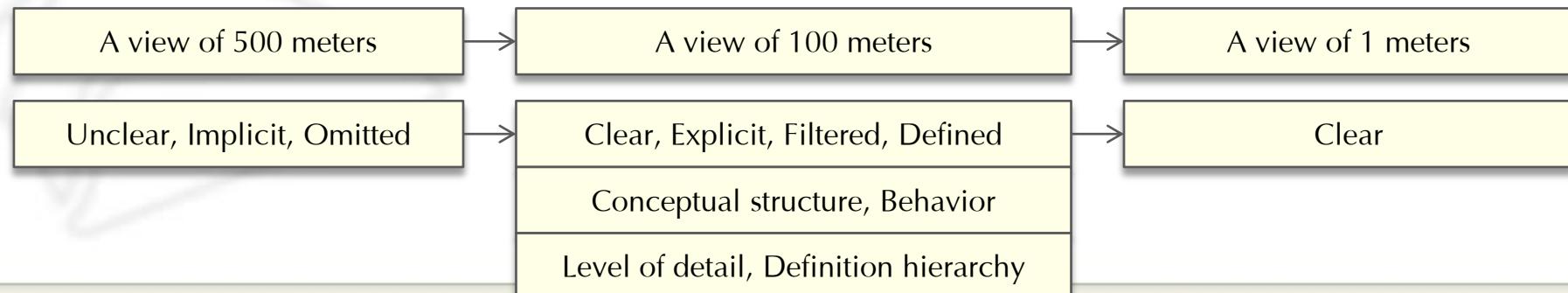
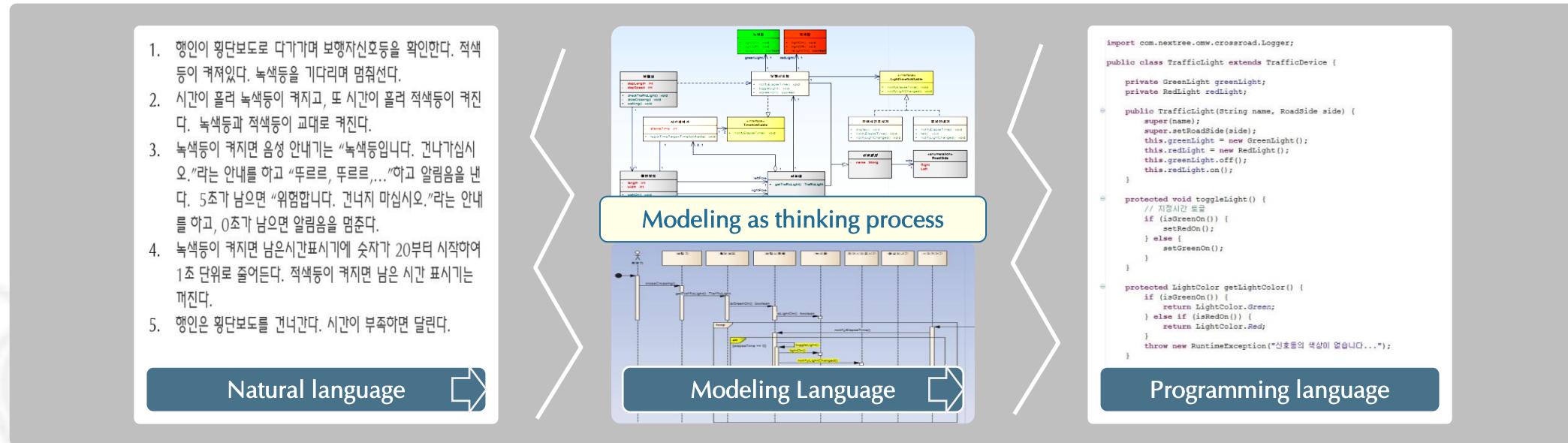
# 모델링 이해(1/4): 개요

- ✓ 표준으로 지정된 두 가지 모델링 언어가 있습니다. → BPMN, UML
- ✓ BPMN은 비즈니스 프로세스를 모델링을 목적으로 사용하는 모델링 언어입니다.
- ✓ UML은 시스템 모델링용 언어입니다. 컴포넌트를 설계하거나 프레임워크를 설계할 때 사용합니다.



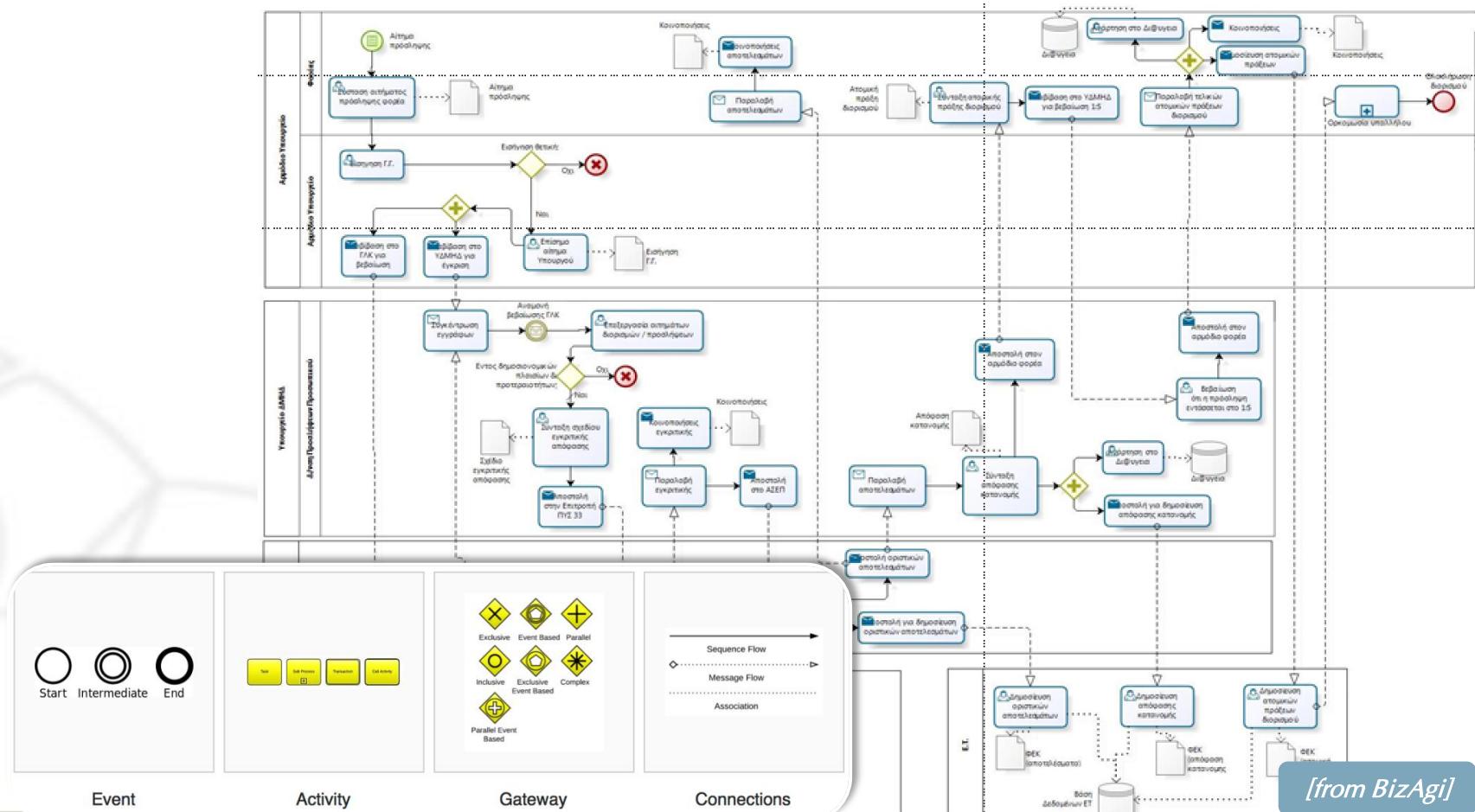
# 모델링 이해(2/4): Language transition

- ✓ 자연어, UML, Java는 서로 다른 목적을 가진 언어이지만 순서대로 참조를 합니다.
- ✓ 자연어로부터 모델링 언어를 자동 생성하거나, 모델링 언어로부터 프로그래밍 언어를 생성하려는 시도가 있었습니다. 일부 가능성도 확인하였지만, “고유의 목적”을 가진 언어 간의 경계를 없애려는 시도는 무의미합니다.



# 모델링 이해(3/4): Business Process Model 1

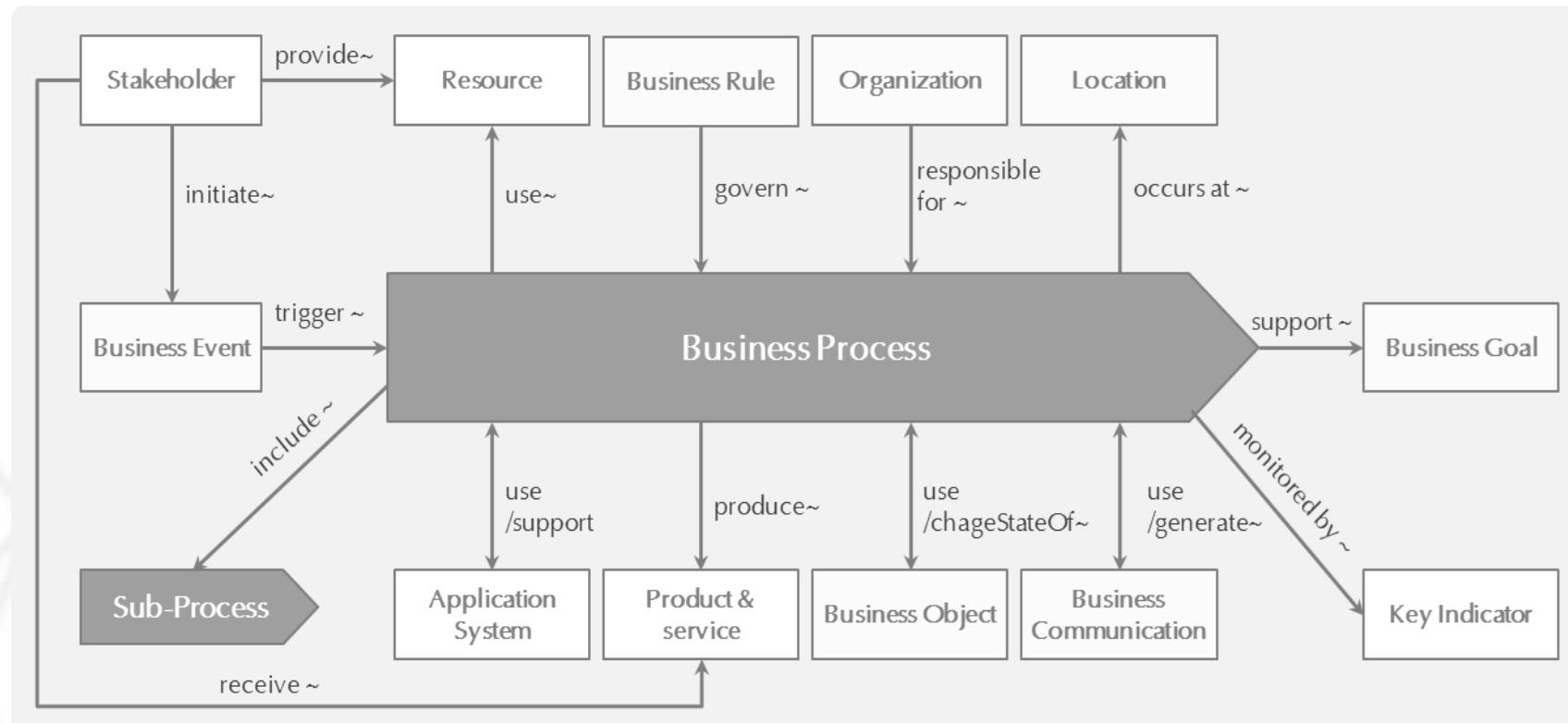
- ✓ BPMN의 일차 목적은 모든 비즈니스 이해관계자가 자연스럽게 이해할 수 있는 표준 표기법을 제공하는 것입니다. [from wikipedia.org]
- ✓ BPMN은 네 가지 유형의 개념이 바탕으로 합니다. Event, Activity, Gateway, and Connection.



[from BizAgil]

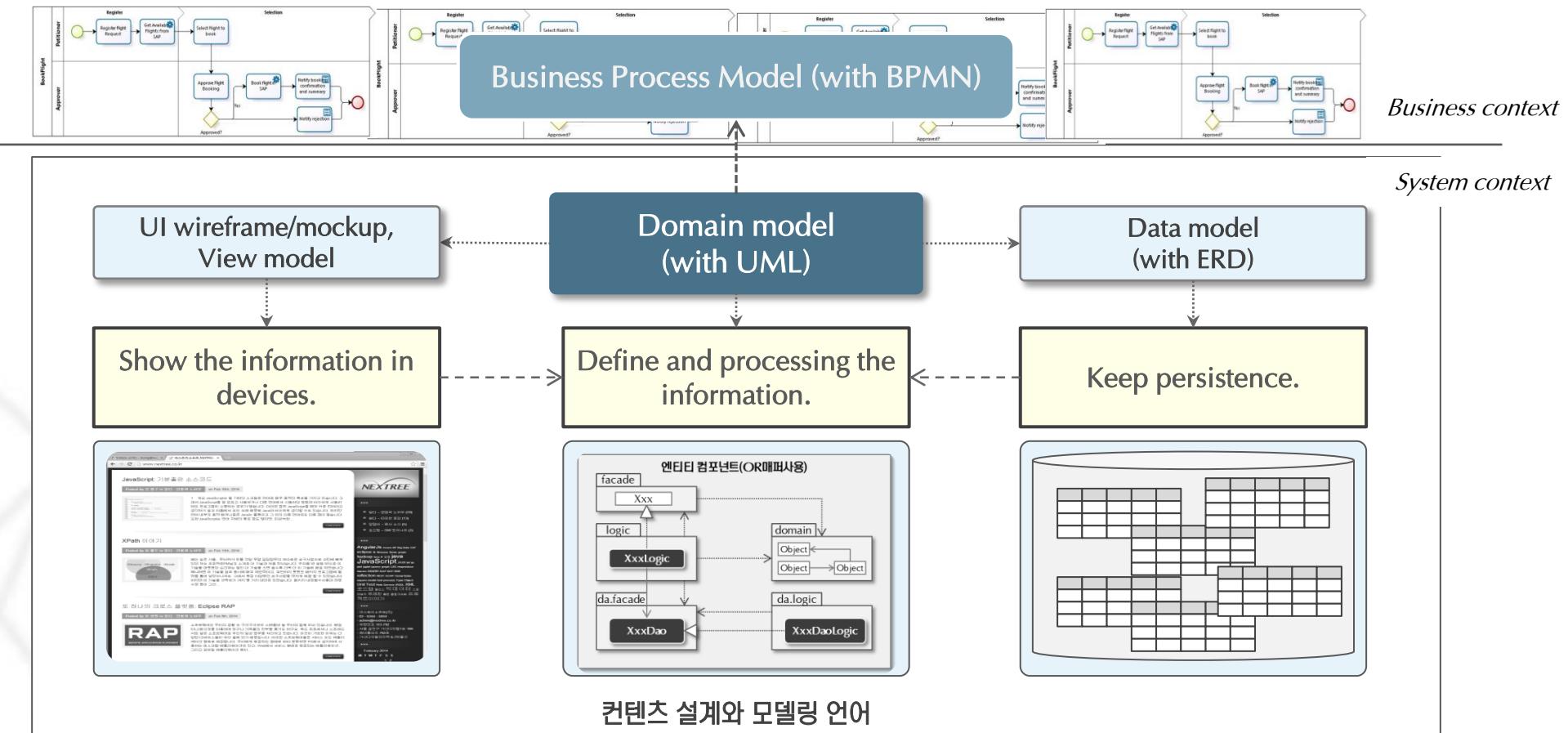
# 모델링 이해(4/4): Business Process Model 2

- ✓ 비즈니스 모델은 프로세스 뿐만 아니라 다양한 모델 요소를 포함하고 있습니다.
- ✓ BPMN은 그 중에서 프로세스 만을 모델링 대상으로 삼고 있습니다.
- ✓ 비즈니스 6대 모델에는 조직, 목표, 위치, 프로세스, 이벤트, 커뮤니케이션 등이 포함됩니다. ← 매우 다양함



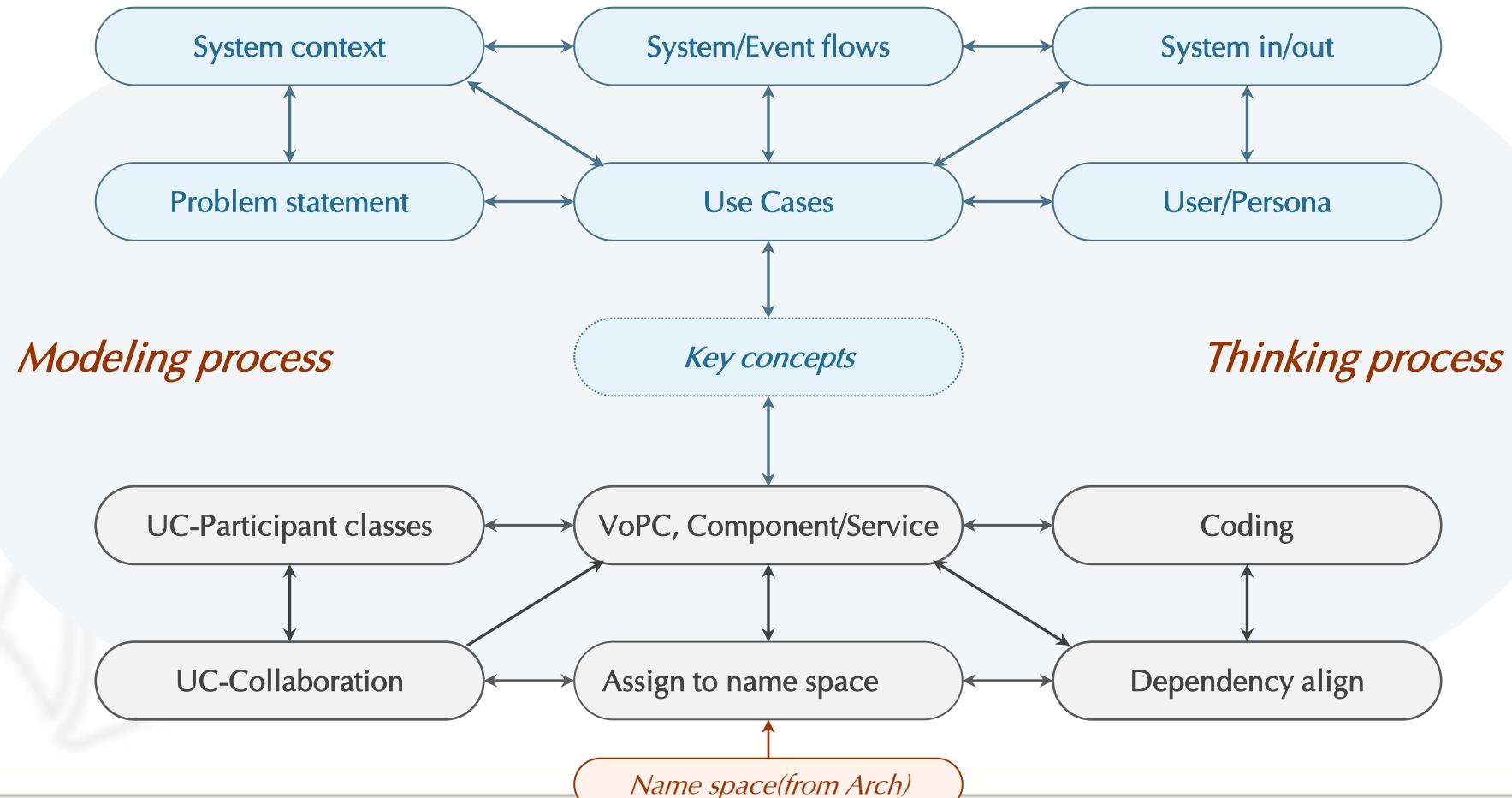
# 도메인 모델링 – 개요

- ✓ 도메인 모델은 모든 모델의 한 가운데 있으며, 업무를 표현하며, UML을 사용합니다.
- ✓ 데이터 모델은 도메인 모델을 참조하며, 도메인 모델 범위 안에서 설계를 하며, 저장 장치의 특성을 반영합니다.
- ✓ 뷰 모델은 클라이언트 디바이스의 영향을 받으며, 도메인 모델을 기반으로 합니다.



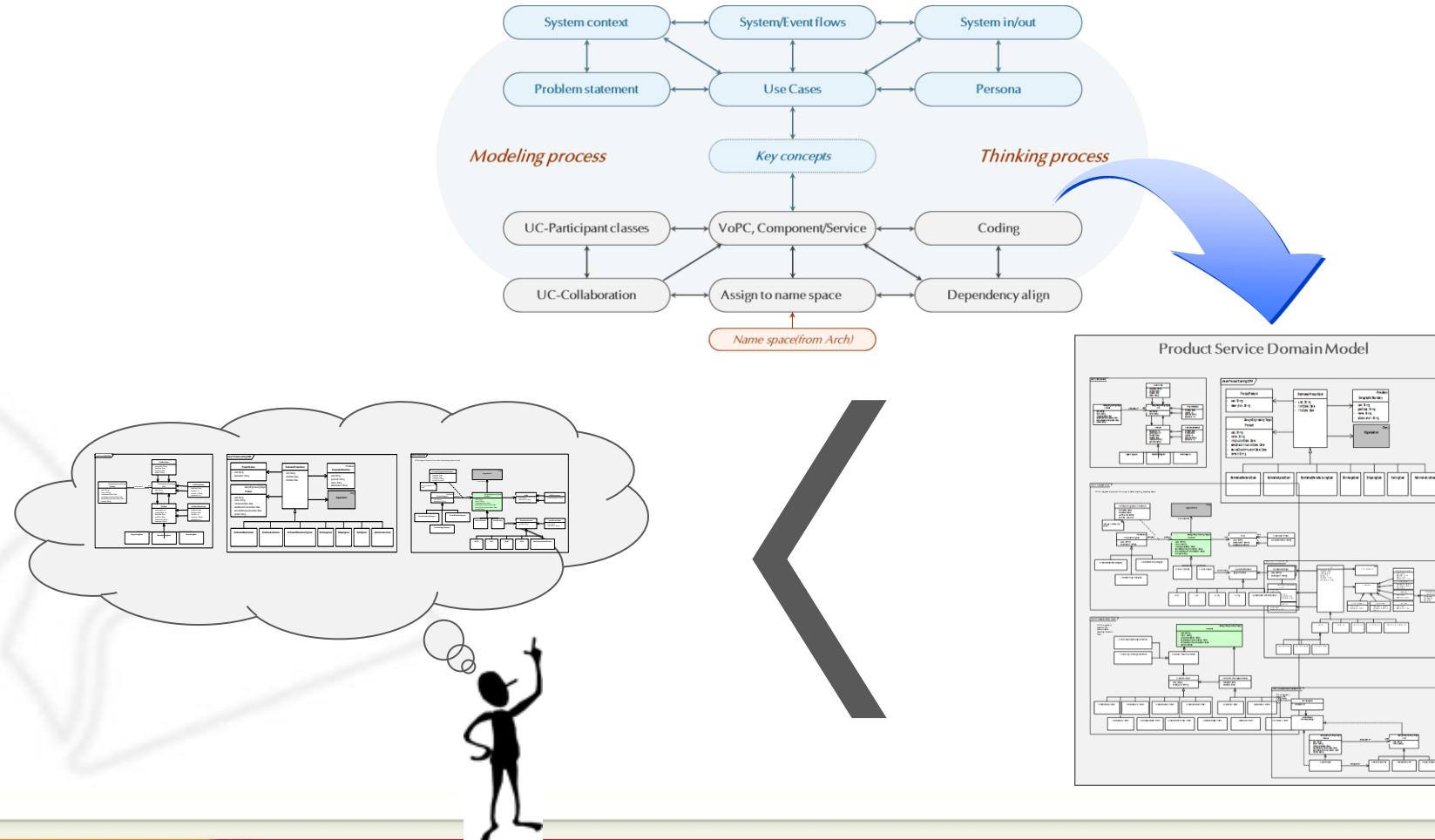
# 도메인 모델링 – 생각하는 과정

- ✓ 도메인 모델링은 도메인을 이해하고 모델러의 뇌 속에 정리하여 넣은 과정입니다.
- ✓ 모델링 절차는 도메인에 있는 대상을 다양한 관점과 다양한 상세도 수준에서 바라볼 수 있도록 도와 줍니다.
- ✓ 모델링은 단순히 클래스 다이어그램과 시퀀스 다이어그램을 그리는 활동이 아님을 알아야 합니다.



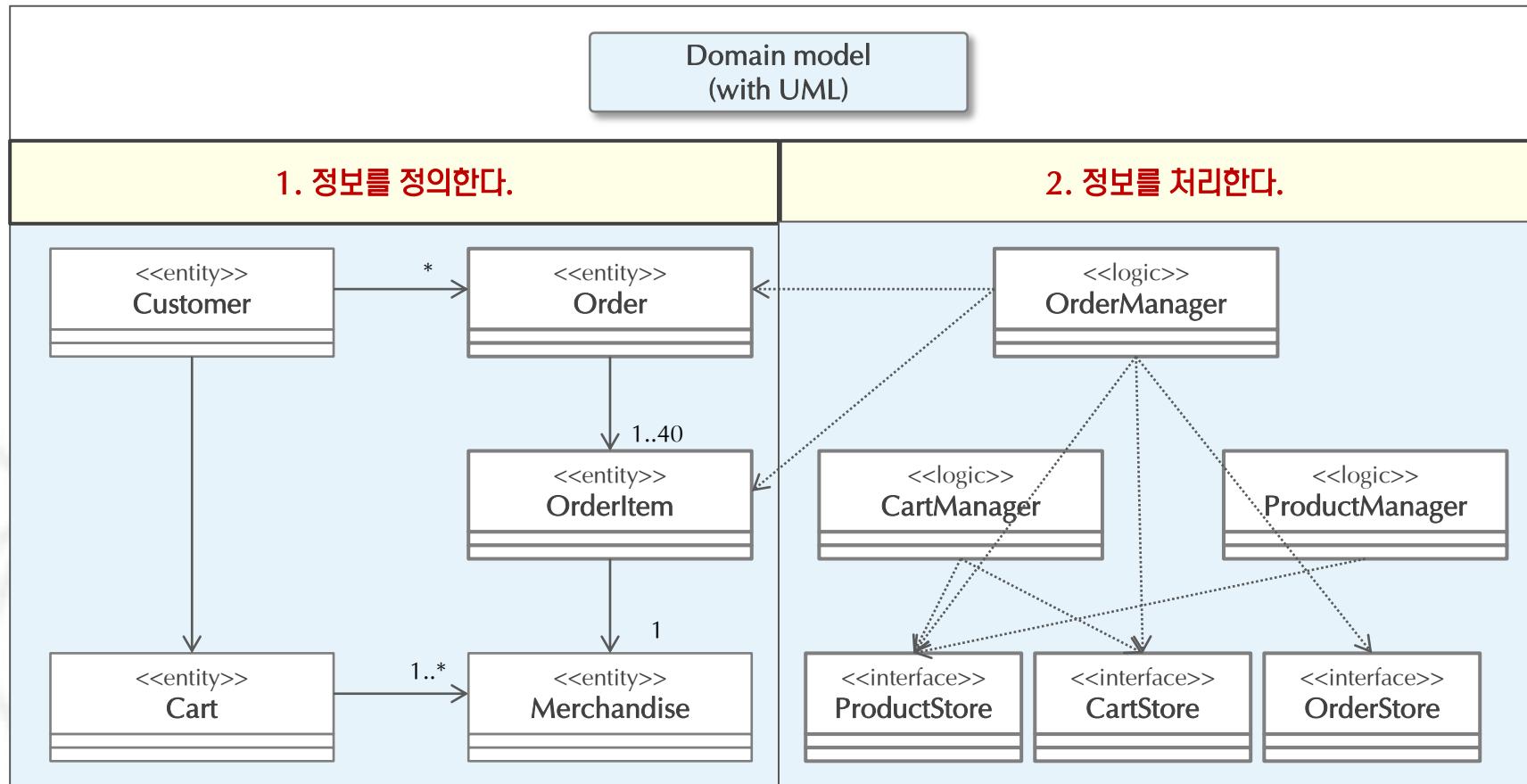
# 도메인 모델링 – 생각하는 과정

- ✓ 다양한 input 정보를 참조하여 생각하는 과정을 거치면 모델을 볼 수 있습니다.
- ✓ 하지만 이 모델 보다 더 중요한 것은 모델러의 머리 속에 질서정연하게 자리 잡은 도메인 개념입니다.
- ✓ 이제 모델러는 코딩을 할 수 있으며, 고객과의 대화를 통해 이해의 폭과 깊이를 더해 갈 수 있습니다.



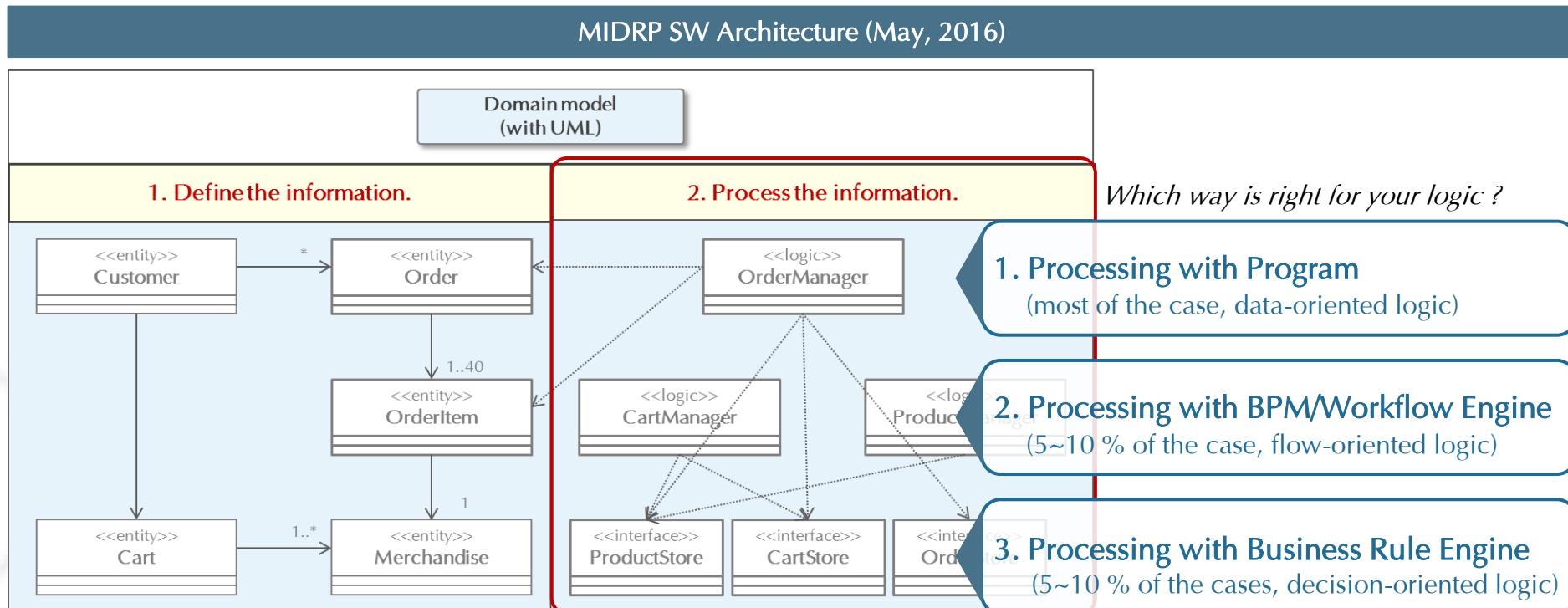
# 모델링 핵심 요소: 정의와 처리

- ✓ 비즈니스 컴포넌트를 구성하는 핵심 객체는 엔티티 객체와 로직 객체입니다.
- ✓ 엔티티 객체는 개념과 개념 간의 관계를 담고 있고, 로직 객체를 처리 절차를 담고 있습니다.
- ✓ 도메인 모델러는 도메인을 정확하게 이해하고, 그 내용을 모델로 표현하여야 합니다.



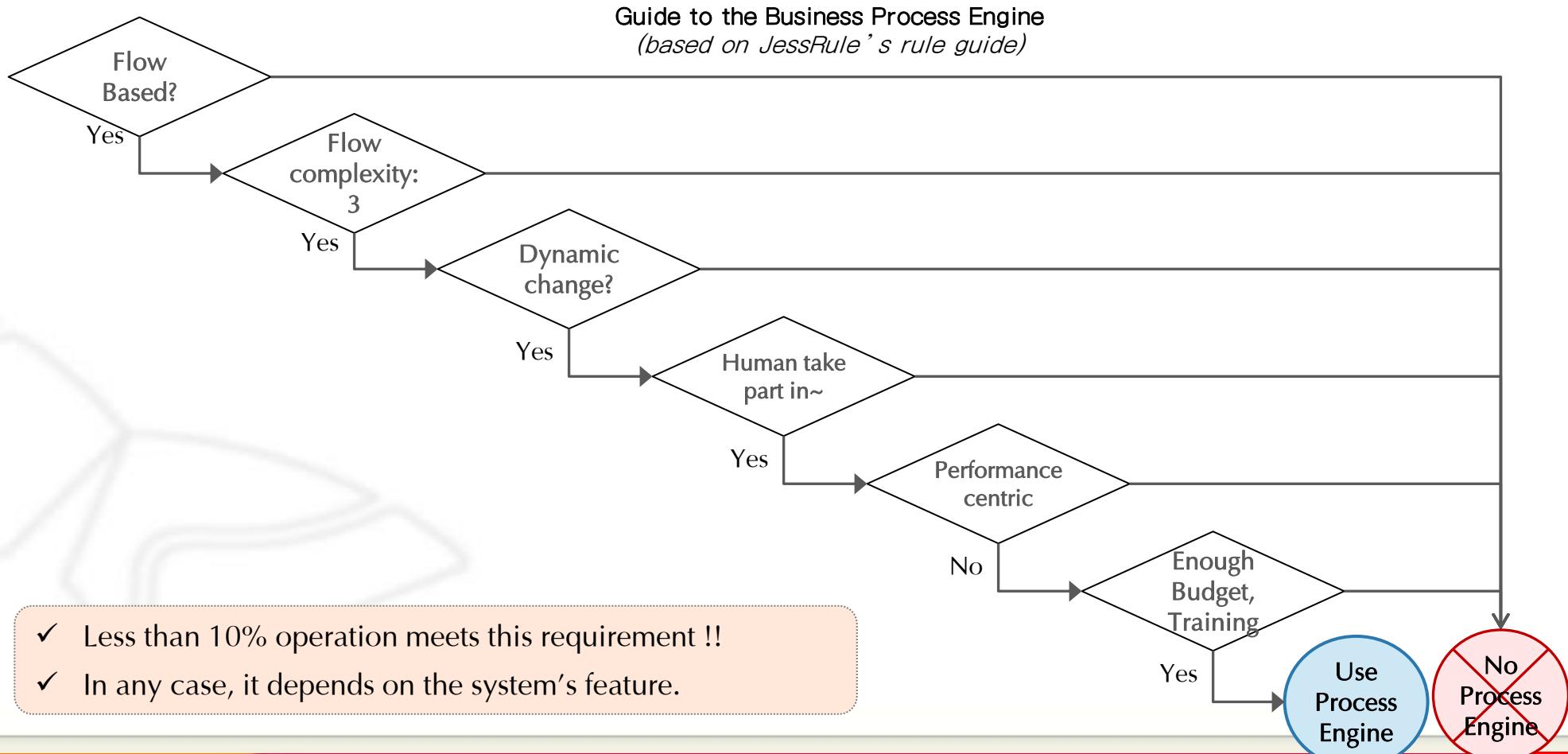
# 모델링 핵심 요소: 정의와 처리

- ✓ 로직 객체는 세 가지 처리 로직으로 분류할 수 있습니다. → 데이터 처리, 흐름 처리, 판단 처리
- ✓ 어떤 경우에는 흐름 처리나 판단 처리를 전문으로 하는 모듈이 필요할 수도 있습니다.
- ✓ 하지만, 어떤 경우든 프로그램으로 처리할 수 있습니다.



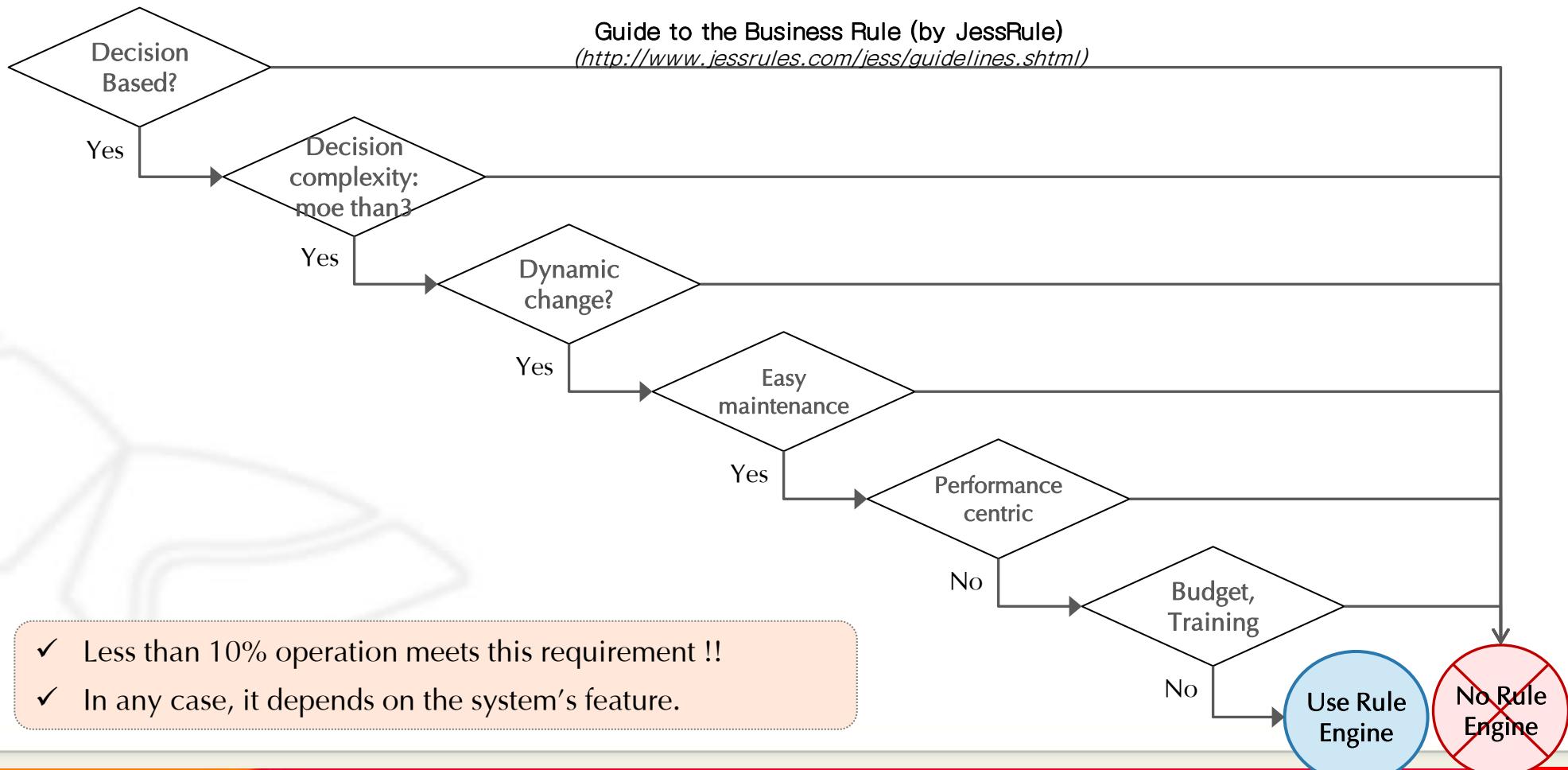
# 모델링 – 의사 결정: 프로세스 엔진

- ✓ 때로는 흐름이 복잡하거나 자주 변경하는 경우, 흐름 처리를 전문으로 하는 엔진 모듈을 사용할 수 있습니다.
- ✓ 프로그램으로 처리할 것인가 프로세스 엔진을 사용할 것인가에 대한 판단이 필요합니다.
- ✓ JessRule에서 제시한 룰 선택 가이드를 참조하여 프로세스 엔진 선택을 위한 의사결정 트리를 만들어 보았습니다.



# 모델링 – 의사 결정: 룰엔진

- ✓ JessRule 에서 제시한 룰엔진 사용여부 판단을 위한 의사결정 트리는 아키텍트들이 자주 참조하는 자료입니다.
- ✓ 여섯 가지 질문에 대답을 하다보면 룰엔진을 사용할 지 프로그램으로 개발할지 여부를 결정을 할 수 있습니다.
- ✓ 시스템의 특성에 따라 결과는 다르겠지만, 보통의 시스템에서 룰 엔진이 필요한 경우는 10% 이내입니다.



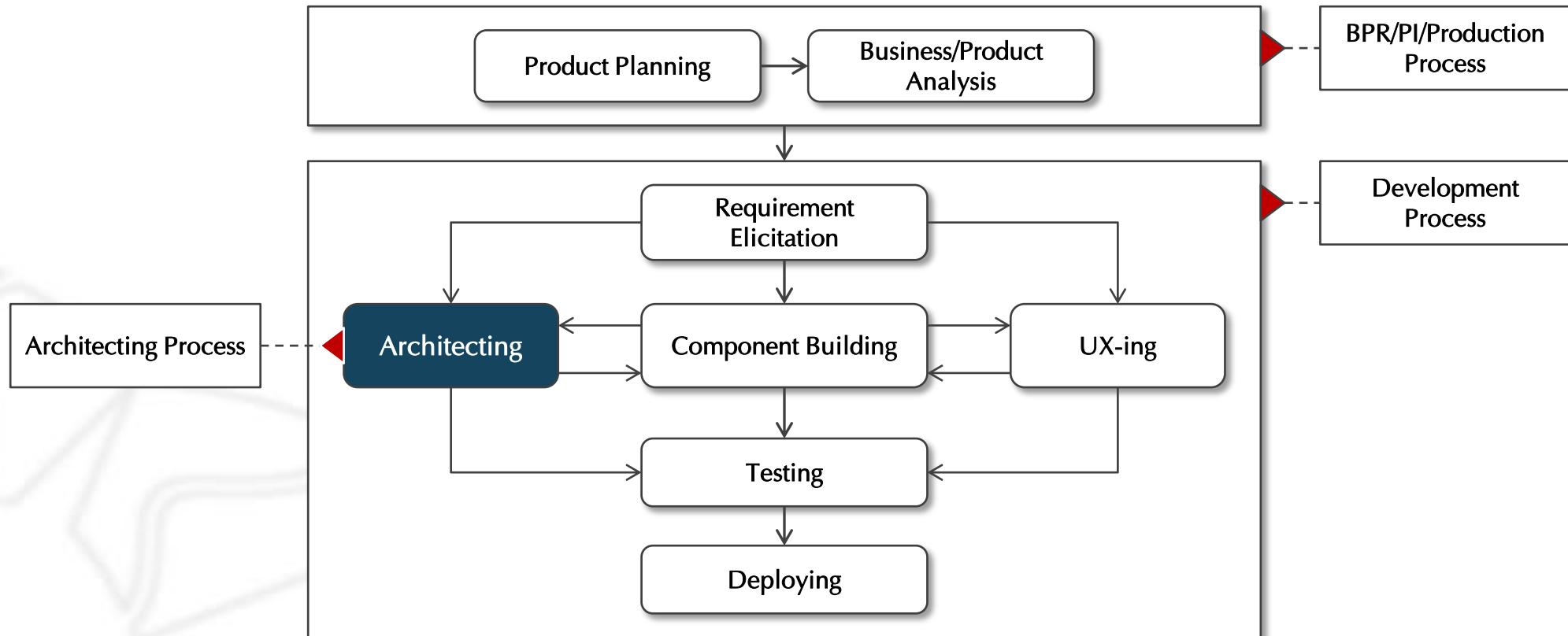
## 6. 아키텍팅 프로세스

---

- ✓ Development process
- ✓ Why Architecting Process ?
- ✓ Architecting Process History
- ✓ Reference Architecting Process - ADD
- ✓ IEEE 1471 개념 모델 확장
- ✓ New Architecting Process
- ✓ 토의

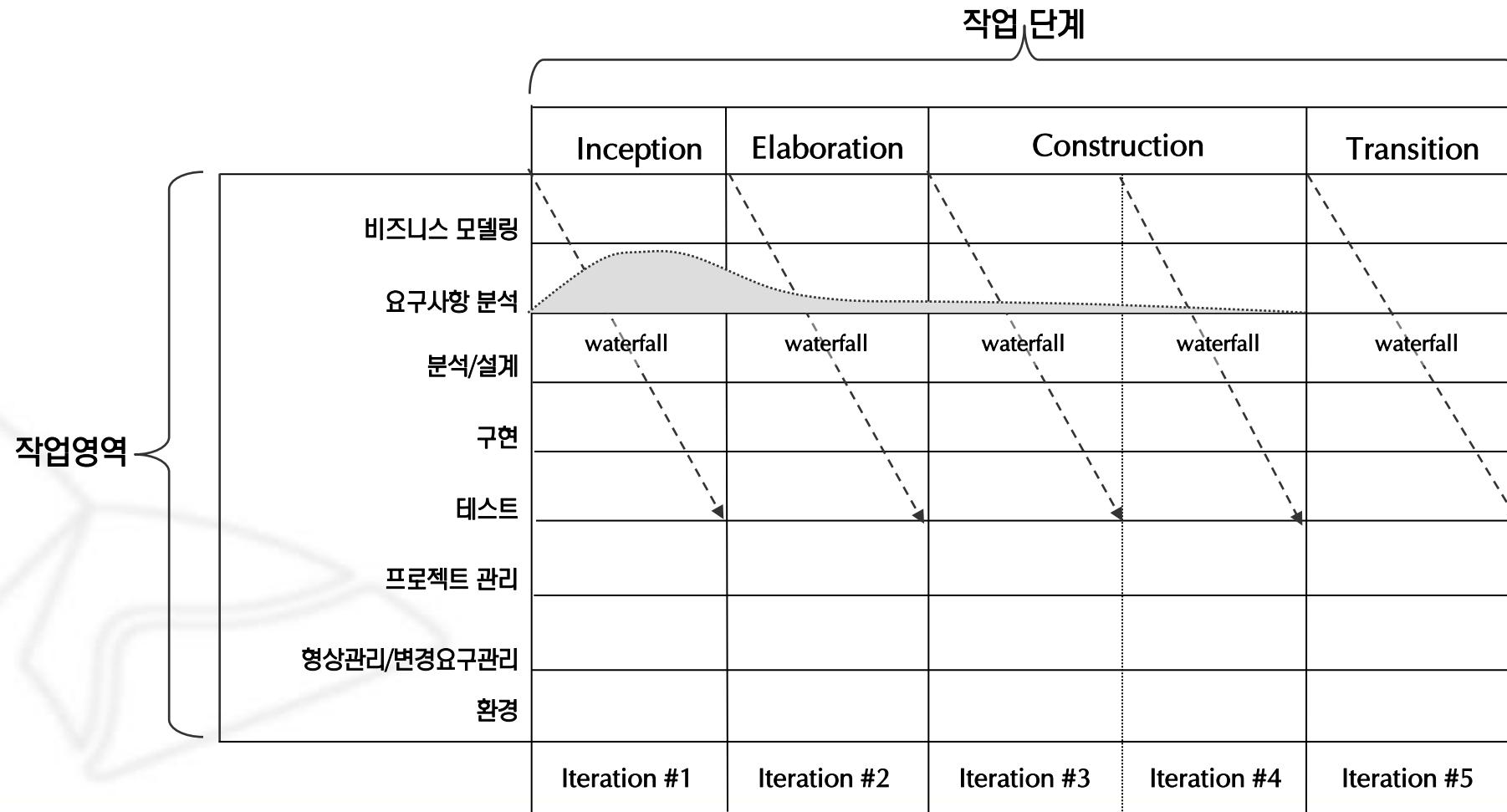
# Development Process(1/4) – Nextree 스타일

- ✓ SW를 기반으로 하는 제품이나 업무용 Application을 개발할 때, 다양한 프로세스가 필요합니다.
- ✓ 아키텍팅 활동의 목표 시스템의 틀과 원칙을 설계하는 중요한 활동입니다.
- ✓ 아키텍팅 프로세스는 개발 프로세스의 하위 프로세스이며, 아키텍처 설계, RI 등을 포함합니다.



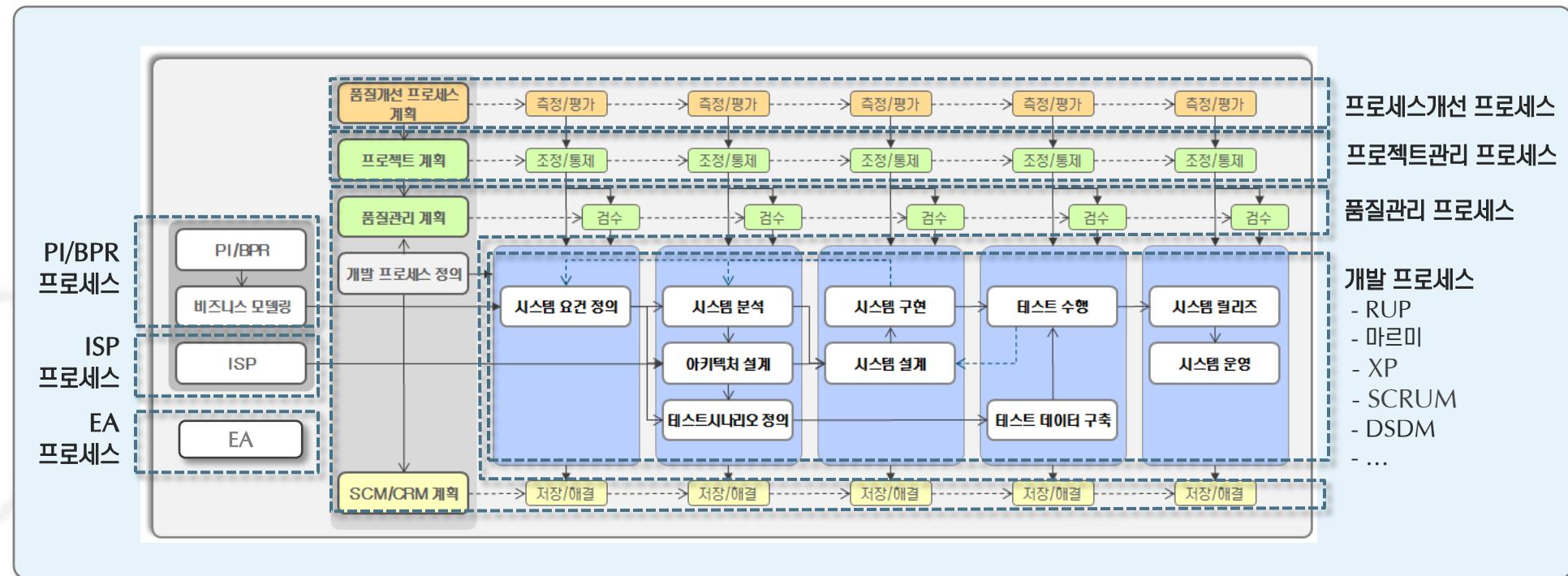
# Development Process(2/4) – RUP 스타일

- ✓ RUP는 Objectory를 모태로 하여 가장 완성된 형태의 개발 프로세스로 발전했습니다.
- ✓ Iterative & Incremental 사상을 기반으로 개발을 진행함으로써, Waterfall을 탈피하기 시작합니다.



