소프트웨어 제품 계열에 적용되는 SW 아키텍처

배 준 수

ivarbae@2e.co.kr

019-215-7086

2e Consulting
Efficiency & Effectiveness

Ⅰ. **Product Line**의 정의

Ⅱ. **PL**에서의 **Architecture** 요구 사항 및 **Architect**의 역할

Ⅲ. 아키텍처 설계 프로세스

Ⅳ. 아키텍처 설계 패턴

Ⅴ. 사례로 본 **Product Line**의 효과

Product Line의 정의

I

제품 계열(**Product Line**)이란?

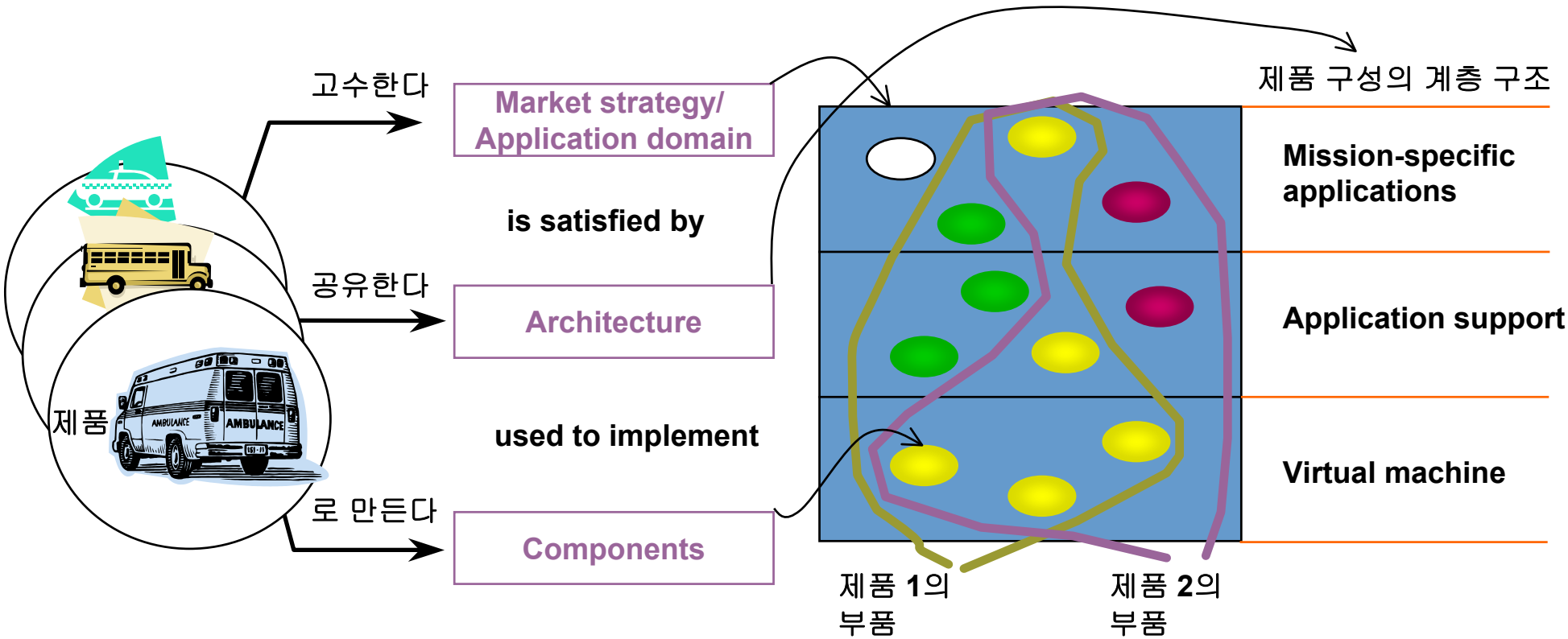
Product Line 도입 근거

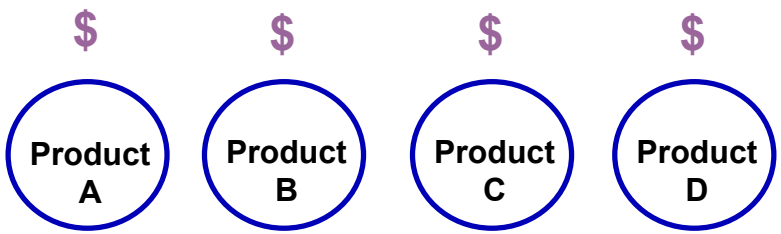