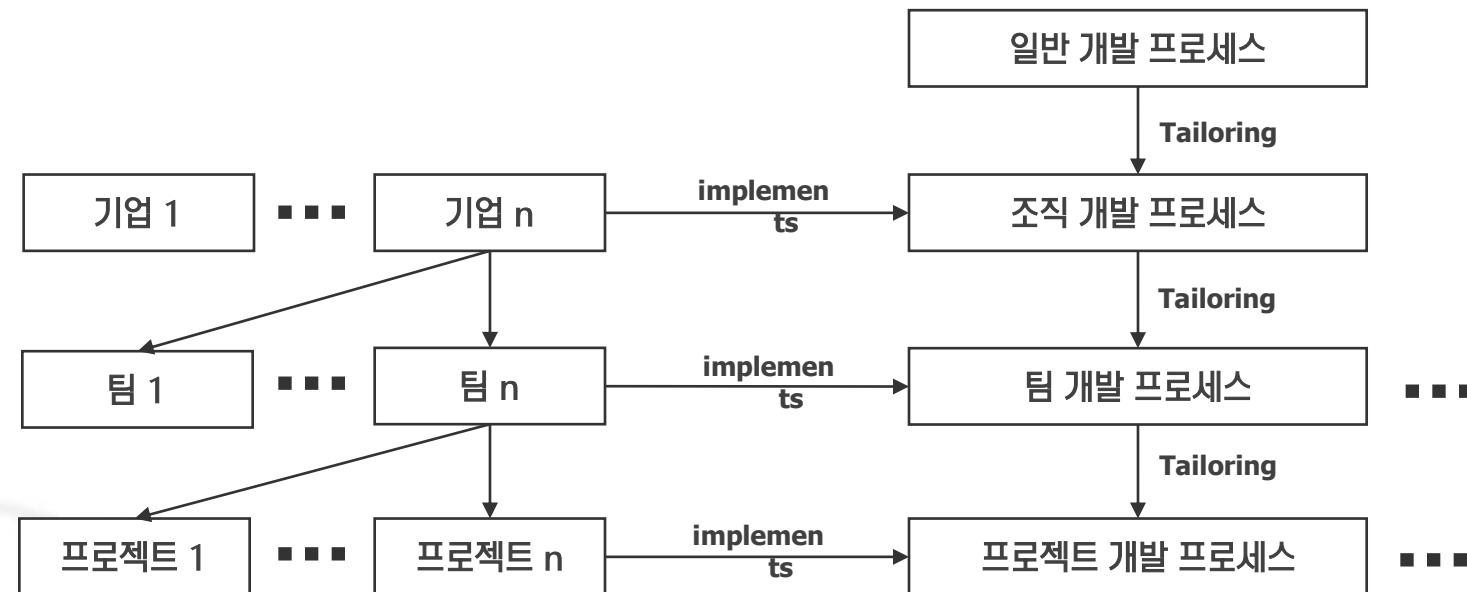
# Development Process(3/4) – 대규모 조직

- ✓ 대규모 조직은 다양한 종류의 프로세스를 가지고 있으며, 개발 프로세스는 그 중에 하나입니다.
- ✓ 프로젝트 유형에 따라 개발 프로세스를 선택하여 조정한 후에 사용하는 것이 일반적입니다.
- ✓ 프로세스는 역할 그룹(관리, 개발, 도메인, 솔루션, 시스템 엔지니어링 등) 별로 서로 다릅니다.



# Development Process(4/4) – 프로세스 조정

- ✓ 조직 개발 프로세스는 일반 개발 프로세스로부터 조정(tailoring)한 결과입니다.
- ✓ 팀 개발 프로세스는 조직의 표준 개발 프로세스를 팀의 개발작업 특성에 맞추어 놓은 것입니다.
- ✓ 프로젝트 개발 프로세스는 팀의 개발 프로세스를 해당 프로젝트의 특성에 맞추어 놓은 것입니다.



# Why Architecting Process ? (1/2)

- ✓ 프로세스는 태권도의 품새(Form)와 비슷합니다.
- ✓ 품새는 순서, 자세를 강조하지만, 실전에서는 스피드, 파워, 순간 판단력이 중요합니다.
- ✓ 실전은 품새가 조정(Tailoring)된 결과로 볼 수 있습니다.



# Why Architecting Process ? (2/2)

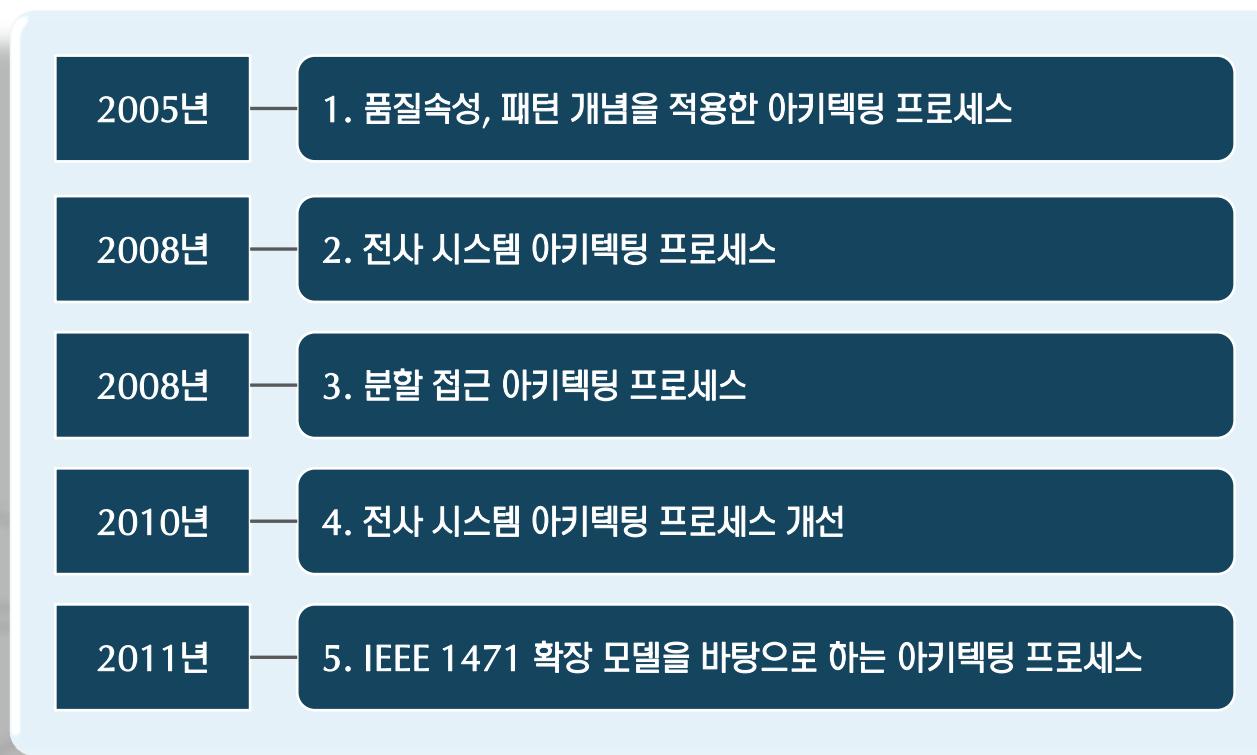
- ✓ 일단 품새가 만들어지면,
- ✓ 경험자의 노하우를 정형화된 형태로 품새에 담을 수 있으며,
- ✓ 수련자는 경험자의 기술을 품새를 통해서 전달받을 수 있습니다.
- ✓ 이런 과정을 통해서 경험과 지식 축적되고 전달되는 선순환 과정을 반복하면서 작업 성숙도가 높아집니다.



▲ 세계태권도연맹(WTF)이 품새 경기에 프리스타일 종목 도입을 추진하고 있다. (자료사진)

# Architecting Process History (1/5)

- ✓ 우리는 아키텍팅 프로세스가 필요하다. 경험을 축적하고 전달하는 수단으로.
- ✓ 2005년 이후 다섯 차례에 걸쳐 아키텍팅 프로세스를 발전시켜 왔습니다.
- ✓ 개선할 때마다, 품질 속성, 전사 시스템, 분할 설계 등의 키워드가 있었습니다.
- ✓ 최근에는 IEEE 1471 – 확장모델을 기반으로 개발 프로세스를 개선하였습니다.

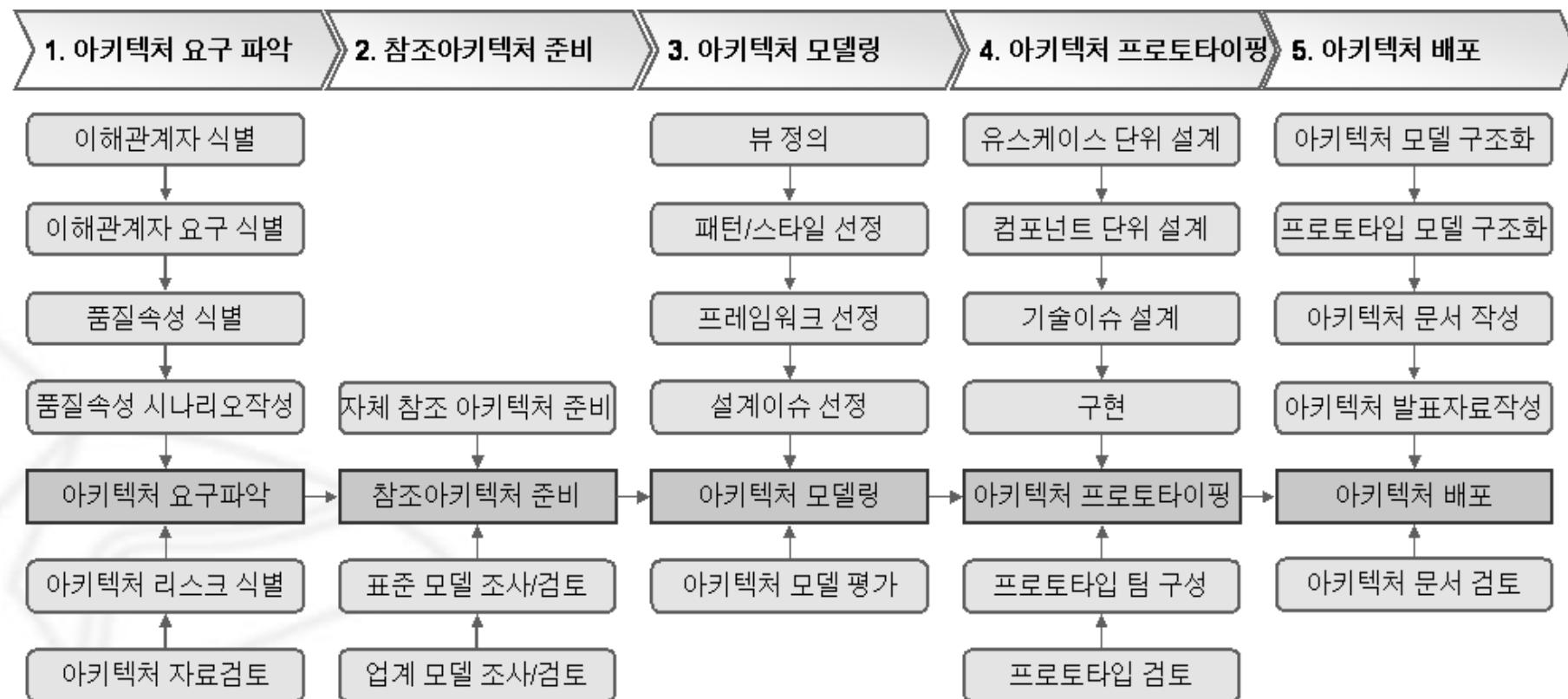


# Architecting Process History (2/5)

- ✓ 2005년 금융회사의 차세대 프로젝트 과정에서 정의했습니다.
- ✓ 품질속성, 패턴, 프로토타입을 특징으로 하는 프로세스입니다.
- ✓ 전사 범위로 기획을 했으나 전사 시스템 아키텍팅 프로세스의 순환적인 특징을 반영하지 못합니다.

2005년

- 1. 품질속성, 패턴 개념을 적용한 아키텍팅 프로세스

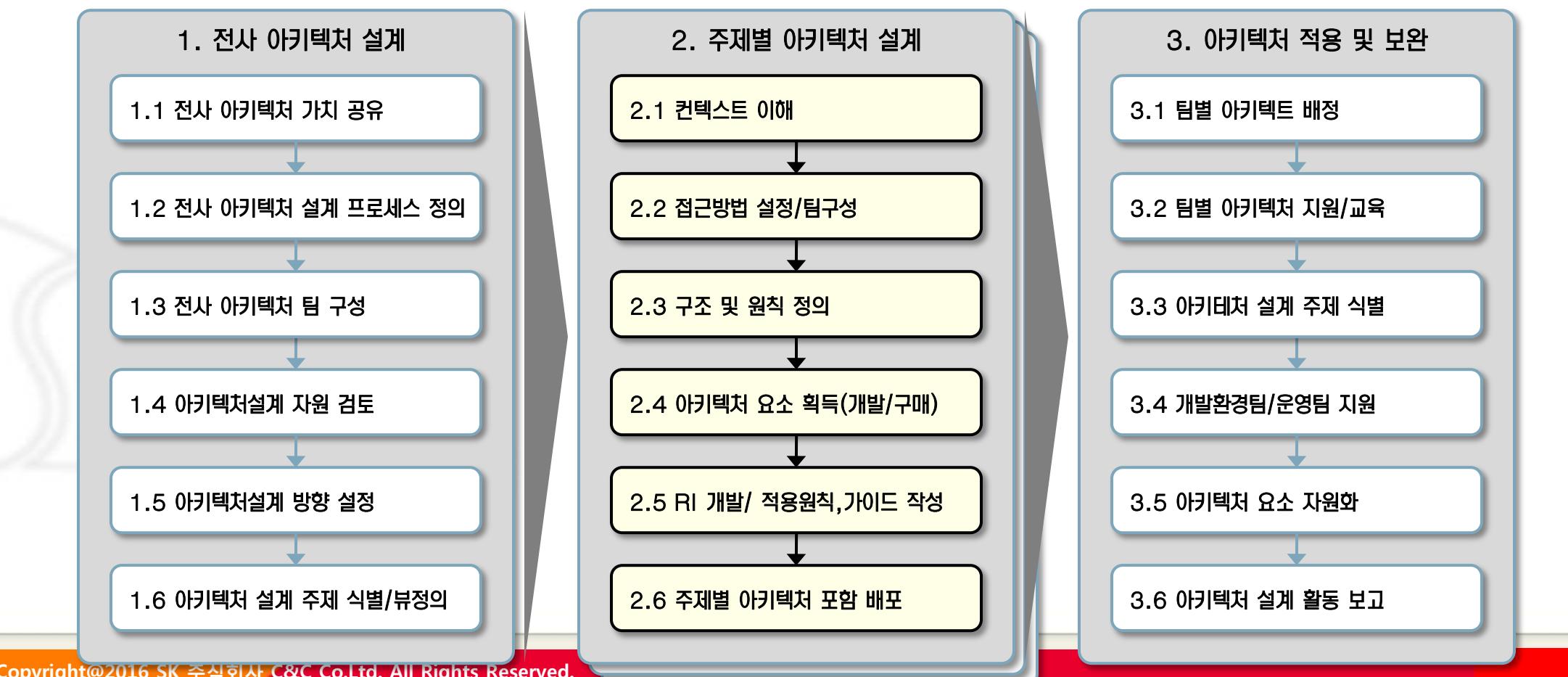


# Architecting Process History (3/5)

- ✓ 전사 시스템 아키텍팅의 순환적인 특성을 반영했습니다.
- ✓ 독립형 시스템 아키텍팅 프로세스는 [아키텍처 요소 획득]으로 한정하였습니다.
- ✓ 아키텍처 적용(enforcement) 개념을 프로세스에 도입하였습니다.

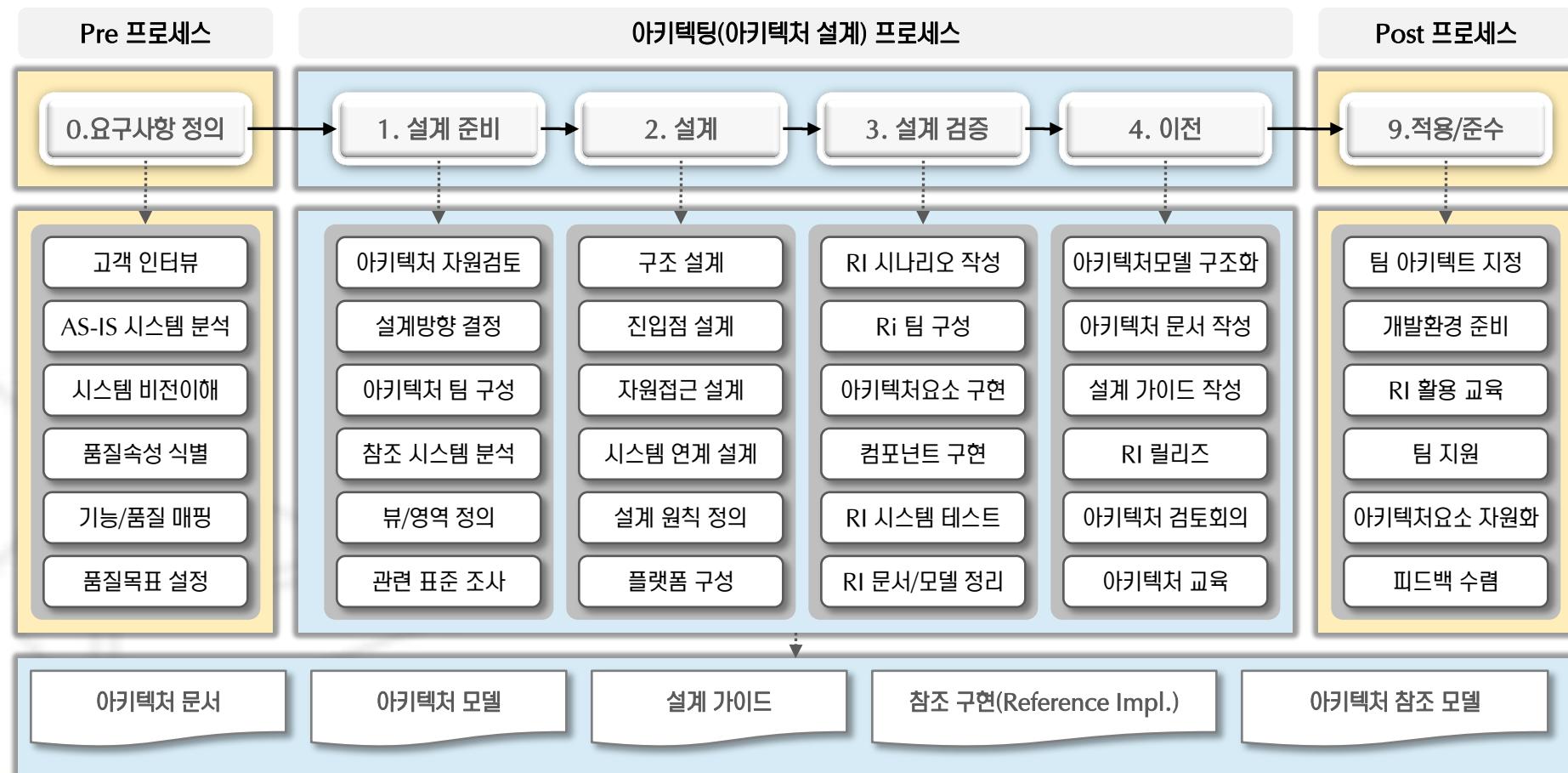
2008년

2. 전사 시스템 아키텍팅 프로세스



# Architecting Process History (4/5)

- ✓ 독립형 애플리케이션 아키텍처 설계 시 분할-정복개념 적용하였습니다. 2008년 - 3. 분할 접근 아키텍팅 프로세스
- ✓ 구조, 진입점, 자원접근, 시스템 연계, 플랫폼 적용, 설계 원칙 등으로 나누어 설계하였습니다.
- ✓ 뷰는 4 + 1 로 고정하였습니다.



# Architecting Process History (5/5)

✓ 기존의 전사 시스템 아키텍팅 프로세스를 개선하였습니다.

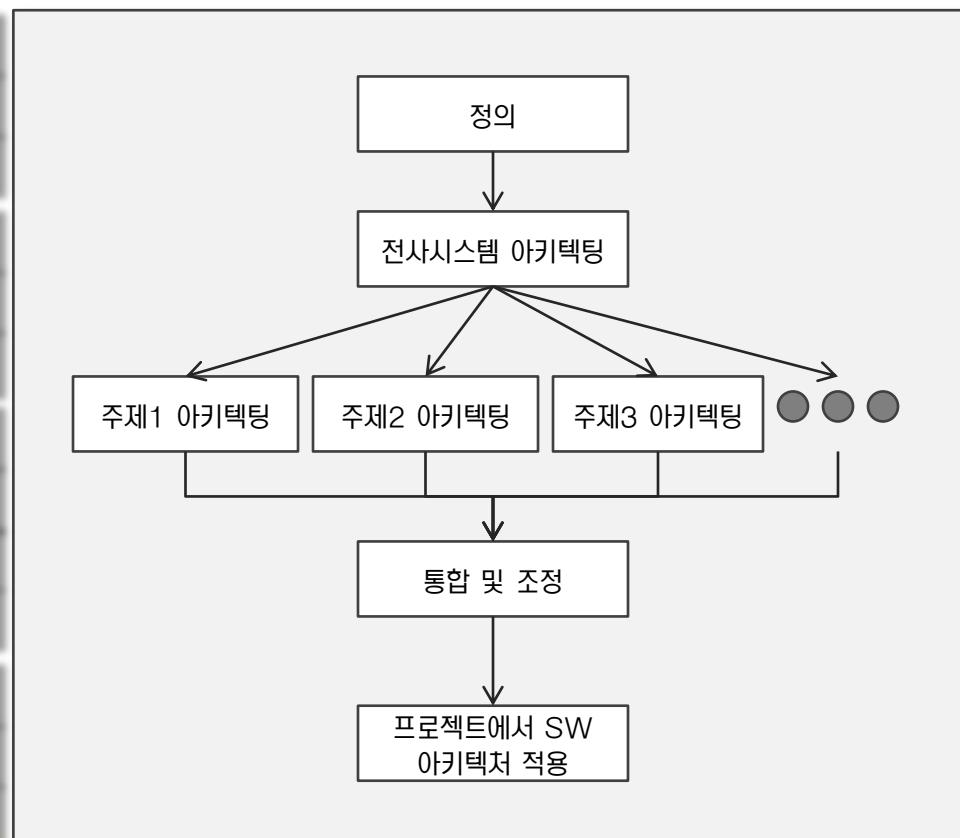
2010년

4. 전사 시스템 아키텍팅 프로세스 개선

✓ 00사 차세대 프로젝트 준비를 위해 기존의 아키텍팅 프로세스를 개선하였습니다.

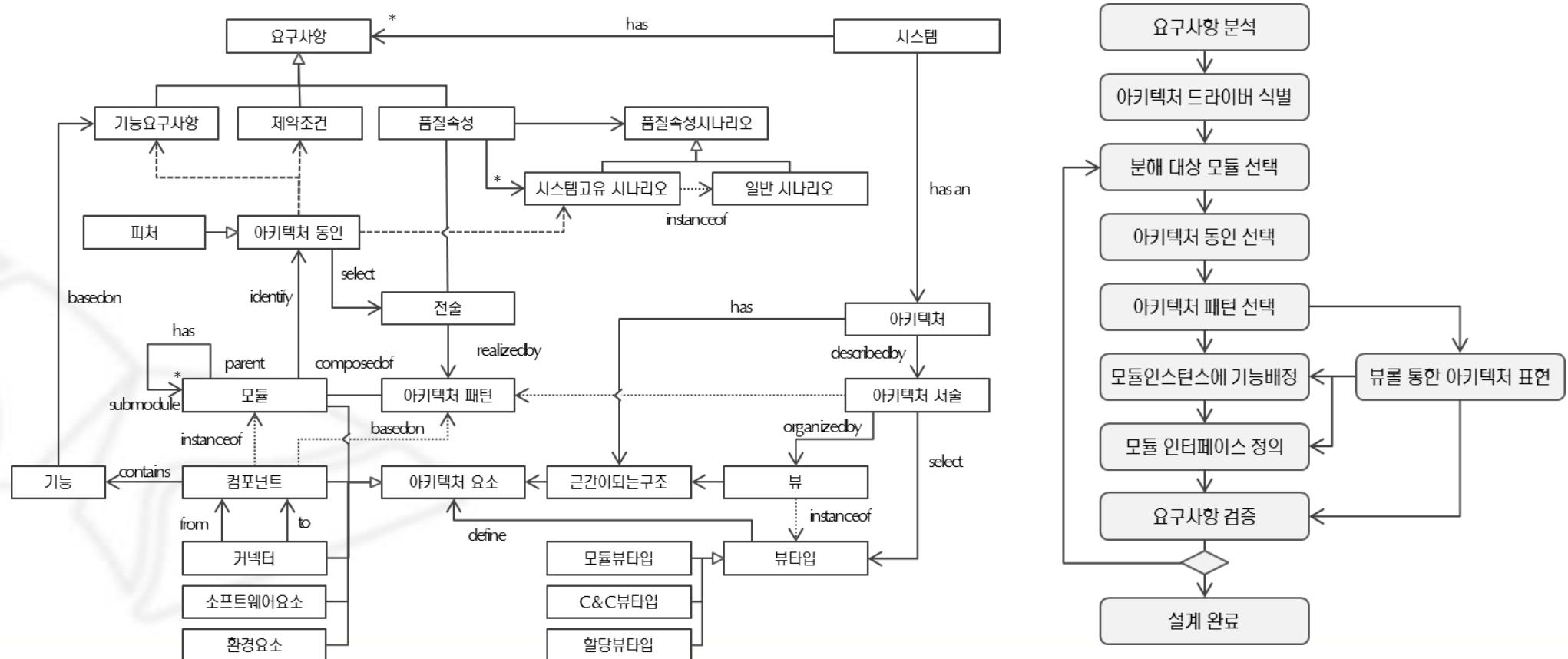
✓ 주제별 아키텍팅이라는 개념을 그대로 사용했습니다.

|                |   |
|----------------|---|
| 1. 정의          | 1.1 다양한 아키텍처 개념 정의 및 조정<br>1.2 전사 시스템 아키텍팅 프로세스 정의<br>1.3 아키텍팅 접근방법 정의 – 뷰와 모델    |
| 2. 전사 시스템 아키텍팅 | 2.1 AS-IS 시스템 아키텍처 표현 및 분석<br>2.2 전사 시스템 아키텍처 설계<br>2.3 아키텍팅 주제 식별 및 우선순위 결정      |
| 3. 주제별 아키텍팅    | 3.1 주제 영역 연구 및 접근방법 결정<br>3.2 SW 아키텍처 설계<br>3.3 아키텍처 요소 획득<br>3.4 RI를 통한 검증       |
| 4. 통합 및 조정     | 4.1 주제별 아키텍팅 결과 통합 및 조정<br>4.2 전사 아키텍처 가이드 업데이트<br>4.3 아키텍처 적용(enforcement) 계획 수립 |

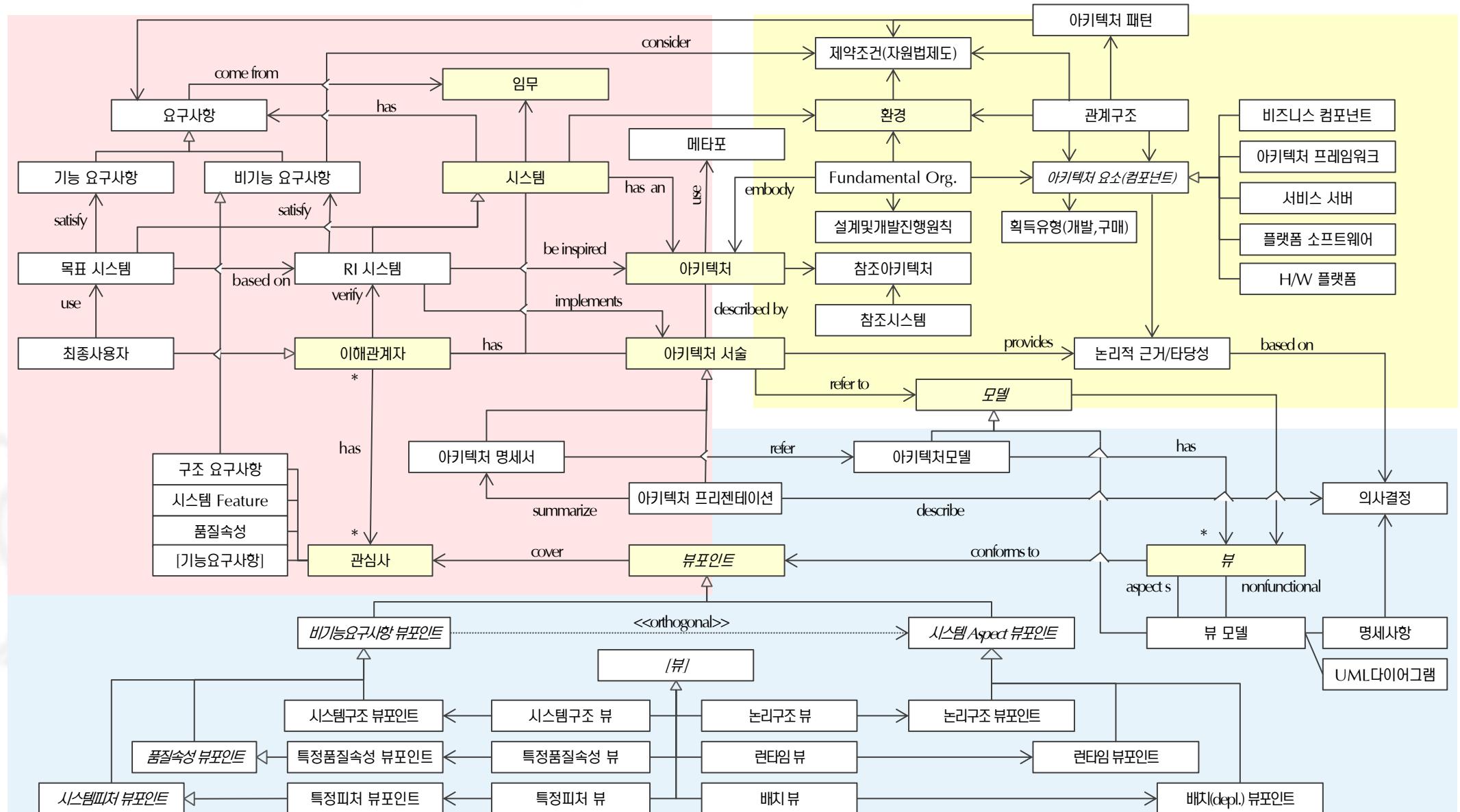


# Reference Architecting Process – ADD

- ✓ ADD 프로세스는 현대의 다양한 아키텍팅 프로세스의 바탕이 되는 프로세스입니다.
- ✓ 소규모 시스템 또는 Embedded 시스템 개발에서는 아직도 활발하게 사용되는 프로세스입니다.
- ✓ 프로젝트에서 개발하려는 목표 시스템과 개발팀의 특성에 맞추어 조정(tailoring) 하여야 사용합니다.



# IEEE 1471 – 개념 모델 확장

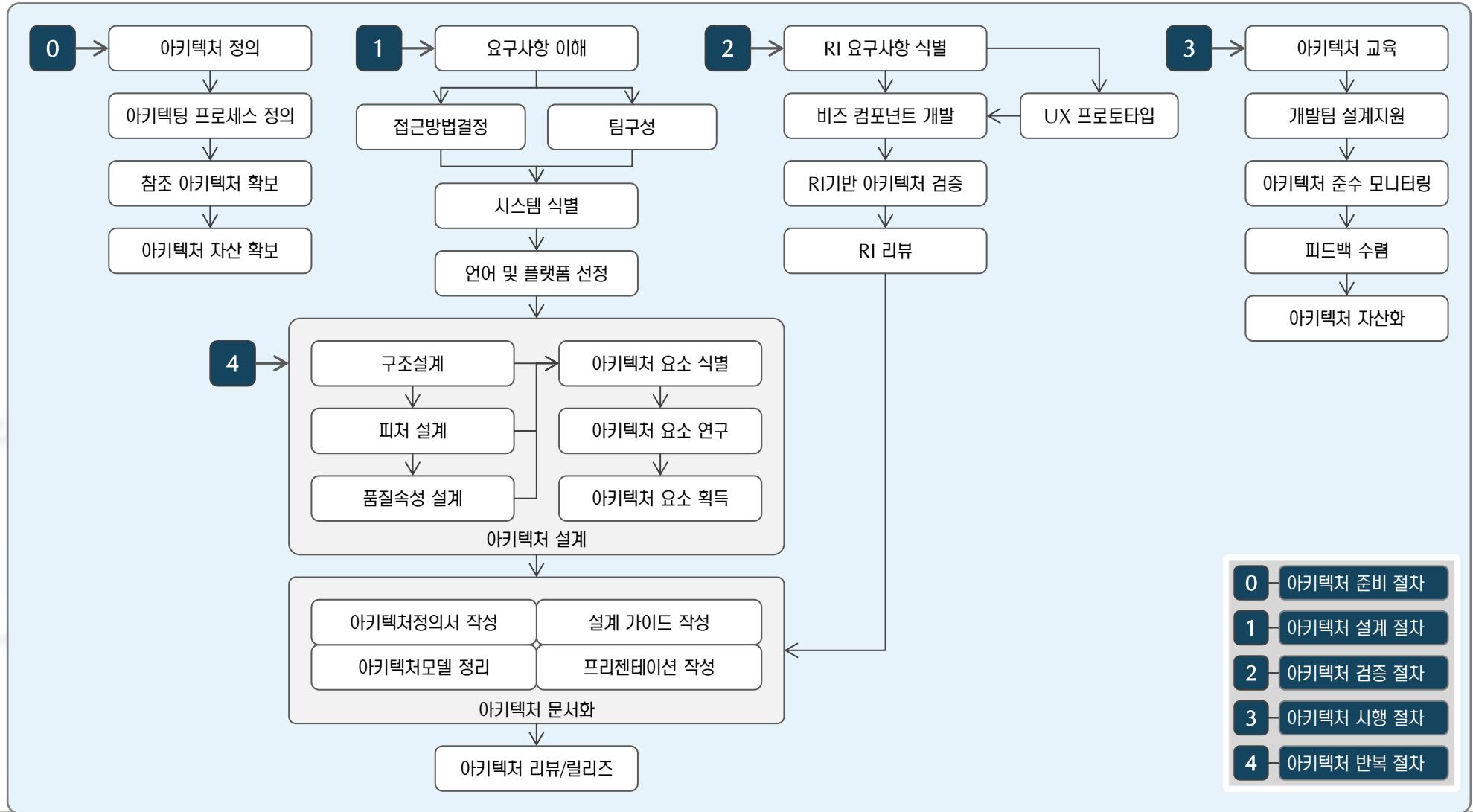


# New Architecting Process

✓ 아키텍팅 프로세스는 IEEE 1471 확장 모델을 기반으로 합니다.

2011년

5. IEEE 1471 확장 모델 기반 아키텍팅 프로세스

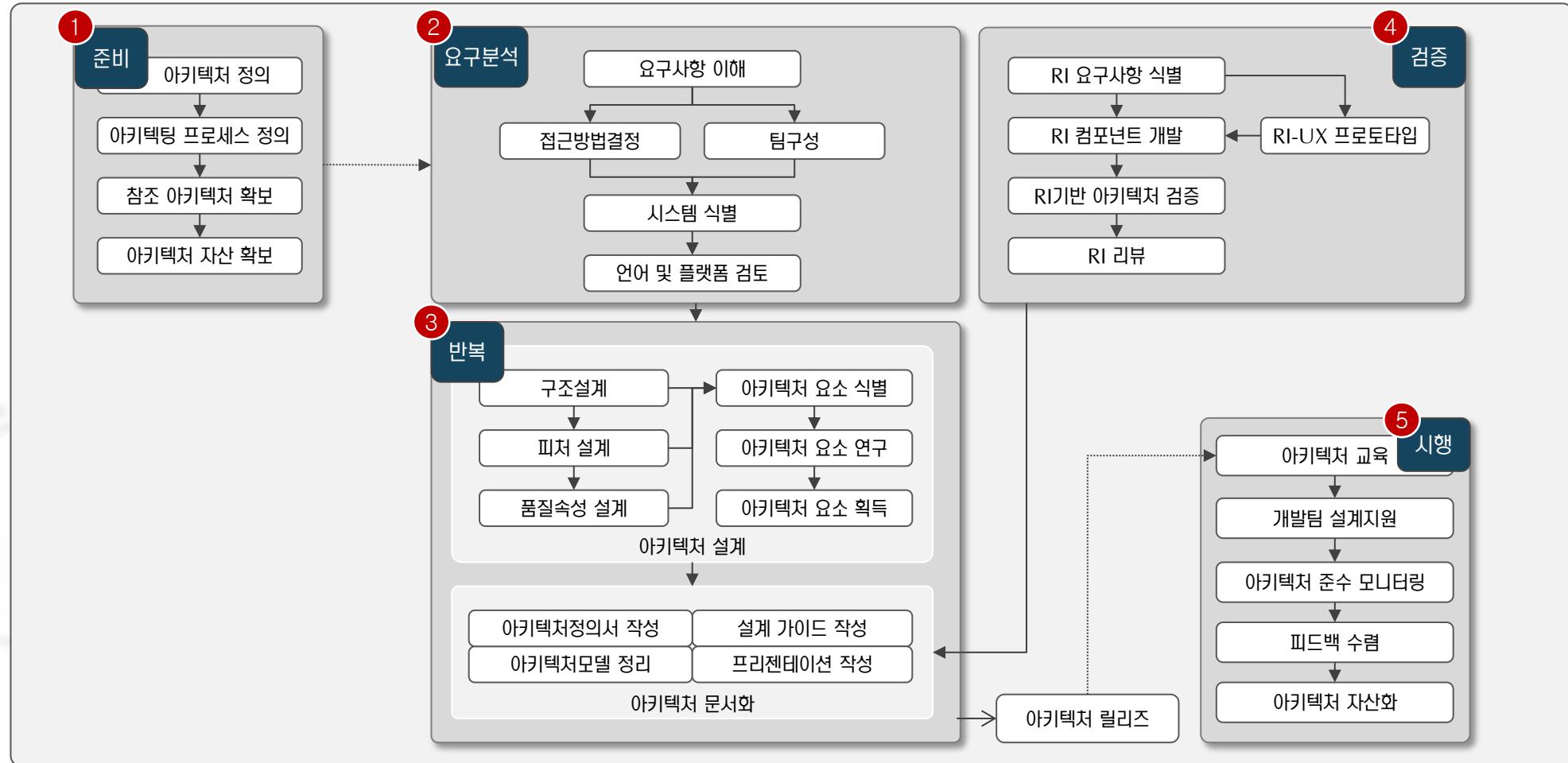


# New Architecting Process(refined)

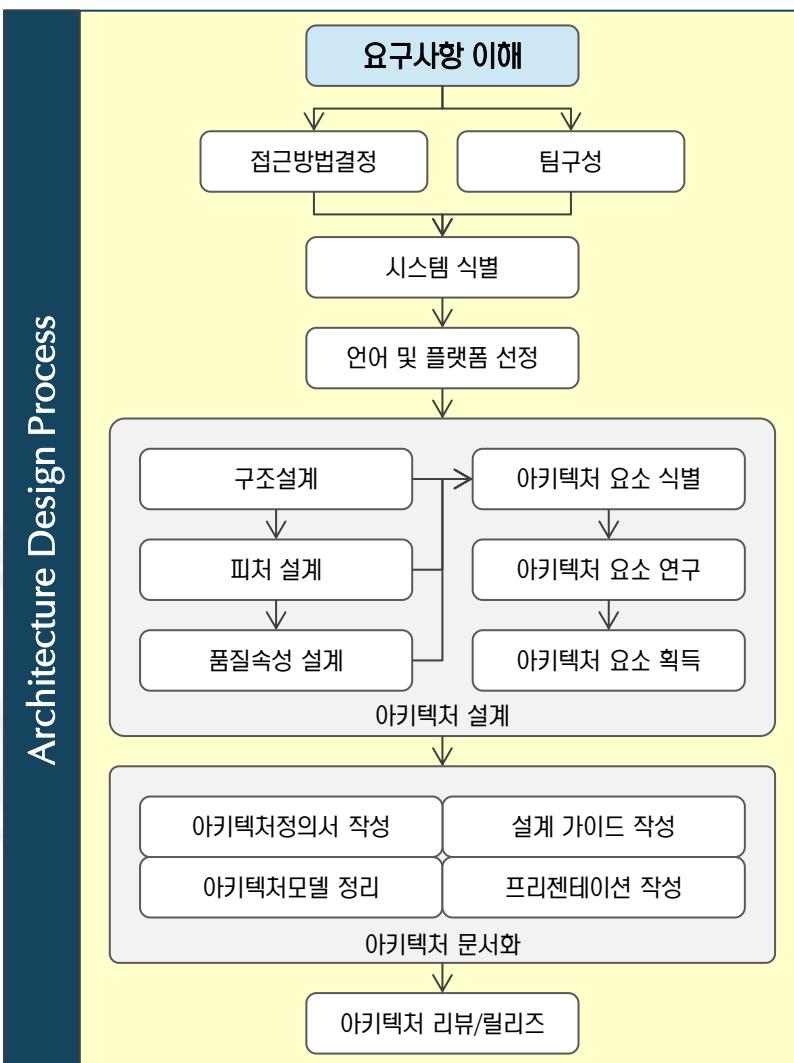
✓ 아키텍팅 프로세스는 IEEE 1471 확장 모델을 기반으로 합니다.

2011년

5. IEEE 1471 확장 모델 기반 아키텍팅 프로세스



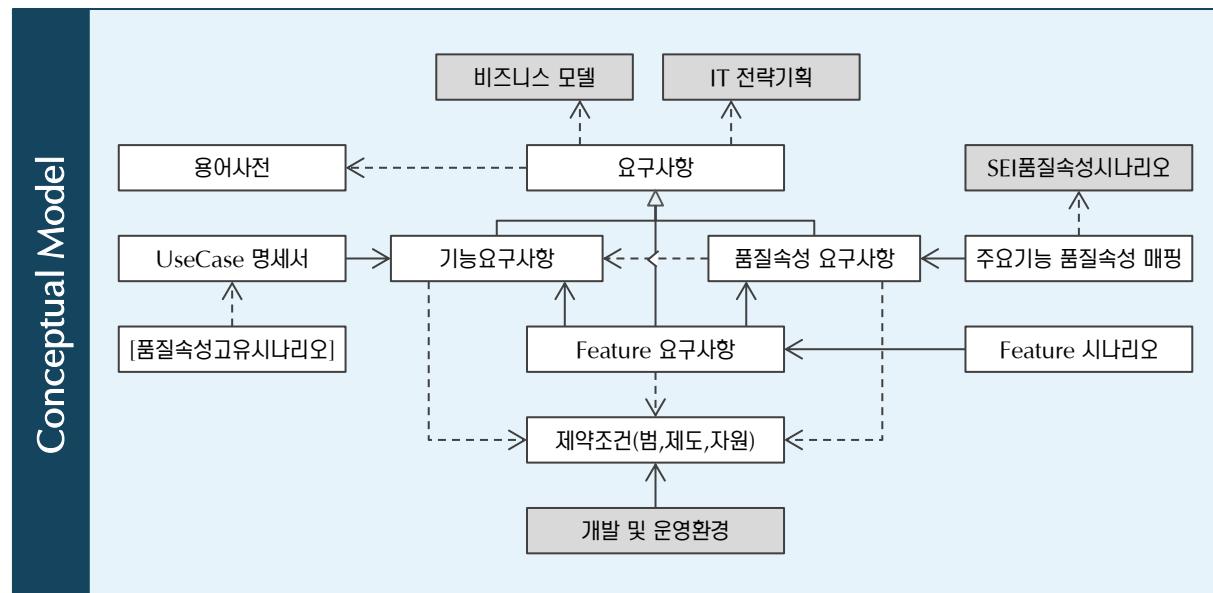
# 요구사항 이해(1/5)



## Activity Guide

- ❖ 아키텍처 요구사항은 품질속성과 피처(Feature) 임
- ❖ 아키텍처 요구사항은 환경이 제공하는 제약조건을 고려함
- ❖ 품질속성 요구는 주요 기능-품질속성 맵으로 정리함
- ❖ SEI 스타일의 품질속성 시나리오를 사용할 수도 있음
- ❖ Feature 요구는 시나리오를 활용함

## Conceptual Model



## 요구사항 이해(2/5)

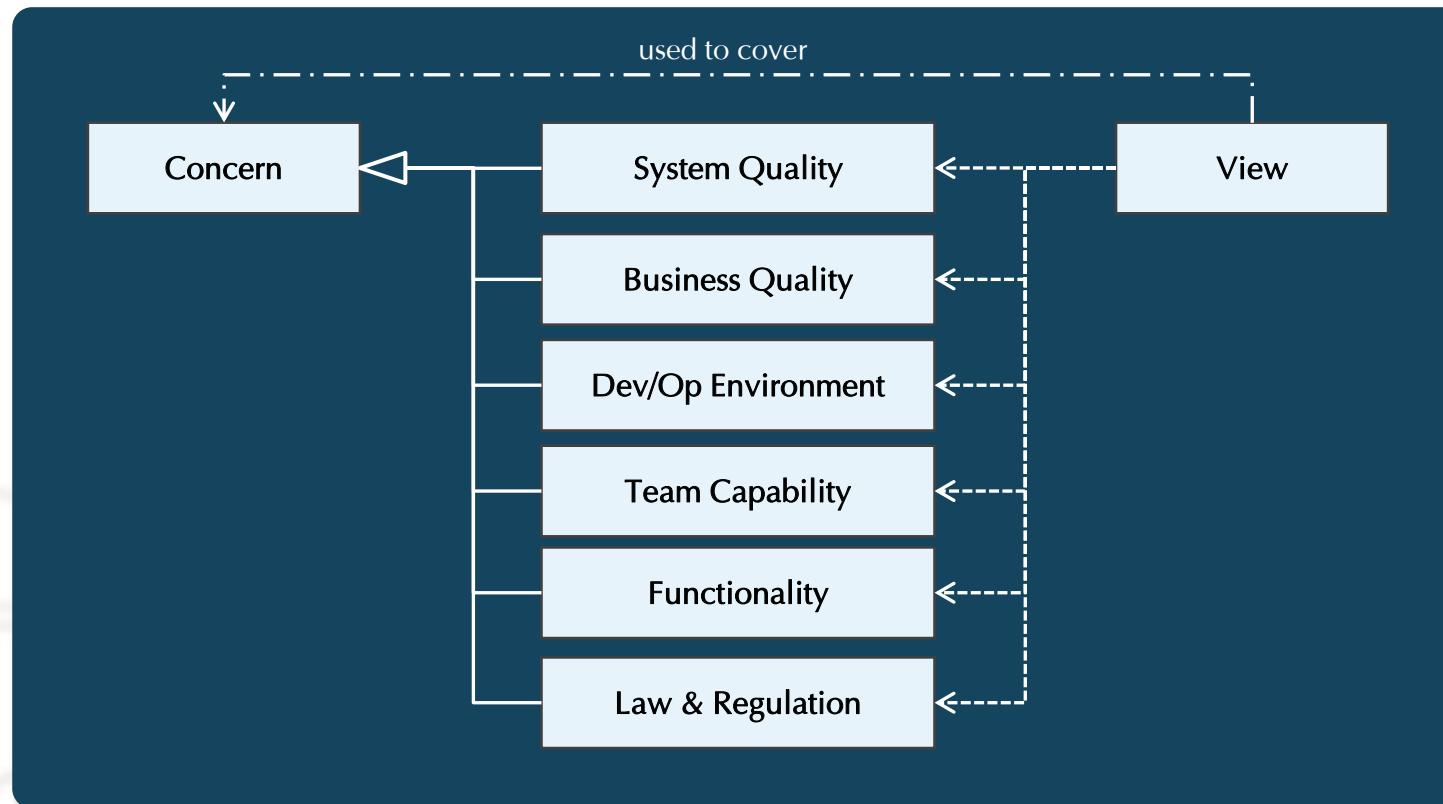
- ✓ 품질과 기능의 차이와 역할에 대한 이해는 아키텍처 설계의 출발점입니다.
- ✓ 기능과 품질은 직교(orthogonal) 함
  - 상판:기능
  - 교각:품질
- ✓ 품질은 주로 Cross-cutting 이슈입니다.
- ✓ 두 경계선이 흐릿한 경우도 있습니다.
- ✓ 피처(Feature)는 두 속성의 조합입니다.

[ 세계에서 가장 높은 다리인 미요교(Le Viaduc de Millau)]



## 요구사항 이해(3/5)

- ✓ 이해관계자의 관심사는 매우 다양하며, 시스템 영역에서 포함되는 것과 그렇지 않은 것들이 있습니다.
- ✓ 여러 관심사를 특정 기준으로 묶음짓고 이를 바라보는 관점을 정의하는데 그것을 뷰(view)라고 합니다.
- ✓ 관심사는 포괄적인 개념으로 시스템 품질이 주를 이루고 있지만, 기능성, 비즈니스 환경, 운영 환경, 법과 제도를 포함하고 있습니다.



# 요구사항 이해(4/5)

- ✓ 품질 속성은 조직 별, 컨텍스트 별로 서로 다른 정의를 가질 수 있습니다.
- ✓ 모든 품질 속성은 관심사가 될 수 있지만, 모든 관심사가 품질 속성은 아닙니다.
- ✓ 프로젝트 팀은 목표 시스템과 관련된 품질 속성을 정의함으로써 의사소통을 원활하게 할 수 있습니다.

## 시스템의 품질 속성(From SEI)

- 가용성(availability)
- 변경 용이성(modifiability)
- 수행성능(performance)
- 보안성(security)
- 테스트 용이성(testability)
- 사용성(usability)
- 상호운영성(interoperability)
- 이식성(Portability)
- 범위성(Scalability)

## 다양한 품질속성...

### 비즈니스의 품질속성 (From SEI)

- 시장 출하 시기(time to market)
- 비용과 장점(cost and benefit)
- 시스템의 예정 생명주기(projected lifetime of the system)
- 목표 시장(targeted market)
- 최초 공개 일정(rollout schedule)
- 기존 시스템과의 통합(integration with legacy systems)

### 시스템의 품질 속성(From TOGAF)

#### Availability

- manageability
- serviceability
- performance
- reliability
- recoverability
- locatability

#### Assurance:

- security
- integrity
- credibility

#### Usability

- International operation

#### Adaptability

- interoperability
- scalability
- portability
- extensibility

# 요구사항 이해(5/5)

- ✓ 하나의 기능은 여러 품질 속성과 관련이 있습니다.
- ✓ 서비스(기능) 제공 목표를 달성하기 위해, 요구되는 품질 목표를 달성하여야 합니다.
- ✓ 때로는 품질 목표가 기능 설계 또는 업무 구조 또는 절차와 관련이 있을 수 있습니다.
- ✓ 따라서, 아키텍팅 활동을 위해 특정 업무 기능에 대한 이해가 필요할 수도 있습니다.

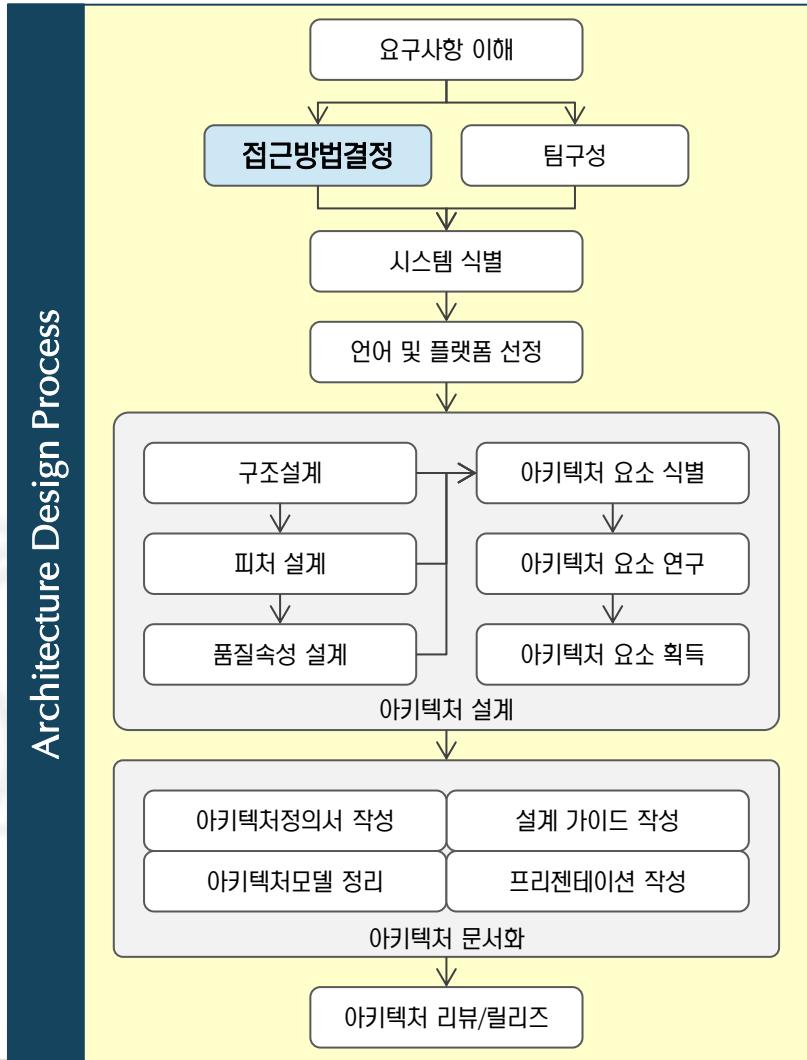
| 구분   |      |           | 경매등록  | 경매물품조회 | 입찰               | 낙찰금액지불    | 경매마감   |
|------|------|-----------|-------|--------|------------------|-----------|--------|
| 품질속성 | 명세내용 | 단위        |       |        |                  |           |        |
| 가용성  | 중요도  | 상중하       | 중     | 상      | 상                | 중         | 상      |
|      | 허용범위 | 다운시간/년    | 30H   | 10H    | 20H              | 30H       | 1H     |
| 성능   | 중요도  | 상중하       | 하     | 상      | 상                | 하         | 상      |
|      | 허용범위 | 초(sec)    | 3     | 1      | 1                | 5         | 1      |
| 범위성  | 중요도  | 상중하       | 하     | 상      | 중                | 하         | 하      |
|      | 허용범위 | 동시 사용자수   | 5천    | 10만명   | 1만명              | 5천명       | 5천명    |
| 보안   | 중요도  | 상중하       | 중     | 하      | 중                | 상         | 하      |
|      | 허용범위 | 인증/인가/암호화 | 인증/인가 | 인증     | 인증/인가            | 인증/인가/암호화 | 해당사항없음 |
| 확장성  | 중요도  | 상중하       | 중     | 상      | 중                | 중         | 하      |
|      | 허용범위 | 작업일       | 3     | 3      | 3                | 5         | 3      |
| 관리성  | 중요도  |           |       |        | Cross-cutting 이슈 |           |        |
|      | 허용범위 |           |       |        |                  |           |        |
| 사용성  | 중요도  | 상중하       | 중     | 상      | 상                | 중         | 하      |
|      | 허용범위 | 사용자만족도    | 80%   | 95%    | 90%              | 80%       | 해당사항없음 |

[주요 기능과 품질 속성 간의 연관관계]

아키텍팅  
포인트

# 접근방법 결정(1/2)

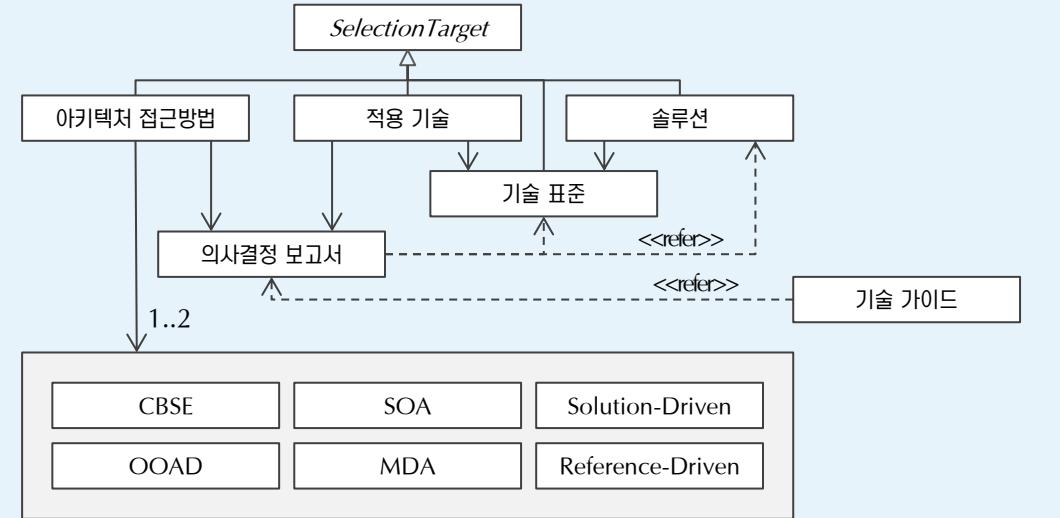
✓ 시스템 구축을 위한 접근 방법, 기술, 표준, 솔루션 등에 대한 방향을 결정합니다.



Activity Guide

- ❖ 기준이 되는 접근방법을 의사결정보고서를 통해 결정함
- ❖ 적용기술, 관련 표준, 핵심 솔루션 등에 대한 Study를 진행함
- ❖ [주의] 초기에 모든 솔루션을 검토할 필요는 없음
- ❖ 이 활동이 종료되면 접근방법에 대한 공감대가 형성되고, 주요 기술, 표준, 솔루션 등에 대한 기본적인 이해를 공유함
- ❖ 마케팅용 Keyword를 걸러내고 팀을 올바른 방향으로 리드하는 활동

Conceptual Model



## 접근방법 결정(2/2)

- ✓ 프로젝트 초기 팀이 필요로 하는 것은 정확한 방향성과 접근방법입니다.
- ✓ 접근방법이 결정된 후, 주요 기술에 대한 조사 및 정리 후 공유하는 과정이 필요합니다.
- ✓ 기술은 관련 표준과 관련 솔루션을 동반하는 경향이 있으므로 너무 깊이 들어가지 않도록 주의합니다.
- ✓ 조직의 아키텍트 그룹에서 [주요 기술 동향 및 분석 보고서]를 정기적으로 발간하면 도움이 됩니다.

### [접근방법] CBD가 아닌 CBSE 접근방법

1. 컴포넌트 개요
2. 컴포넌트 정의
3. 컴포넌트 식별
4. 컴포넌트 관계 규칙
5. 컴포넌트 내부 구조
6. 컴포넌트 개발 프로세스

사례참조

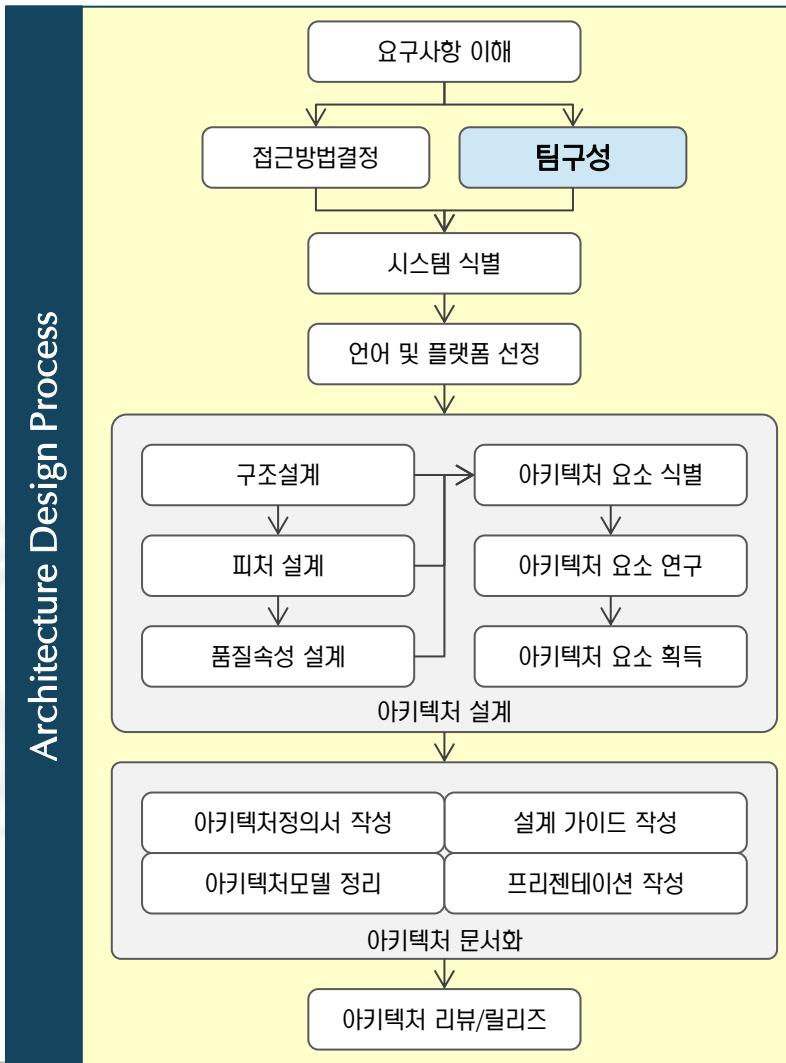
### [기술] Identity & Access Management

1. Identity 관리 개념 정의
2. Identity 관리 프로젝트 접근방법
3. 정의 단계
4. Identity 관리
5. Access 관리
6. Identity 관리 이슈사항
7. 시나리오

사례참조

# 팀구성(1/3)

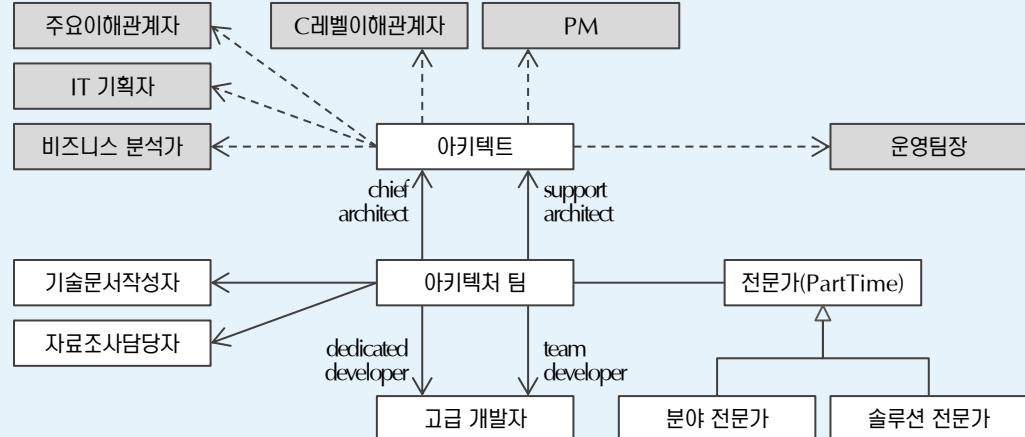
✓ 소프트웨어 아키텍처 팀을 구성합니다.



Activity Guide

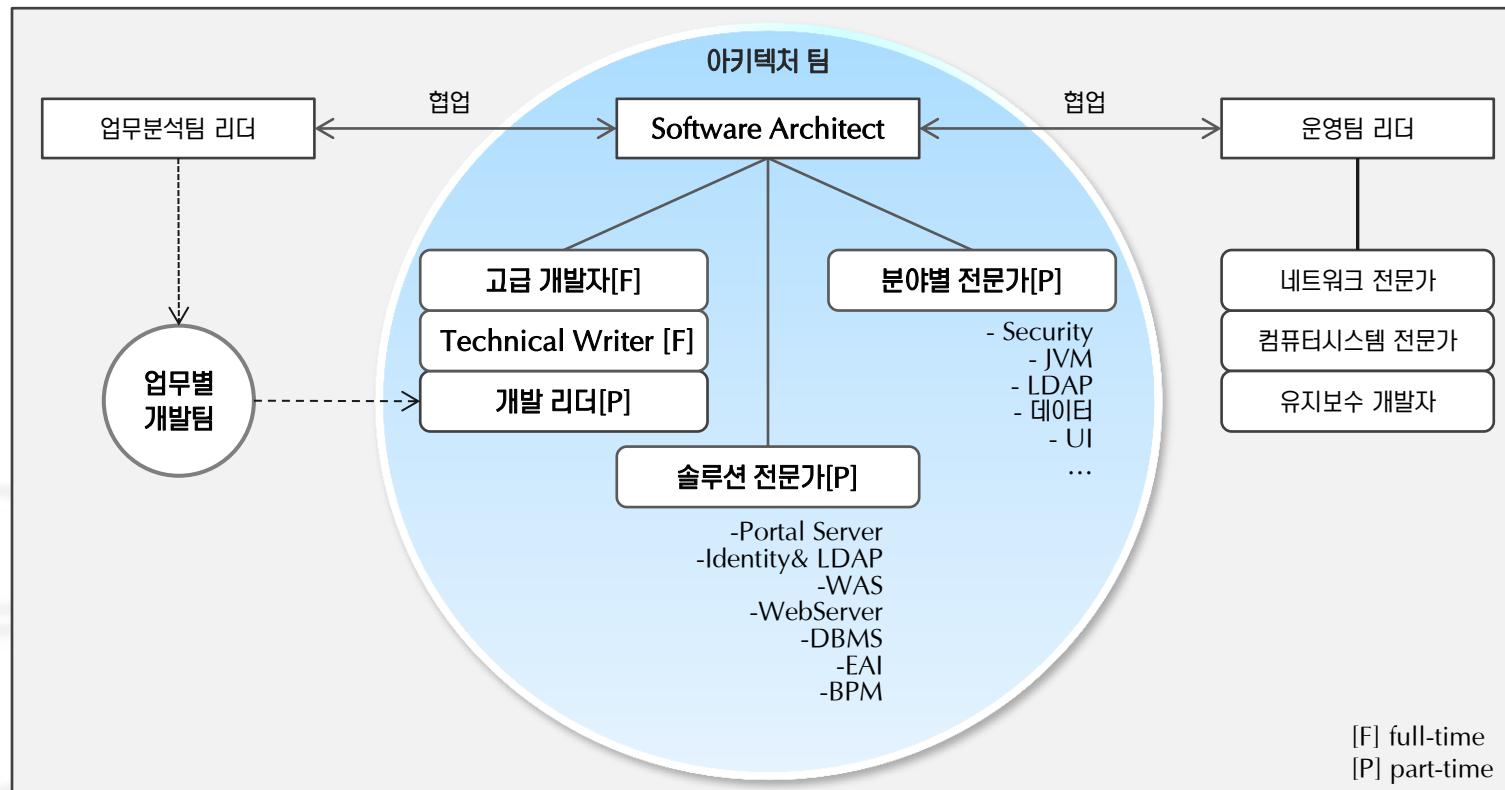
- ❖ 팀은 개발 역량과 전문화 역량, 솔루션 역량을 중심으로 구성함
- ❖ 기술 문서작성자와 자료 조사 담당자를 통해 문서 수준을 높임
- ❖ 대체로 전문가는 필요한 기간에 파트타임으로 참여를 함
- ❖ 아키텍트는 다양한 역할 담당자, 이해관계자와 의사소통함
- ❖ 아키텍처 팀 규모는 조직의 업무특성과 경험에 의존함
- ❖ 고급개발자 그룹에 각 업무개발팀의 개발리더 참여를 권장함

Conceptual Model



## 팀구성(2/3)

- ✓ 아키텍처 팀 구성은 AA, DA, TA, BA 역할로 채워서는 안됩니다.
- ✓ 아키텍처 팀의 핵심은 고급 개발인력과 파트 타임으로 참여하는 솔루션 전문가, 또는 분야별 전문가입니다.
- ✓ 하드웨어, 네트워크 등은 아키텍처 팀의 범주를 넘어섭니다.
- ✓ 프로젝트와 사용 기술 특성에 따라 아키텍처 팀 규모를 결정합니다.



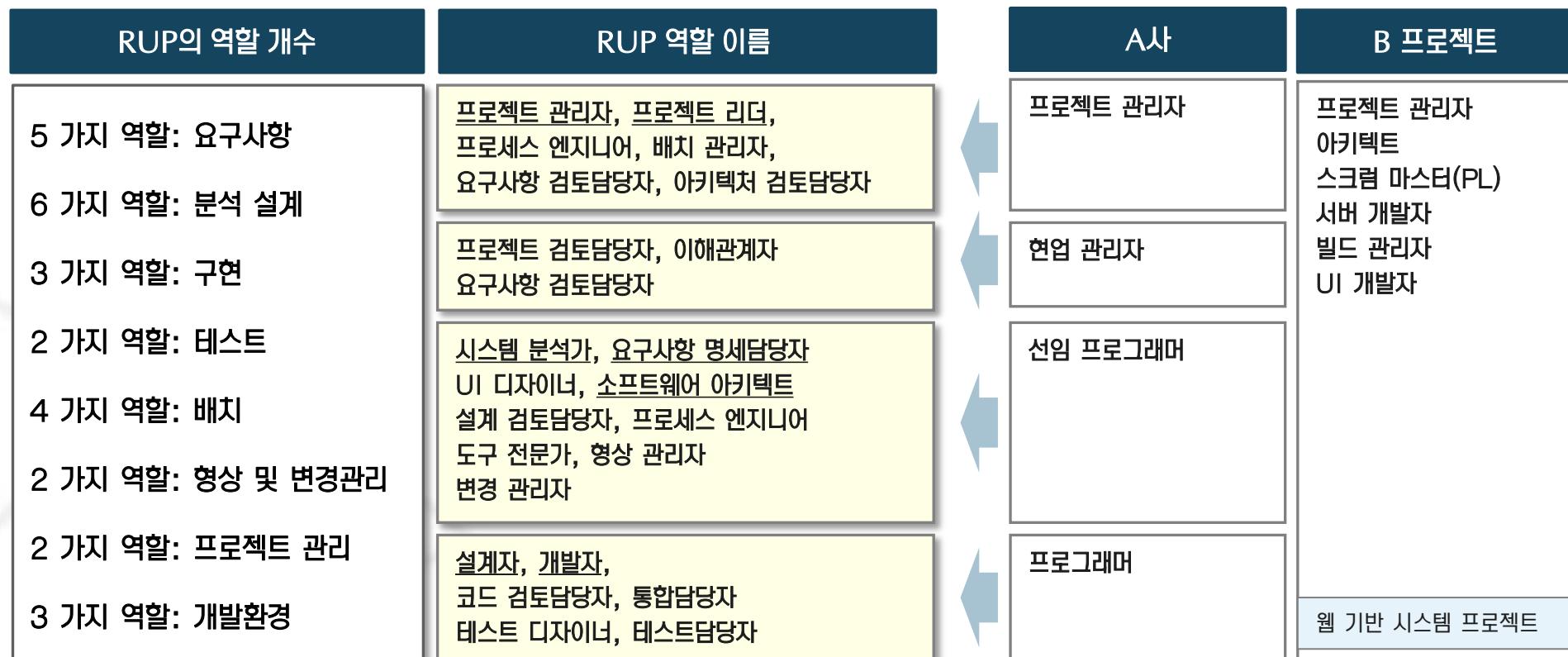
## 팀 구성(3/3)

- ✓ 어떤 유형의 개발이든 최소 1명의 아키텍트 역할을 지정해야 합니다.
- ✓ 애플리케이션 유형: 독립형 시스템, 엔터프라이즈 시스템(System of System)
- ✓ 개발 유형: 신규 개발, 확장 개발, 솔루션 맞춤(customization)
- ✓ 아키텍처 팀의 활동 범위에 따라 규모가 달라질 수 있습니다. ← 아키텍팅 프로세스 조정에 따라

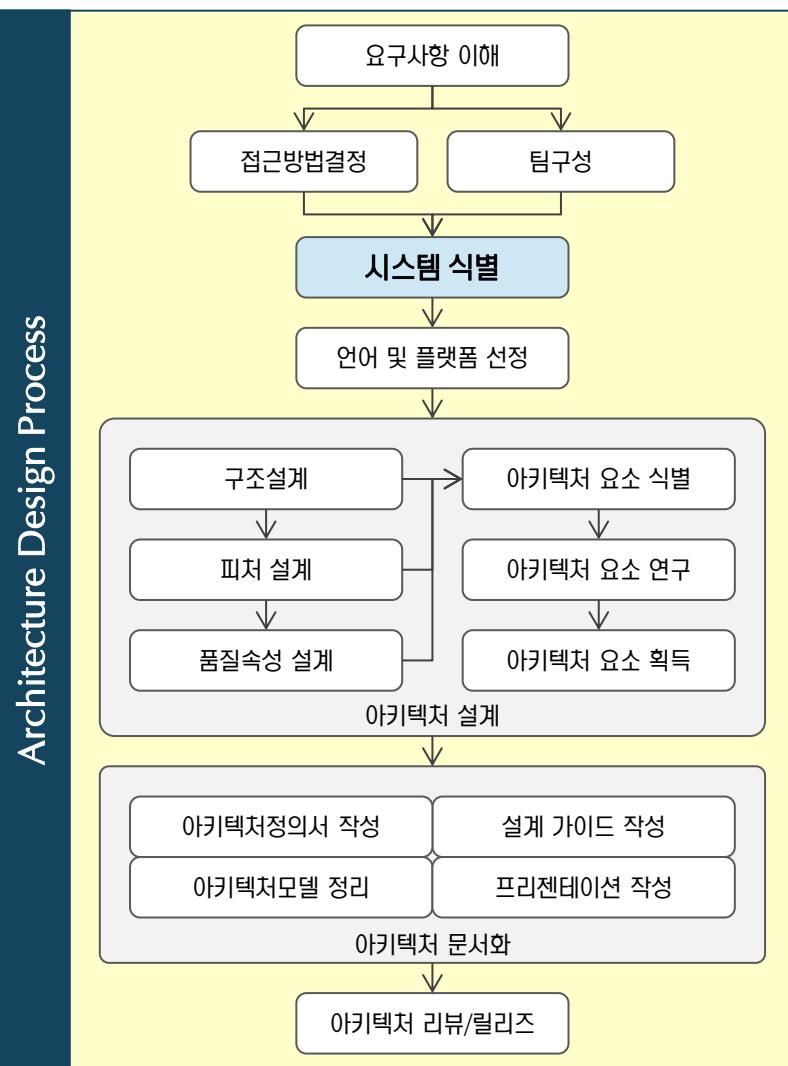
| 시스템 유형<br>/<br>개발 유형 |            | 독립형 시스템    |             | 엔터프라이즈 시스템  |  |
|----------------------|------------|------------|-------------|-------------|--|
| 신규<br>개발             | 소규모(10명내외) | 중규모(50명내외) | 중규모(100명내외) | 대규모(200명이상) |  |
|                      | 2명         | 5~7명       | 12명         | 20명이상       |  |
| 확장<br>개발             | 1명         | 3명         | 7명          | 12명이상       |  |
|                      | 1명         | 3명         | 6명          | 10명이상       |  |
| 솔루션 맞춤               | 1명         | 2명         | 4명          | 8명이상        |  |
|                      | 1명         | 1명         | 2명          | 4명이상        |  |

# 개발 팀과 역할

- ✓ 소프트웨어 공학 기준, 표준 역할이 있으며, 개발 관련 프로세스는 역할 이름을 준수하여야 합니다.
- ✓ 조직의 특성과 프로젝트의 상황에 맞추어 역할을 정의합니다.
- ✓ 프로젝트에 사용한 기술에 따라 역할을 재 정의할 필요가 있습니다. , 예, UI integrator, 웹 퍼블리셔, 웹 기획자



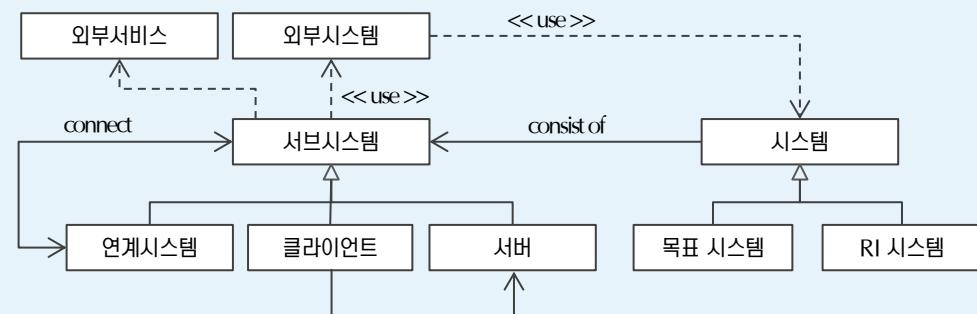
# 시스템 식별(1/3)



## Activity Guide

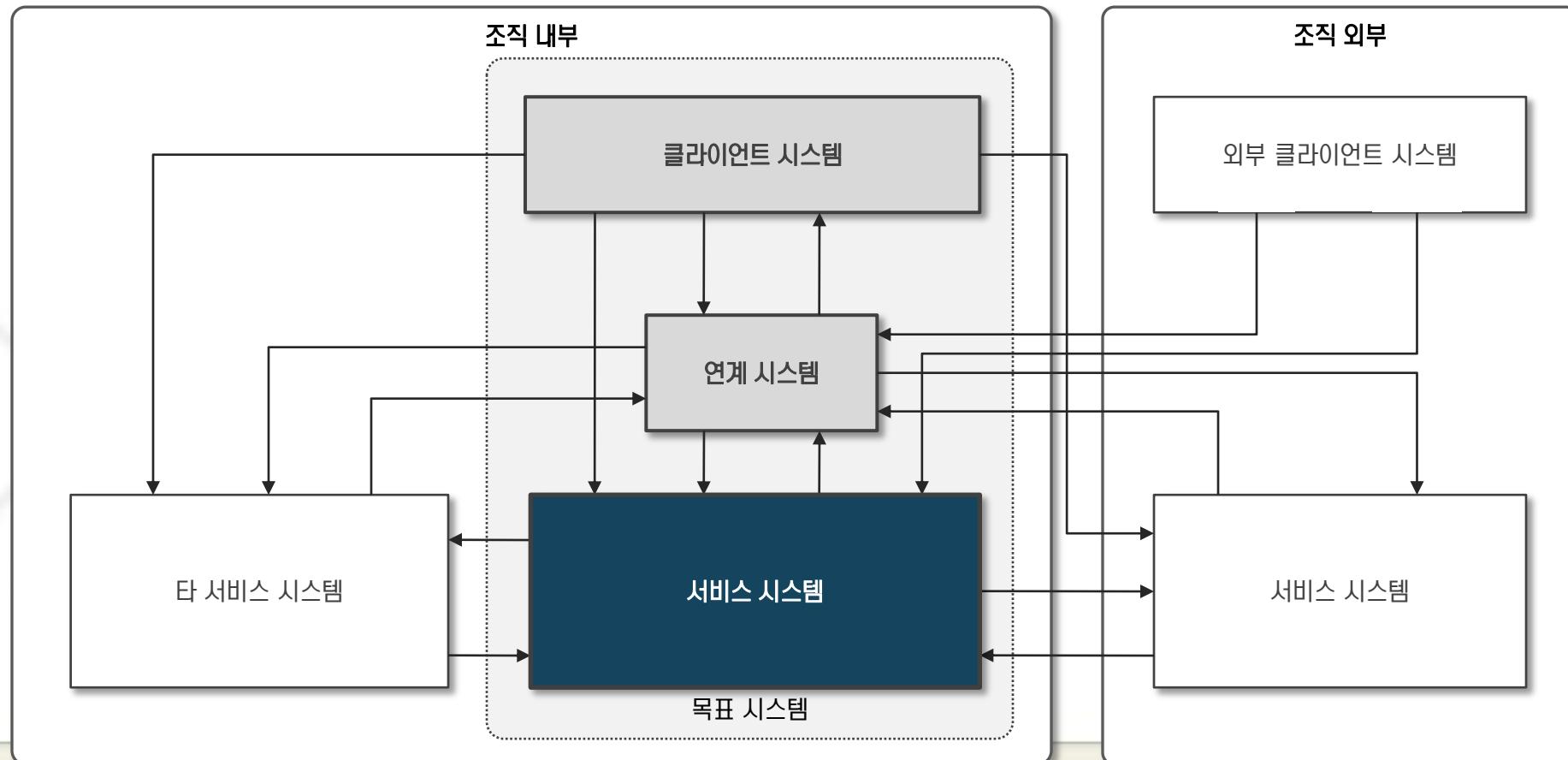
- ❖ 시스템 컨텍스트 안의 다양한 시스템을 식별함
- ❖ 시스템은 서브시스템으로 구성되며,
- ❖ 느슨한 결합을 위해 클라이언트와 서버를 서브시스템으로 식별함
- ❖ 완전한 서비스 지향 아키텍처에서 연계 시스템은 필요없음
- ❖ 고려대상은 목표 시스템과 RI 시스템 두 가지임
- ❖ 구현 범위 밖의 미래 시스템 역시 식별하여야 함

## Conceptual Model



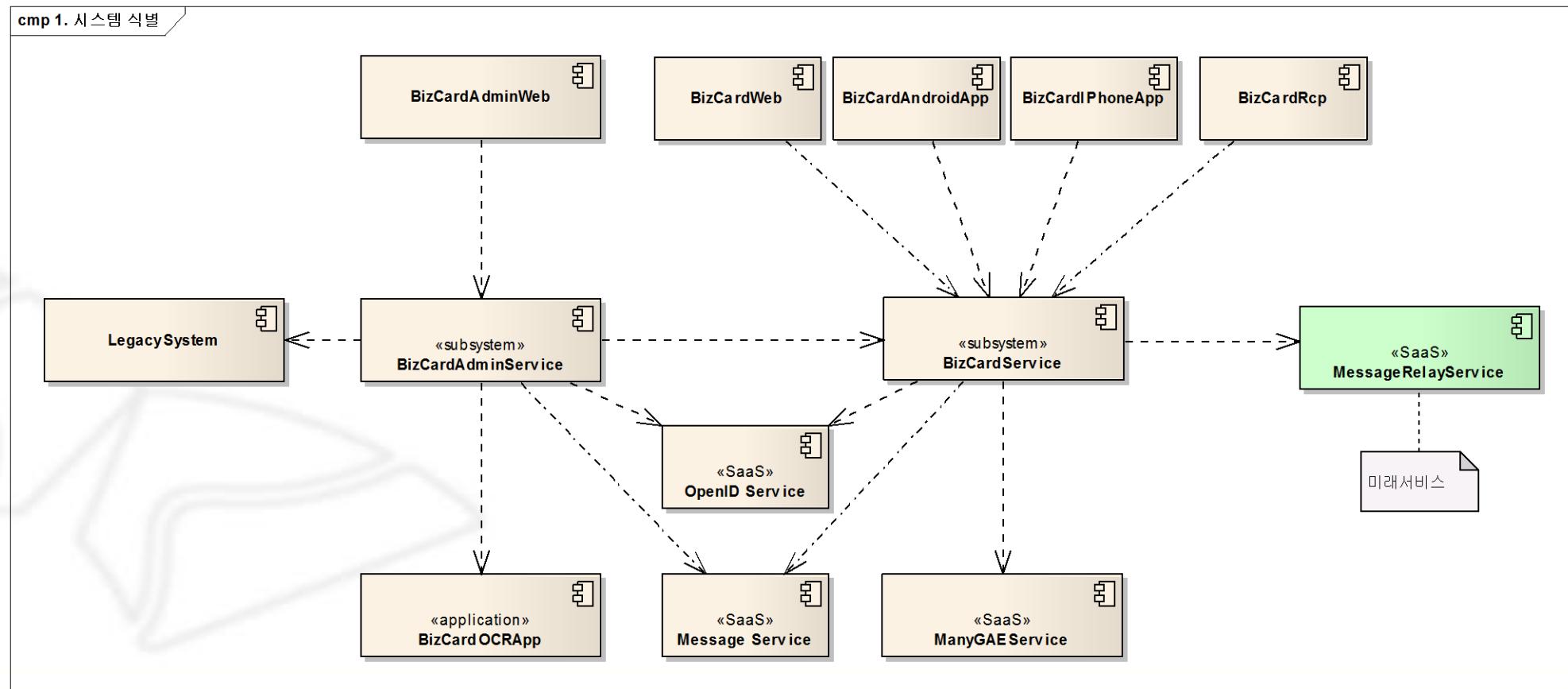
## 시스템 식별(2/3)

- ✓ 우선 시스템 유형을 정의하고, 정의된 기준에 근거하여 시스템을 식별합니다.
- ✓ 조직 내부의 시스템과 외부의 시스템을 나누어 식별하는 것이 바람직합니다.



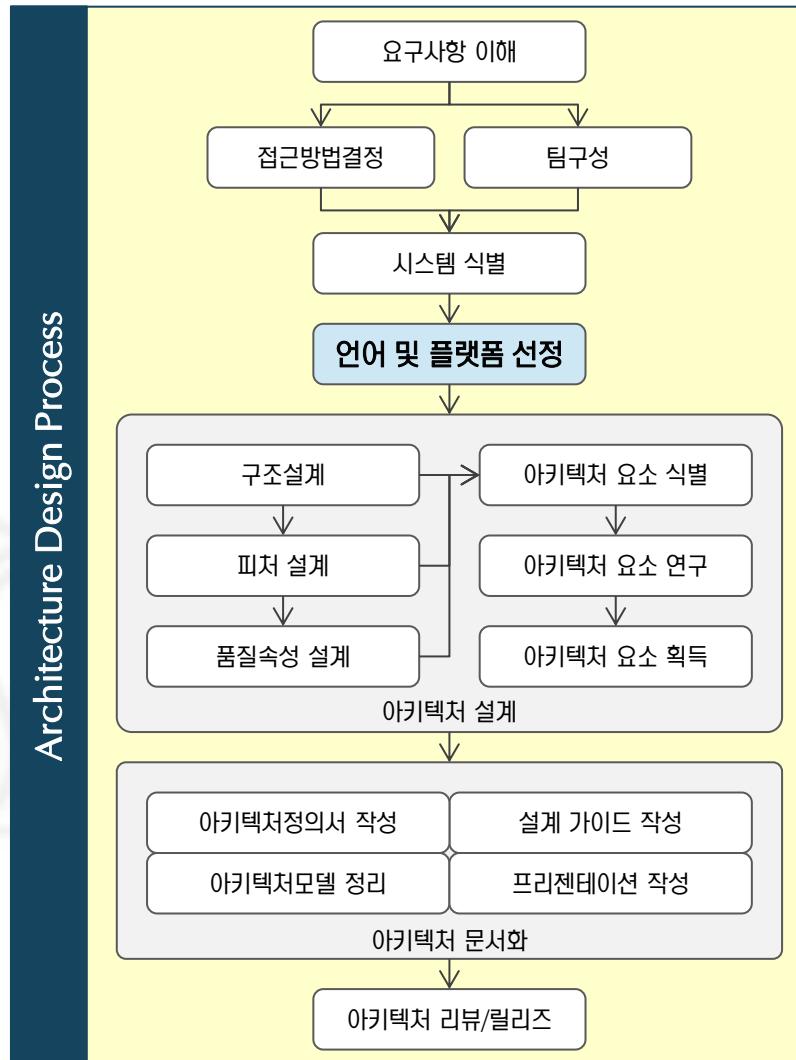
# 시스템 식별(3/3)

- ✓ 예제, 비즈카드 시스템에서 시스템 식별
- ✓ SEI에서 모듈 분해(Decomposition)와 유사한 작업입니다.



# 언어 및 플랫폼 선정(1/2)

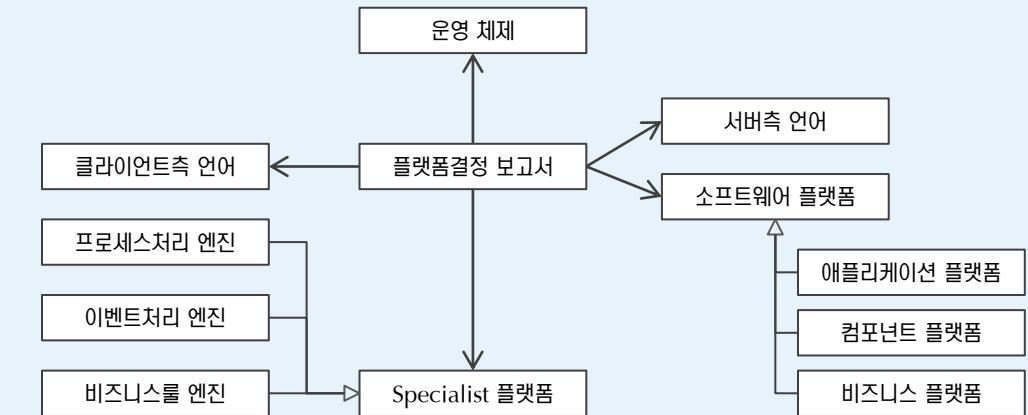
✓ 아키텍처 설계를 위한 프로그램 언어 및 플랫폼을 선정합니다.



## Activity Guide

- ❖ 언어는 클라이언트측과 서버측 언어를 별도로 식별함
- ❖ 소프트웨어 플랫폼을 식별함
- ❖ 엔진으로 명명할 수 있는 특수 플랫폼을 식별함

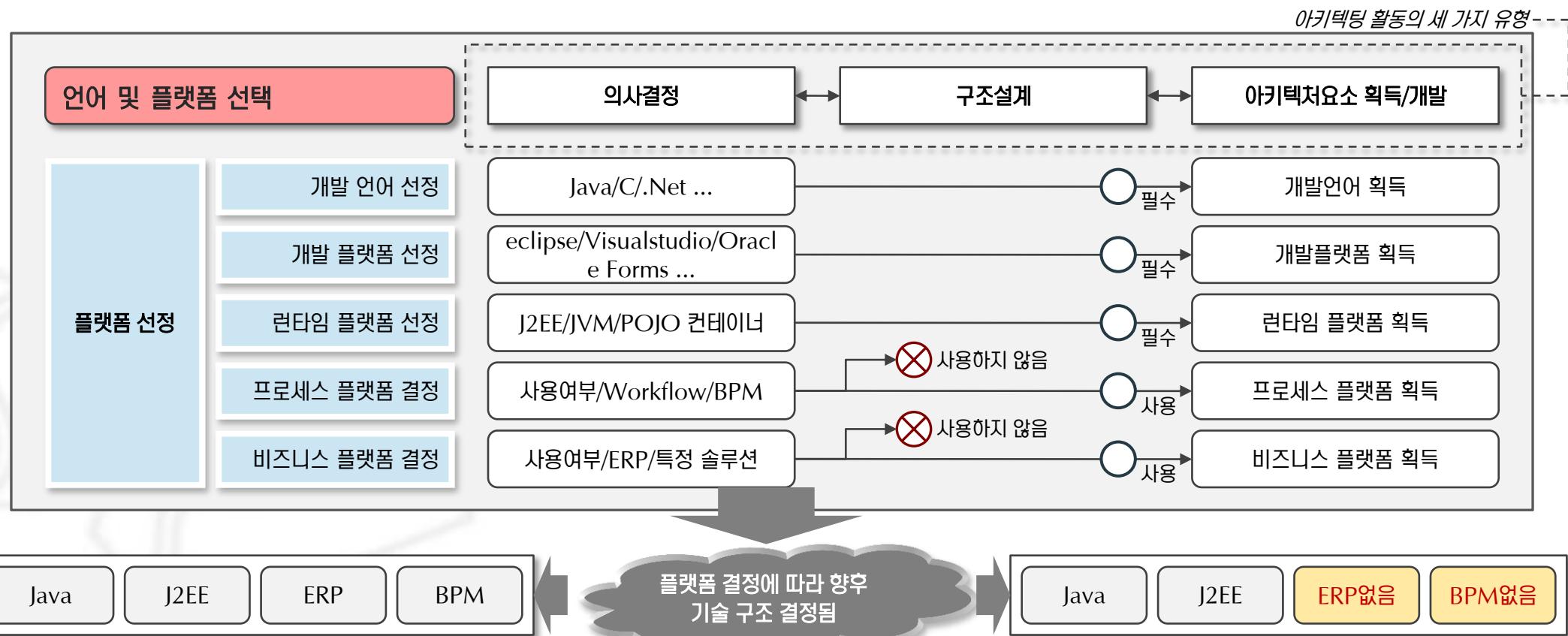
## Conceptual Model



# 언어 및 플랫폼 선택(2/2)

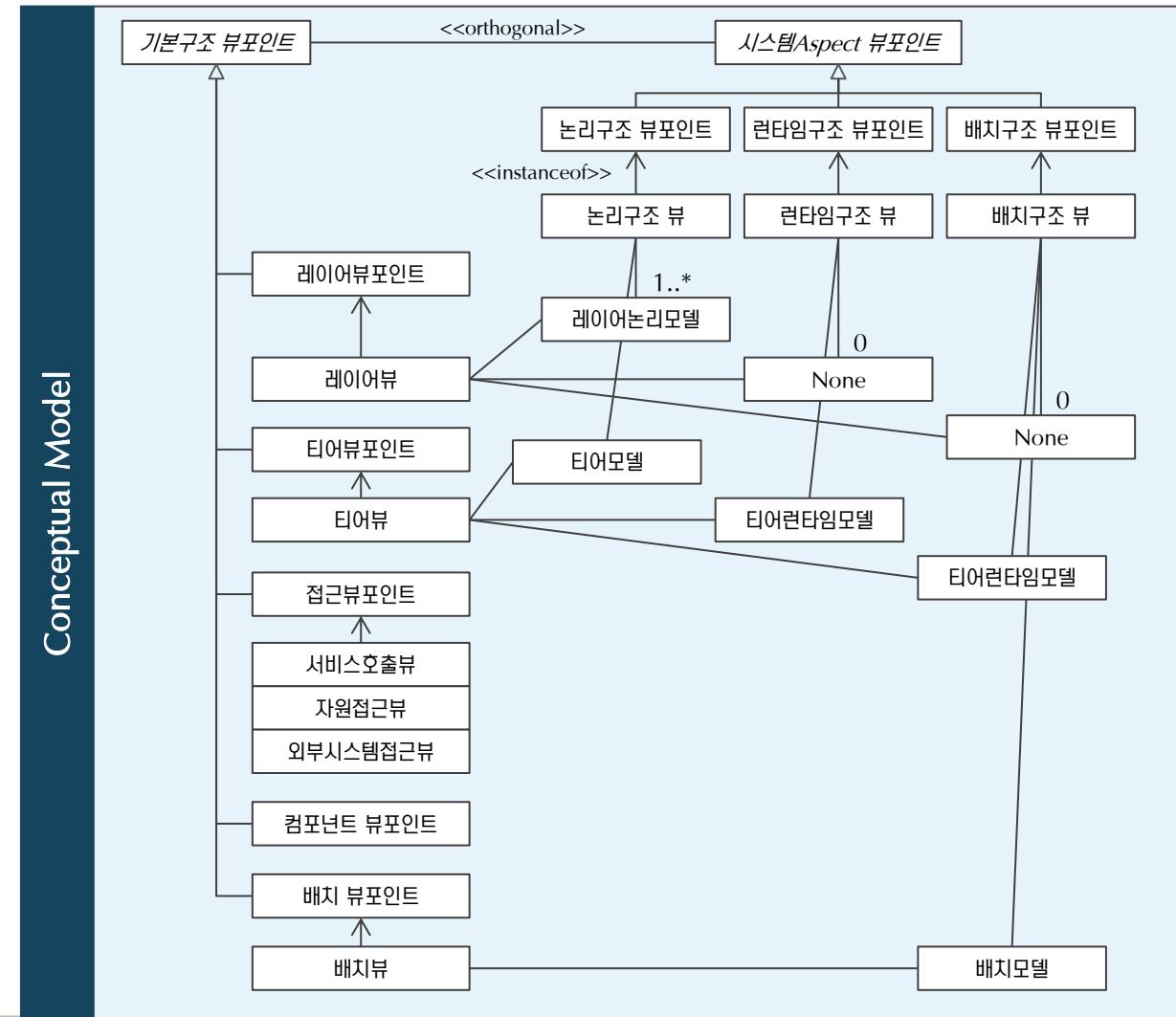
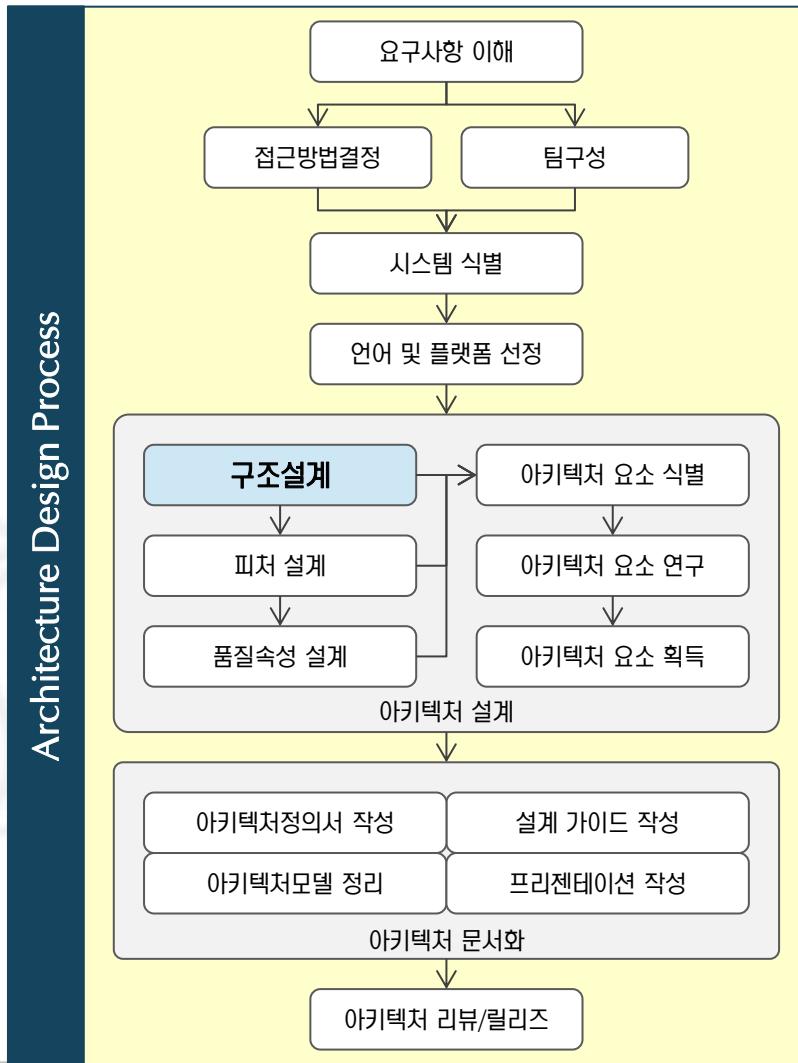
✓ 잘못된 플랫폼 선택은 많은 문제의 근본 원인을 제공하고 있습니다.

- 개발언어와 련타임 플랫폼은 필수 선택항목이며, 프로세스 플랫폼과 비즈니스 플랫폼은 옵션항목입니다.
- 선택된 플랫폼은 기술구조, 개발인력, 사용 소프트웨어에 제약조건을 가하므로 매우 신중해야 합니다.