The Key Concepts

Common Asset의 종류

제품 계열(Product Line)이란?

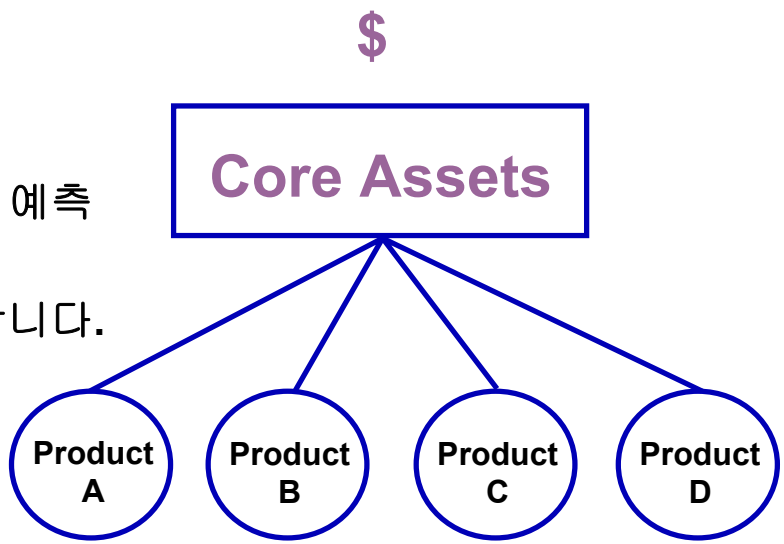
- 제품 계열이란 제품의 묶음(그룹)이며, 특정 업무 영역의 요구 사항을 만족시키는 공통적이며 관리할 수 있는 특성을 공유한다.



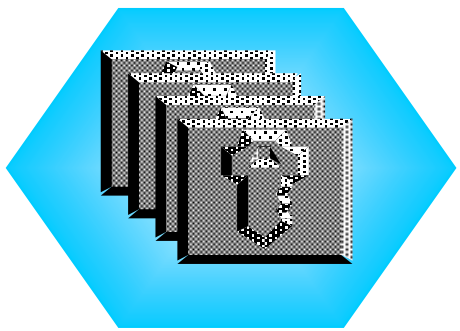


한번에 하나 씩 소프트웨어 제품을 만드는 방식은 여러 사업 기회가 있을 경우 경제적이지 않다.

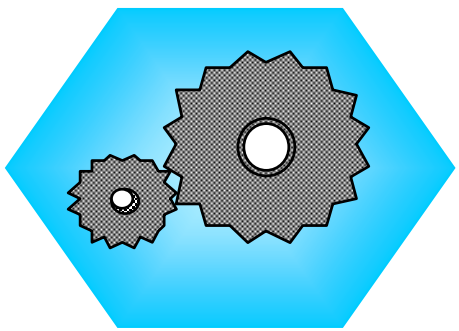
전략적 소프트웨어 재사용으로 예측 가능하며, 저 비용적인 생산 및 유지보수 체계를 수립하여야 합니다.