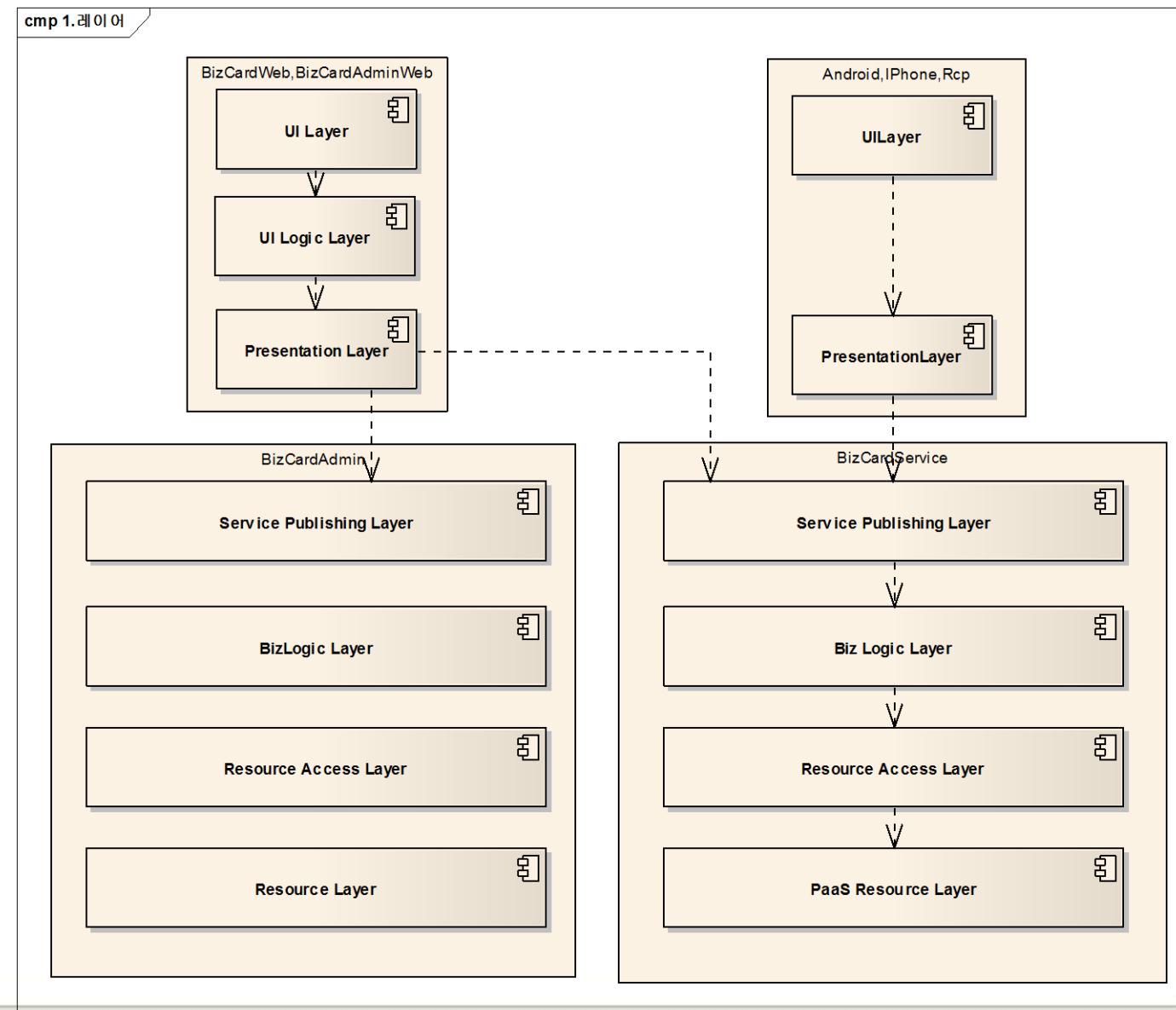
# 구조설계(1/2)

- ✓ 구조설계는 품질요구사항을 고려하지 않더라도 기본적으로 설계되어야 함



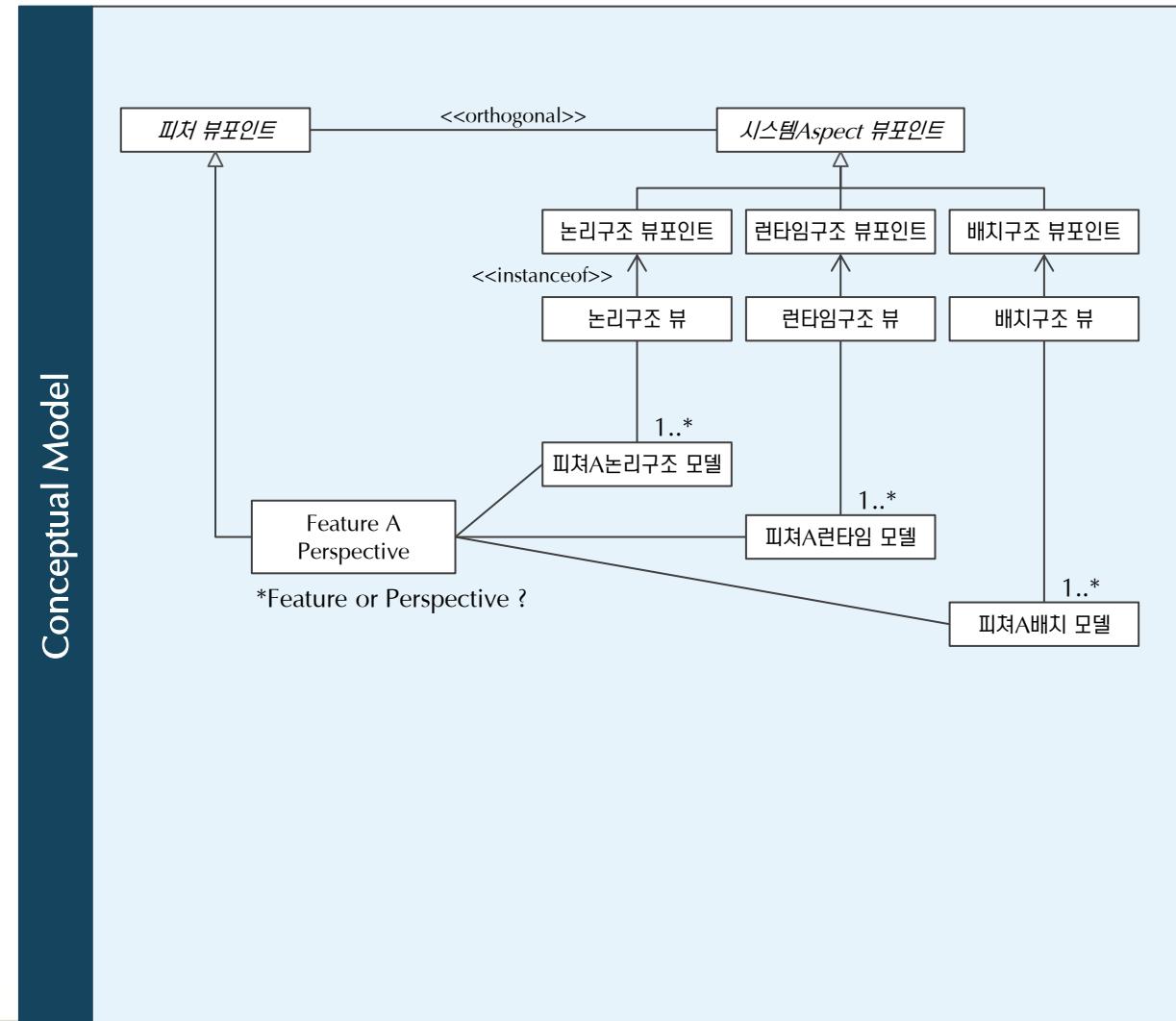
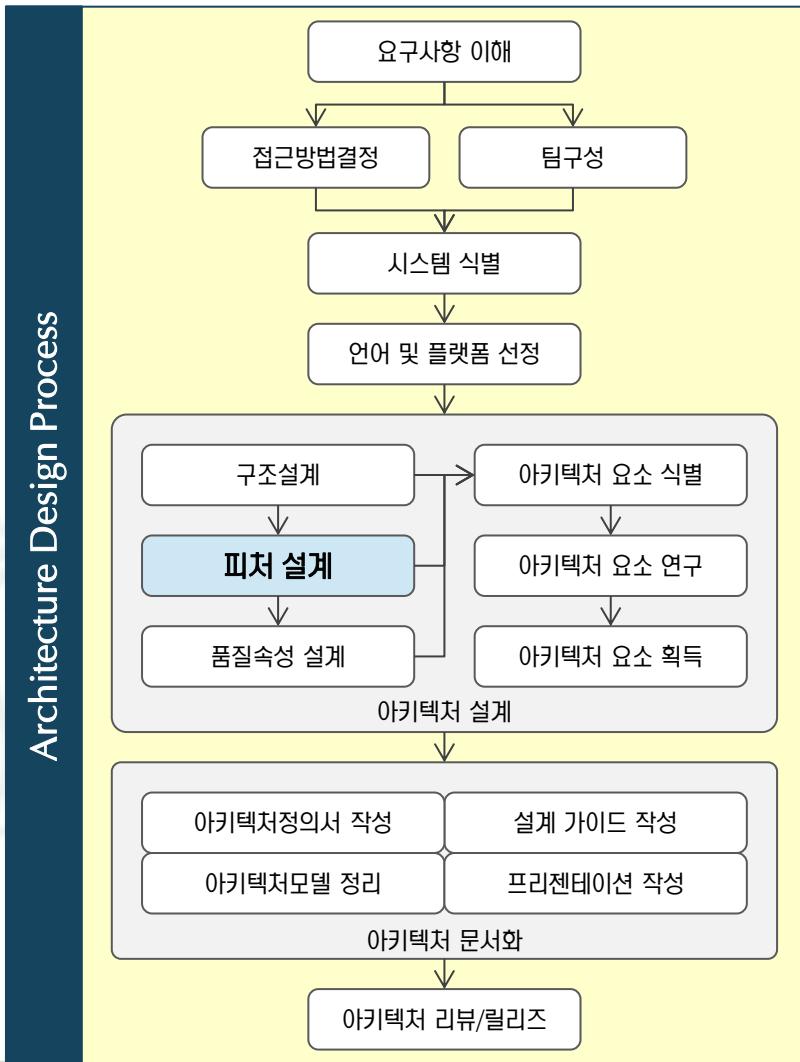
## 구조 설계(2/2)

- ✓ 구조설계의 첫 번째는 레이어 설계입니다.



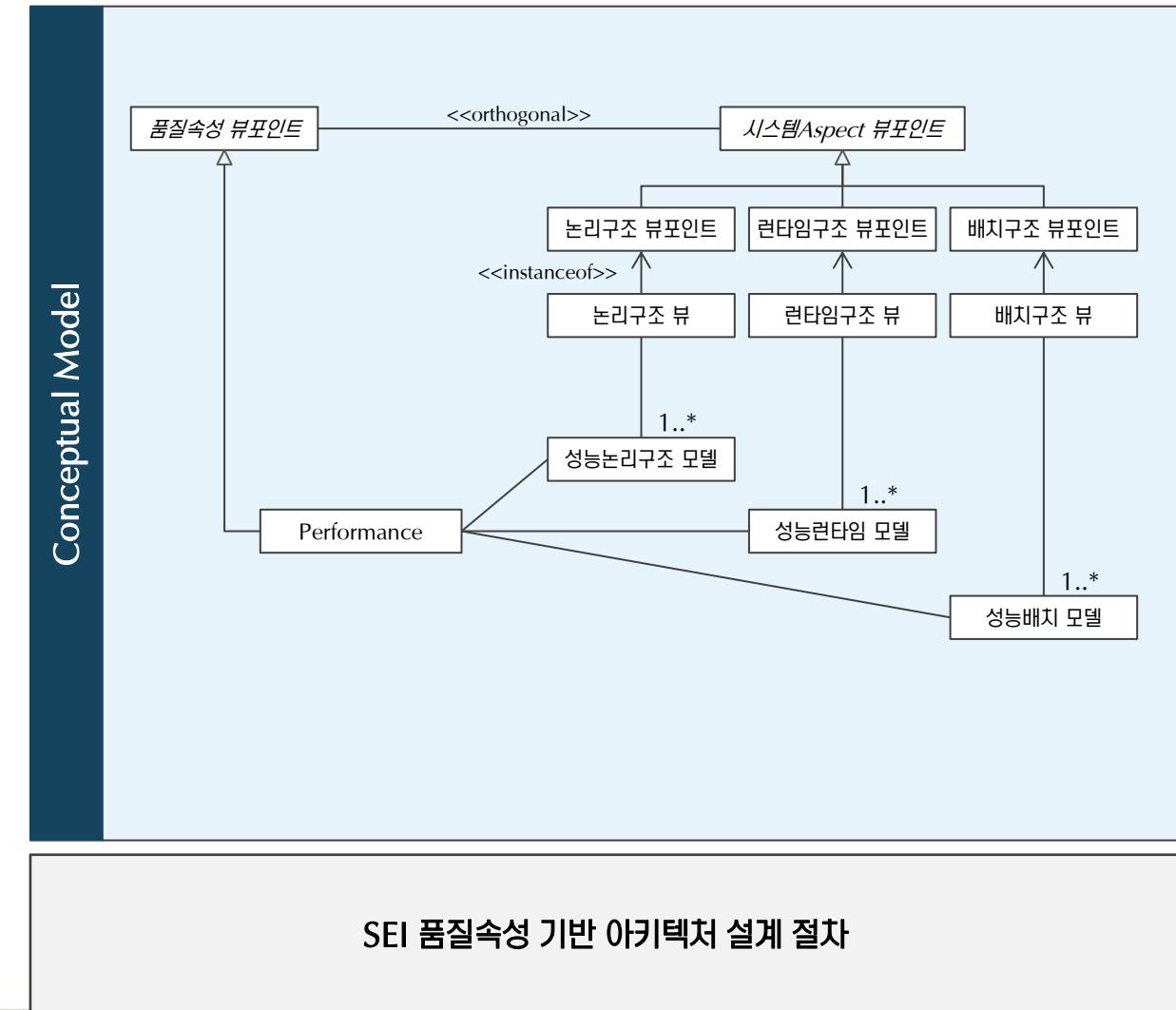
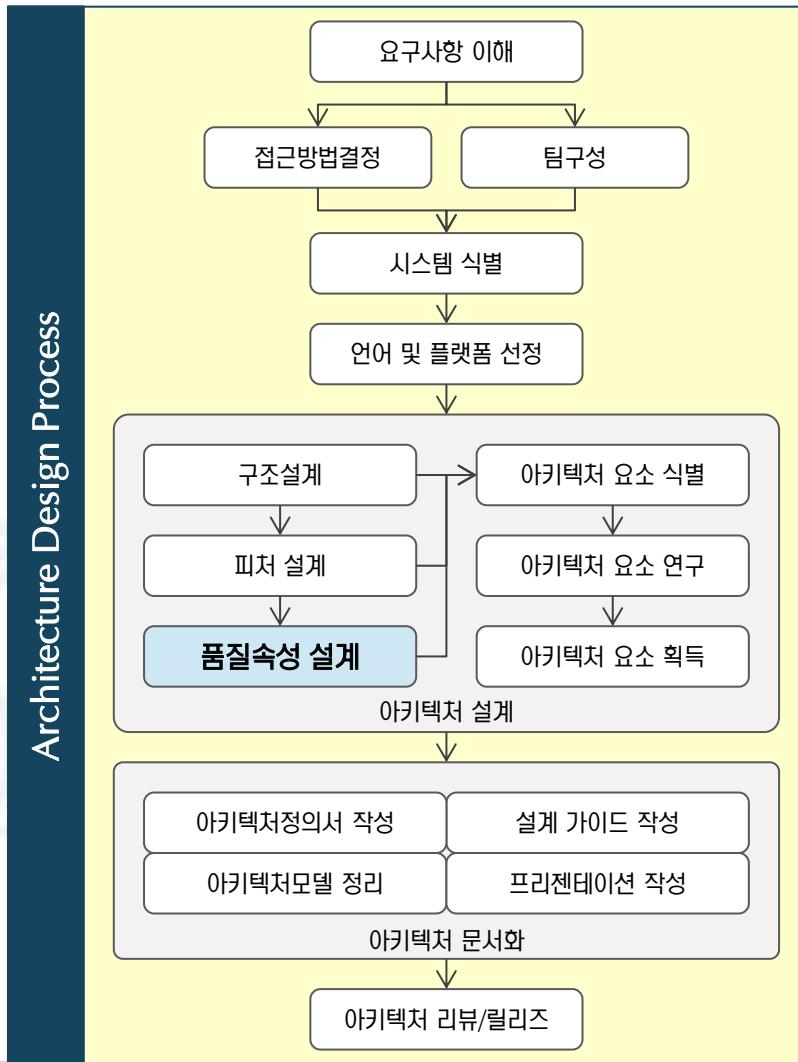
# 피처 설계

- ✓ 피처는 기능과 품질이 다양한 비율로 결합된 형태의 요구사항입니다.



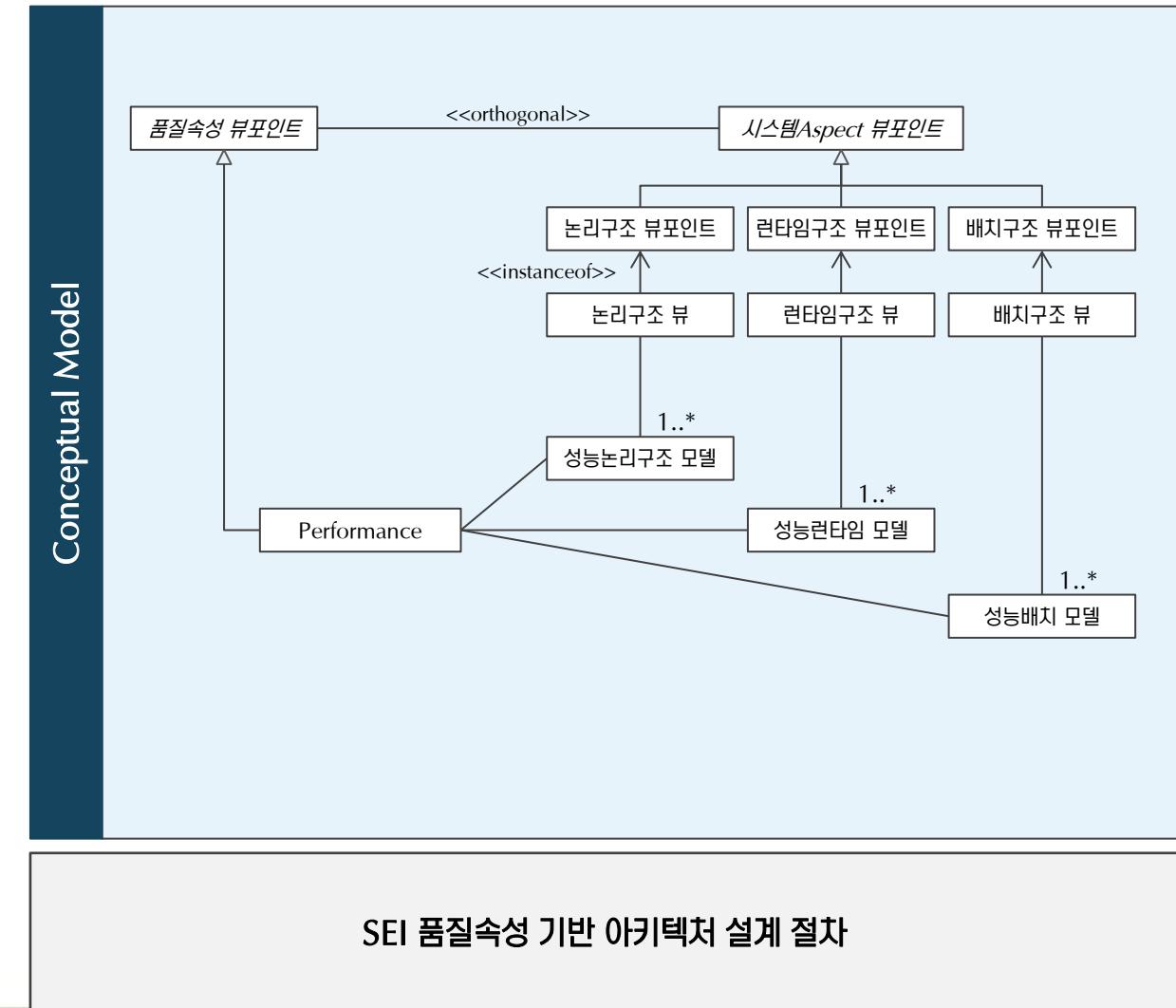
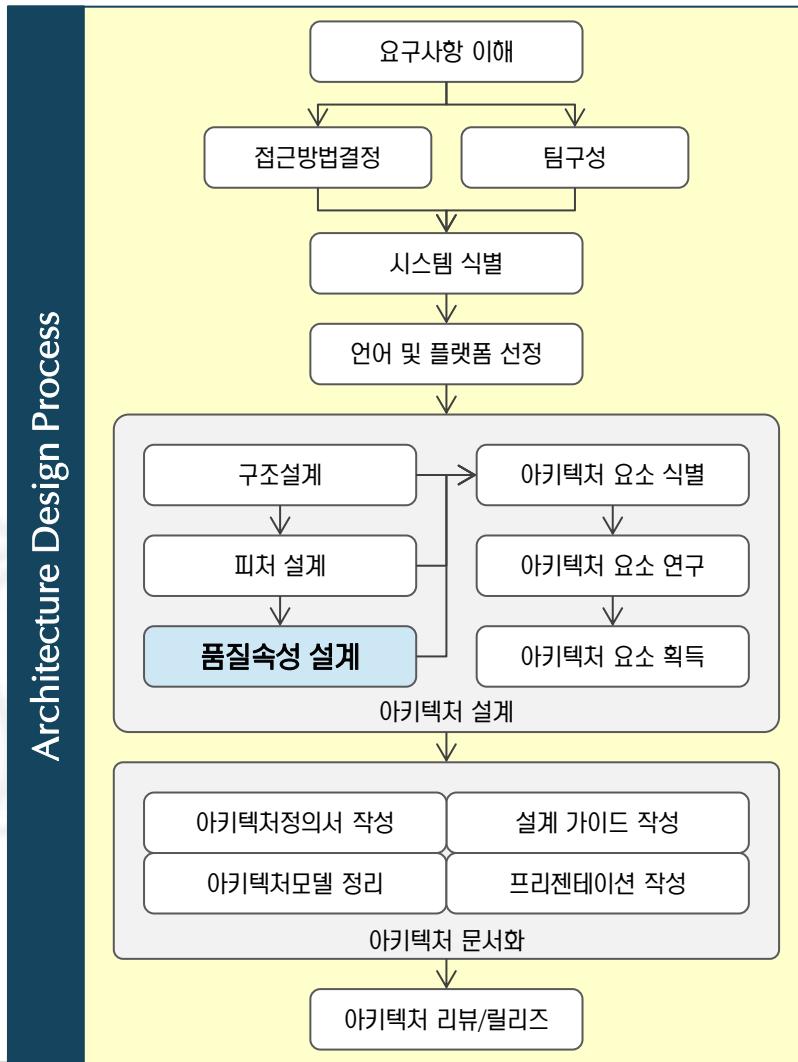
# 품질속성 설계(1/2)

- ✓ 품질속성은 조직이나 프로젝트 별로 서로 다른 세트를 선호하는 경향이 있음



# 품질속성 설계(2/2)

- ✓ 품질속성은 조직이나 프로젝트 별로 서로 다른 세트를 선호하는 경향이 있음



SEI 품질속성 기반 아키텍처 설계 절차

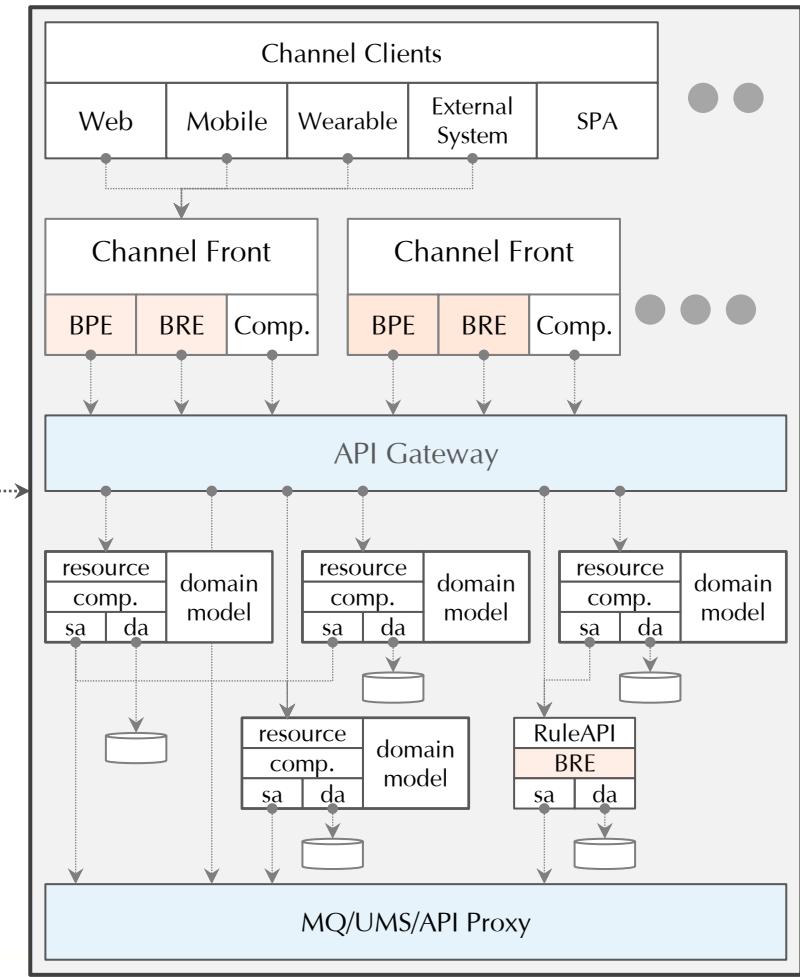
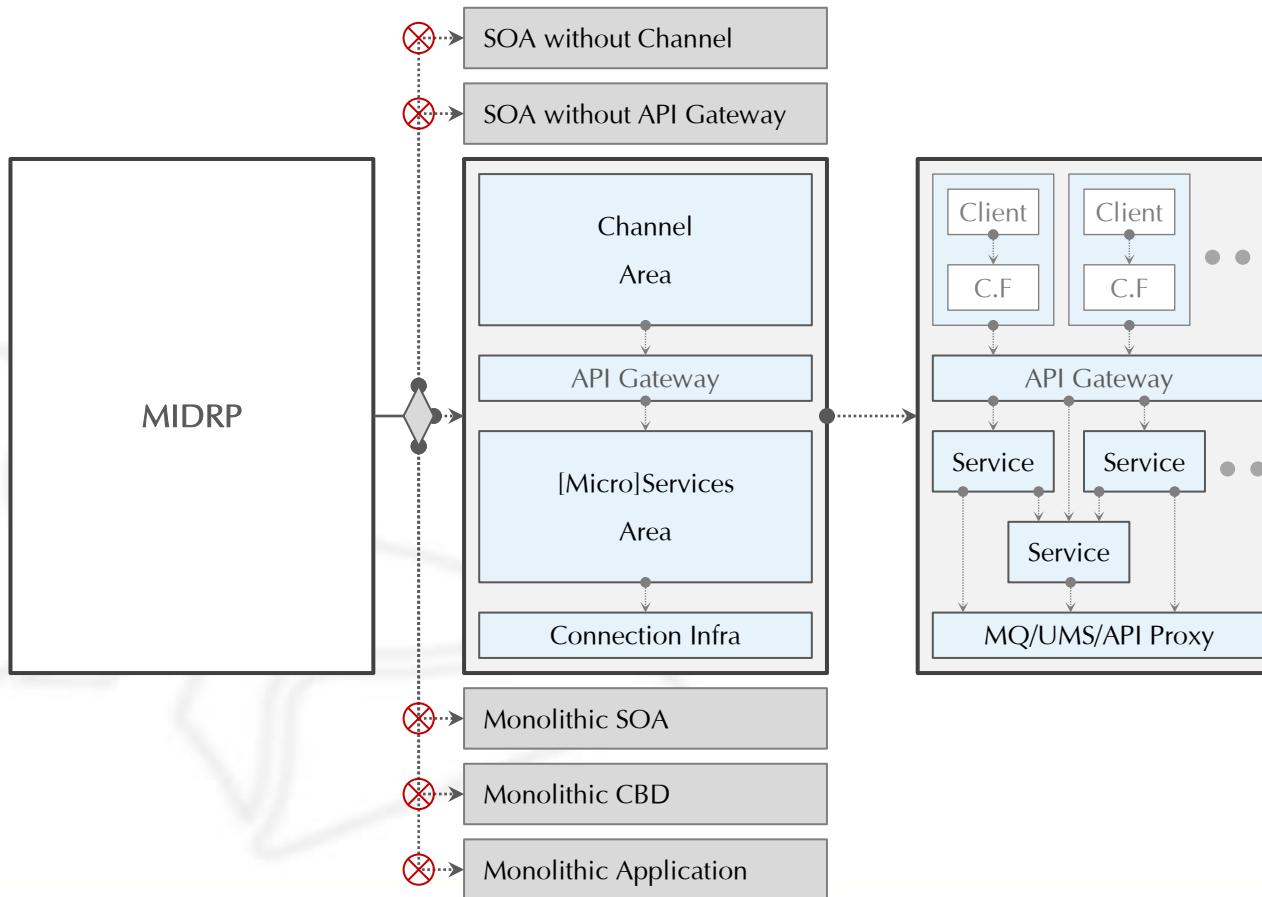
## 7. 아키텍처 설계

- ❖ 컨테이너 관점의 시스템 설계를 이해합니다.
- ❖ 정보 시스템의 본질과 그에 따른 레이어 구조를 이해합니다.
- ❖ 레이어드가 되어 가는 과정을 이해합니다.
- ❖ 레이어와 티어의 차이를 이해합니다.
- ❖ 서브시스템/서비스 유형 별 레이어 설계 유형을 알아 봅니다.

- ✓ 컨테이너 설계 개요
- ✓ 정보시스템 이해
- ✓ 레이어드
- ✓ 레이어와 티어
- ✓ 서브시스템과 레이어

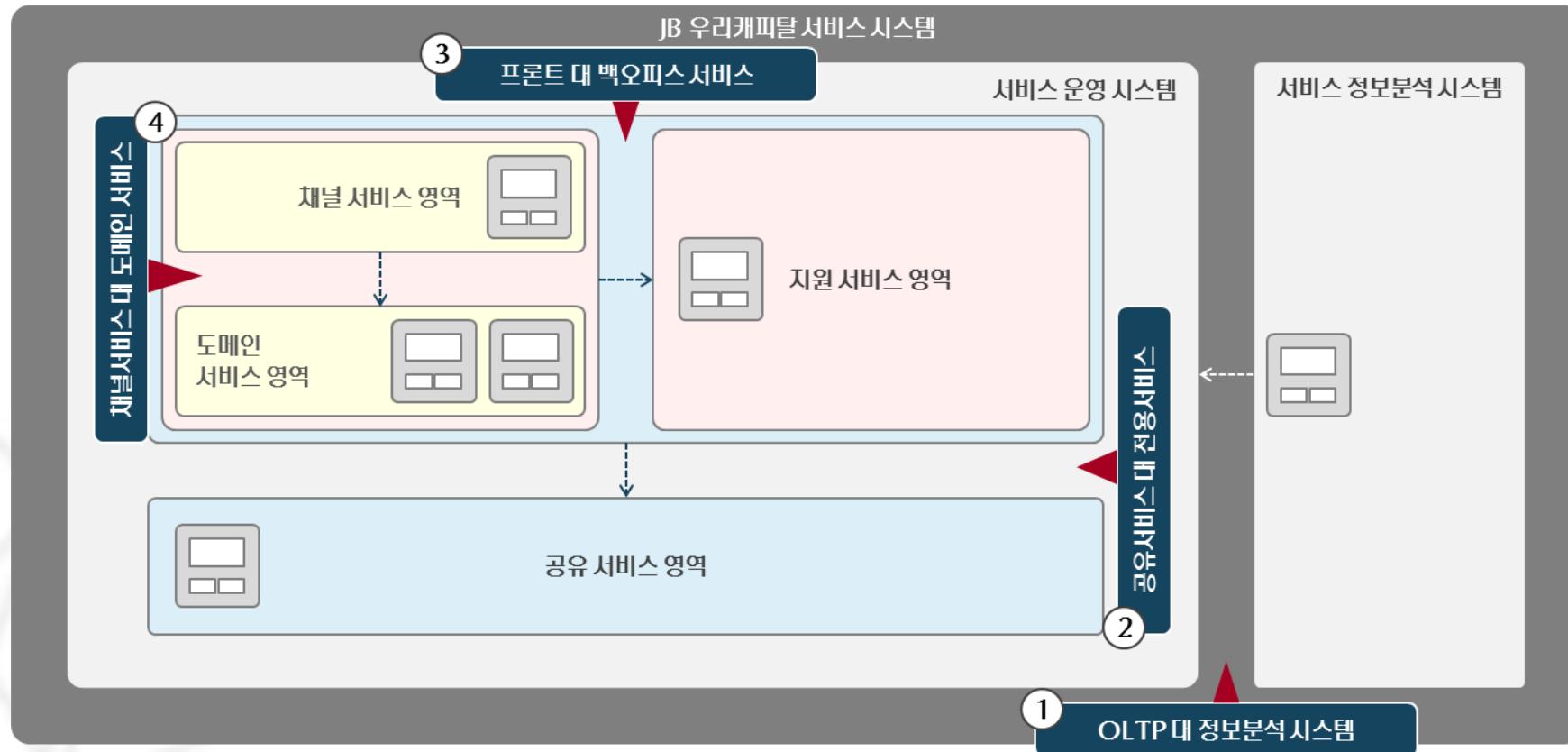
# 컨테이너 설계 개요

- ✓ 시스템을 컨테이너와 컨텐츠라는 관점에서 바라 보았을 때,
- ✓ 컨텐츠 설계는 컴포넌트와 서비스 설계로 접근을 합니다.
- ✓ 컨테이너 설계는 어떻게 할까요? 컨텐츠 설계와는 어떤 차이가 있을까요?



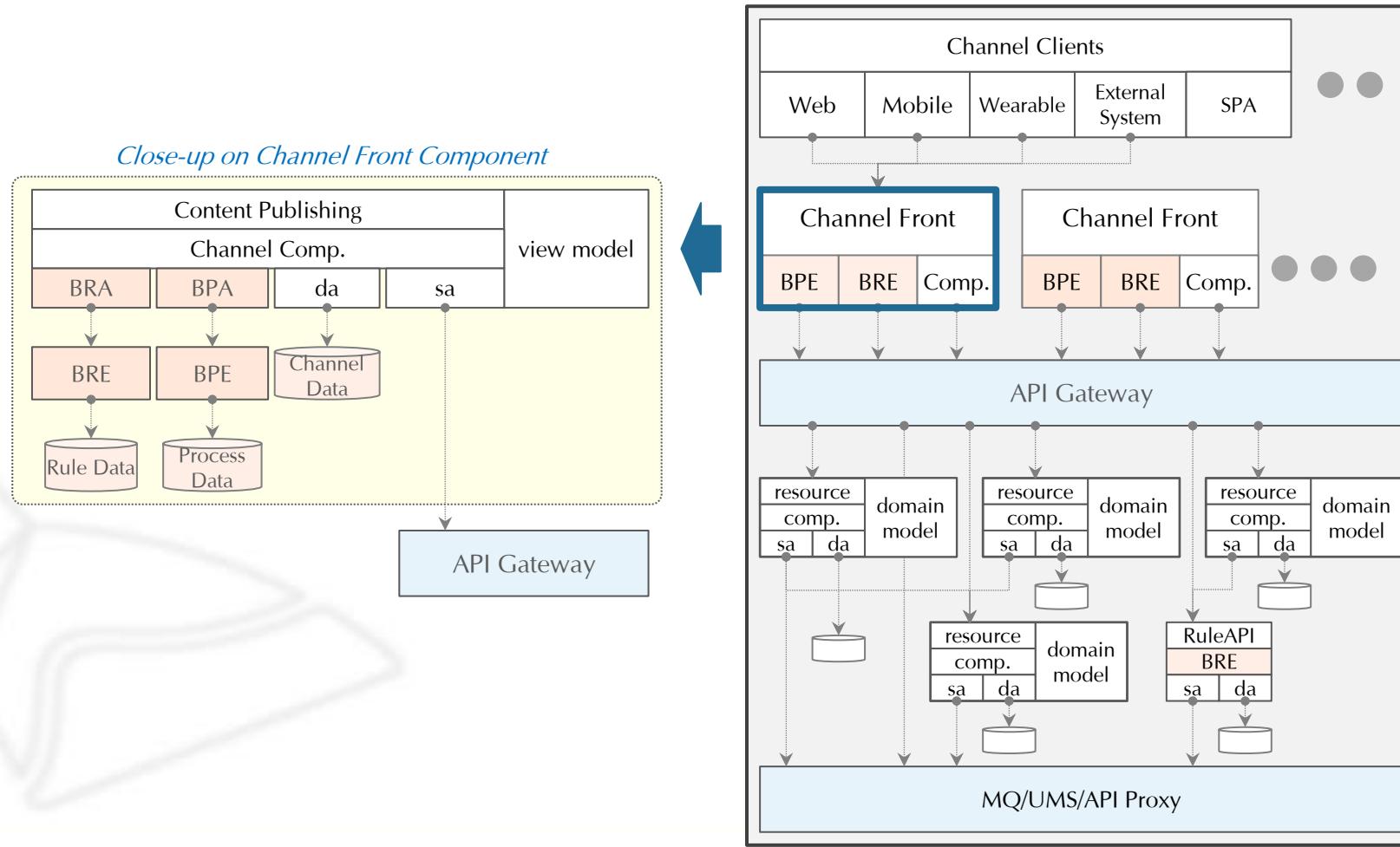
# 컨테이너 설계 개요

- ✓ 시스템을 컨테이너와 컨텐츠라는 관점에서 바라 보았을 때,
- ✓ 컨텐츠 설계는 컴포넌트와 서비스 설계로 접근을 합니다.
- ✓ 컨테이너 설계는 어떻게 할까요? 컨텐츠 설계와는 어떤 차이가 있을까요?



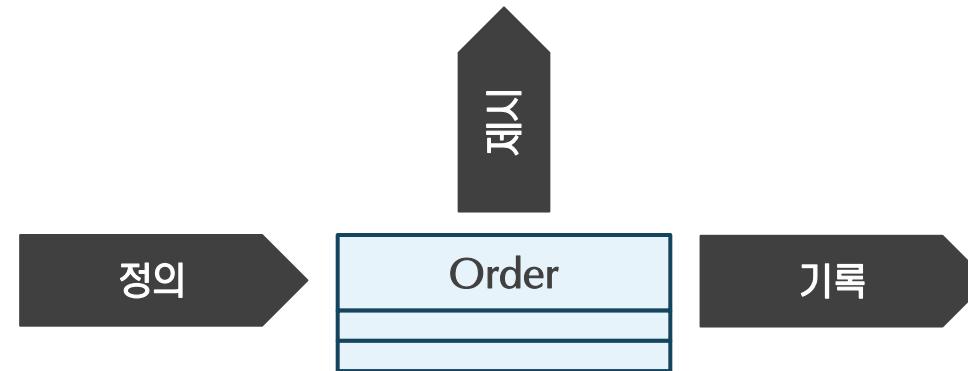
# 컨테이너 설계 개요

- ✓ 컨테이너를 구성하는 요소는 어떤 것이 있을까요?
- ✓ 컨테이너 관점에서 보면 컨텐츠인 서비스도 컨테이너를 구성하는 하나의 요소입니다.



# 정보 시스템 이해(1/3)

- ✓ 아키텍트 관점에서 정보의 정의는 무엇일까요?
- ✓ 정보에 대한 충분한 개념, 영속성을 갖기 위한 기록, 유용성을 갖기 위한 제시, 세 가지를 생각할 수 있습니다.
- ✓ 이런 정보의 특성에 대한 이해는 정보 시스템을 설계하는데 도움이 됩니다.
- ✓ 정보 관리 시스템은 이 세 가지 특성을 기반으로 설계합니다.

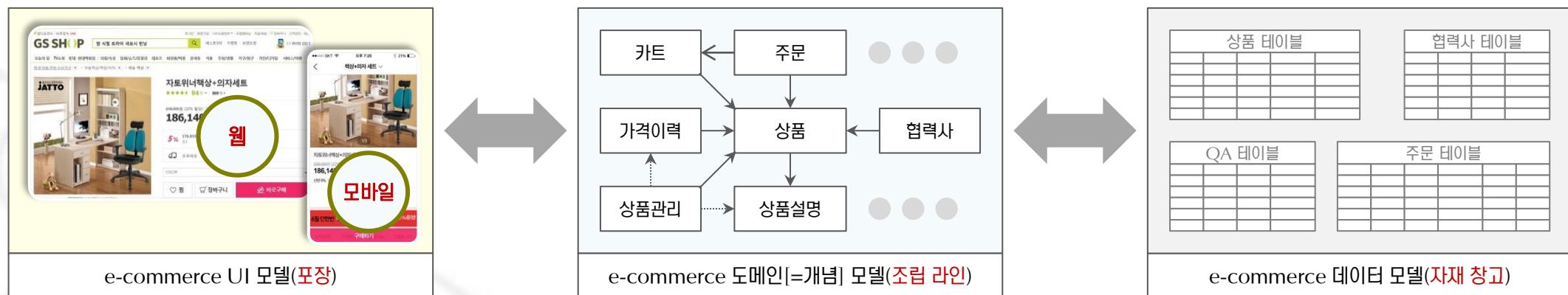


정보란 개념이 정의되어야 하고, 영속성을 가져야 하고, 필요할 때 제시할 수 있어야 한다.

# 정보 시스템 이해(2/3)

- ✓ 정보 시스템을 구축할 때, 개념을 정의(도메인 모델링)하고 컴포넌트를 개발하고, 데이터와 UI 부분을 개발합니다.
- ✓ 하지만, 이러한 정보의 특성에 대해 서로 다른 생각을 가지고 시스템 구축에 접근하는 경우도 있습니다.
- ✓ 데이터를 중심으로 설계하는 경우도 있고(1995 ~ 2000, 정보 공학), UI를 중심으로 설계하는 경우도 있습니다.
- ✓ 업무가 복잡할수록 올바른 개념 정의가 중요하게 되었고, 그 결과 DDD(Domain Driven Design)가 주류가 되었습니다.

정보 시스템의 고민 1 : 상품, 주문 등의 개념을 세 가지 서로 다른 방식으로 표현을 해야 합니다.

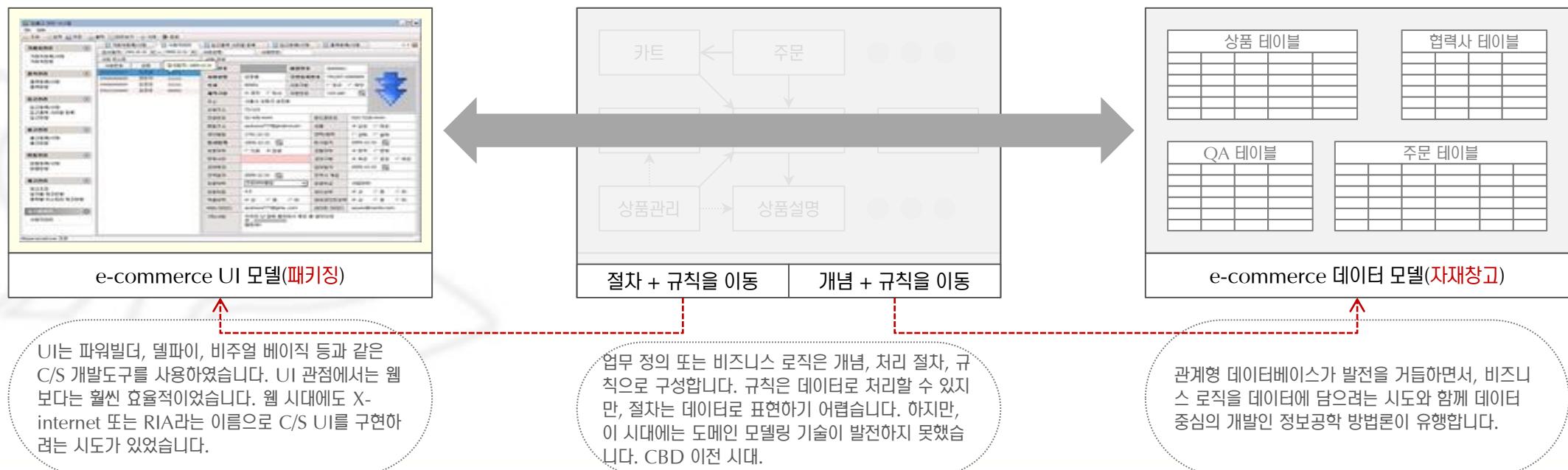


SW 설계 원칙: 관심사 분리(Separation of concern)

# 정보 시스템 이해(3/3)

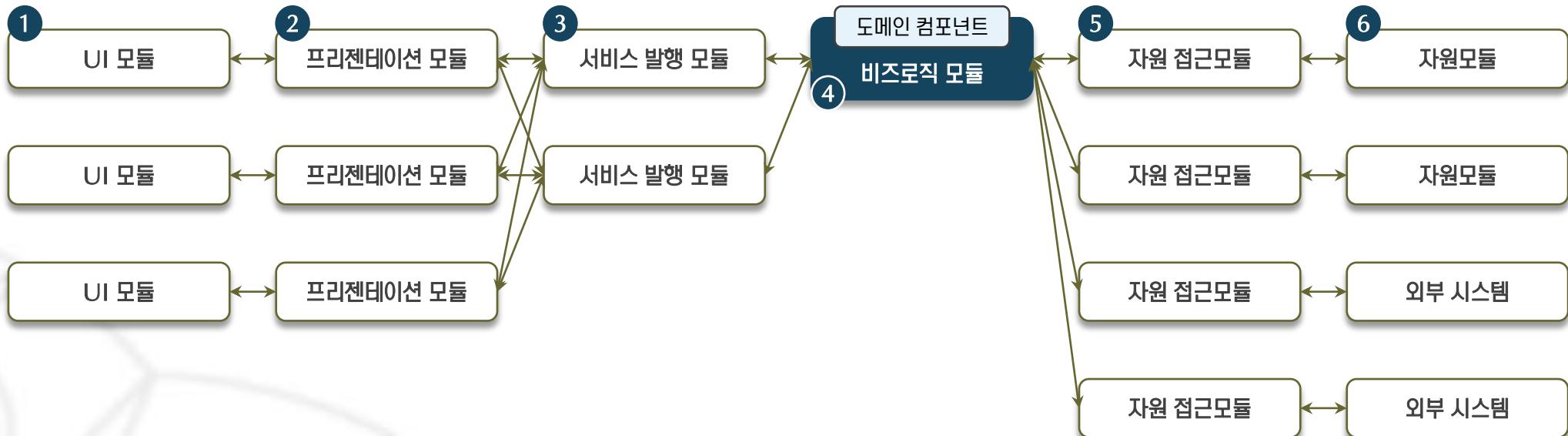
- ✓ 1990년대 중반부터 RDBMS가 널리 사용되면서 데이터의 중요성이 강조하는 개발 접근방법이 유행하였습니다.
- ✓ 그 시대에는 데이터를 중심으로 하는 개발 접근 방법인 정보 공학(Information Engineering) 방법이 주류가 되었습니다.
- ✓ 개발의 시작은 데이터 모델링으로 출발하고, 이후 UI 개발 후 연결하는 방식입니다.
- ✓ 아키텍처 관점에서는 2 티어 구조를 가지는 클라이언트-서버(사실상 DB 서버) 개발 방식입니다.

복잡한 정보를 담을 수 있지만, 잦은 변경을 처리하기에는 너무나 많은 노력이 필요하여, 비즈니스 변화를 제 때에 지원하지 못하면서 한계를 드러냅니다. ← 빅뱅 프로젝트의 원인



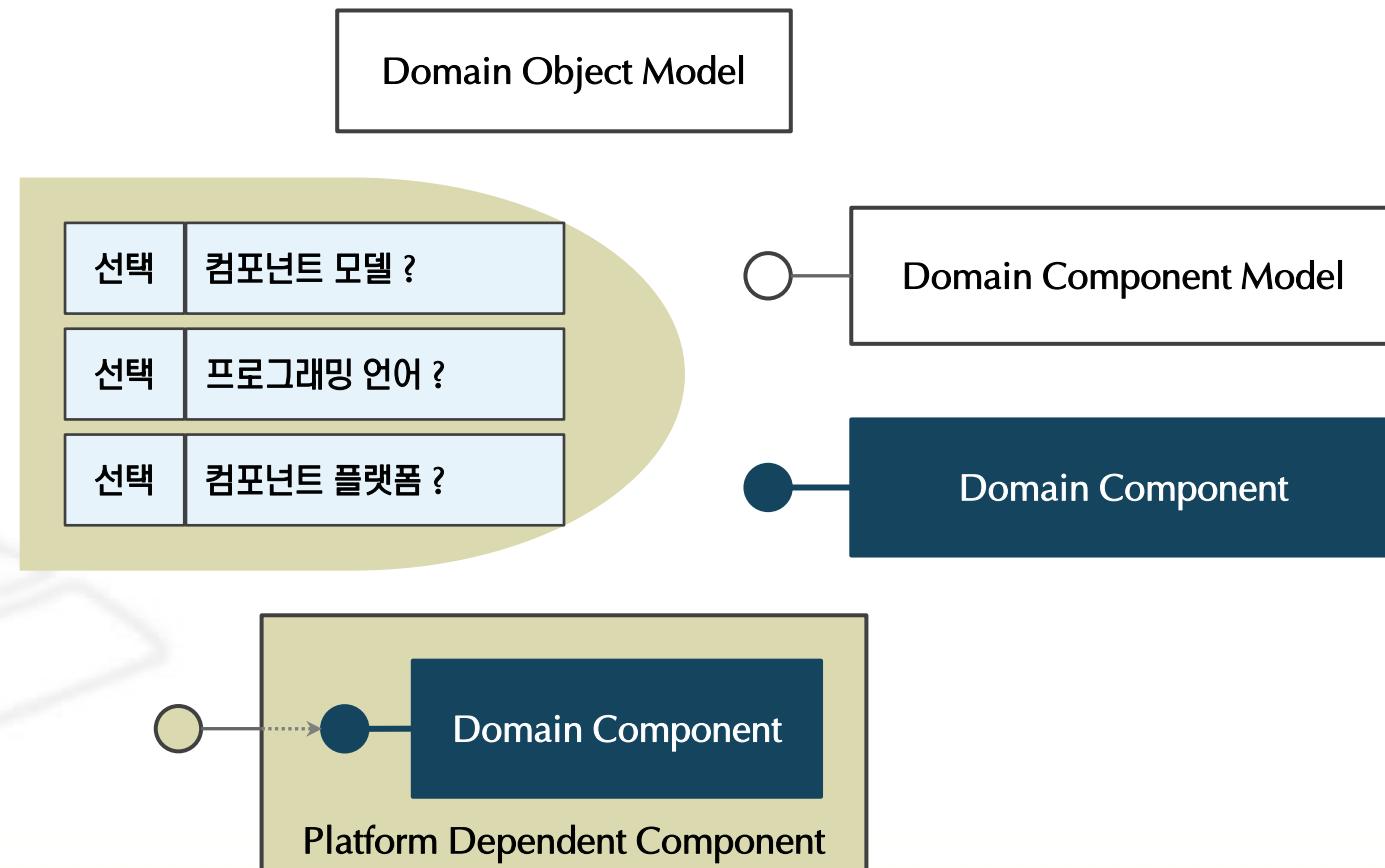
# 레이어드(1/14): 개요

- ✓ 잘 설계된 애플리케이션은 Loose coupling, High cohesion 원칙을 잘 지켰을 겁니다.
- ✓ 서로 다른 시스템 구성 요소인 도메인과 기술이 잘 분리되었을 겁니다. 분리와 결합이 자유롭습니다.
- ✓ 기술 요소의 변화는 레이어를 기준으로 자유로우며, 도메인은 기술의 변화에 영향을 받지 않습니다.
- ✓ 도메인 컴포넌트는 각 레이어 기술 요소로부터 중립적이며, 의존성을 갖지 않습니다.



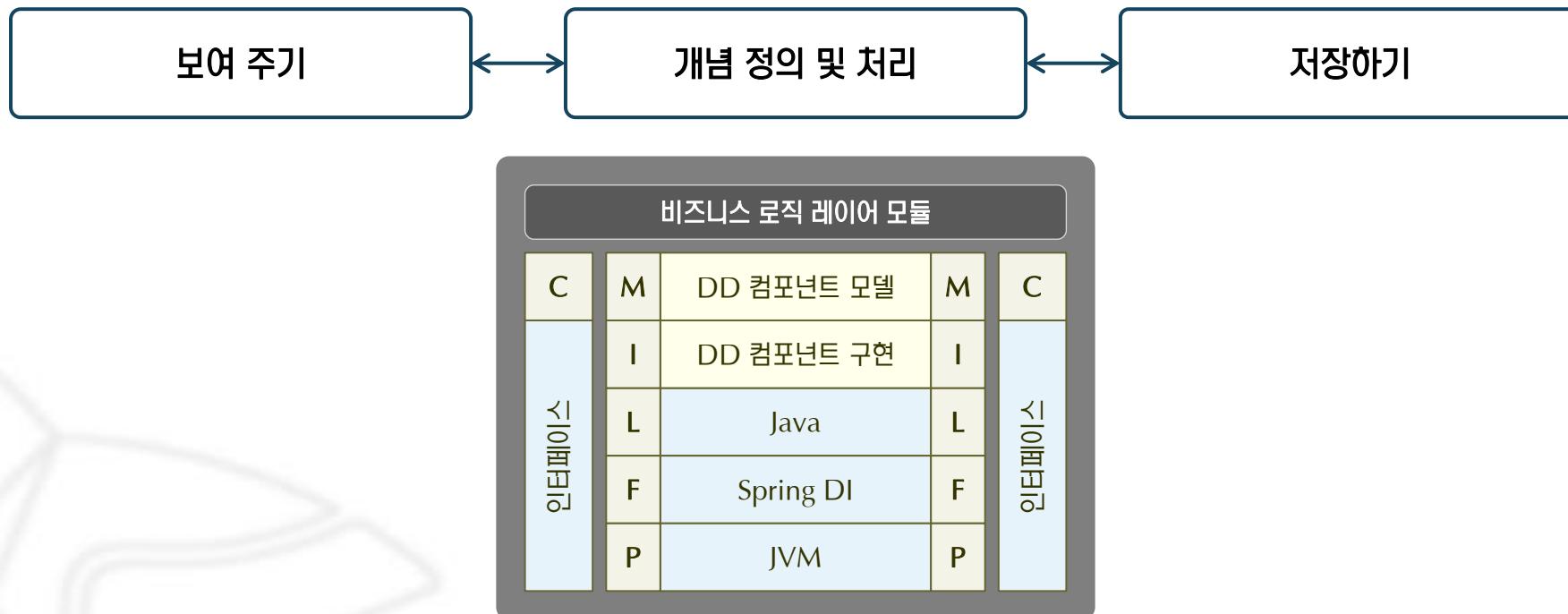
## 레이어드(2/14): 컴포넌트 구조 설계

- ✓ 도메인 객체 모델을 담을 컴포넌트 모델을 어떤 구조로 할 것인지 선택해야 합니다.
- ✓ 목표 시스템이 어느 수준의 기술 독립성을 가져야 하는지, 팀원의 기술 수준 등에 따라 컴포넌트 모델을 결정합니다.
- ✓ 컴포넌트를 구현할 언어와 컴포넌트 플랫폼을 선택해야 합니다.
- ✓ 도메인 모델링과 도메인 컴포넌트 구현은 분석가가 할 수도 있습니다.



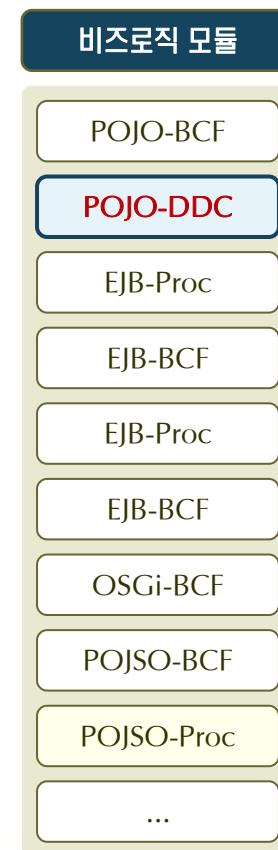
# 레이어드(3/14): 비즈니스 로직 레이어 모듈 1

- ✓ 도메인 컴포넌트를 감싸고 있는 플랫폼 종속적인 컴포넌트는 비즈니스 로직이 됩니다.
- ✓ 각 레이어 모듈에서 선택 가능한 요소는 대략 여섯 가지 정도입니다.
- ✓ 플랫폼, 프레임워크/라이브러리, 표현 언어, 사용자 정의 모델, 모델 구현체, 외부로의 연결 등은 모두 선택 사항입니다.



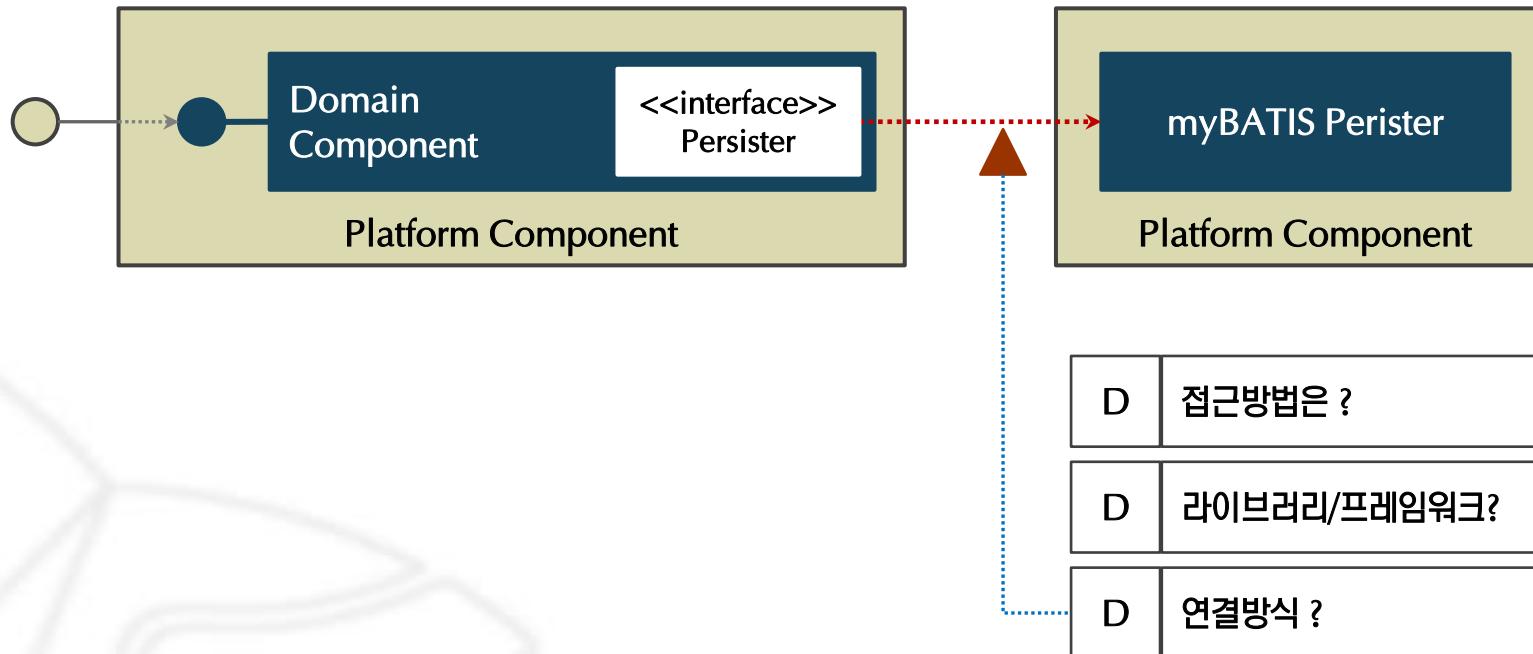
# 레이어드(4/14): 비즈니스 로직 레이어 모듈 2

- ✓ 여섯 가지 선택 요소는 결국 레이어 모듈의 특징을 결정하게 합니다.
- ✓ 비즈니스 로직 레이어를 구성하는 모듈은 컴포넌트 모델과 플랫폼 선택에 따라 다음과 같이 다양할 수 있습니다.
- ✓ 우리는 POJO 기반의 Domain [Driven] Component를 선택했습니다.



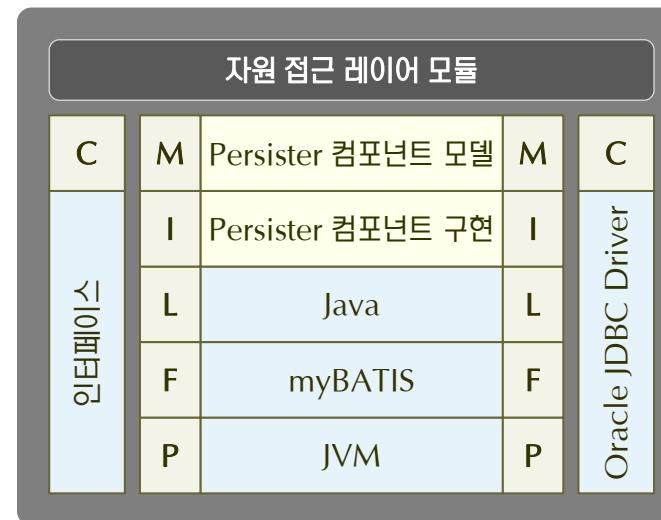
# 레이어드(5/14): 자원접근 레이어 모듈 1

- ✓ 다음은 플랫폼 컴포넌트로부터 자원에 접근하는 구조를 설계합니다.
- ✓ 자원 접근 영역이 완전히 투명해야 하는가 아니면 어느 정도의 의존성을 허용하는가에 대해 고민을 합니다.
- ✓ 어떤 경우든 자원 접근 모듈을 대체 가능하도록 설계하는 것이 필요합니다.



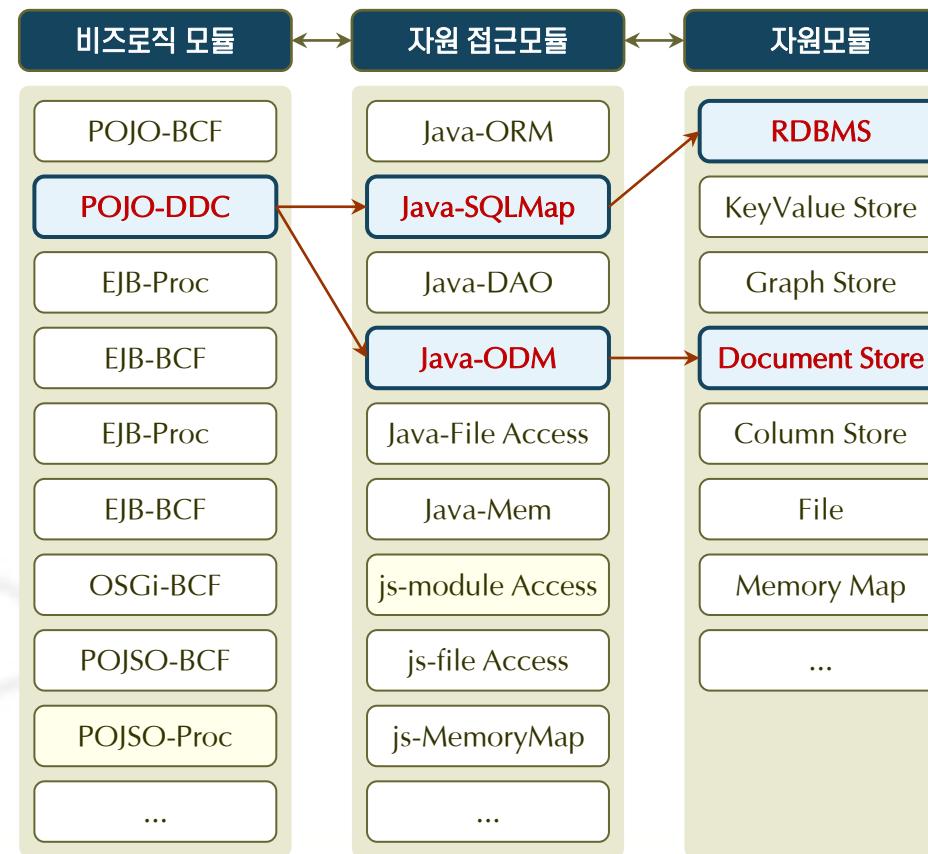
# 레이어드(6/14): 자원접근 레이어 모듈 2

- ✓ 고객의 요구와 시스템 환경 등을 고려하여 두 가지 접근 방법을 선택합니다.
- ✓ 읽기 쓰기는 오라클 DB로 읽기 전용은 MongoDB로 접근하도록 설계를 했습니다.
- ✓ 이 모듈은 동일한 Persister 인터페이스를 구현하므로, 필요할 경우 서로 대체할 수 있습니다.



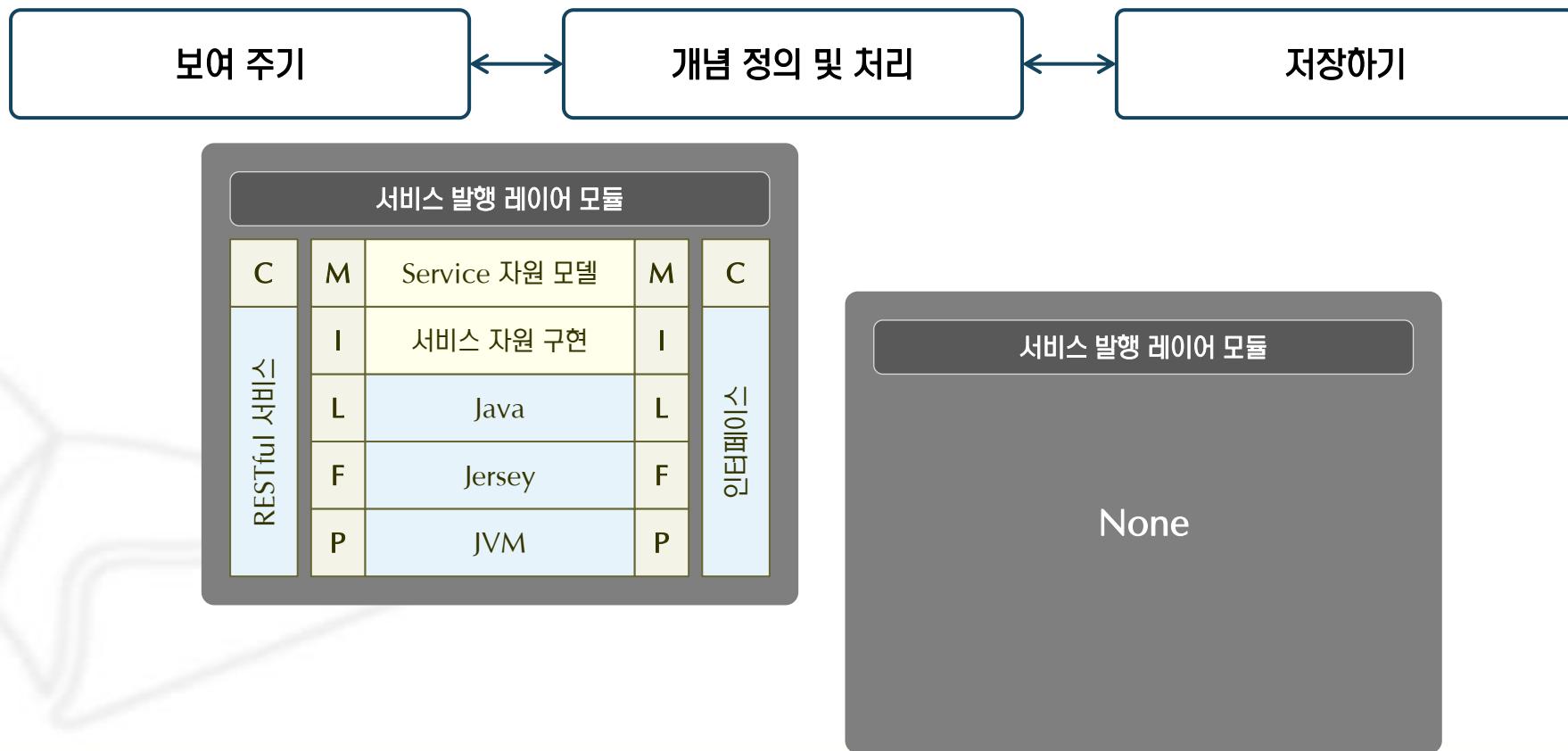
# 레이어드(7/14): 자원접근 레이어 모듈 3

- ✓ 자원 모듈과 자원 접근은 매우 밀접한 관계가 있으므로 함께 선택합니다.
- ✓ 선택 가능한 자원 접근 방법 중에 두 가지를 선택했습니다.
- ✓ 트랜잭션 등에 대한 고민은 틀을 완성할 때까지는 뒤로 미룹니다.



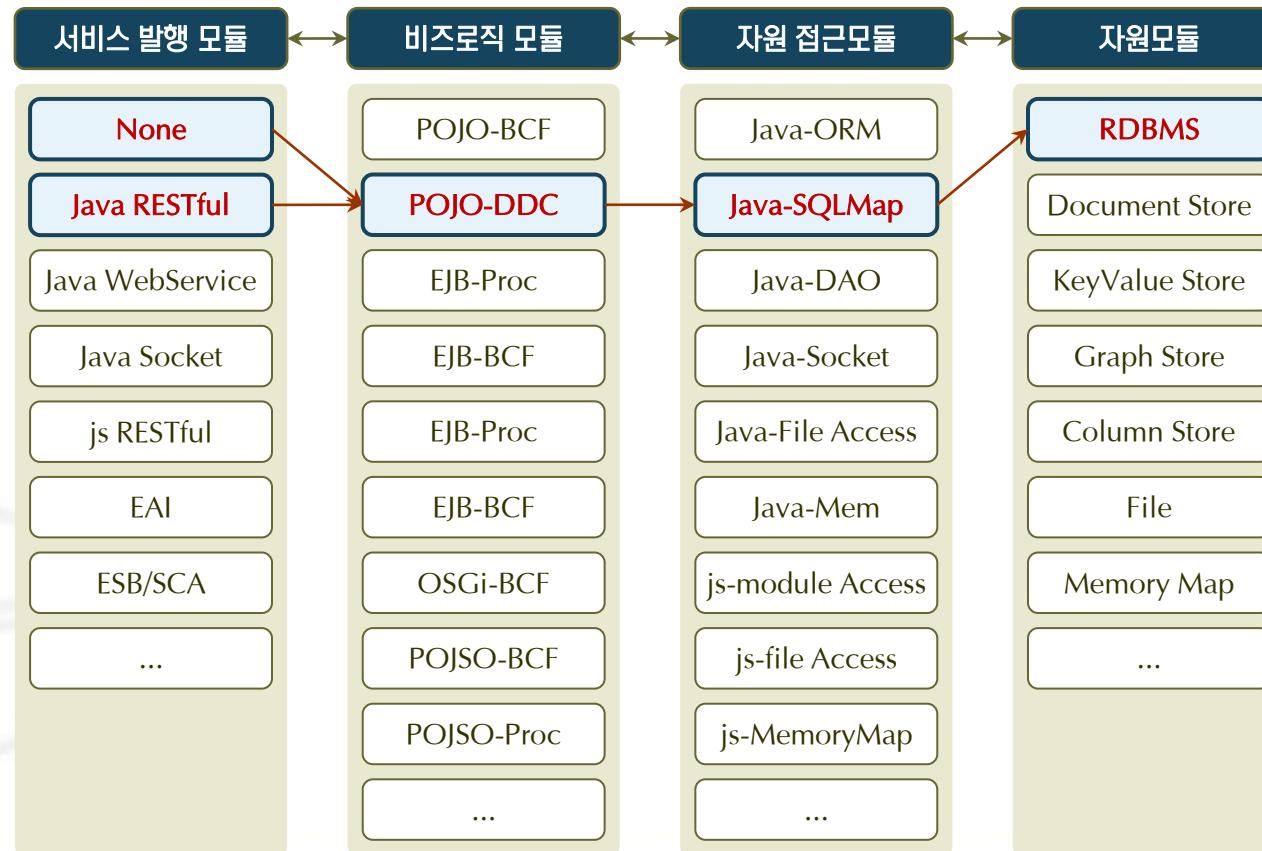
# 레이어드(8/14): 서비스 발행 레이어 모듈 1

- ✓ 정의한 정보나 처리된 결과 정보를 외부로 내보내는 레이어입니다.
- ✓ UI나 프리젠테이션 레이어로부터 직접 플랫폼 커포넌트에 접근한다면 레이어 모듈은 "None"입니다.
- ✓ 웹 페이지의 Javascript 모듈에서 REST 서비스를 이용하여 데이터를 가져가는 경로도 준비했습니다.



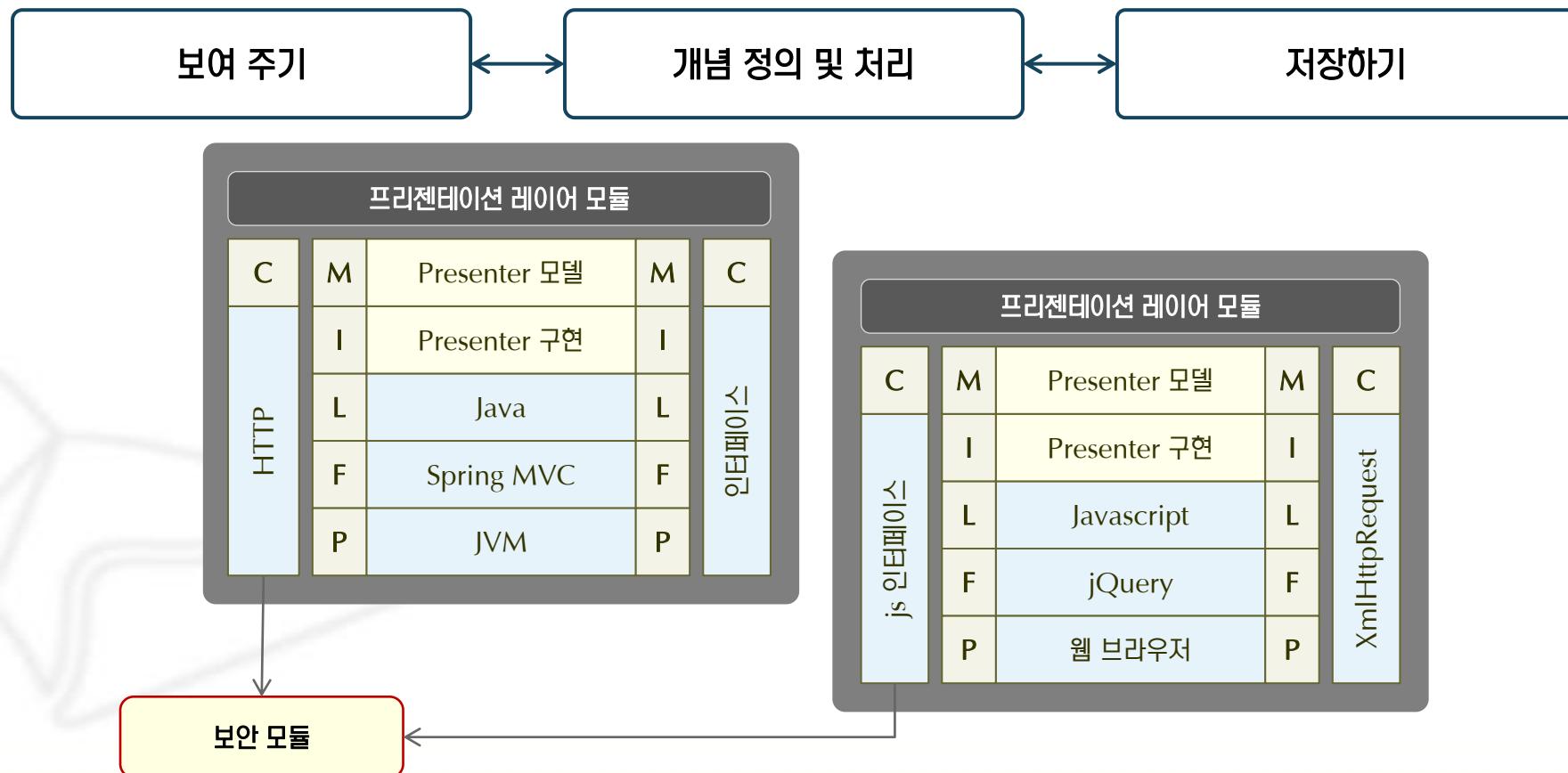
# 레이어드(9/14): 서비스 발행 레이어 모듈 2

- ✓ 프리젠테이션 모듈에서 컴포넌트로 직접 접근 경로와 RESTful 서비스로 접근하는 경로를 선택했습니다.
- ✓ 그 외 다양한 서비스 발행 방법이 있습니다.
- ✓ 요즘은 Node.js 기반의 Javascript RESTful 서비스 발행도 많이 선택되고 있습니다.



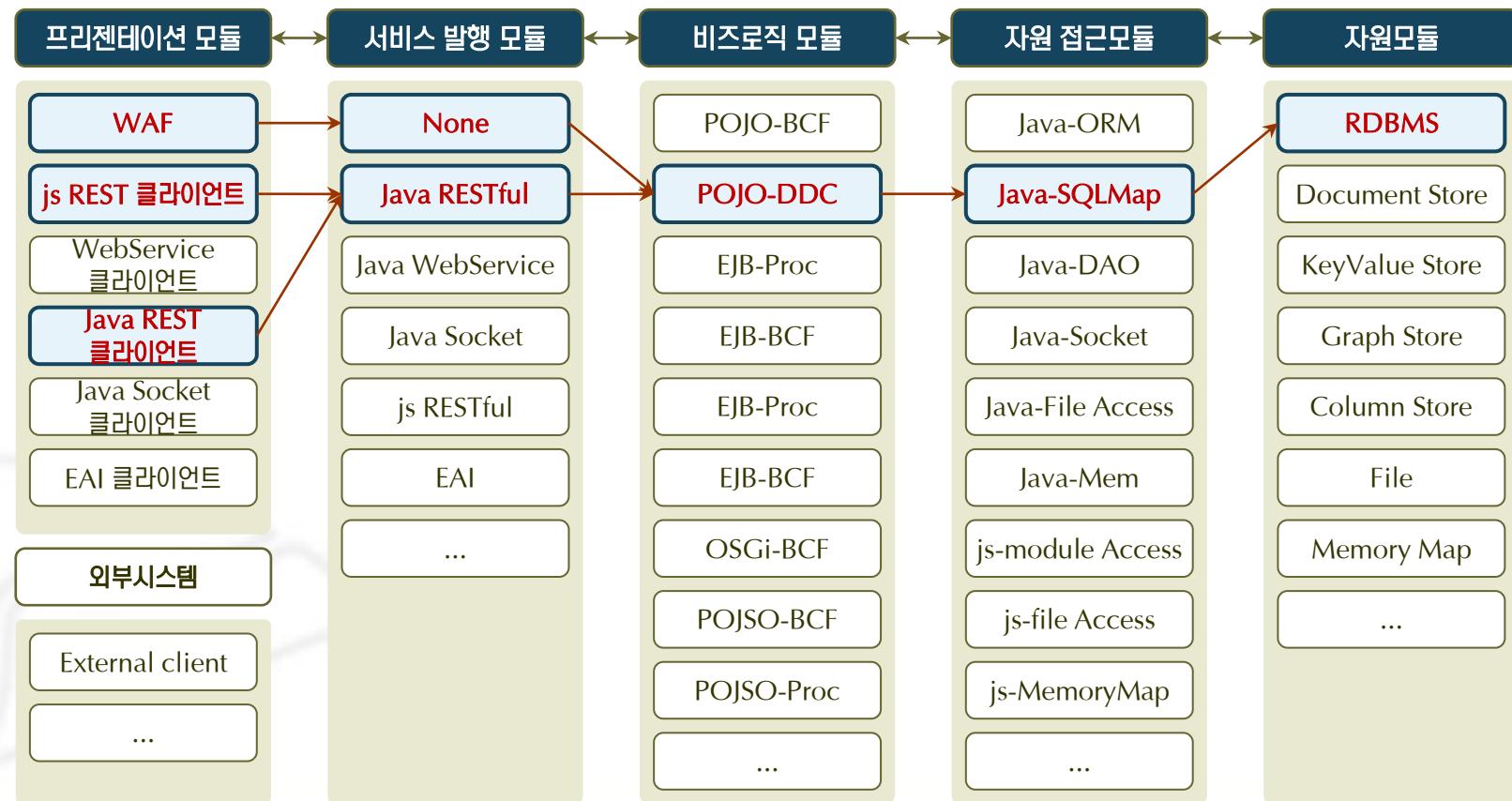
# 레이어드(10/14): 프리젠테이션 레이어 모듈 1

- ✓ 프리젠테이션 모듈은 두 가지를 선택하였습니다.
- ✓ Java와 Web Application Framework 위의 Presenter 모듈이 페이지 단위 UI 요청 처리를 합니다.
- ✓ 웹 브라우저에 내려가 있는 jQuery 모듈이 서버와 Ajax 통신을 하여 데이터를 가져 옵니다.



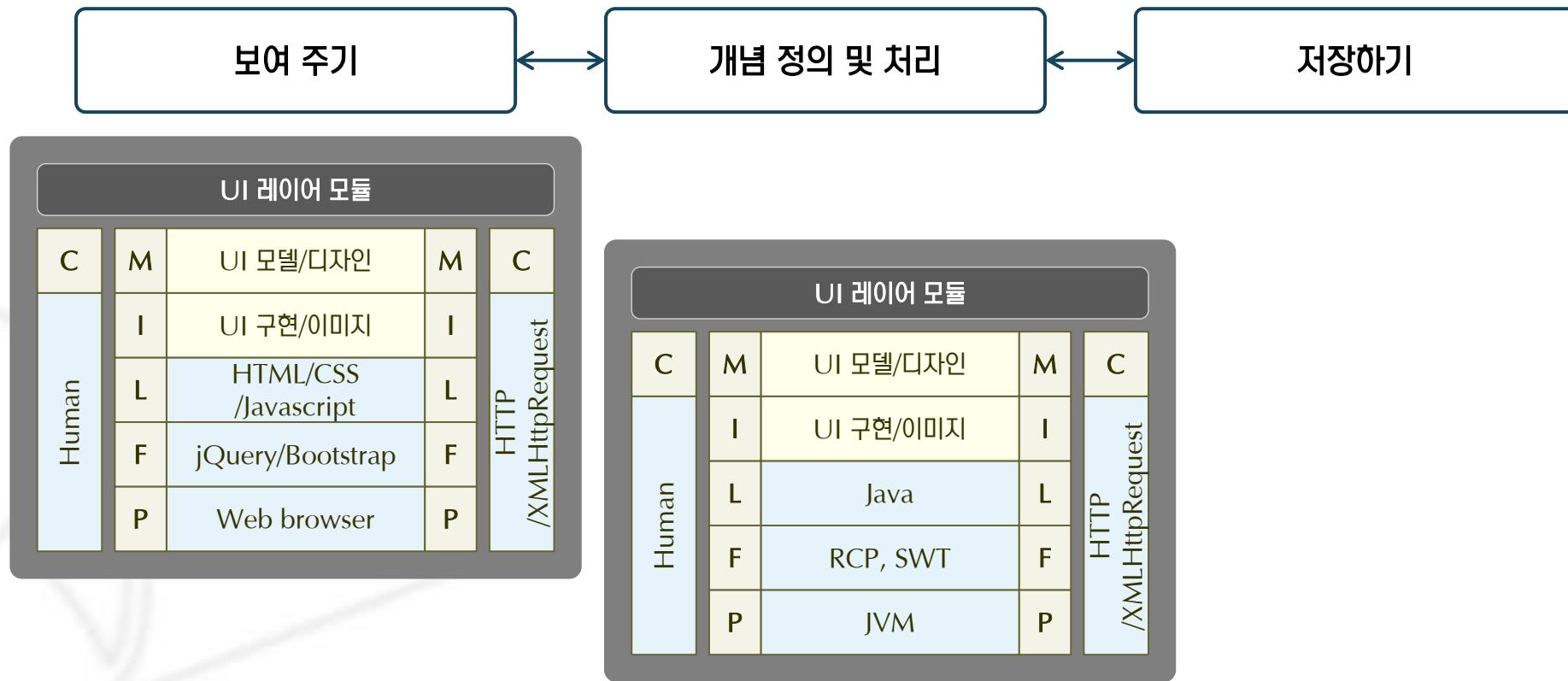
# 레이어드(11/14): 프리젠테이션 레이어 모듈 2

- ✓ 프리젠테이션 레이어는 UI 레이어와의 구분이 어려운 레이어입니다.
- ✓ 프리젠테이션 레이어에서는 두 개의 서로 다른 경로를 선택했습니다.
- ✓ WAF는 서버 사이드에서, Javascript RESTful 클라이언트는 웹 브라우저에서 동작합니다.



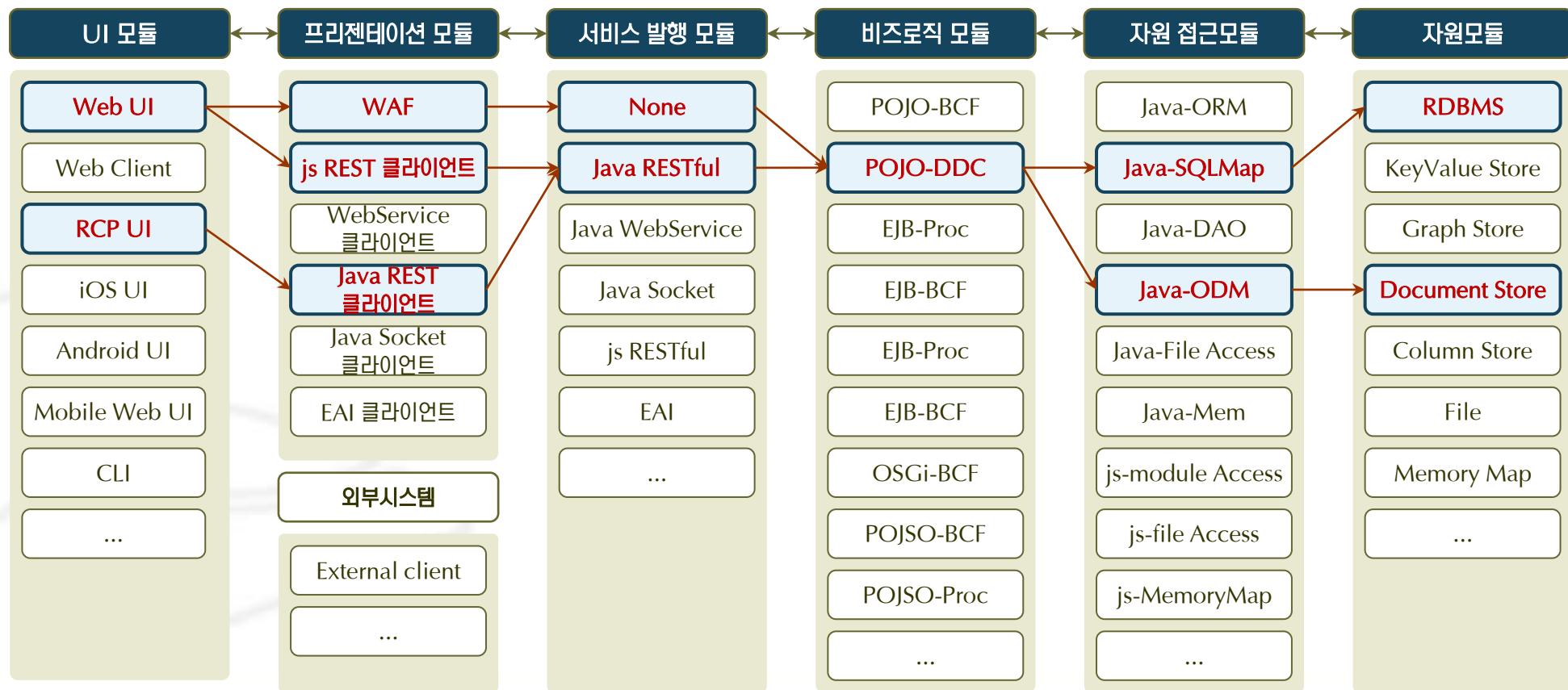
# 레이어드(12/14): UI 레이어 모듈 1

- ✓ UI 레이어를 위한 플랫폼으로 웹 브라우저와 JVM을 선택했습니다.
- ✓ 하나는 전형적인 웹 아키텍처에 AJAX 기능 보완한 구조를 가지고 있습니다.
- ✓ 또 다른 하나는 관리자를 위한 모니터링 기능을 제공하기 위해 eclipse RCP를 선택하였습니다.



# 레이어드(13/14): UI 레이어 모듈 2

- ✓ 다양한 UI 중에 두 가지를 선택하였습니다. 일반 사용자용 웹 UI와 관리자용 RCP UI입니다.
- ✓ 이제 여섯 개의 레이어의 모듈을 모두 [설계 | 선택] 했습니다.
- ✓ 서비스나 프리젠테이션 레이어에서 도메인 컴포넌트의 인터페이스를 사용할 것인가에 대한 선택이 남아 있습니다.
- ✓ 각 레이어 진입점(connector)에 필요한 장치들에 대한 선택이 남아 있습니다.



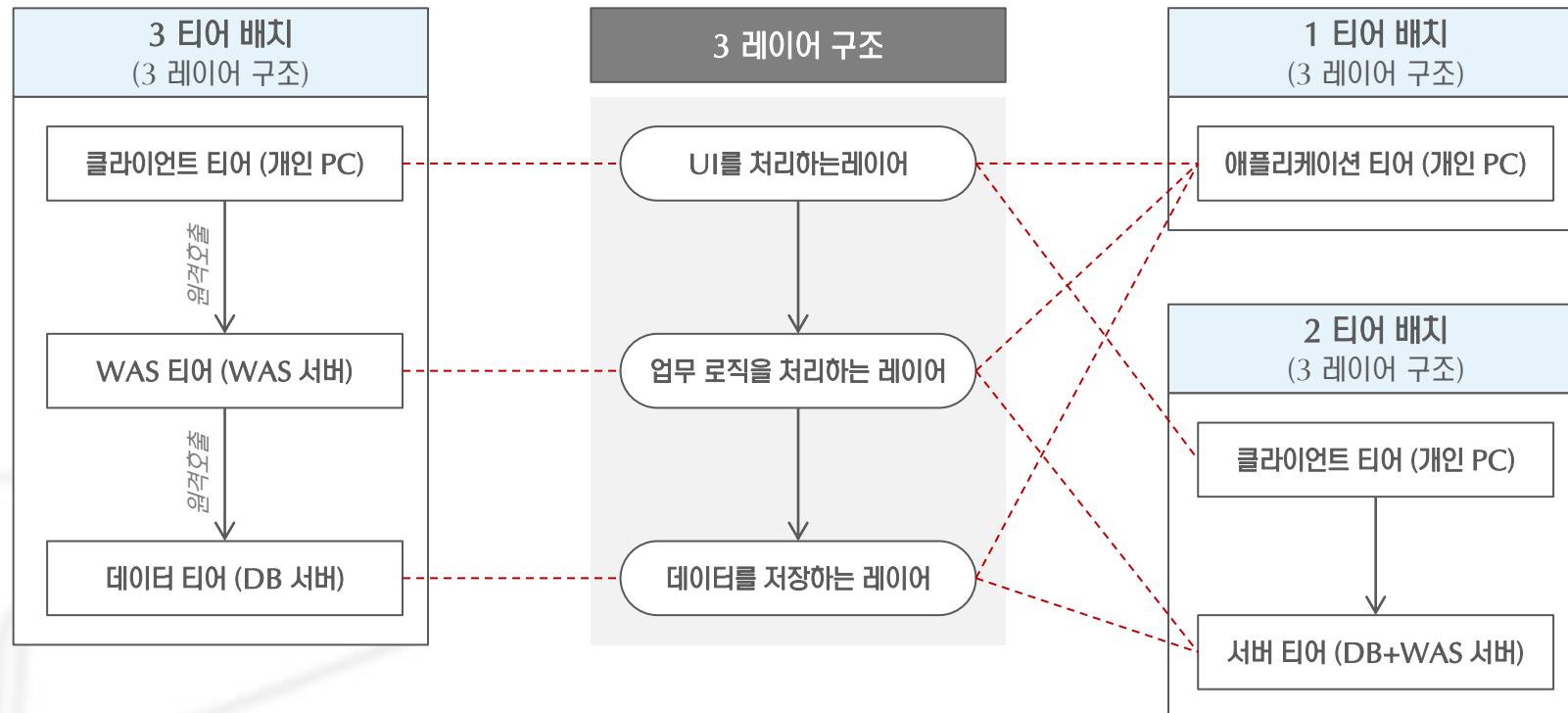
# 레이어드(14/14): 아키텍처 설계 회고

- ✓ 하지만, 우리는 레이어를 잘 분리해서 멋지게 설계한 시스템을 본적이 별로 없습니다.
- ✓ 체계적인 설계 역량이 부족하거나, 시간이 부족하거나, 관심이 부족하기 때문이겠죠.
- ✓ 아무렇게나 지어도 사람이 들어가서 살 수 있듯이, 아무렇게 개발해도 실행되기 때문입니다.
- ✓ 우리는 아키텍처 설계에 대한 고민을 많이 하지 않았으며, 그 결과로 유연성 없는 불량 시스템을 양산하고 있습니다.



# 레이어와 티어(1/4)

- ✓ 티어는 레이어와 비슷하면서도 다른 개념이므로 많은 사람들이 혼용해서 사용합니다.
- ✓ 레이어는 시스템의 논리 구조를 표현하고, 티어는 물리적인 서버를 표현합니다.
- ✓ 3레이어 구조로 설계한 시스템을 3티어, 2티어, 1티어로 배치(deployment)할 수 있습니다.
- ✓ 레이어 구성에 따라 레이어와 티어와의 관계를 설정할 수 있습니다. 아래 3 레이어 설계의 예를 살펴봅니다.



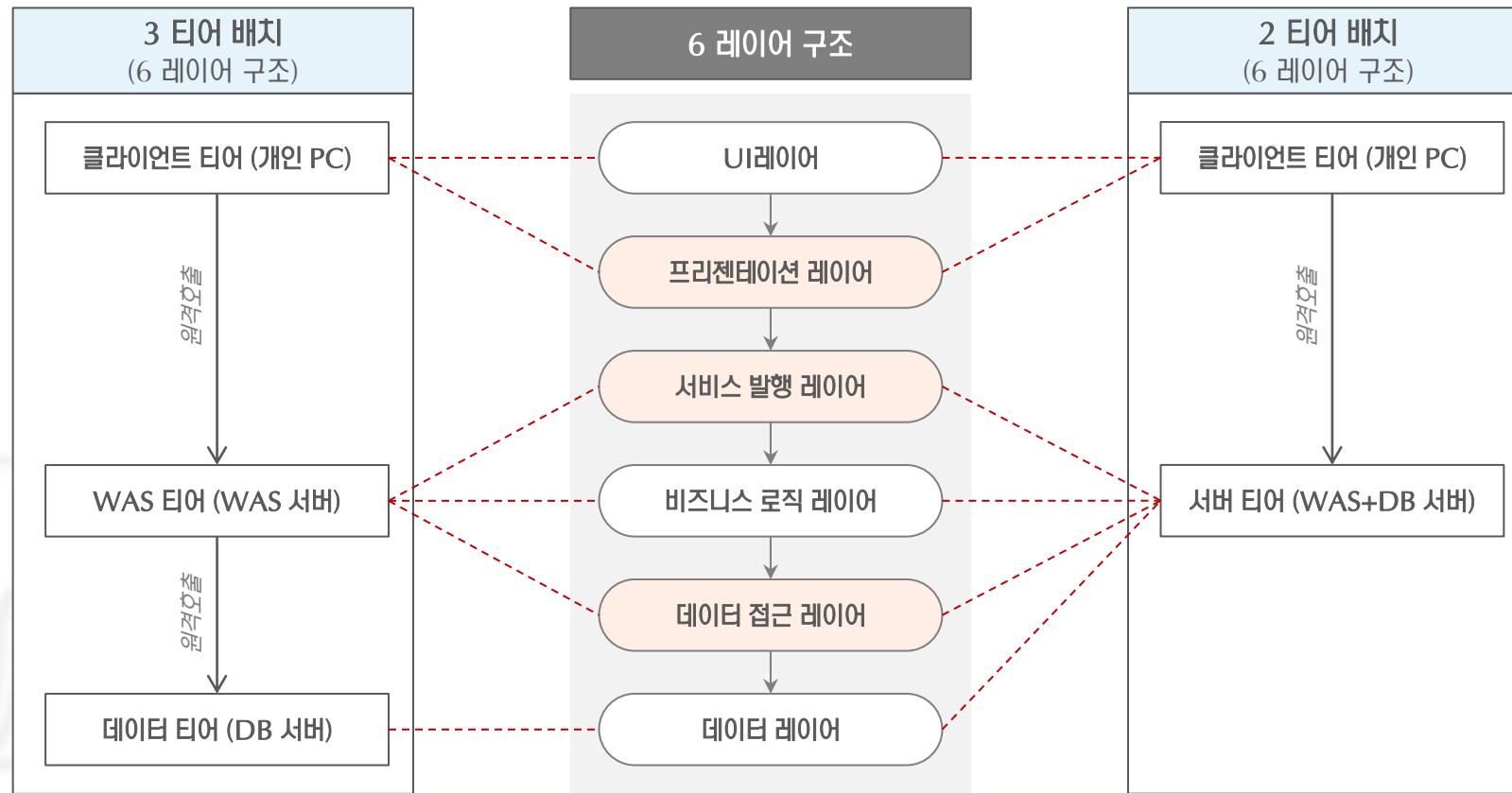
# 레이어와 티어(2/4)

- ✓ 레이어를 나눌 때, “서로 다른 작업은 서로 다른 레이어에서 한다.”는 원칙으로 세 개의 레이어를 나누었습니다.
- ✓ 레이어를 나누는 또 다른 기준은 “인접 레이어의 변화 충격을 흡수하는 레이어를 둔다.”입니다.
- ✓ 데이터 저장 방식이 변경될 때, 비즈니스 로직 대신 그 변화 충격을 흡수하는 “데이터 접근 레이어”가 필요합니다.
- ✓ 다양한 서비스 제공(웹 서비스 기반, REST 기반, TCP 기반, 등)에 따른 변화는 “서비스 발행 레이어”에서 소화합니다.



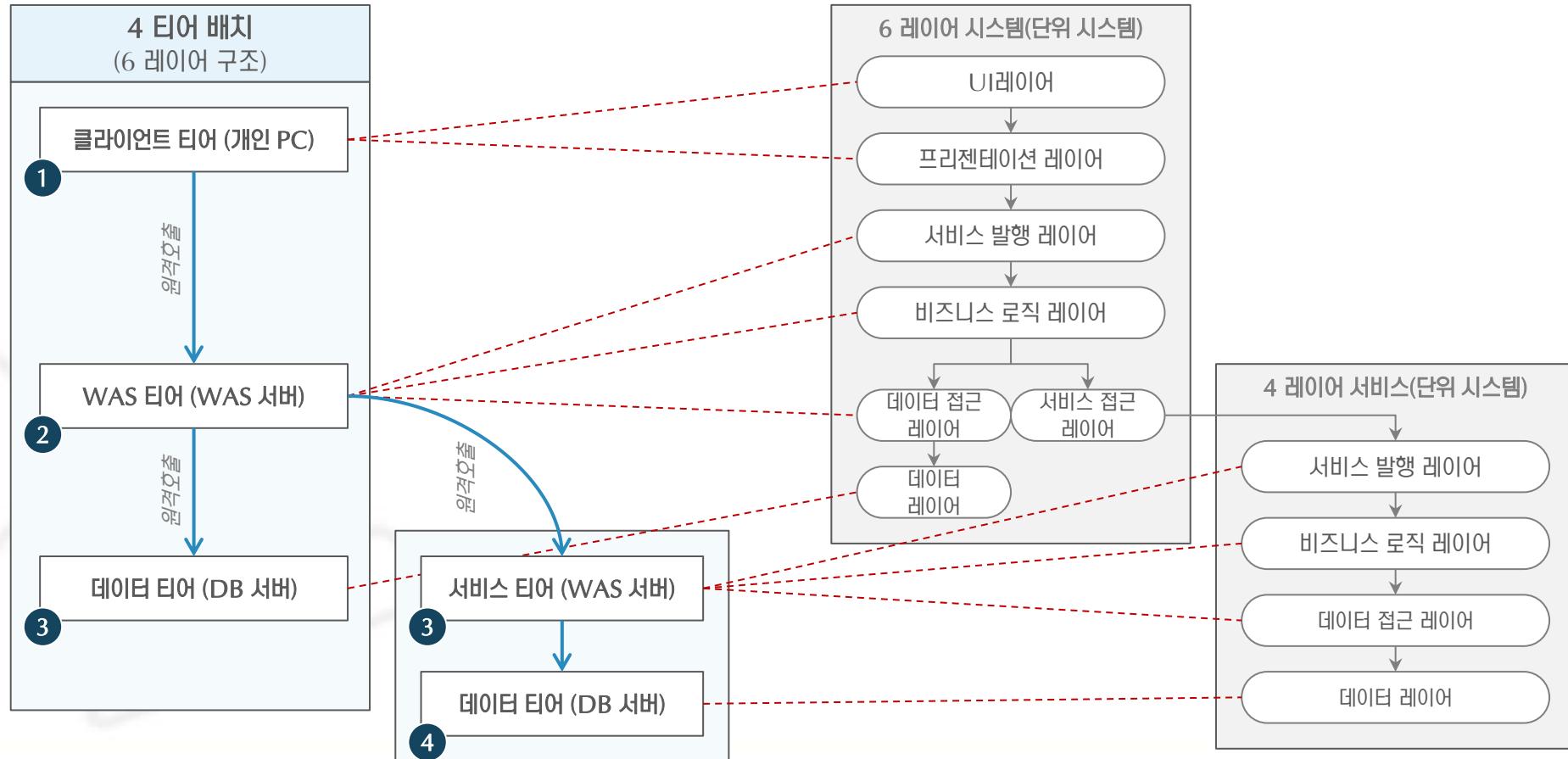
# 레이어와 티어(3/4)

- ✓ 레이어가 늘어나더라도 티어의 변화가 없을 수도 있습니다. 6 레이어로 변경되었으나 티어는 여전히 3티어입니다.
- ✓ “처리 대상(UI, 로직, 데이터)”이 주로 티어와 일치하고, “변화 대응”을 위한 레이어는 별도의 티어를 만들지 않습니다.
- ✓ 다른 관점에서 티어를 실행 프로세스와 거의 일치하기도 합니다.
- ✓ 이 예를 보면, 티어와 레이어는 관련이 있지만, 일치하지는 않음을 알 수 있습니다.



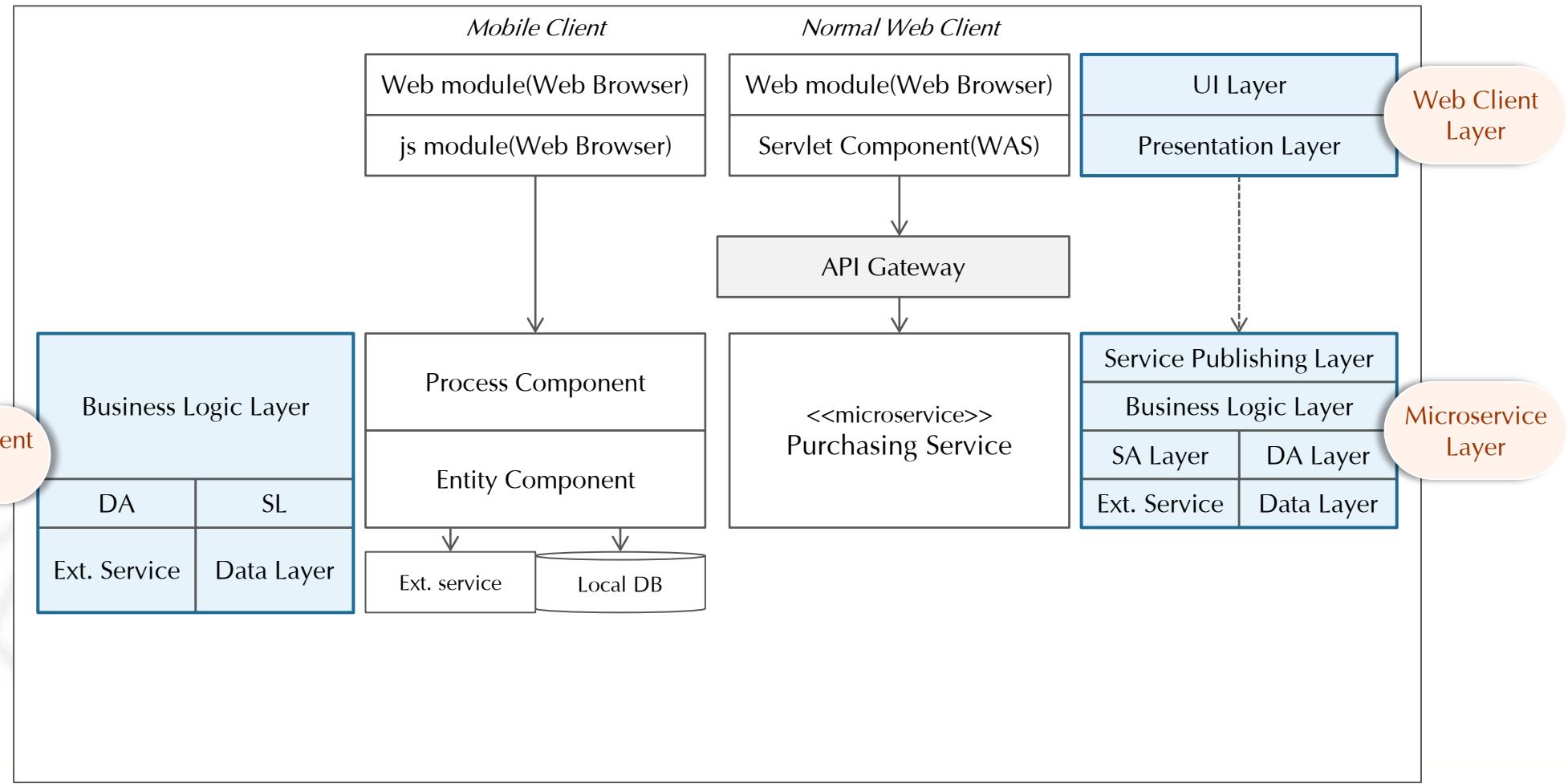
# 레이어와 티어(4/4)

- ✓ 레이어 개수는 단위 시스템(시스템, 서브 시스템, 서비스, 마이크로 서비스) 안에서 결정합니다.
- ✓ 티어 개수는 하나의 기능을 완결하는데 필요한, 모든 호출 경로를 포함하는 최대 티어 개수로 결정합니다.
- ✓ 아래 시스템은 4티어 아키텍처를 가지고 있으며, 각 단위 시스템 별로 6레이어, 4레이어로 설계하였습니다.



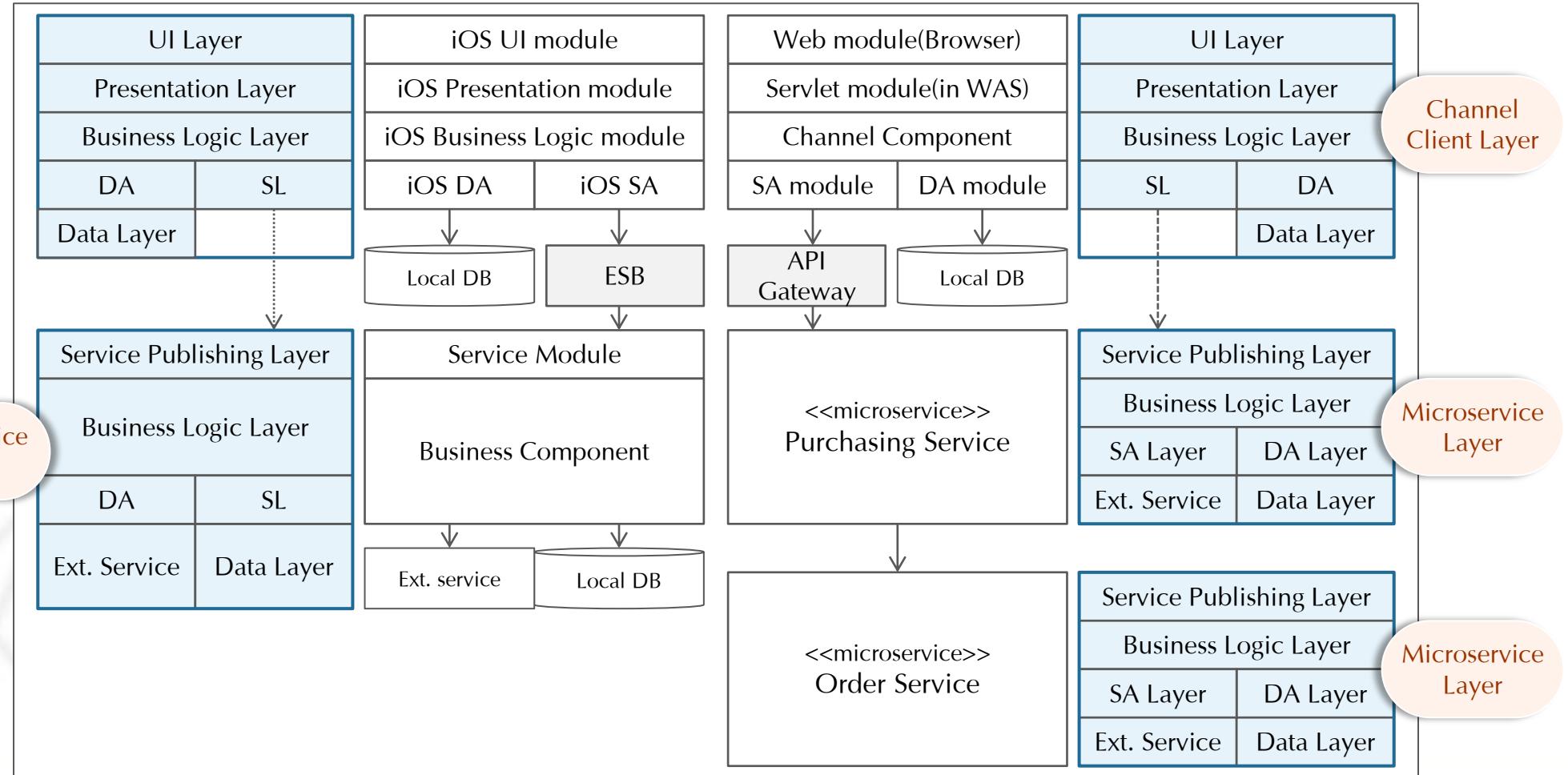
# 서브 시스템과 레이어(1/2)

- ✓ 컨테이너 관점에서 바라 보았을 때, 다양한 유형의 서브 시스템/서비스를 볼 수 있습니다.
- ✓ 각 서브 시스템/서비스 별로 서로 다른 레이어 구조를 가질 수 있습니다. 로직의 유무에 따라 레이어 구성이 달라집니다.



# 서브 시스템과 레이어(2/2)

- ✓ 로직 처리를 포함하는 클라이언트와 로직을 포함하지 않는 클라이언트의 레이어 구조는 다를 수 밖에 없습니다.
- ✓ 서버 사이드의 서브 시스템/서비스와 클라이언트 사이드 서브 시스템/서비스의 레이어 구조가 다를 수 밖에 없습니다.



## 8. 컴포넌트 모델링

- ❖ 비즈니스 컴포넌트를 구성하는 요소를 알아봅니다.
- ❖ 기술의 흐름과 컴포넌트 구조의 변화를 알아봅니다.

- ✓ [old] 컴포넌트 정의
- ✓ [old] 컴포넌트 내부 구조
- ✓ 컴포넌트 구성
- ✓ 컴포넌트 설계
- ✓ 무늬만 컴포넌트
- ✓ BCF 컴포넌트
- ✓ JPA 도메인 컴포넌트
- ✓ Pure 도메인 컴포넌트

# 컴포넌트 정의(1/3)

- ✓ 컴포넌트에 대한 다양한 정의가 존재하며, 특히 조직별로 차이가 있으며, 그로 인한 의사소통의 어려움이 존재합니다.
- ✓ 컴포넌트를 보는 다양한 관점(하드웨어, CPU에서 실행 중인 프로세스, 구현체, 논리적인 설계단위)이 존재합니다.
- ✓ 이런 이유로 인해 컴포넌트라는 용어를 피하고 예전의 모듈이나 번들이라는 일반적인 개념을 도입하는 경우도 있습니다.

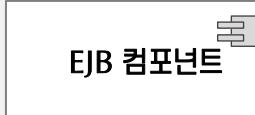
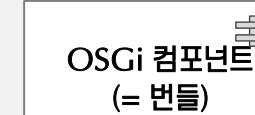
| 모듈(module)  | 컴포넌트(component)   | 번들(bundle)   |
|---|---|--|
|   | <br>실행개체 ? 논리적인 설계 개체 ?   |  |
| <ul style="list-style-type: none"><li>❖ 과거로부터 현재까지 조립이 가능한 소프트웨어 부품으로 기능의 집합이라는 의미로 가장 많이 사용하는 용어임</li><li>❖ 컴포넌트가 그 의미를 대체하다가, 컴포넌트가 너무 광범위하게 사용되는 까닭에 과거의 모듈이라는 의미를 다시 사용하는 경향도 생겨나고 있음</li></ul> | <ul style="list-style-type: none"><li>❖ IT 산업에서 HW와 SW 영역 등을 가리지 않고 많이 사용하는 일반적인 용어임</li><li>❖ 해당 컴포넌트의 의미를 이해하기 위해서는 컴포넌트를 언급하고 있는 컨텍스트를 먼저 확인하여야 함</li><li>❖ NEXT에서 컴포넌트는 UML에서 정의한 컴포넌트의 의미를 따름</li><li>❖ 설계모델 단위와 구현 단위가 일치하는 요소임</li></ul> | <ul style="list-style-type: none"><li>❖ OSGi 진영은 SW 모듈화를 가장 잘 실현하는 표준화 연합체이다. OSGi에서 캡슐화되어서 재사용할 수 있는 단위를 컴포넌트라고 하지 않고 번들이라고 부른다.</li><li>❖ 높은 수준으로 일반화한 표현으로 OSGi 외 다른 곳에서는 잘 사용하지 않는 용어임</li><li>❖ NEXT에서 인터페이스 버전 문제해결을 위해 OSGi를 고려할 가치가 있음</li></ul> |

UML에서 컴포넌트는 모듈방식(규격화된 부품을 조립하여 만들 수 있는)의 부품으로, 컨텐츠를 캡슐처럼 감싸고 있다. 컴포넌트는 행위를 요구(required) 인터페이스와 제공(provided) 인터페이스 관점에서 정의하며 이 두 가지 인터페이스가 일치한다면 그 컴포넌트는 대체 가능하다. 이러한 아이디어는 컴포넌트 기반 시스템의 플러그-앤-플레이를 위한 기초가 되며, 소프트웨어 재사용을 촉진한다. UML에서는 컴포넌트의 입도(granularity)에 대한 제약조건을 주지 않는다. 따라서 컴포넌트는 숫자 변환기 같이 작은 것에서부터 문서관리시스템처럼 큰 것도 가능하다.

/From wikipedia – Component(UML)/

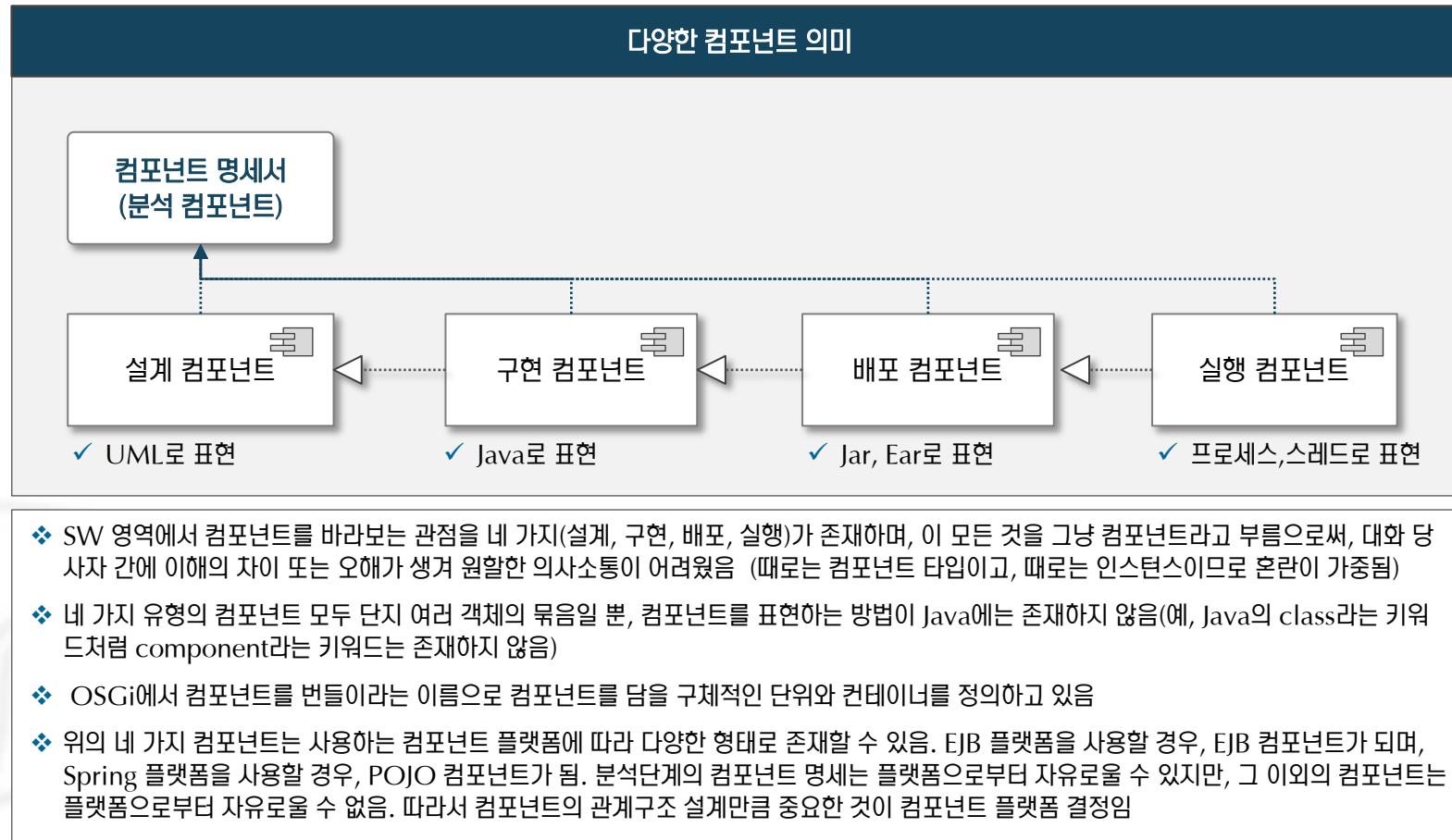
# 컴포넌트 정의(2/3)

- ✓ 구현에 사용한 기술에 따라, POJO 컴포넌트, EJB 컴포넌트, 웹 컴포넌트, OSGi 컴포넌트 등으로 분류할 수 있습니다.
- ✓ 아키텍처 설계 과정에서 어떤 컴포넌트 구현기술을 사용할 것인지에 대한 의사결정을 해야 합니다.
- ✓ 엔터프라이즈 시스템의 경우, 각 시스템의 특성에 따라 여러 가지 컴포넌트가 공존하는 경우가 많습니다.

| POJO 컴포넌트   | EJB 컴포넌트  | Web 컴포넌트  | OSGi 컴포넌트   |
|---|---|---|---|
|    |    |    |    |
| <p>실행개체 ? 논리적인 설계 개체 ?</p> <ul style="list-style-type: none"><li>❖ POJO는 Plain Old Java Object 의 약자로 순수한 자바 객체를 의미한다.</li><li>❖ POJO 컴포넌트 실행을 위해서는 POJO 컨테이너가 필요한데, Spring은 대표적인 POJO 컨테이너이다.</li></ul> | <p>실행개체 ? 논리적인 설계 개체 ?</p> <ul style="list-style-type: none"><li>❖ J2EE 명세서의 EJB를 구현한 컴포넌트이다. EJB 2.1 명세서 당시 컴포넌트가 너무 무겁고 느려 수행 성능 요건을 충족하지 못하여 설계자의 외면을 받았다.</li><li>❖ EJB 컴포넌트 실행을 위해서는 EJB 컨테이너가 필요하다. JBoss와 같은 오픈소스 EJB 컨테이너도 있고, WebLogic, WebSphere와 같은 상용 EJB 컨테이너도 있다.</li></ul> | <ul style="list-style-type: none"><li>❖ 웹 컴포넌트는 웹화면을 구성하는 JSP, JavaScript, Java 객체 등으로 구성된 컴포넌트이다.</li><li>❖ 웹 컴포넌트 실행을 위해서는 Tomcat과 같은 웹 컨테이너가 필요하다. POJO 컨테이너는 웹 컨테이너와 함께 실행되기도 한다.</li></ul> | <ul style="list-style-type: none"><li>❖ 번들은 OSGi 플랫폼에 설치되고 실행되는 컴포넌트로써, 라이프사이클을 가지며, 고유한 identity를 가진다.</li><li>❖ 번들은 서로 다른 버전의 인터페이스를 제공함으로써, 버전 업그레이드로 인한 서비스 변화를 효과적으로 지원한다.</li></ul> |

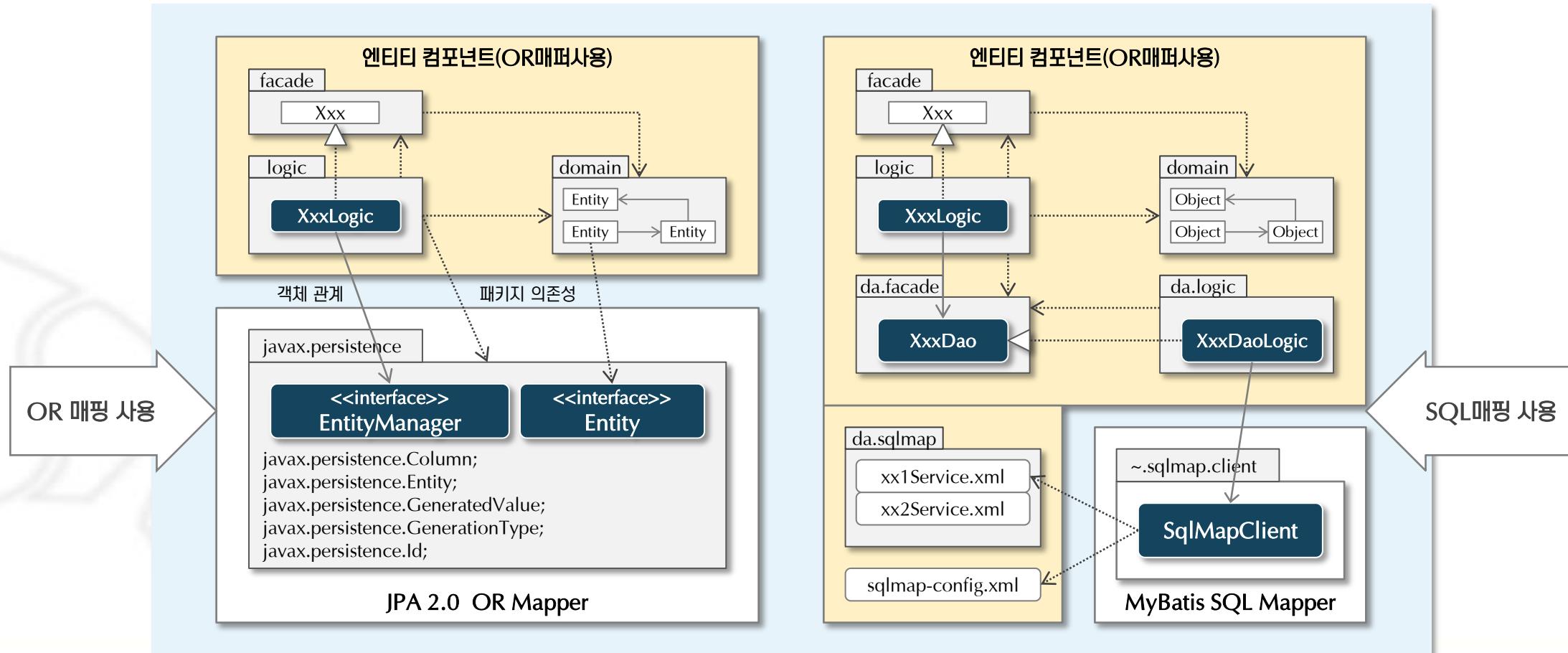
# 컴포넌트 정의(3/3)

- ✓ [what] 컴포넌트라고 할 수 있을 정도로 컴포넌트는 다양한 대상을 지칭하므로, [what]에 대한 표현이 중요합니다.
- ✓ 컴포넌트는 개발 각 단계별로 서로 다른 이슈를 가지고 있습니다. (예, 설계 단계는 확장성, 배포 단계는 의존성 등)
- ✓ 컴포넌트 개발에 투입되는 공수(effort)는 전체 프로젝트의 70~80% 가 될 정도로 많은 비중을 차지합니다.



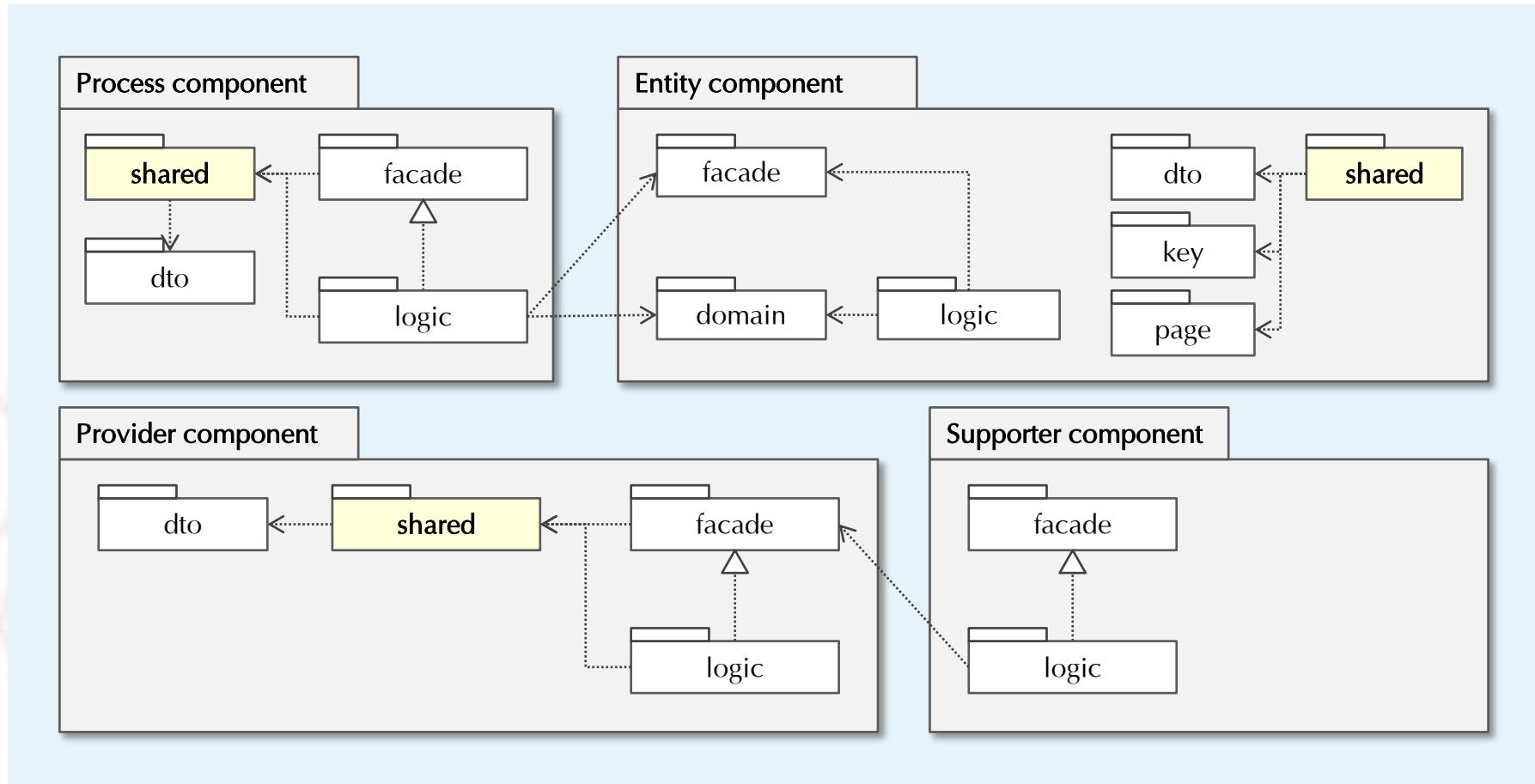
# 컴포넌트 내부 구조(old)

- ✓ 엔티티 컴포넌트는 데이터 접근구조에 따라 패키지 내부구조에 차이가 발생합니다.
- ✓ OR 매팅을 이용한 엔티티는 데이터 접근을 JPA 표준인 EntityManager에 의존하므로 별도의 패키지가 없습니다.
- ✓ SQL 매팅을 이용할 경우, SQL 매팅 파일, Dao 인터페이스와 구현 등을 위한 패키지 구성이 필요합니다.



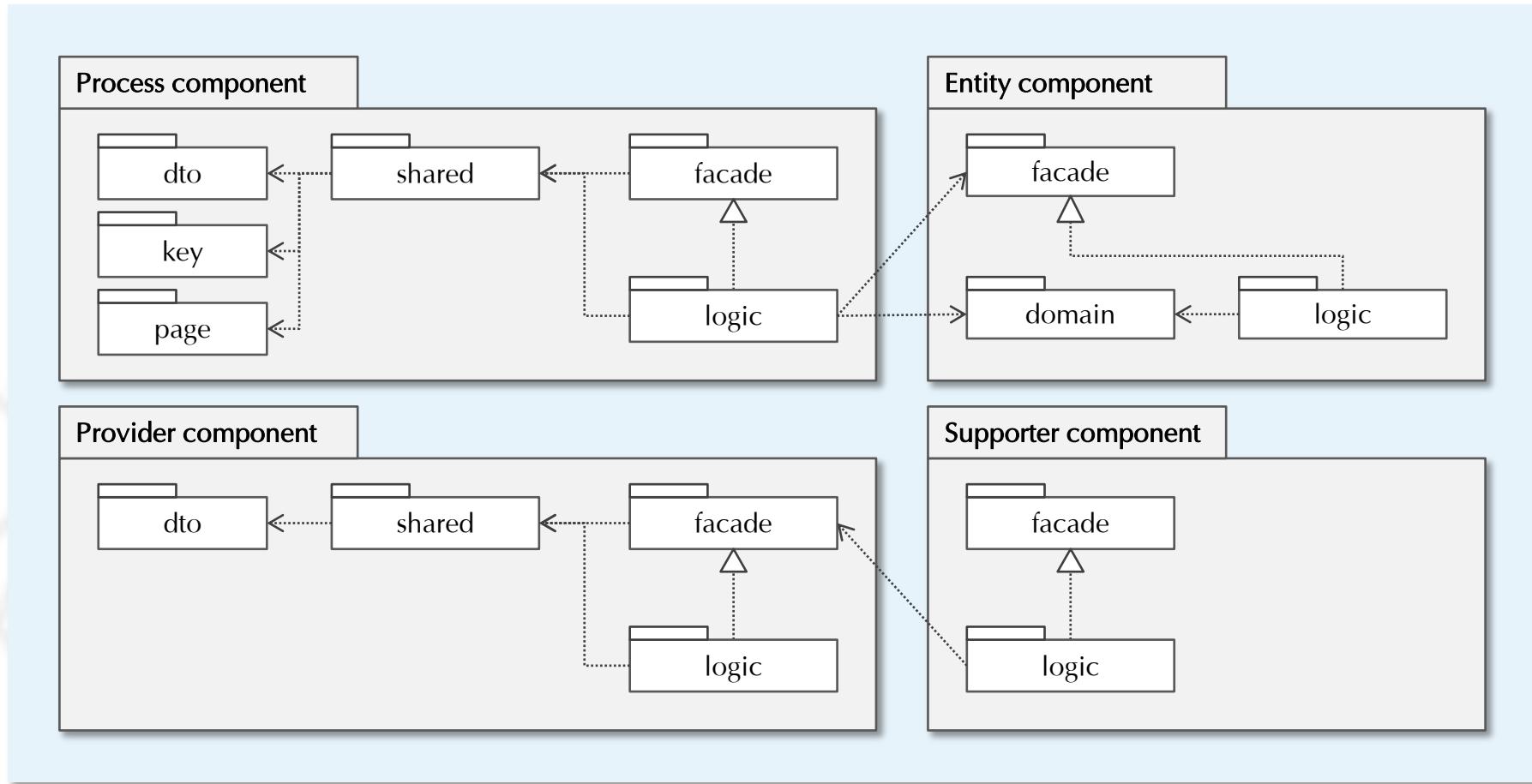
# 컴포넌트 내부 구조(old) – 엔티티 중심

- ✓ 컴포넌트 유형별로 서로 다른 내부구조를 가지는데 이것은 컴포넌트의 역할과 관계가 있습니다.
- ✓ 프로세스 타입 컴포넌트 내부에 있는 shared 영역을 엔티티 타입 컴포넌트의 내부에도 두어야 할지에 대한 고민이 필요하며, Supporter 컴포넌트는 shared 영역이 없으며 가장 단순한 구조를 가져야 합니다.



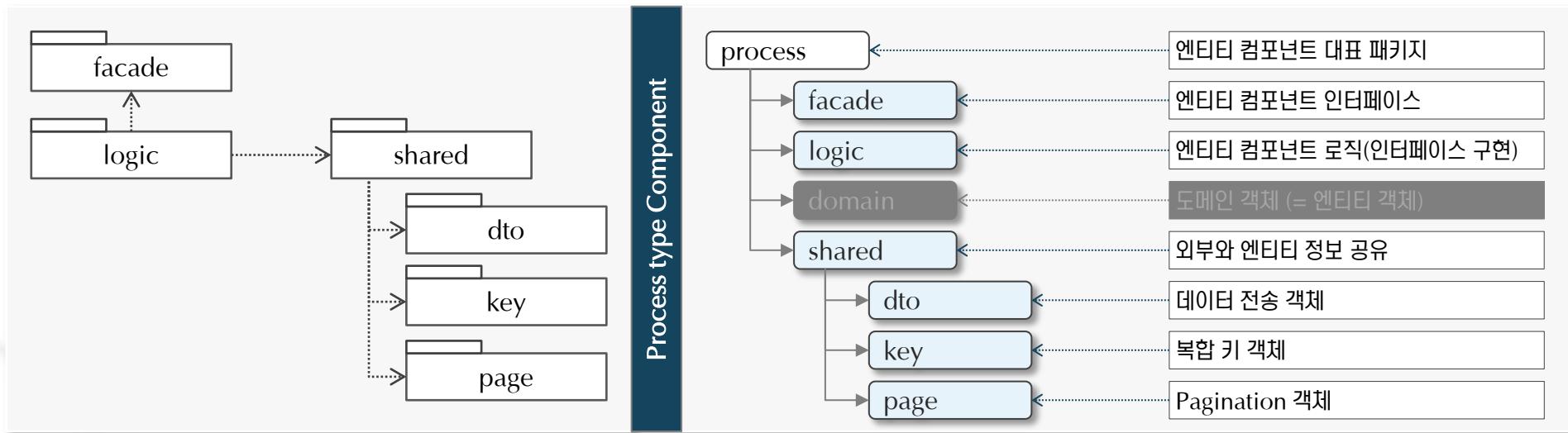
# 컴포넌트 내부 구조(old) – 프로세스 중심

- ✓ Entity 컴포넌트는 공유정보를 가지지 않고, 모든 공유정보는 Process 컴포넌트 내부로 두는 구조입니다.
- ✓ 하나의 프로세스 컴포넌트가 여러 Entity 컴포넌트를 사용하는 경우, Process 컴포넌트의 공유영역이 복잡해집니다.
- ✓ 설계의 원칙을 준수하지만 현실적인 어려움이 발생하는 구조입니다.



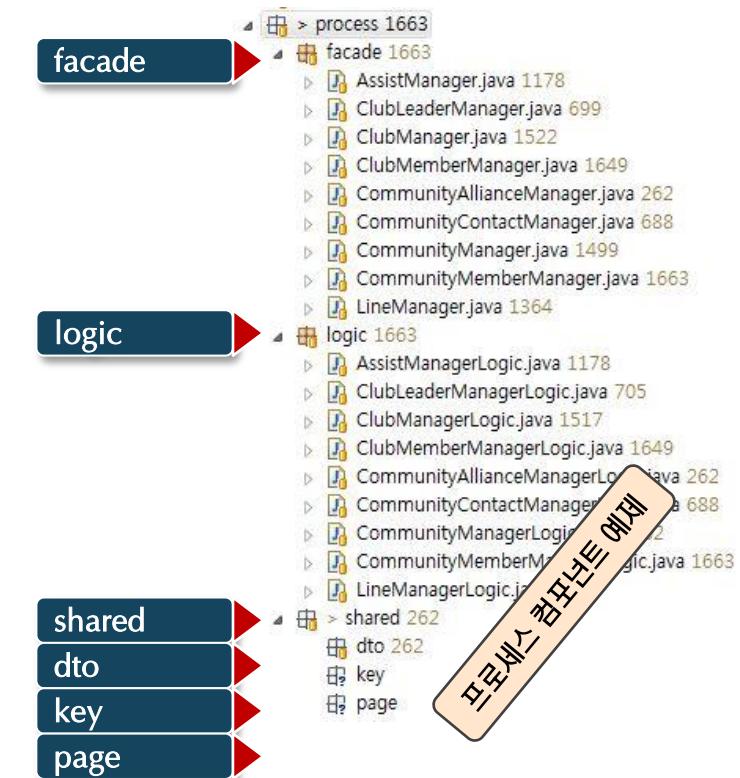
# 컴포넌트 내부 구조(old) – 프로세스 타입 컴포넌트 (1/2)

- ✓ 프로세스 컴포넌트의 특징은 도메인 객체를 가지고 있지 않다는 것입니다.
- ✓ 복합 프로세스(composite process), 프로세스, 제공 프로세스(provide process) 타입 모두 동일한 구조를 가집니다.
- ✓ 프로세스 컴포넌트가 가지고 있는 dto는 응용 수준의 dto이며, 엔티티 컴포넌트의 기본 dto와는 성격이 다릅니다.



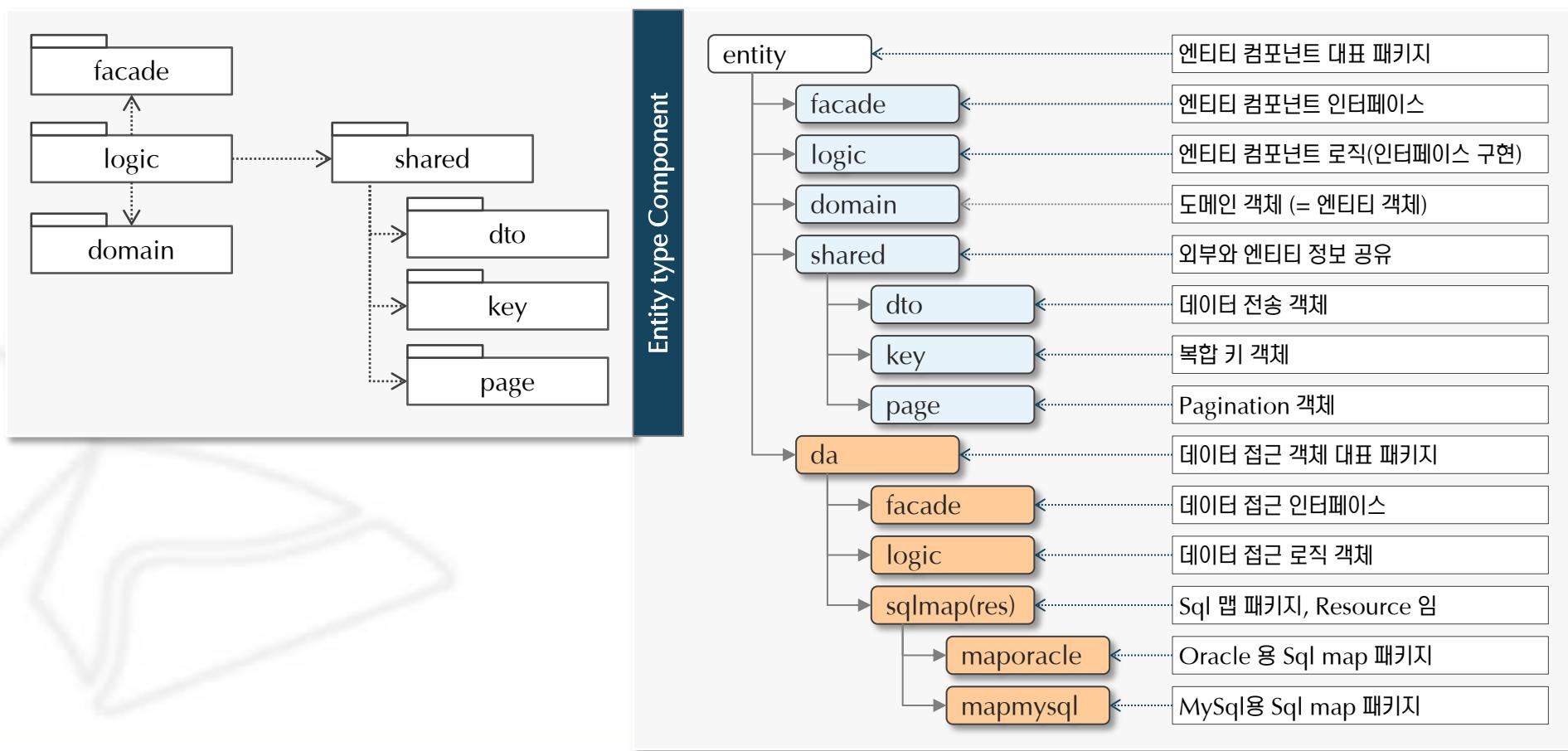
# 컴포넌트 내부 구조(old) – 프로세스 타입 컴포넌트(2/2)

- ✓ 프로세스 컴포넌트는 여러 개의 인터페이스 [클래스]를 가지고 있습니다.
- ✓ 인터페이스 구현체에 해당하는 logic 객체를 가지고 있습니다.
- ✓ 도메인 객체와 dto 간의 값 설정을 지원하는 helper 객체는 없으며, dto 객체에서 해당 기능을 지원합니다.



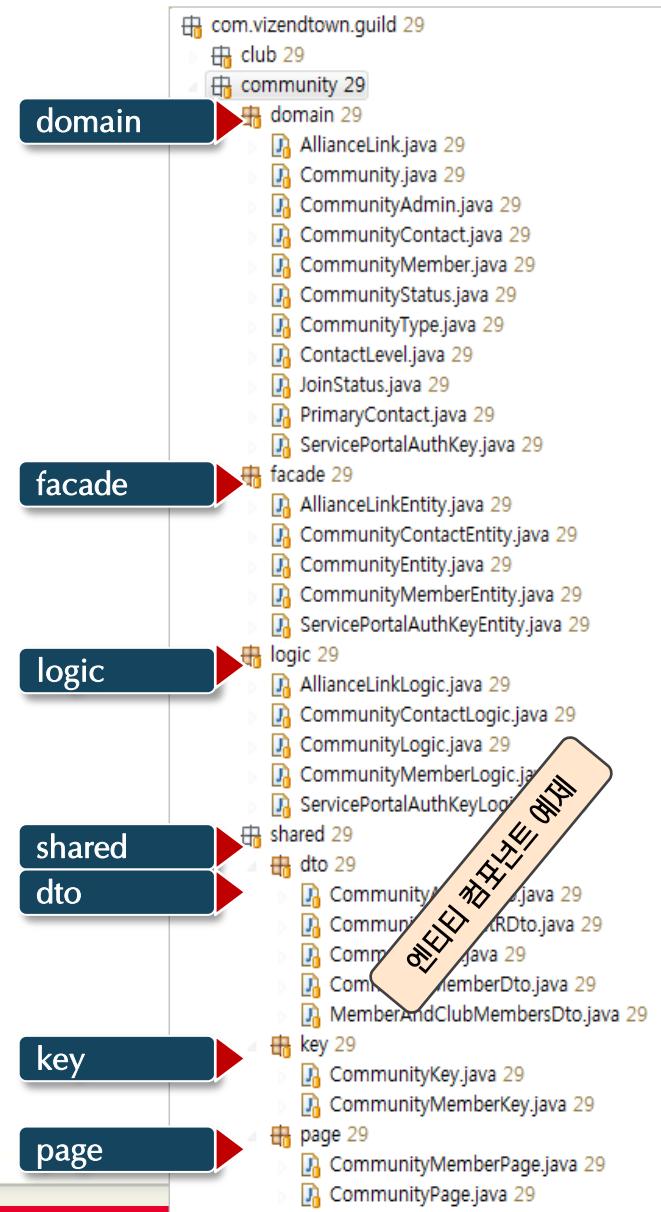
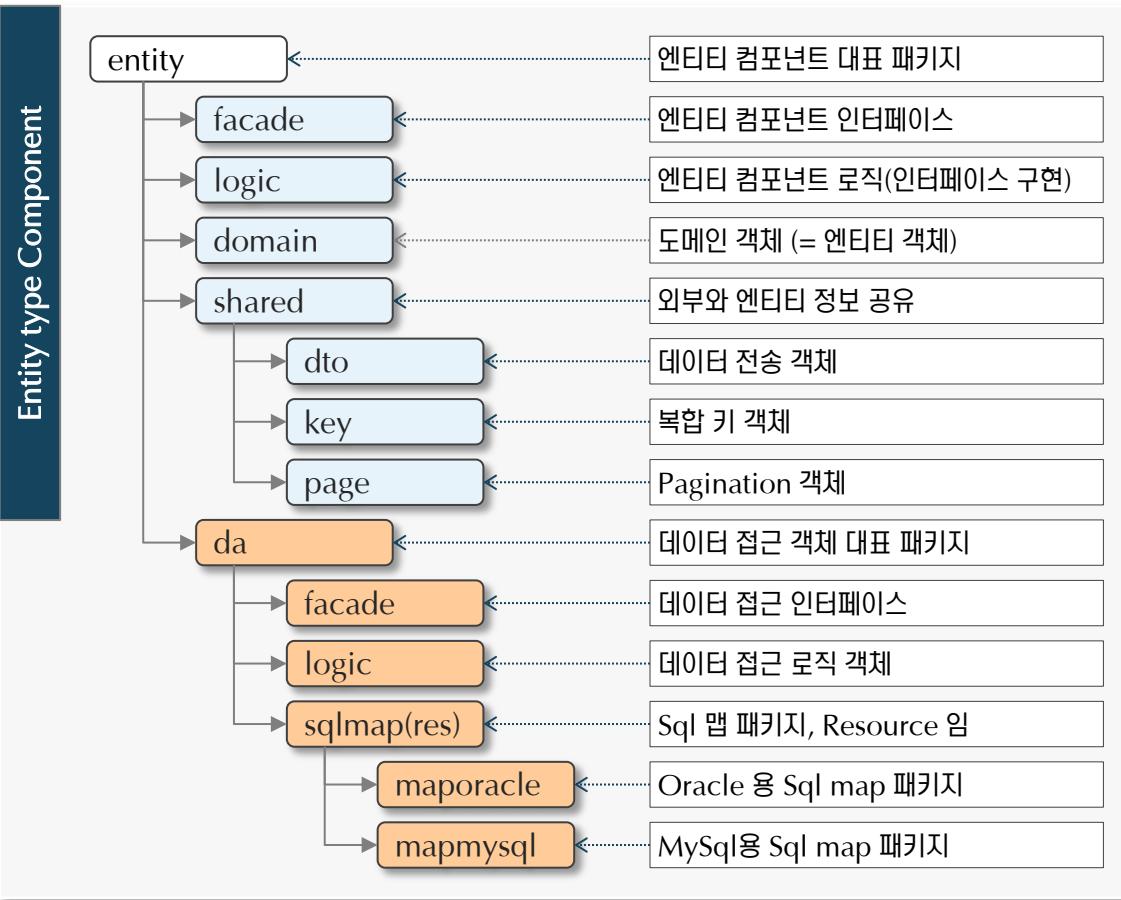
# 컴포넌트 내부 구조(old) – 엔티티 타입 컴포넌트 (1/2)

- ✓ 엔티티 타입 컴포넌트는 도메인 객체를 가지고 있습니다.
- ✓ 데이터 접근 방법에 따라, JPA 기반 접근일 경우는 별도의 데이터 접근 패키지가 필요 없습니다.
- ✓ SQL 매핑 방식으로 데이터에 접근할 경우, 아래 쪽의 da 패키지가 필요합니다.



# 컴포넌트 내부 구조(old) – 엔티티 타입 컴포넌트 (2/2)

- ✓ 엔티티 타입 컴포넌트는 3 ~ 8 개 정도의 도메인 객체를 가지고 있으며,
- ✓ 3개内外의 인터페이스를 가지고 있으며,
- ✓ 도메인 객체와 엇비슷한 개수의 DTO 객체를 가지고 있습니다.



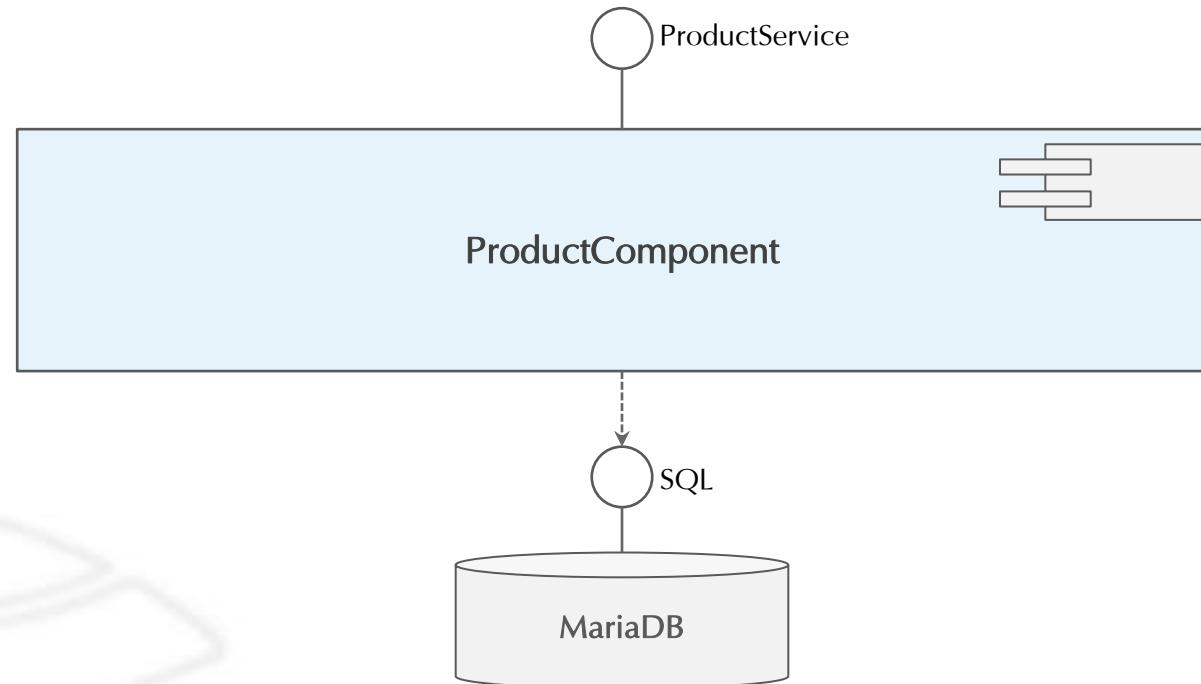
# 컴포넌트 내부 구조(old) – 유ти리티 타입 컴포넌트

- ✓ 유ти리티 컴포넌트는 두 가지가 있습니다. → 시스템 수준 유ти리티 컴포넌트, 도메인 수준 유ти리티 컴포넌트
- ✓ 시스템 유ти리티는 유형별로 패키지를 나눌 수 있으며, 패키지 구성은 아키텍트가 결정합니다.
- ✓ 도메인 유티리티는 “util”이라는 단일 패키지 안에 둡니다.



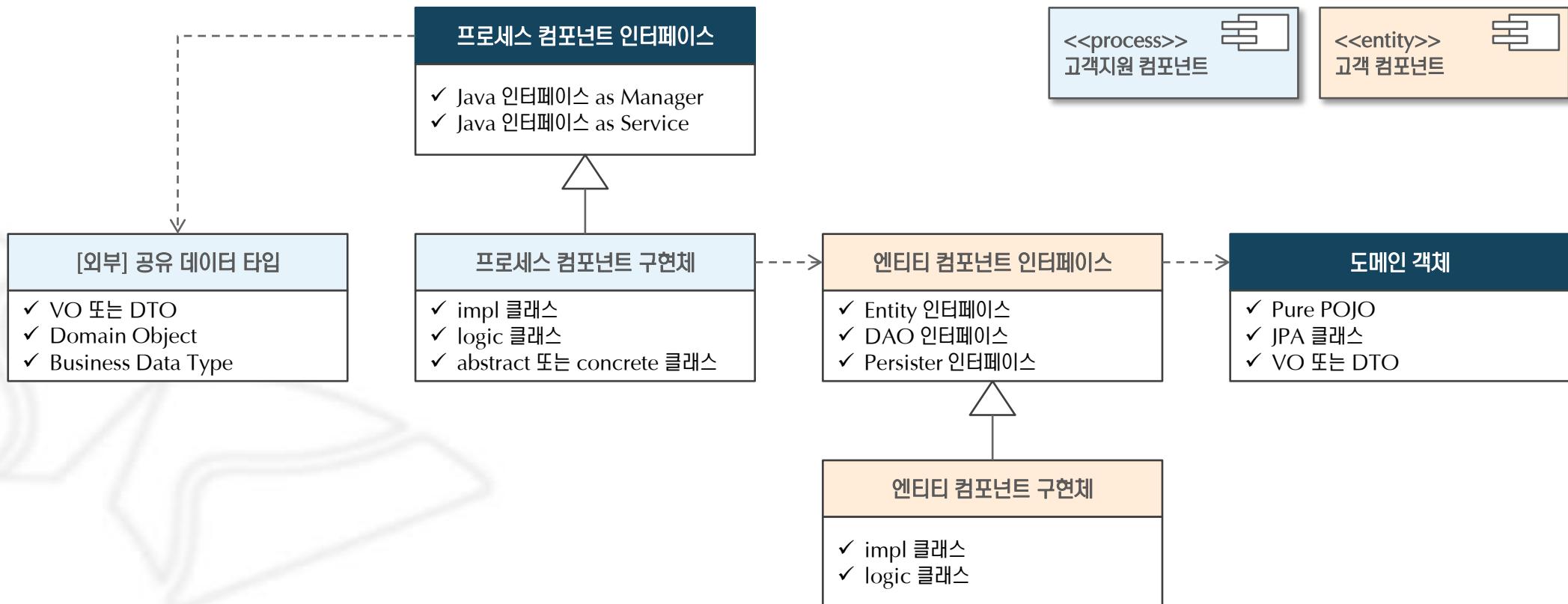
# 컴포넌트 구성(1/3)

- ✓ 비즈니스 컴포넌트 속에는 무엇이 있을까요?
- ✓ 정보에 해당하는 도메인 엔티티 객체, 로직에 해당하는 도메인 로직 객체, 데이터 저장소에 접근하는 객체, ...
- ✓ 여러 가지 유형의 인터페이스와 객체들이 규칙적으로 정렬되어 있을 겁니다.
- ✓ 컴포넌트 내부를 구성하는 요소(객체들의 그룹)는 어떤 것들이 있는지 생각해 봅시다.



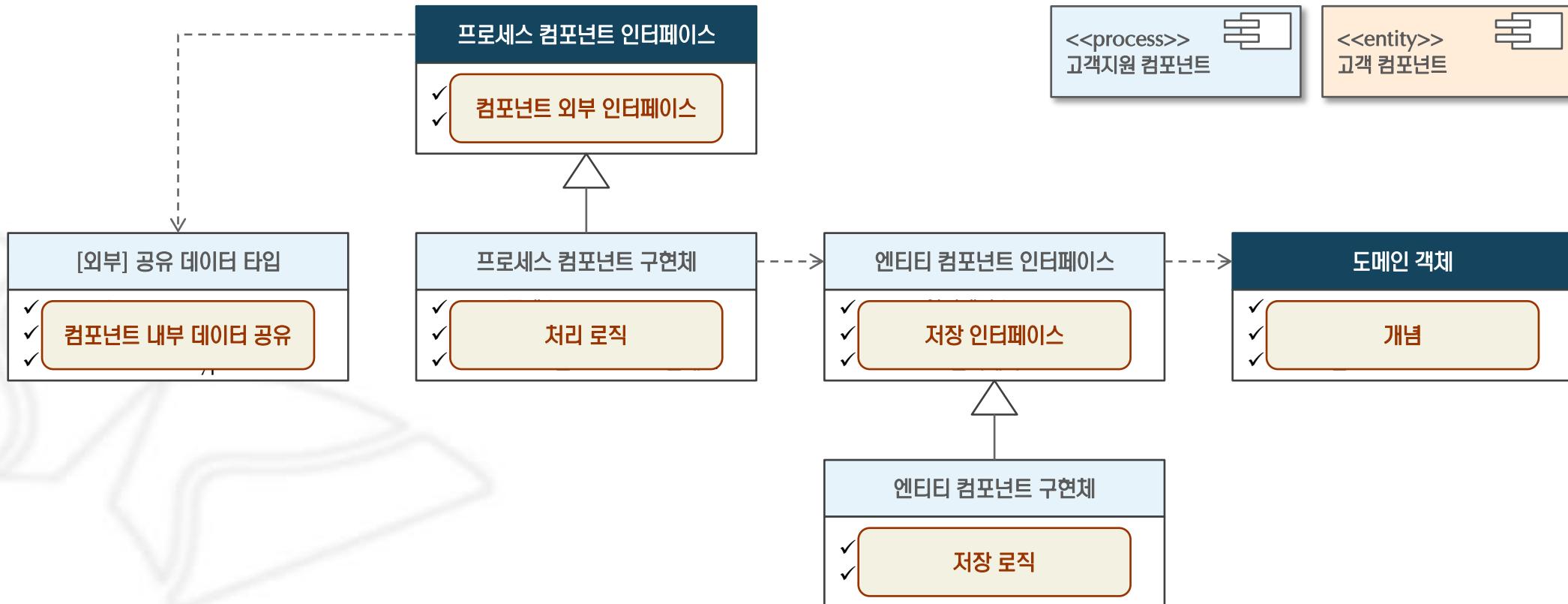
## 컴포넌트 구성(2/3) – 요소

- ✓ 비즈니스 컴포넌트를 구성하는 요소는 프로세스/엔티티 컴포넌트의 인터페이스와 구현체, 공유 타입과 도메인 객체입니다.
- ✓ 각 요소를 구현하는 방식에 따라 컴포넌트의 특징을 결정할 수 있습니다.
- ✓ 컴포넌트 구조 설계 시, 사용하는 요소의 이름은 서로 달라서 혼란스럽긴 하지만, 결국은 아래 여섯 가지 요소 중 하나입니다.
- ✓ 가장 이상적인 컴포넌트에서부터 무늬만 컴포넌트 까지 아래의 여섯 가지 구성 요소의 조합으로 결정됩니다.



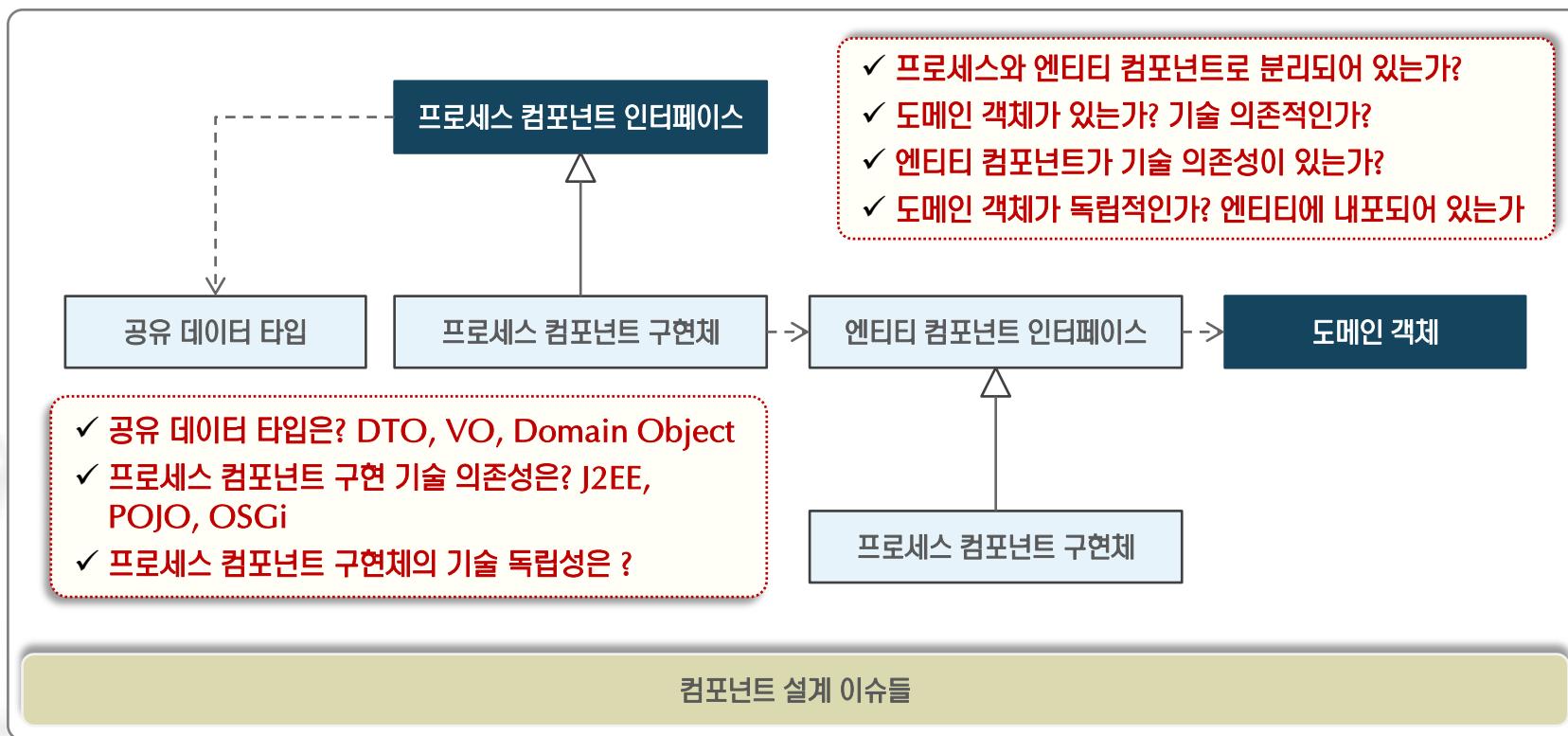
## 컴포넌트 구성(3/3) – 요소 별 목적

- ✓ “Loose coupling, High cohesion” 설계 목표를 달성하기 위해 컴포넌트 구성 요소는 각자의 존재 이유가 있습니다.
- ✓ 컴포넌트의 외부로 노출되는 구성요소, 비즈니스 로직을 처리하는 구성요소, 지속성을 담당하는 요소와,
- ✓ 컴포넌트가 책임지고 있는 주요 개념을 담당하는 “도메인 객체”로 구성됩니다.
- ✓ 각자의 역할을 가지고 전체 컴포넌트 그룹에 참여합니다. 이러한 요소의 역할을 조정하면 서로 다른 특성을 보여 줍니다.



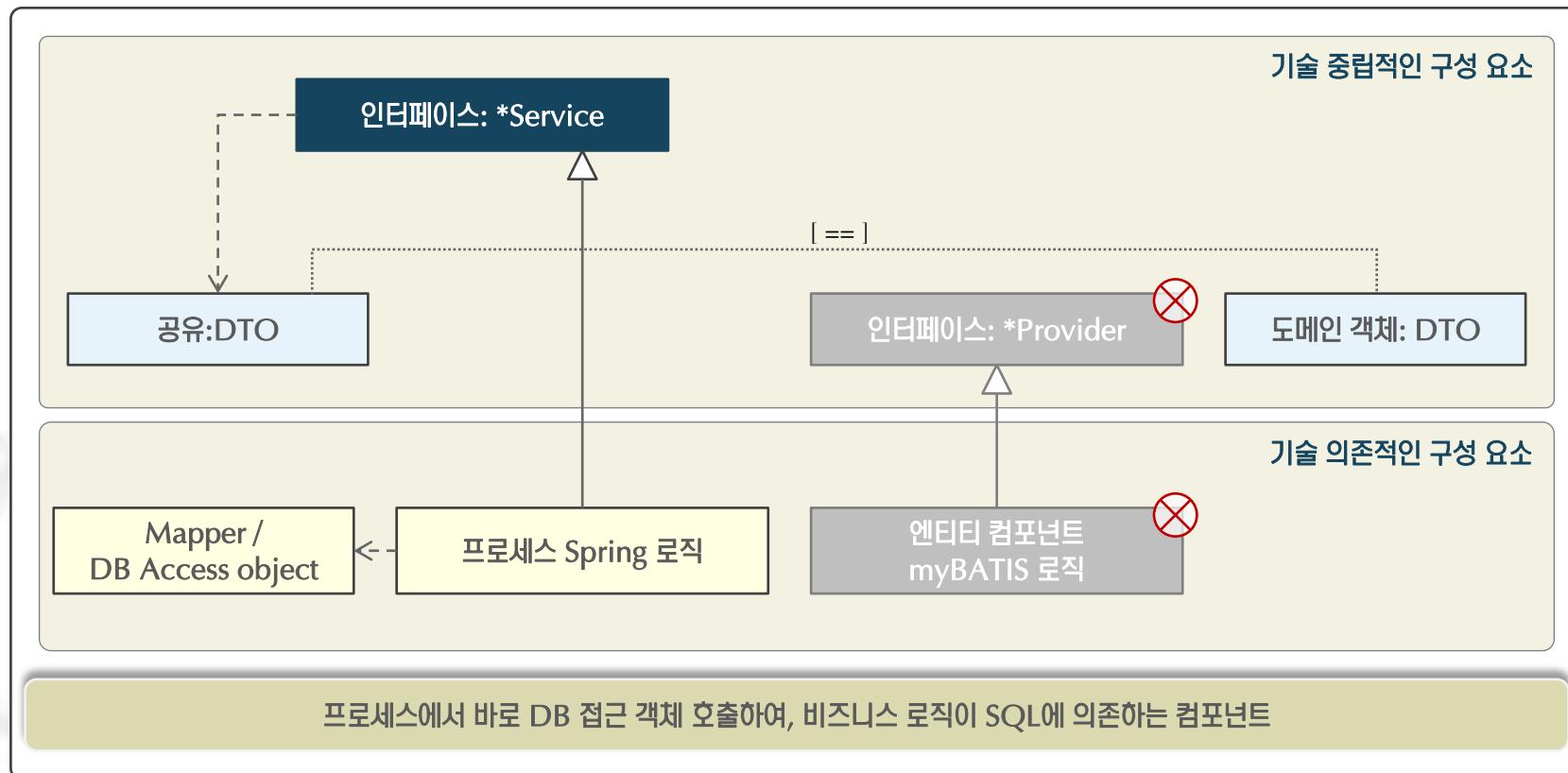
# 컴포넌트 설계 이슈

- ✓ 프로세스와 엔티티 컴포넌트로 분리되어 있는가? 분리되어 있다면, 엔티티 컴포넌트를 무엇이라 부르는가?
- ✓ 도메인 객체가 존재하는가? 존재한다면 기술 종속적인가, 아니면 PURE POJO인가?
- ✓ 도메인 객체가 엔티티 컴포넌트 내부에 있는가? 아니면 외부에 독립적으로 존재하는가?
- ✓ 프로세스 컴포넌트의 비즈니스 로직은 기술 독립적인가? 독립적이라면 추상 클래스인가, 실제 클래스인가?



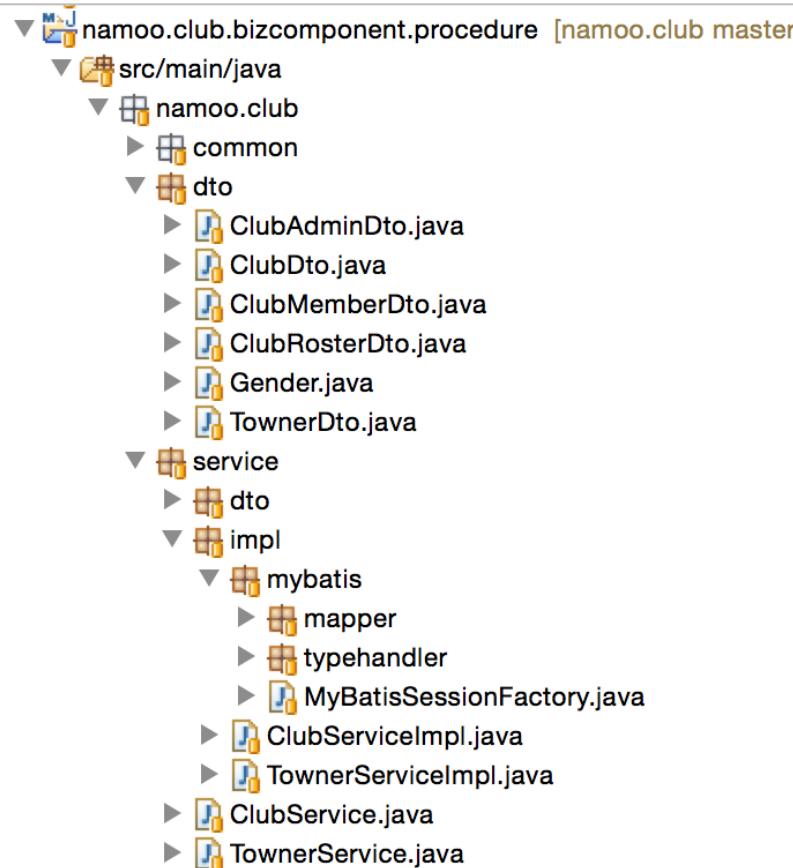
# 무늬만 컴포넌트(1/2)

- ✓ 컴포넌트 내부에 프로세스 로직과 엔티티 로직이 분리되어 있지 않습니다.
- ✓ 이런 구조는 기존의 절차 지향 구조를 컴포넌트 인터페이스 속으로 감춘 것에 지나지 않습니다.
- ✓ 도메인 객체가 없을 뿐만 아니라, DTO는 비즈니스 본질을 담기 보다는 UI에서 요구하는 정보를 담아 줄 뿐입니다.
- ✓ SI 프로젝트에서 이런 구조를 심심치 않게 봅니다. 어찌되었건 myBATIS는 사용을 했으니 문제 될 것이 없다고 합니다.



## 무늬만 컴포넌트(2/2)

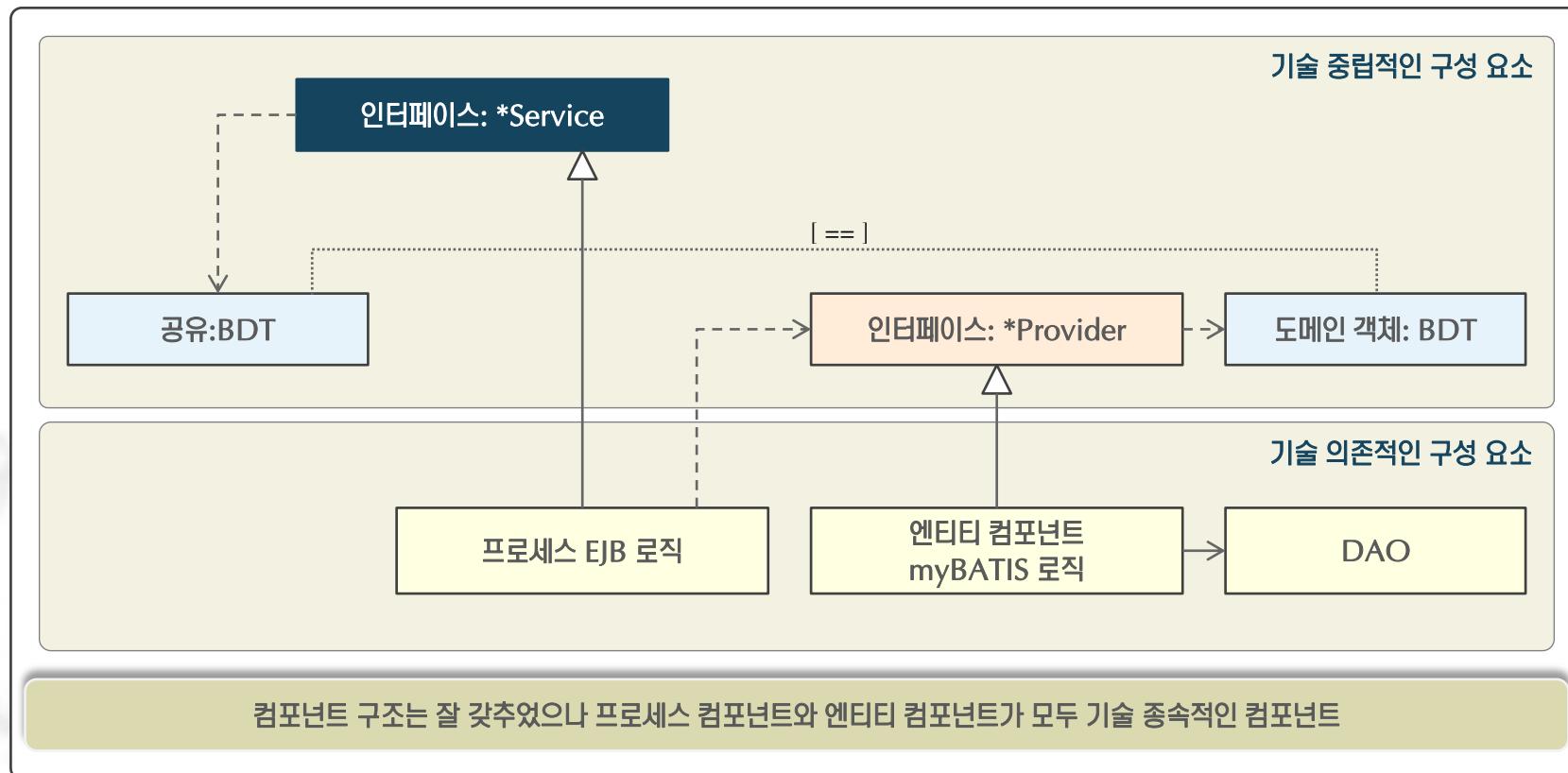
- ✓ 비즈니스 로직에서 Persistence를 책임지는 객체를 직접 다루도록 설계하였습니다.
- ✓ SQL에 많이 의지 합니다.
- ✓ 저장 방식에 의존적입니다.



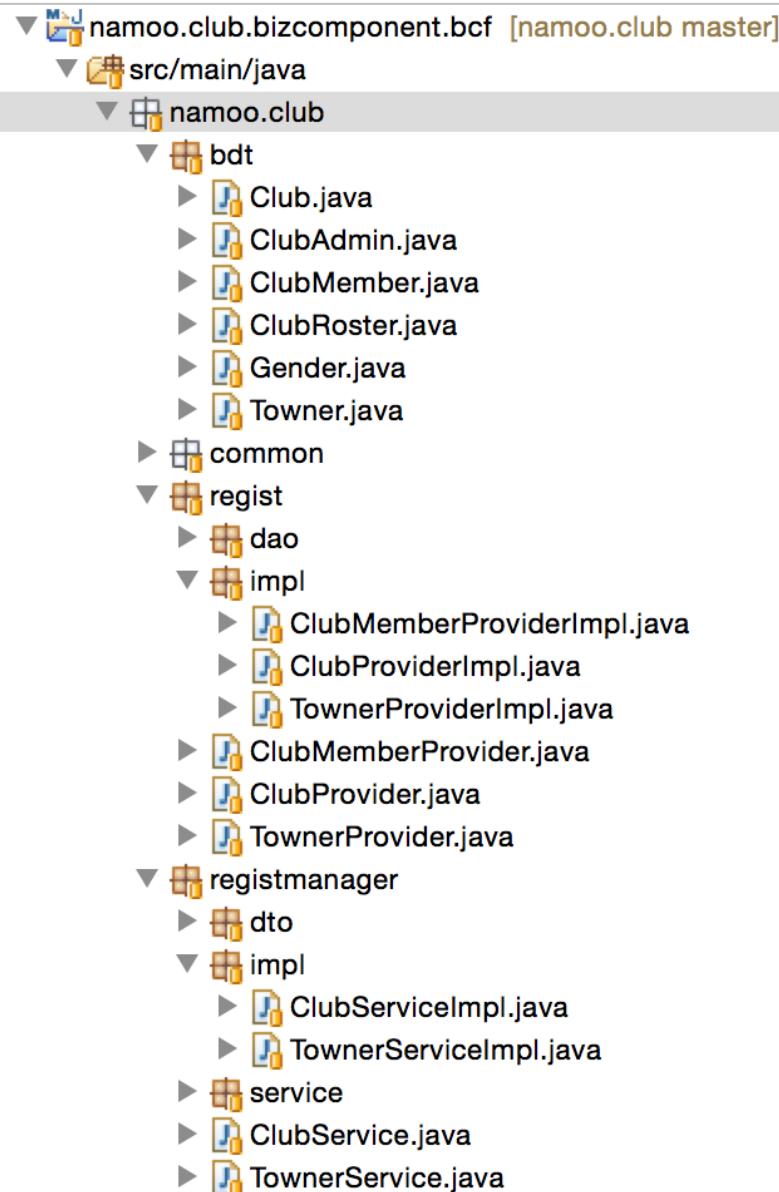
```
public class ClubServiceImpl implements ClubService {  
    //  
    @Override  
    public String registerClub(ClubCDto clubCDto) throws NamooClubException {  
        //  
        SqlSession session = MyBatisSessionFactory.getSession();  
  
        try {  
            ClubMapper clubMapper = session.getMapper(ClubMapper.class);  
            TownerMapper townnerMapper = session.getMapper(TownerMapper.class);  
            ClubMemberMapper clubMemberMapper = session.getMapper(ClubMemberMapper.class);  
  
            if (clubMapper.selectClubByName(clubCDto.getClubName()) != null) {  
                throw new NamooClubException("Already existed club --> " + clubCDto.getClubName());  
            }  
  
            TownerDto townner = null;  
            try {  
                townner = townnerMapper.selectTownerByEmail(clubCDto.getAdminEmail());  
            }  
            catch (EmptyResultException e) {  
                throw new NamooClubException(e);  
            }  
  
            ClubDto club = clubCDto.createDomain(new ClubAdminDto(townner));  
            clubMapper.insertClub(club);  
  
            ClubMemberDto clubMember = new ClubMemberDto(club, townner);  
            clubMemberMapper.insertClubMember(club.getId(), clubMember);  
            return club.getId();  
        } finally {  
            session.close();  
        }  
    }  
}
```

# BCF 컴포넌트(1/2)

- ✓ 도메인 개념을 표현하는 도메인 객체 관점 보다는 외부와의 공유 관점에 초점을 둔 Business Data Type을 정의합니다.
- ✓ 프로세스 컴포넌트는 플랫폼에 종속적입니다. 이런 구조를 가질 때는, 플랫폼을 변경할 가능성이 없을 때입니다.
- ✓ 엔티티 인터페이스는 중립 영역에 있으며, 구현은 지속성 기술에 종속적입니다.
- ✓ 물론 DAO를 통해서 DB에 접근함으로써, DB 변경에 따른 유연성을 가질 수 있는 구성입니다.



# BCF 컴포넌트(2/2)



```
public class ClubServiceImpl implements ClubService {
    //
    private ClubProvider clubProvider;
    private ClubMemberProvider clubMemberProvider;
    private TownerProvider townerProvider;

    /**
     * @return
     */
    public ClubServiceImpl() {
        //
        RegistManagerServiceFactory serviceFactory = RegistManagerServiceFactory.getInstance();

        this.clubProvider = serviceFactory.getClubProvider();
        this.clubMemberProvider = serviceFactory.getClubMemberProvider();
        this.townerProvider = serviceFactory.getTownerProvider();
    }

    /**
     * @Override
     */
    public String registerClub(ClubCDto clubCDto) throws NamooClubException {
        //
        if (clubProvider.isExistClubByName(clubCDto.getClubName())) {
            throw new NamooClubException("Already existed club --> " + clubCDto.getClubName());
        }

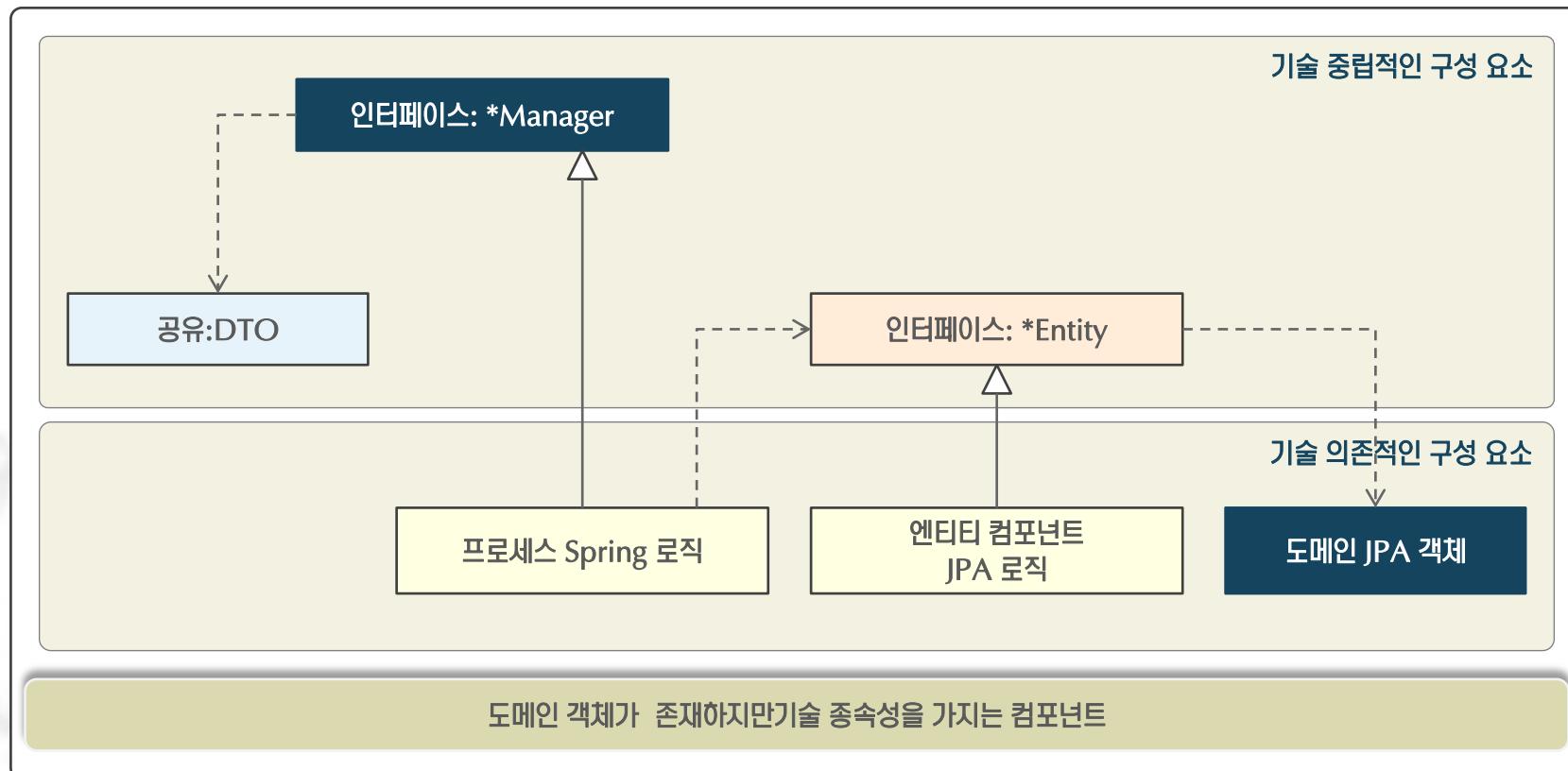
        Towner towner = null;
        try {
            towner = townerProvider.retrieveByEmail(clubCDto.getAdminEmail());
        } catch (EmptyResultException e) {
            throw new NamooClubException(e);
        }

        Club club = clubCDto.createDomain(new ClubAdmin(towner));
        clubProvider.create(club);
        ClubMember clubMember = new ClubMember(club, towner);
        clubMemberProvider.addMember(club.getId(), clubMember);

        return club.getId();
    }
}
```

# 도메인 컴포넌트(1/3)

- ✓ 도메인 객체가 분명하게 존재하며, 개념을 잘 담고 있습니다. 하지만, 기술에 의존적입니다.
- ✓ 예를 들면, JPA를 사용하여 지속성을 처리할 때, 도메인 객체에 JPA annotation이 포함됩니다.
- ✓ 프로세스와 엔티티 컴포넌트 구분이 명확합니다. 프로세스 로직도 별도로 존재하지만, 기술 종속적입니다.
- ✓ 컴포넌트 플랫폼과 지속성 플랫폼 또는 솔루션을 변경하지 않을 때 유용한 컴포넌트 구조입니다.



# 도메인 컴포넌트(2/3)

```
@Override  
@Transactional(propagation = Propagation.REQUIRED)  
public String registCitizen(String metroId, CitizenCDto citizenCDto) {  
    //  
    // validation  
    if(citizenEntity.existByLoginId(citizenCDto.getLoginId())){  
        throw new VizendTownException("loginId already exists! >> loginId : "+citizenCDto.getLoginId());  
    }  
  
    Castellan castellan = null;  
  
    if (!castellanEntity.existByAuthEmail(citizenCDto.getEmail())) {  
        Nation nation = nationEntity.retrieveByMetroId(metroId);  
        castellan = new Castellan(  
            nation.getId(),  
            townIdGenerator.generateCastellanId(nation.getId()),  
            citizenCDto.getLoginName(),  
            citizenCDto.getEmail());  
        castellanEntity.create(castellan);  
    } else {  
        castellan = castellanEntity.retrieveByAuthEmail(citizenCDto.getEmail());  
    }  
  
    String citizenId = townIdGenerator.generateCitizenId(metroId);  
    Citizen citizen = citizenCDto.createDomain(castellan, metroId, citizenId);  
    citizen.setPassword(EncryptUtils.encrypt(citizenCDto.getPassword()));  
  
    return citizenEntity.create(citizen);  
}
```

# 도메인 컴포넌트(3/3)

vizend-town-reign [townrepo master ↑1]

```
src/main/java
  com.vizend.town.reign
    nation
    people
      domain
        Castellan.java
        Citizen.java
        CivicAdmin.java
        PublicServant.java
        StatusCode.java
        Towner.java
      domainbackup
    facade
      CastellanEntity.java
      CitizenEntity.java
      CivicAdminEntity.java
      ServantEntity.java
      TownerEntity.java
    logic
      CastellanEntityLogic.java
      CitizenEntityLogic.java
      CivicAdminEntityLogic.java
      ServantEntityLogic.java
      TownerEntityLogic.java
    shared
      dto
        CastellanCDto.java
        CastellanRDto.java
        CitizenCDto.java
        CitizenRDto.java
        CivicAdminCDto.java
        CivicAdminRDto.java
        ServantCDto.java
        ServantRDto.java
        TownerRDto.java
      helper
```

```
@Repository
public class CitizenEntityLogic implements CitizenEntity {
    //
    @PersistenceContext
    private EntityManager em;

    //-----
    // methods

    @Override
    public String create(Citizen citizen) {
        //
        em.persist(citizen);
        return citizen.getId();
    }

    @Override
    public void createBackup(CitizenBackup citizenBackup) {
        //
        em.persist(citizenBackup);
    }

    @Override
    public Citizen retrieve(String citizenId) {
        //

        String queryStr = "FROM Citizen WHERE id = :citizenId";
        Query query = em.createQuery(queryStr);
        query.setParameter("citizenId", citizenId);

        return SafeQuery.getUniqueResult(query);
    }
}
```

```
@Entity
public class Citizen {
    //
    @Id
    @Column(name = "CITIZEN_ID")
    private String id; // MetroId-ABC123

    @Column(nullable = false)
    private String metroId;

    @Column(nullable = false)
    private String loginId;

    @Column(nullable = true)
    private String loginName;

    @Column(nullable = false)
    private String password;

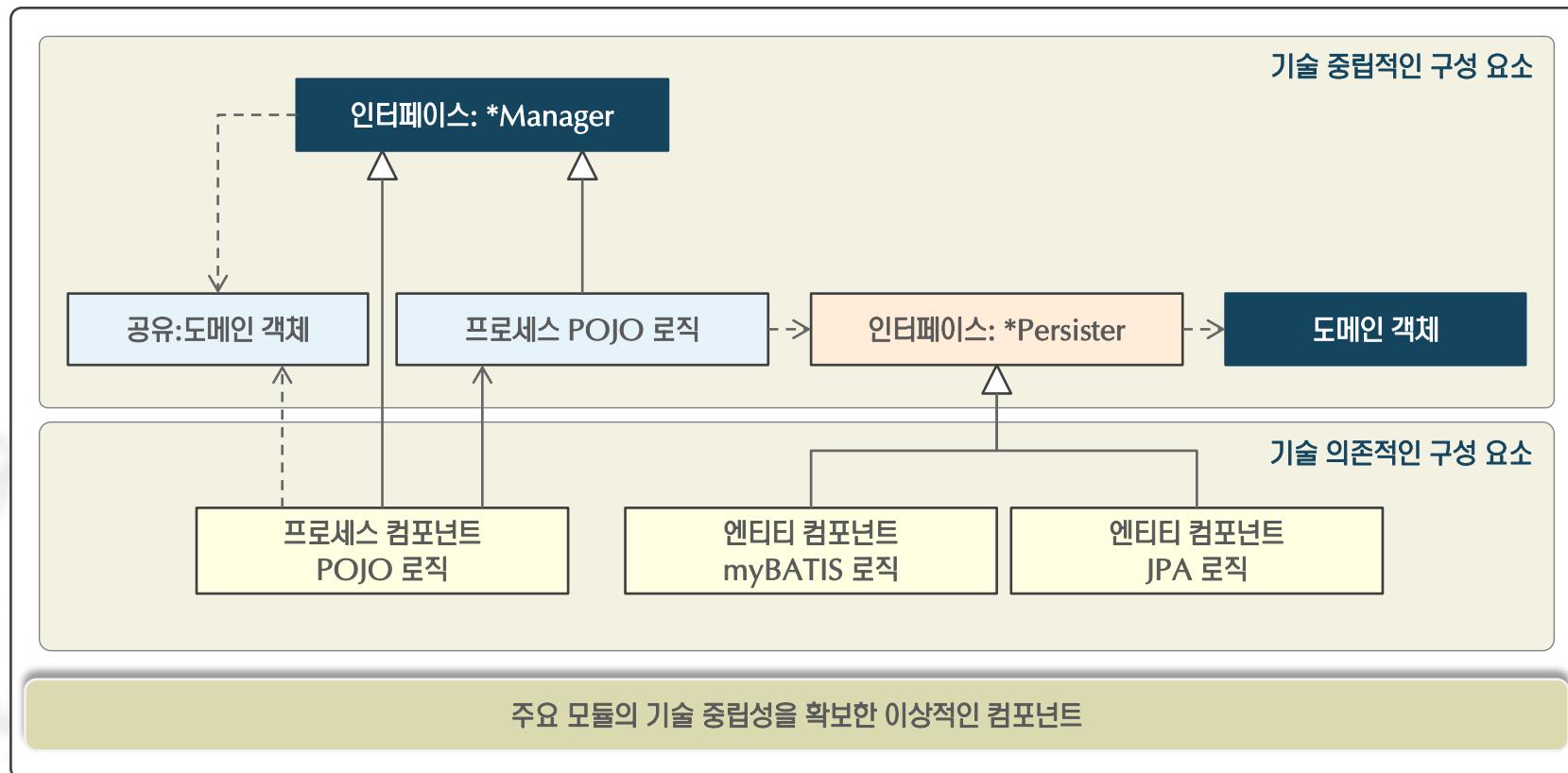
    @Column(nullable = true)
    private String email;

    @Column(nullable = true)
    private String phoneNumber;

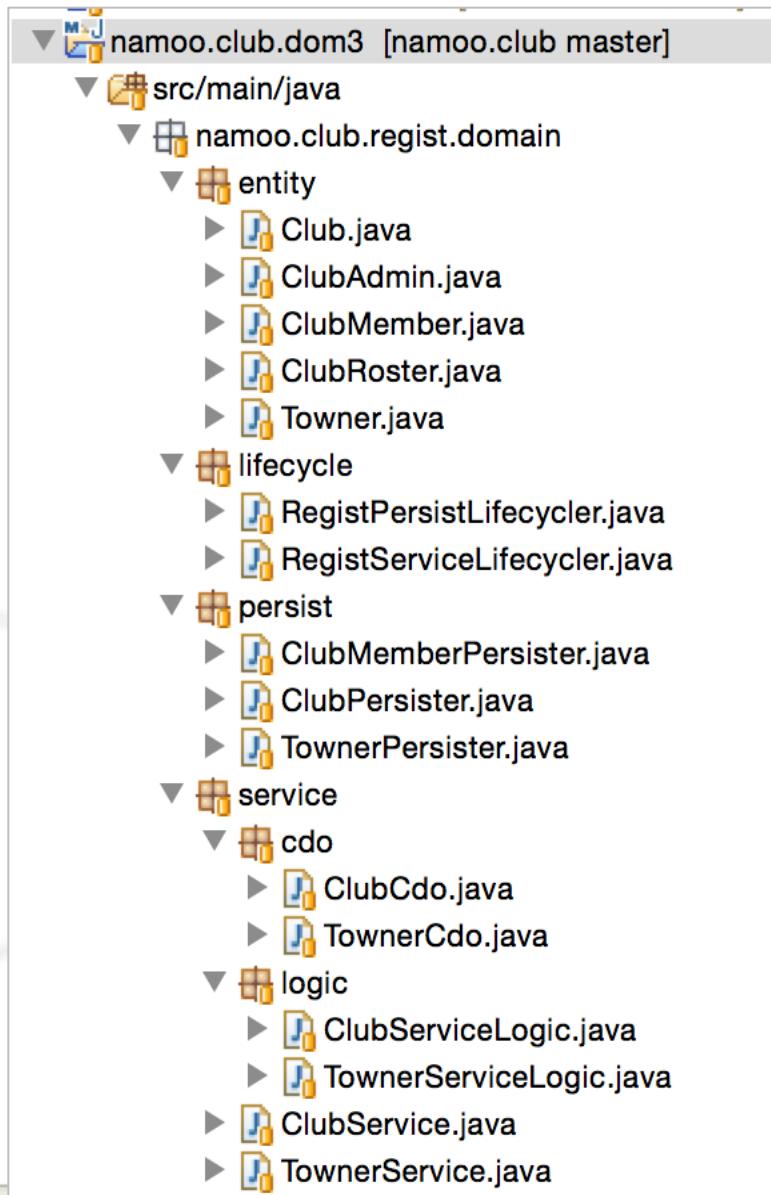
    @Column(nullable = true)
    private String photoId;
```

# PURE 도메인 컴포넌트(1/15)

- ✓ 처리 로직이 “프로세스 POJO 로직”으로 구성되어, 어떤 플랫폼에서도 로직을 사용할 수 있습니다.
- ✓ 지속성 처리가 외부의 구현체로 설계하여, 어떤 지속성 구현체가 오든 기술 중립적인 구성 요소는 영향을 받지 않습니다.
- ✓ 가장 이상적인 형태의 컴포넌트 구성이며, 도메인 객체는 지속성이 아닌 외부로 공유되는 구조입니다.
- ✓ 도메인 객체 모델을 오픈하여 공유함으로써 DDD의 “Ubiquitous Language”로 사용합니다.



# PURE 도메인 컴포넌트(2/15)



```
public class ClubServiceLogic implements ClubService {
    //
    private ClubPersister clubPersister;
    private ClubMemberPersister clubMemberPersister;
    private TownerPersister townerPersister;

    /**
     * @param RegistPersistLifecycle lifecycler
     */
    public ClubServiceLogic(RegistPersistLifecycle lifecycler) {
        //
        this.clubPersister = lifecycler.getClubPersister();
        this.clubMemberPersister = lifecycler.getClubMemberPersister();
        this.townerPersister = lifecycler.getTownerPersister();
    }

    /**
     * @Override
     */
    public String registerClub(ClubCdo clubCdo) throws NamooClubException {
        //
        if (clubPersister.isExistClubByName(clubCdo.getClubName())) {
            throw new NamooClubException("Already existed club --> " + clubCdo.getClubName());
        }

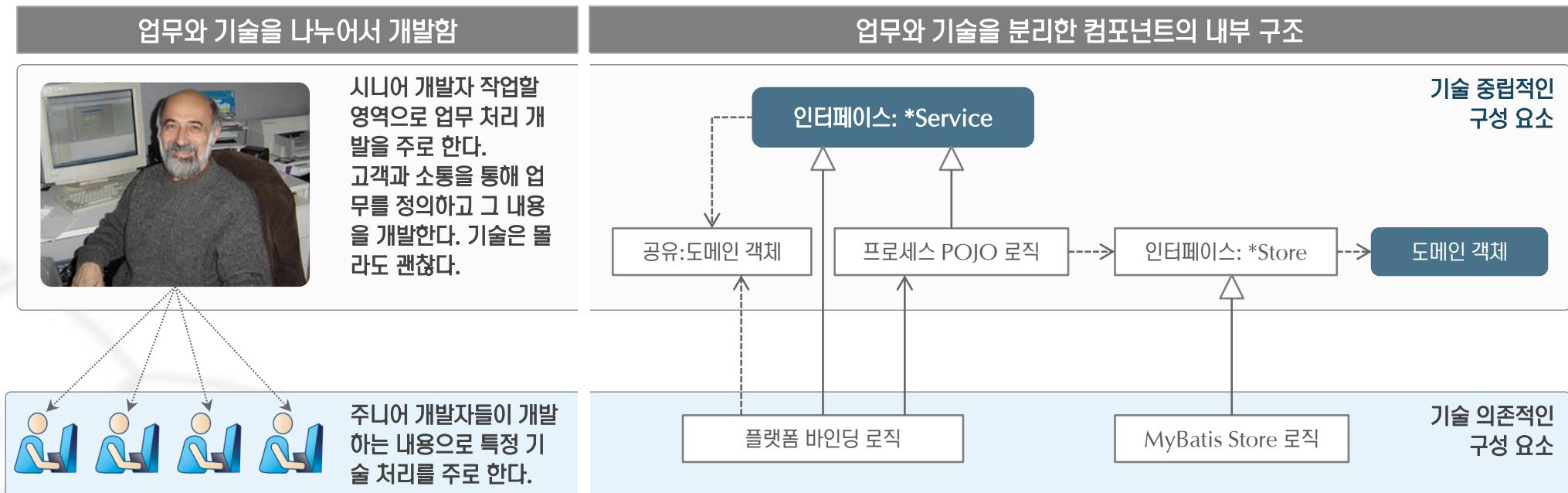
        Towner towner = null;
        try {
            towner = townerPersister.retrieveByEmail(clubCdo.getAdminEmail());
        } catch (EmptyResultException e) {
            throw new NamooClubException(e);
        }

        Club club = clubCdo.createDomain(new ClubAdmin(towner));
        clubPersister.create(club);
        ClubMember clubMember = new ClubMember(club, towner);
        clubMemberPersister.addMember(club.getId(), clubMember);

        return club.getId();
    }
}
```

# PURE 도메인 컴포넌트(3/15) – 업무와 기술 분리

- ✓ 컴포넌트 설계를 통해서 개발자의 역할 배정을 효율적으로 할 수 있습니다. ← 멋지고 경제적인 설계 아이디어
- ✓ 컴포넌트를 업무를 다루는 모듈과 기술에 종속적인 모듈로 나눈다면, 의외의 아주 큰 소득(예산/인력)이 있습니다.
- ✓ 시니어 개발자는 새로운 기술에 약해서 개발에 어려움을 겪고, 주니어 개발자는 업무에 약해서 개발에 어려움을 겪습니다.
- ✓ 결국 시니어 개발자는 개발에서 손을 떼고, 업무를 잘 모르는 주니어 개발자가 개발을 주도합니다. ← 품질 문제 발생

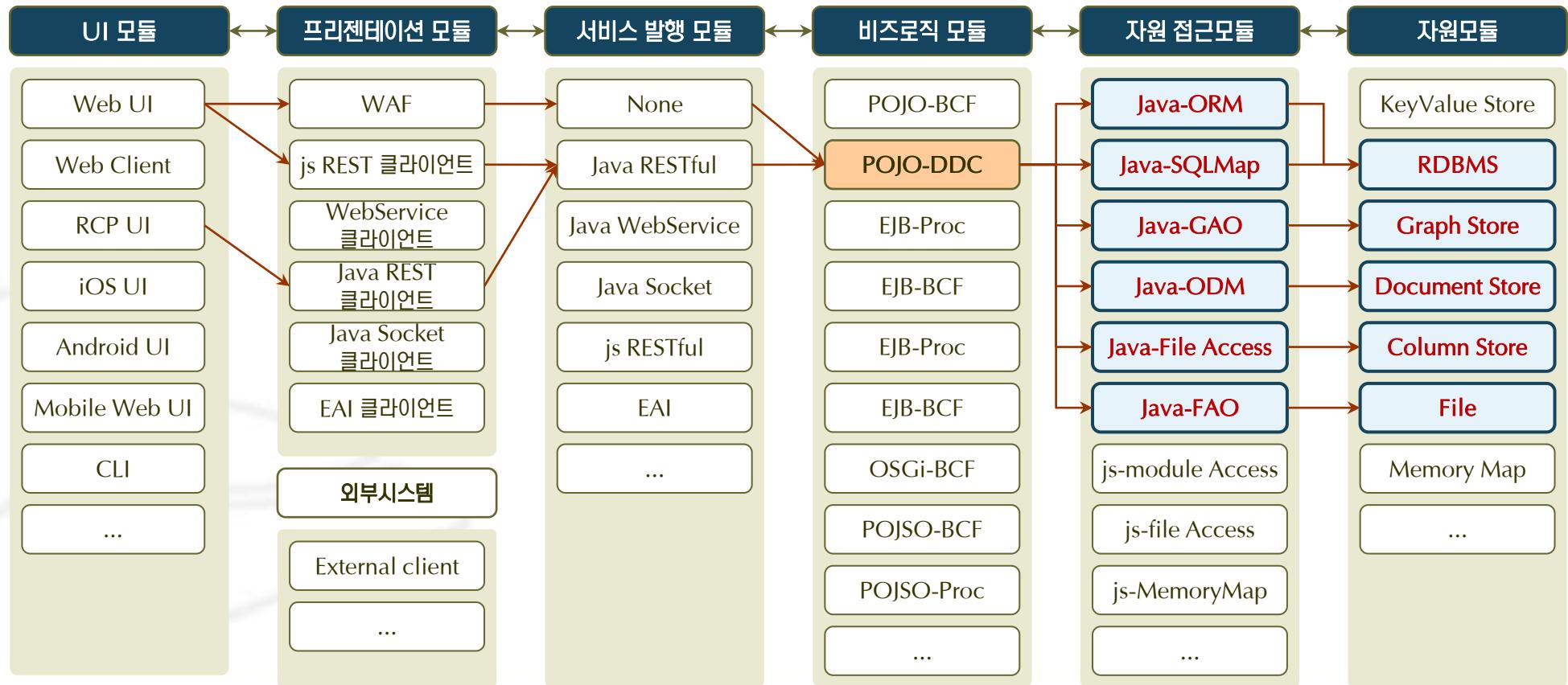


정확한 업무 이해를 바탕으로 하는 개발과 세밀한 기술을 바탕으로 하는 개발을 나누어서 진행할 수 있습니다.

업무와 기술 두 가지 모두에서 비약적인 품질 향상을 얻을 수 있고, 엔지니어를 활용도를 높일 수 있습니다.

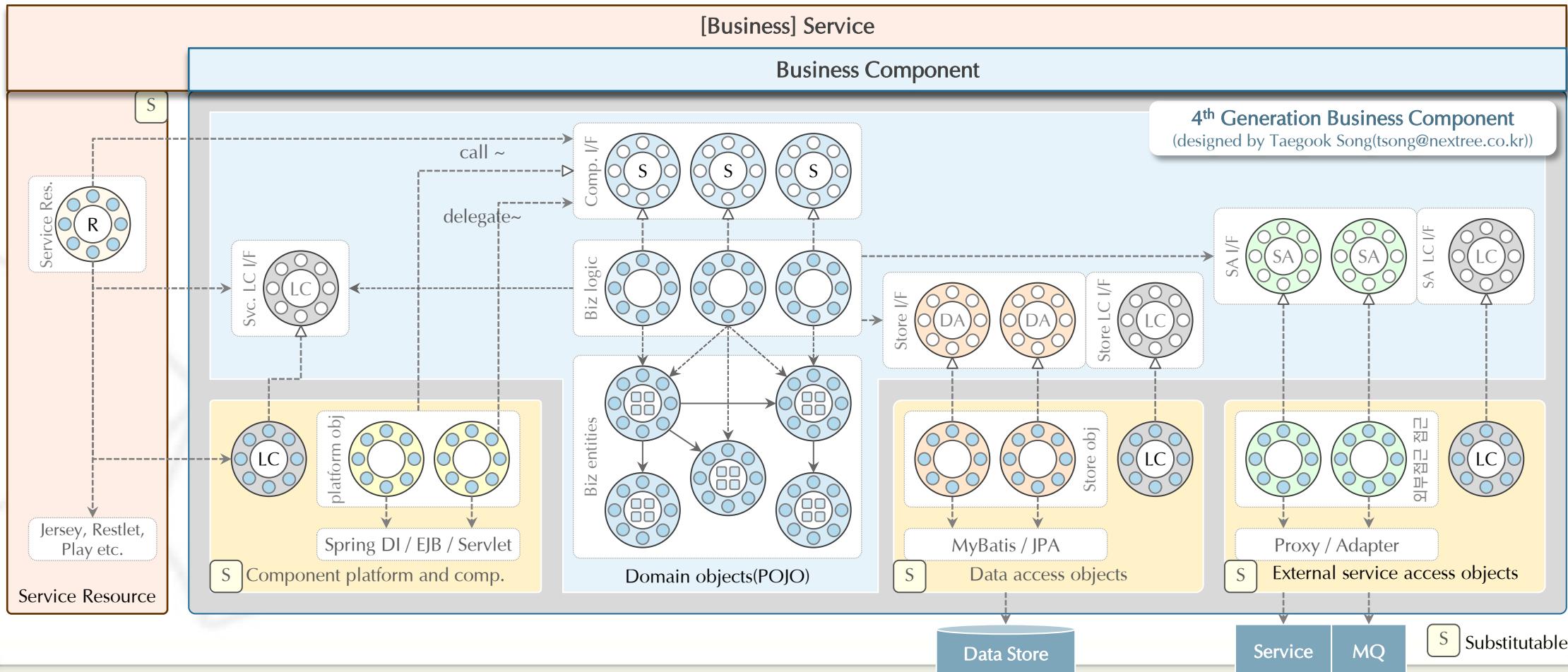
# PURE 도메인 컴포넌트(4/15) – 저장소로부터 자유로움

- ✓ Big data 시대에는 데이터의 특성에 따른 최적의 저장 방식을 찾아야 합니다.
- ✓ 기존의 Relational Database Management System는 데이터 관리 + 데이터 저장의 역할을 동시에 수행했습니다.
- ✓ 데이터의 특성에 따른 최적화된 처리를 위해 관리(management)는 빼고 저장(store)에 집중하고 있습니다.



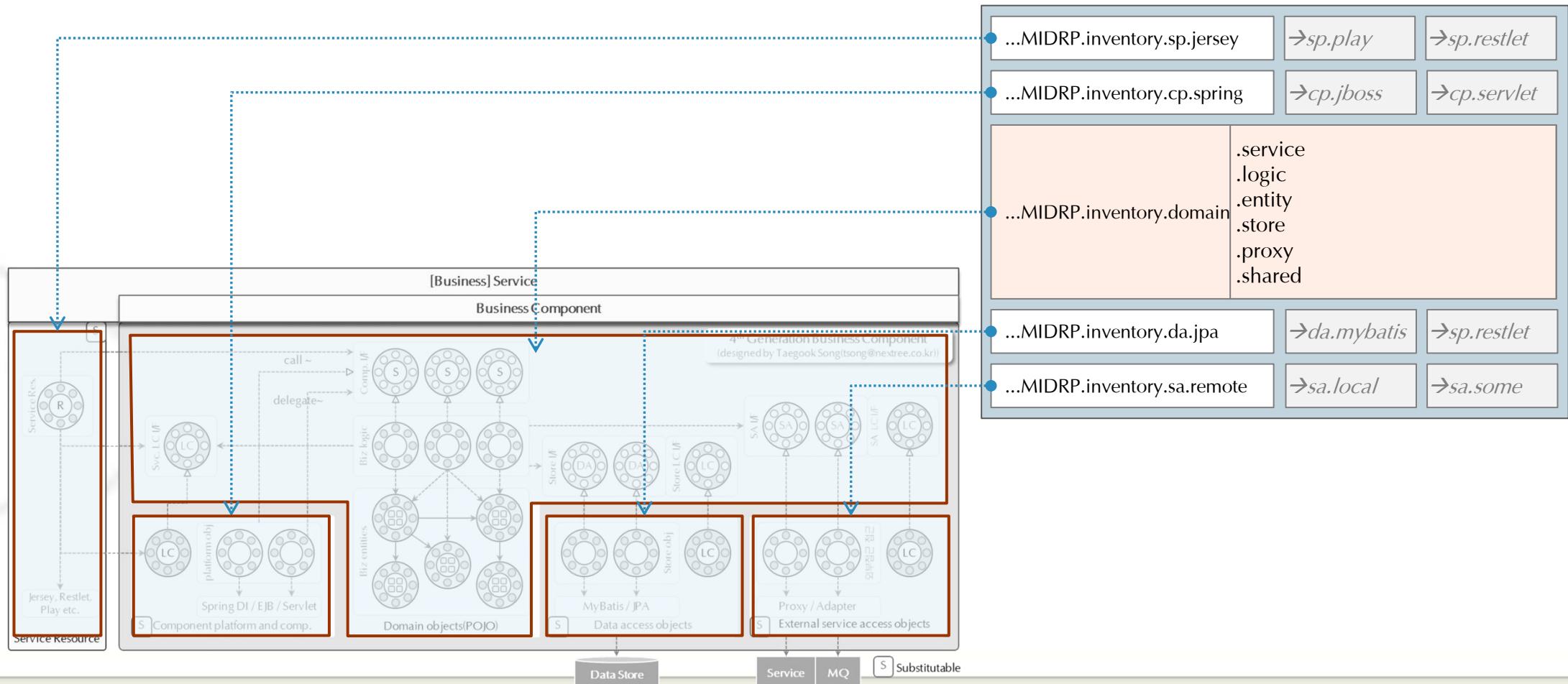
# PURE 도메인 컴포넌트(5/15) – 시각적인 표현

- ✓ 컴포넌트 구조를 시각적으로 표현하여 보면 다음과 같습니다.
- ✓ 컴포넌트와 서비스의 경계를 명확하게 볼 수 있습니다. 서비스는 인터페이스 퍼블리싱 임을 알 수 있습니다.
- ✓ 교제 가능한 부분과 그렇지 않은 부분을 명확하게 알 수 있습니다.



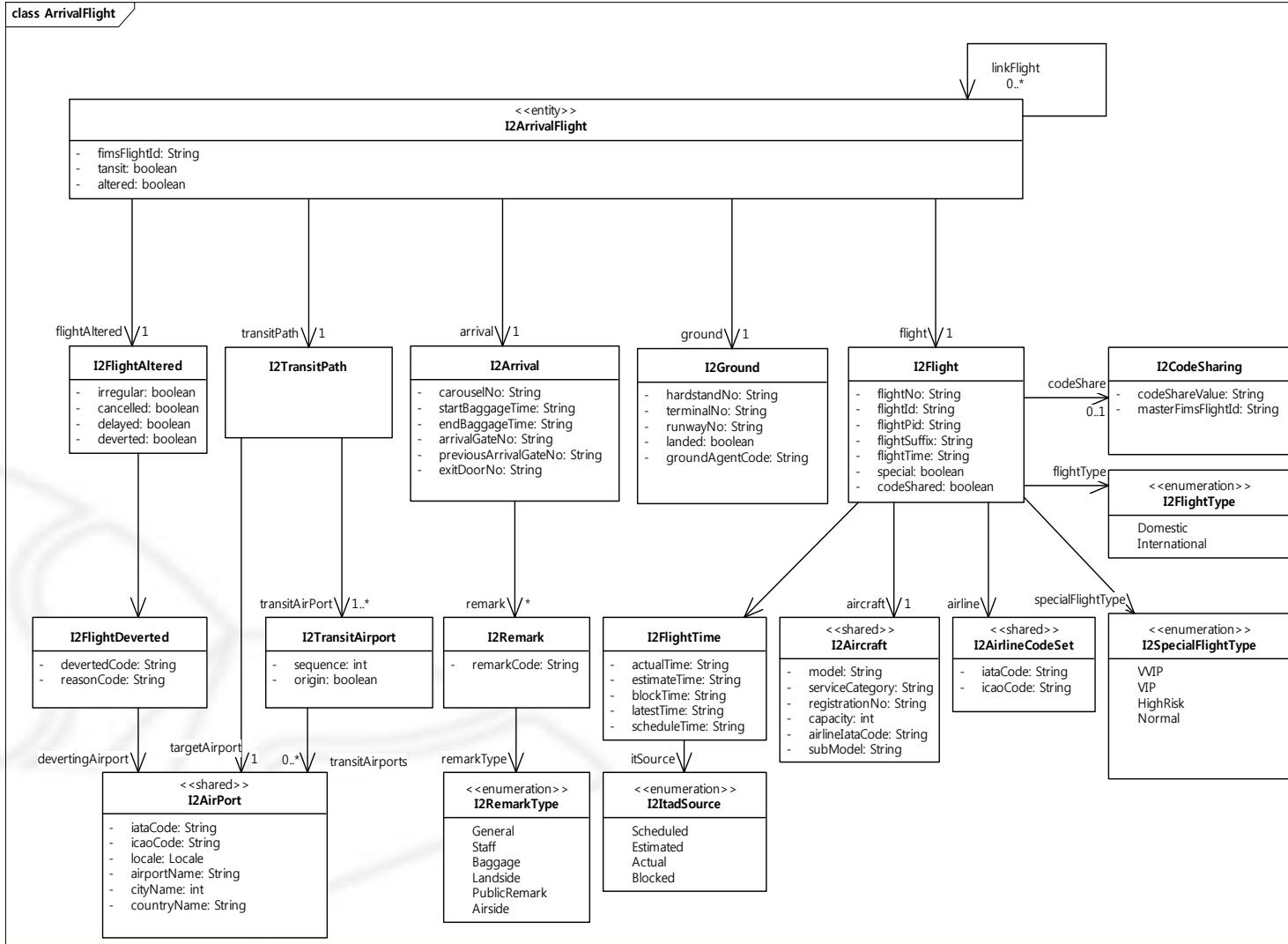
# PURE 도메인 컴포넌트(6/15) – 패키지 맵핑

- ✓ 컴포넌트 내부 구조는 개발 시점의 프로젝트 구성과 패키지 구성과 직접 연결되어 있습니다.
- ✓ 프로젝트 구성은 개발 접근방법, 팀 구성, 구성원 기술 특성, 관리 용이성 등을 반영하여야 합니다.
- ✓ Pure 도메인 컴포넌트는 다양한 프로젝트 구조를 지원할 수 있습니다.

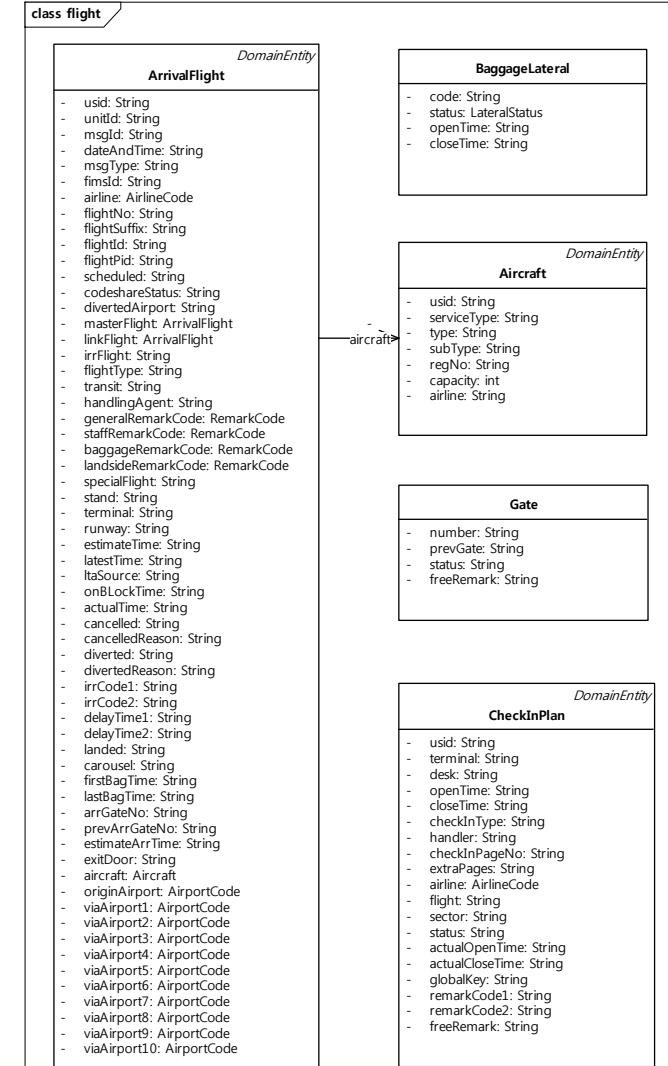


# PURE 도메인 컴포넌트(7/15) – 도메인 모델/데이터 모델 1

Domain model for Arrival Flight

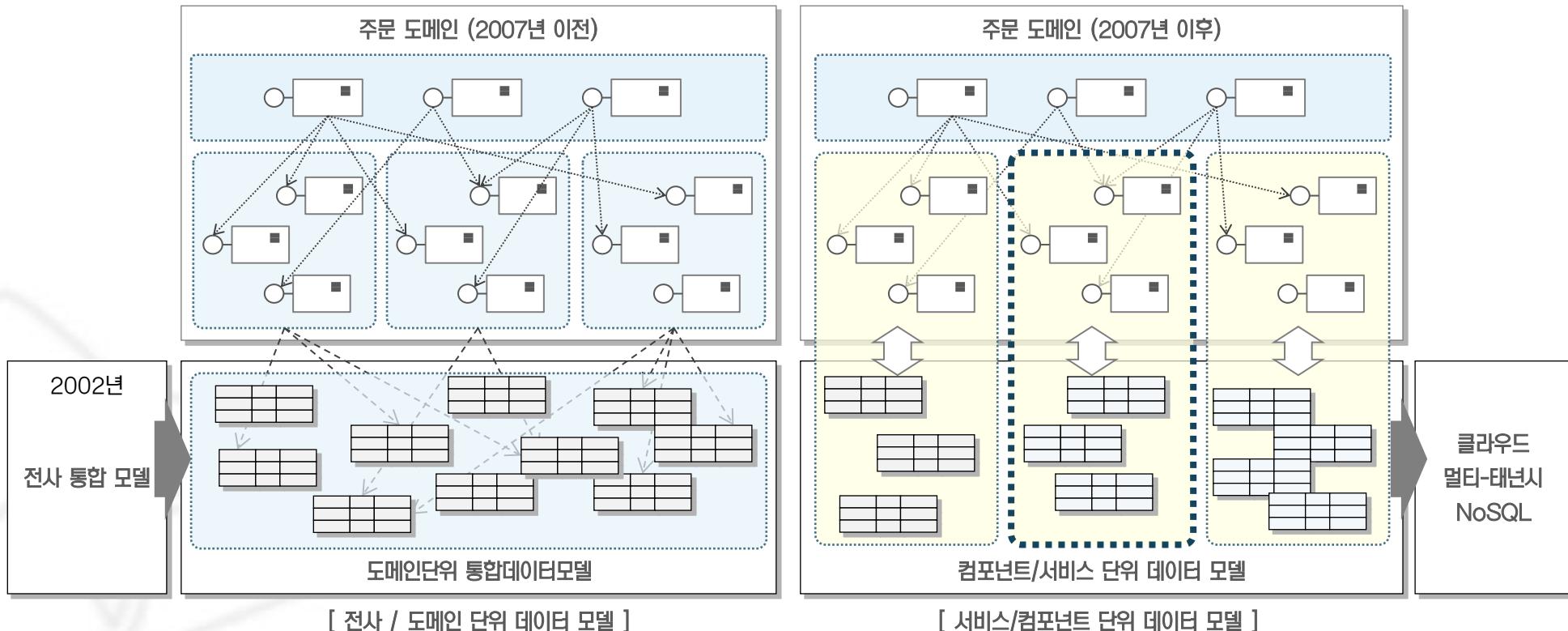


Data model for Arrival Flight



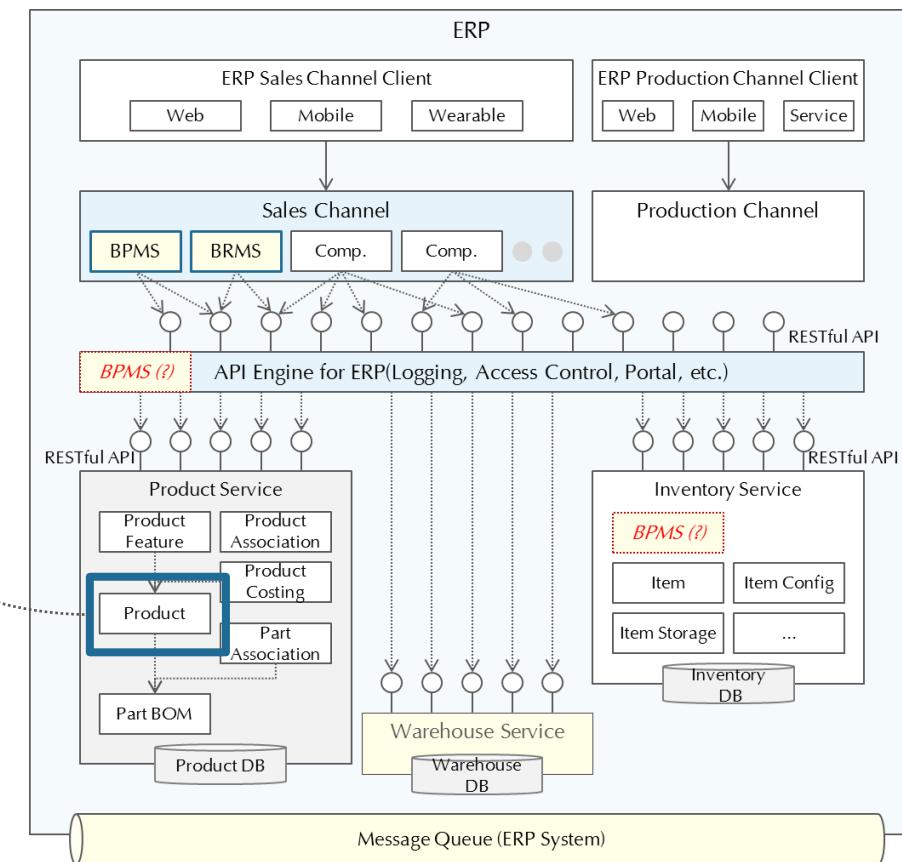
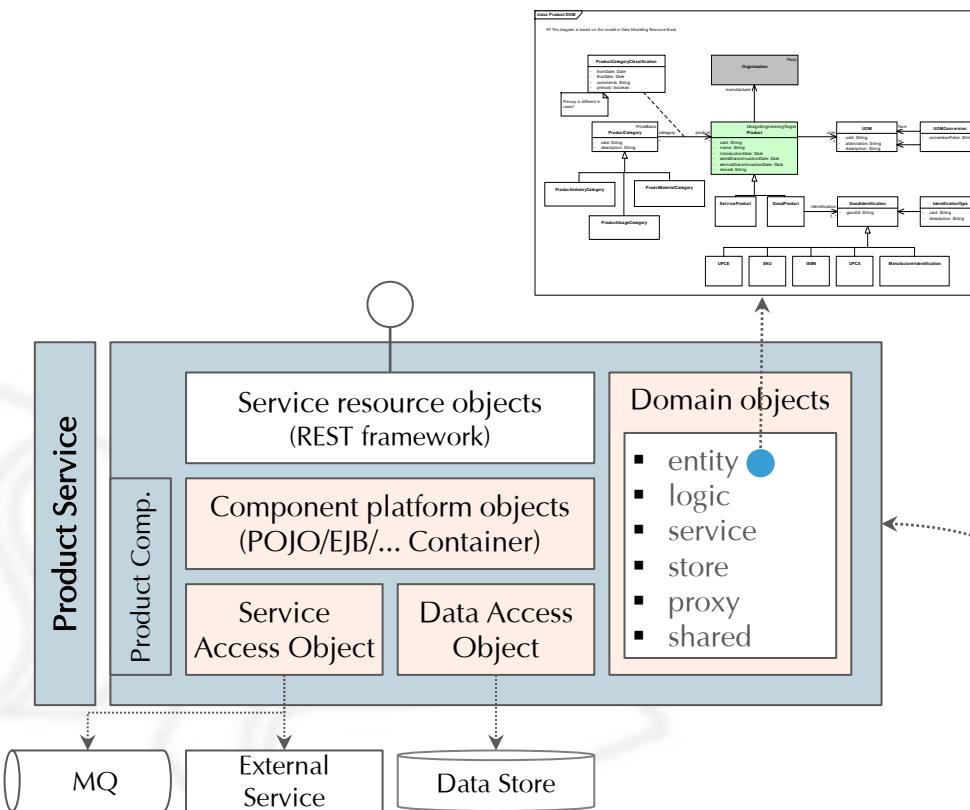
# PURE 도메인 컴포넌트(8/15) – 도메인 모델/데이터 모델 2

- ✓ AS-IS 시스템의 설계는 전사 통합 데이터 모델 사상 위에 비즈니스 컴포넌트를 올린 형태입니다.
- ✓ 전사 통합데이터모델 → 도메인 단위 통합모델 → 서비스/컴포넌트 단위 모델 순으로 시스템 설계가 진화하고 있습니다.
- ✓ 올바른 컴포넌트를 구성하기 위해서 서비스/컴포넌트 단위 데이터 모델로 변경하여야 합니다.



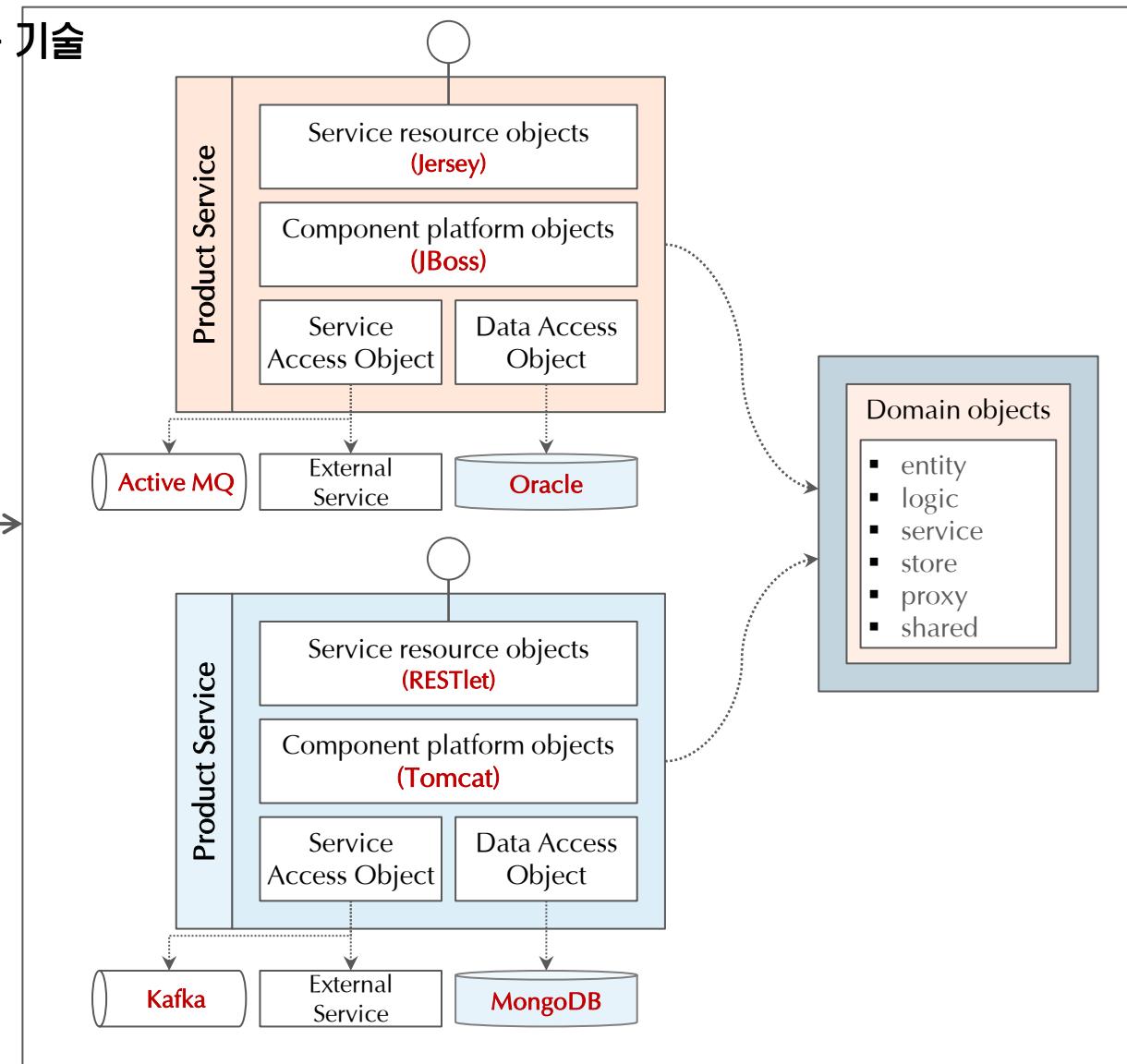
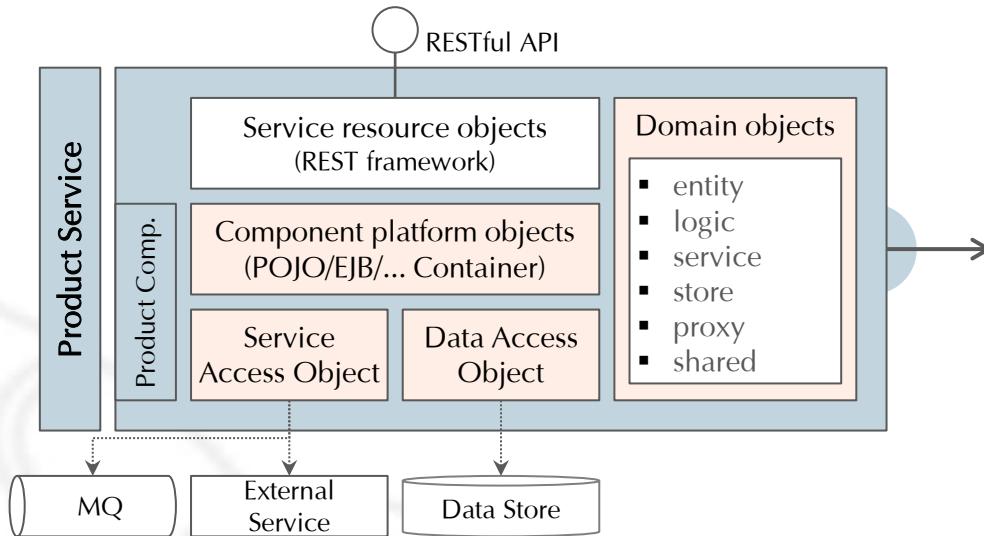
# PURE 도메인 컴포넌트(9/15) – Close up 1

- ✓ 대규 시스템 속에서 컴포넌트와 서비스의 위치를 확인하여 봅니다.
- ✓ 그 속에서 엔티티 클래스들의 위치 또한 확인함으로써, 시스템, 컴포넌트, 서비스를 줌-인, 줌-아웃 하여 봅니다.
- ✓ 현재의 컴포넌트는 그대로 서비스(또는 마이크로서비스)로 확장하는 것을 고려하여 설계합니다.



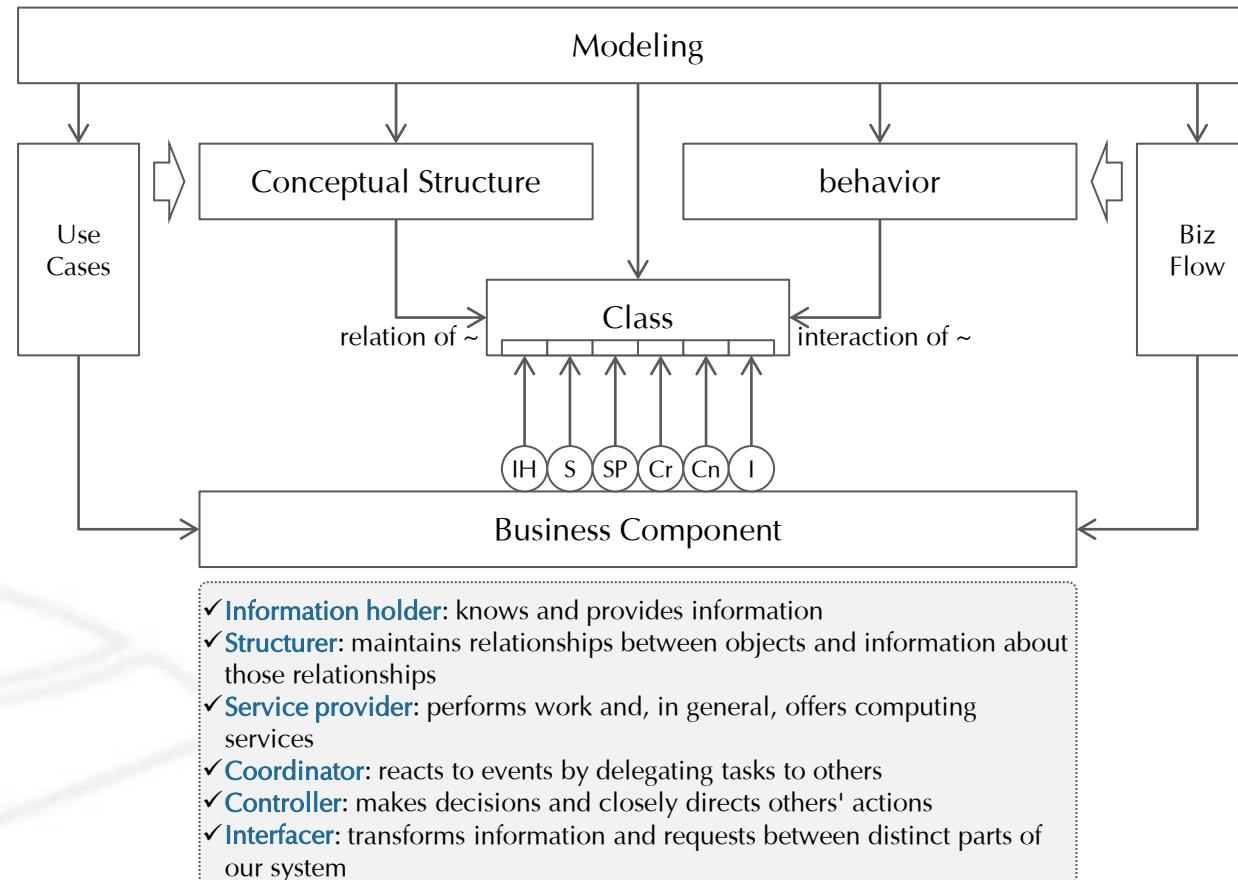
# PURE 도메인 컴포넌트(10/15) – Close up 2

- ✓ 저장 기술과 서비스 발행 기술, 그리고 컴포넌트 플랫폼 기술은 언제든지 교체가 가능하도록 설계하여야 합니다.



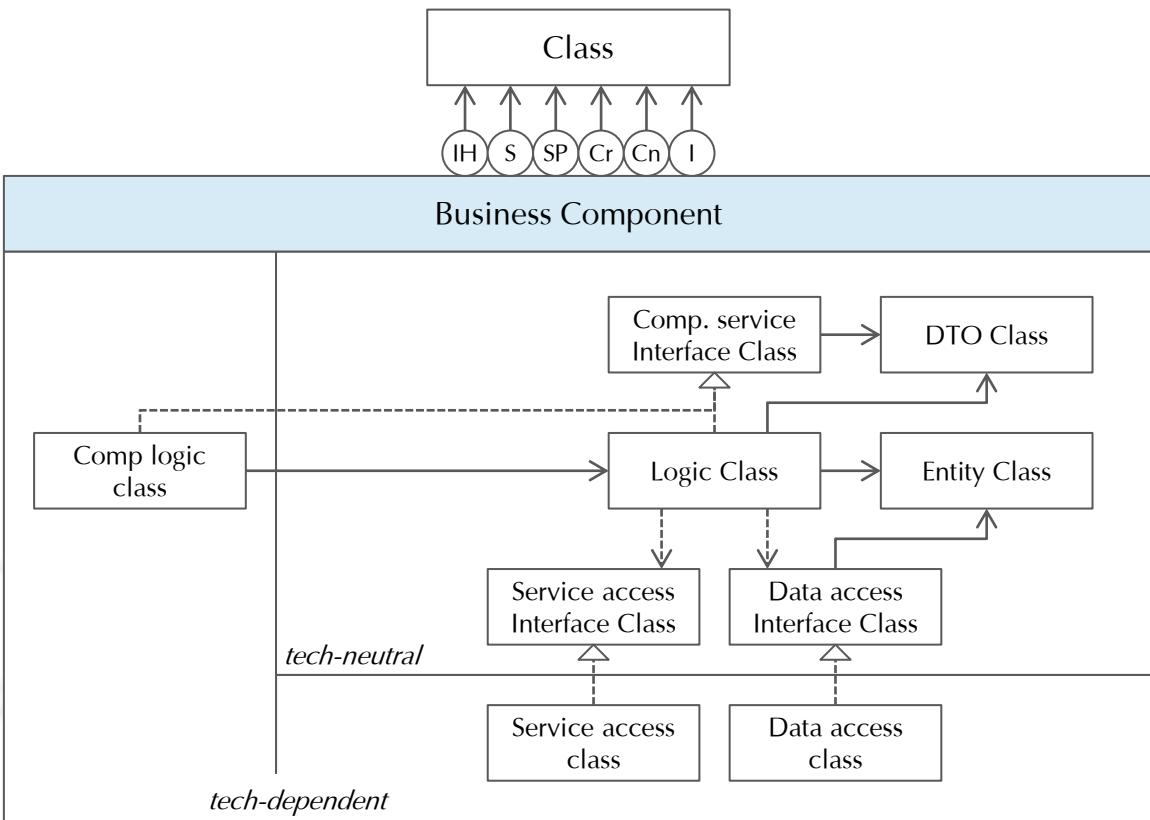
# PURE 도메인 컴포넌트(11/15) – 진화 1

- ✓ 컴포넌트의 구조를 한 단계 더 발전 시키기 위해서 원점에서 다시 생각하여 봅시다.
- ✓ 컴포넌트를 구성하는 클래스는 어떤 종류들이 있는지 생각해 봅니다. → Object Design 참조
- ✓ 컴포넌트는 모델링에서 어떤 관점들을 유지하며 식별하고 명세하고 구조를 설계해야 합니까?



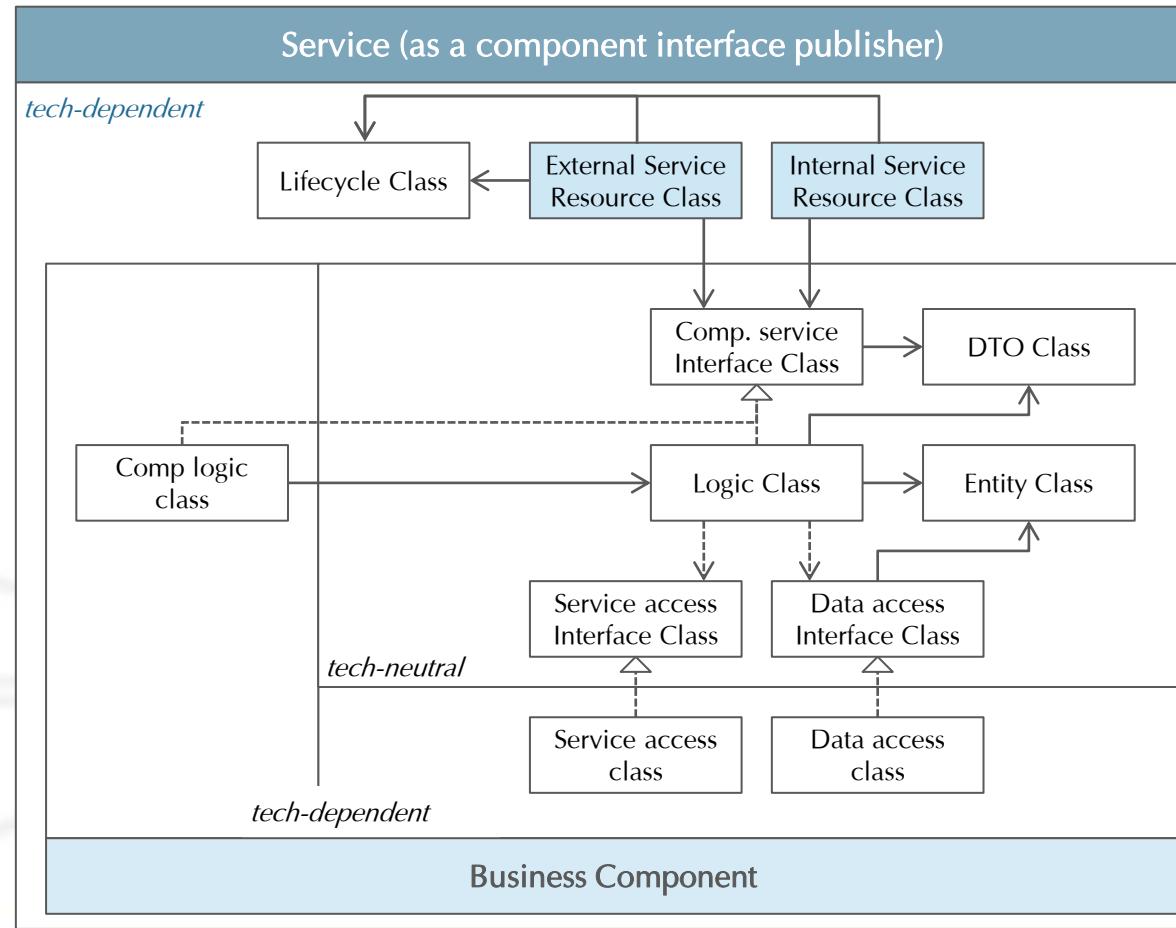
# PURE 도메인 컴포넌트(12/15) – 진화 2

- ✓ 컴포넌트 구조가 프로젝트 구조와 패키지 구조에 직접 영향을 줍니다.
- ✓ 따라서, 아키텍트와 도메인 모델러는 협업을 통해서 이 구조를 설계해야 합니다.
- ✓ 컴포넌트 플랫폼 관련 클래스를 비즈니스 로직으로부터 분리하는 방법을 생각해 봅니다.

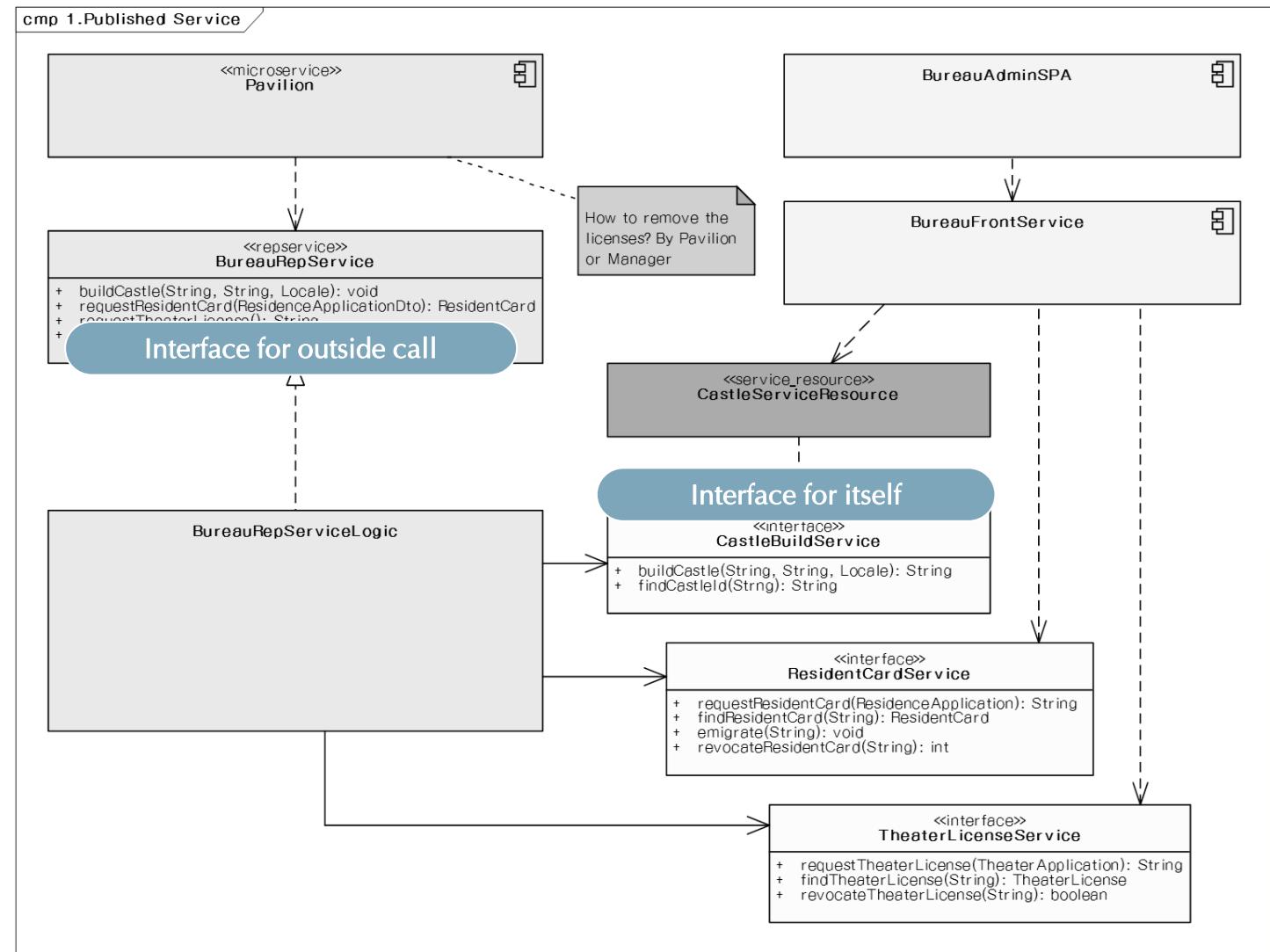
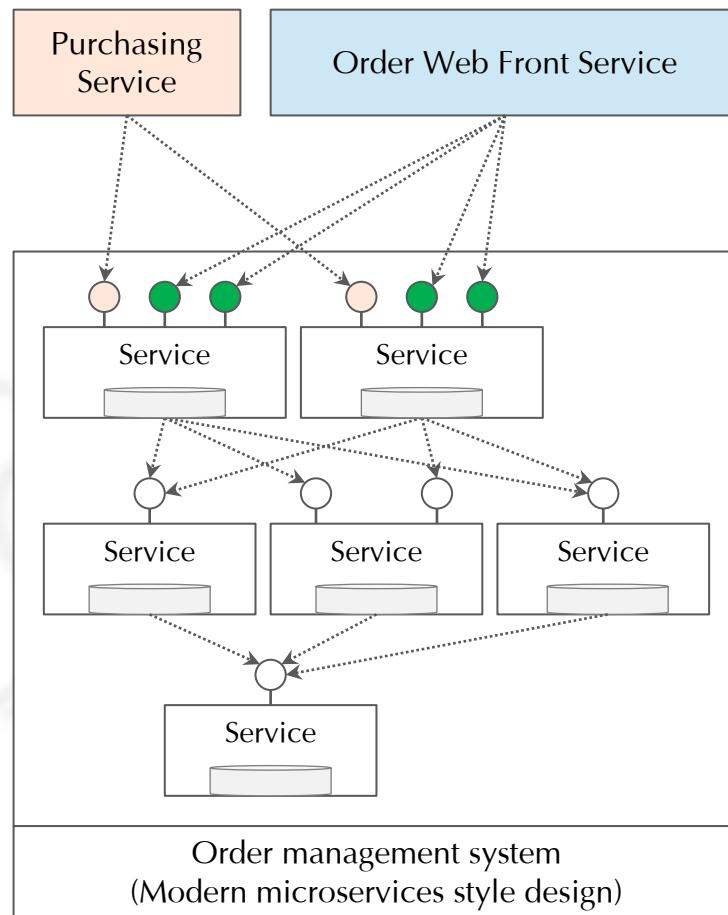


# PURE 도메인 컴포넌트(13/15) – 진화 3

- ✓ 컴포넌트에 서비스 역량을 추가함으로써 서비스가 될 때, 어떤 인터페이스를 추가로 더 생각할 수 있습니까?
- ✓ 자체 인터페이스는 무엇이고 외부 인터페이스는 무엇입니까?
- ✓ 두 개의 인터페이스를 하나로 사용할 때의 장점과 단점은 무엇입니까?

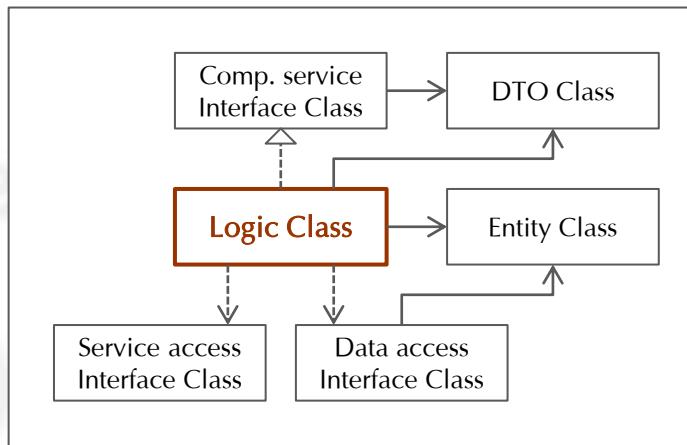


# PURE 도메인 컴포넌트(14/15) – 진화 4



# PURE 도메인 컴포넌트(15/15) – 진화 5

- ✓ 외부 인터페이스를 호출하는 로직 클래스를 생각해 볼 수 있습니다.
- ✓ 로직 클래스는 직접 호출하지 않고 Proxy 클래스를 통해서 호출함으로써 의존성을 제거합니다.
- ✓ 데이터 매팅 레이어가 있듯이, 서비스 매팅 레이어를 둘으로써 외부와의 느슨한 결합 관계를 유지합니다.

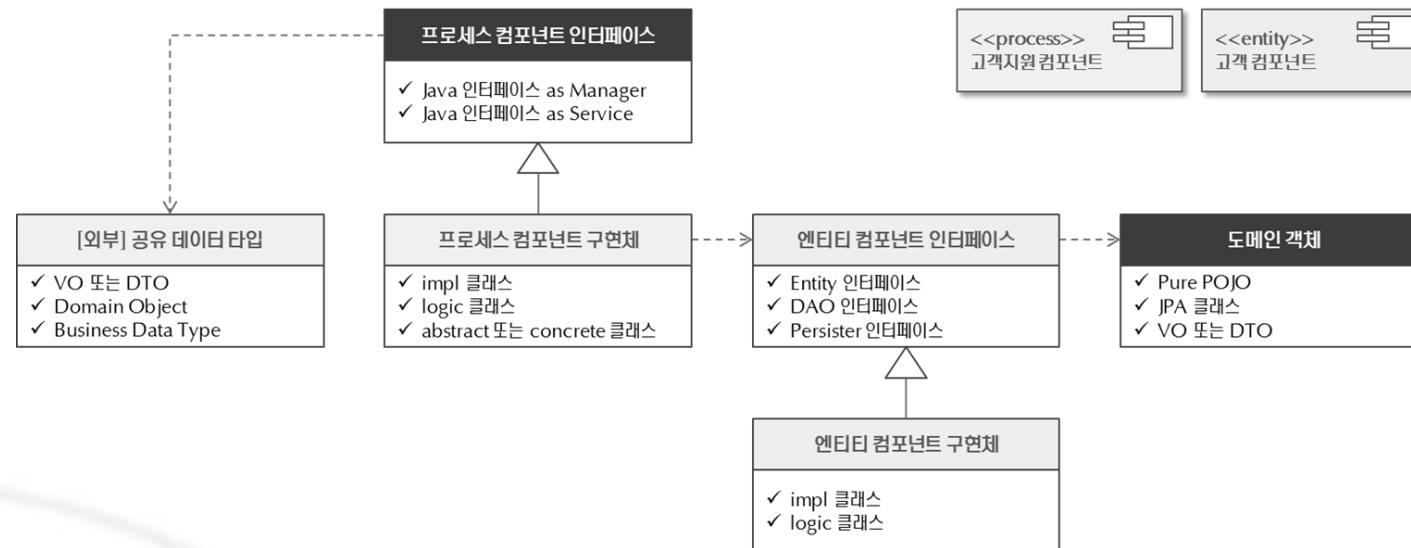


```
@Override  
public String requestTheaterLicense(LicenseRequest licenseRequest) {  
    //  
    String guildId = guildJoinLogic.join(licenseRequest);  
    LicenseIssuePolicy policy = BureauContext.getInstance().getLicenseIssuePolicy();  
  
    String naraId = BureauContext.getInstance().getNaraId();  
    long theaterSequence = sequenceStore.nextTheaterSequence(naraId);  
    String metroId = BureauIdPolicy.initTheaterId(theaterSequence);  
  
    if (policy.isAutomatic()) {  
        TheaterLicense license = TheaterLicense.newInstance(metroId, guildId);  
        String metroName = licenseRequest.getCompanyName();  
        Contact admin = licenseRequest.getContact();  
        townProxy.buildMetro(metroId, metroName, admin);  
        theaterLicenseStore.create(license);  
    } else {  
        // TODO more  
        TheaterApplication application = TheaterApplication.newInstance(guildId);  
        long requestSequence = sequenceStore.nextTheaterRequestSequence(naraId);  
        application.setUsid(BureauIdPolicy.initTheaterRequestId(requestSequence));  
        theaterApplicationStore.create(application);  
    }  
  
    return metroId;  
}
```

*TheaterServiceLogic.java*

# 요약

- ✓ 아키텍처 설계에서 컴포넌트 구조 설계는 아주 중요합니다. 비즈니스를 담는 그릇입니다.
- ✓ 컴포넌트 구조에 따라 품질(확장성, 변경용이성, 유지보수성 등)과 개발 공수에 차이가 있습니다.
- ✓ 컴포넌트 구조는 개발 팀의 기술 수준, 개발 환경, 고객 니즈 등에 따라 서로 다릅니다.
- ✓ 다양한 컴포넌트 구조에 대한 올바른 이해가 있어야, 상황에 맞는 컴포넌트 구조를 설계할 수 있습니다.



✓ Q&A

✓ 토론

## 감사합니다...

- ❖ 송태국 ([tsong@nextree.co.kr](mailto:tsong@nextree.co.kr))
- ❖ 넥스트리컨설팅(주) 대표 컨설턴트
- ❖ 넥스트리소프트(주) 부사장
- ❖ [www.nextree.co.kr](http://www.nextree.co.kr)