Use of a Common Asset Base



in Production



of a Related Set of Products

Common Asset의 종류

- requirements and requirements analysis
- domain models
- software architecture and design
- performance engineering
- documentation
- test plans, test cases, and data
- schedules and budgets
- people: their knowledge and skills
- processes, methods, tools, and estimates
- software components and legacy systems

PL에서의 Architecture 요구 사항 및 Architect의 역할

II

아키텍처 정의

아키텍처 중요성

일반적 프로젝트에서 아키텍처 요구 사항

Product Line 사례

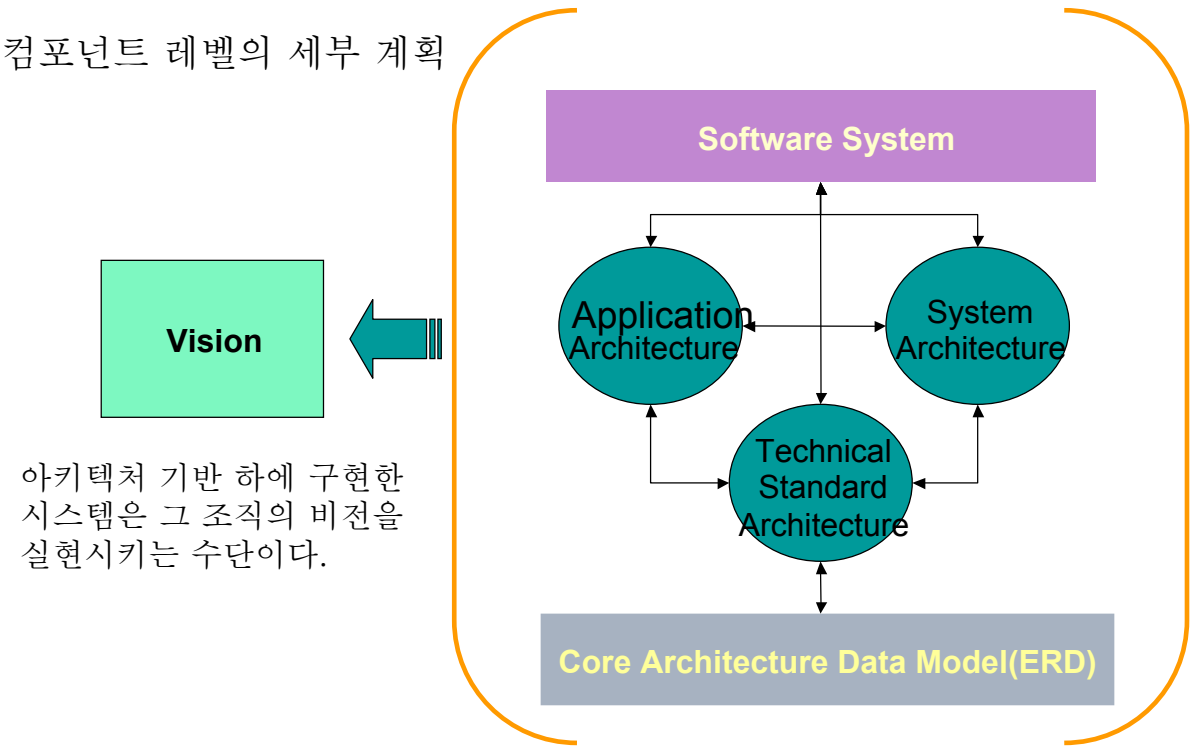
Product Line에서 아키텍처 요구 사항

Architect의 역할




아키텍처 정의

- 컴포넌트들과 그들간의 관계에 대한 구조이며 그 설계와 진화를 지속적으로 관리하는 원칙 및 가이드 라인[DoD(Department Of Defense 1995)]
- 시스템을 구성하는 컴포넌트 및 그들 사이의 관계로 구성된 시스템의 전체적인 구조[EEE Std 1471-2000]
- 시스템의 구조를 설계하고 전개하기 위한 원리 및 지침[TOGAF(The Open Group Architecture Framework)]
- 시스템의 정형화된 명세
- 시스템 구현을 위한 지침을 제공하는 컴포넌트 레벨의 세부 계획



아키텍처 중요성(1)

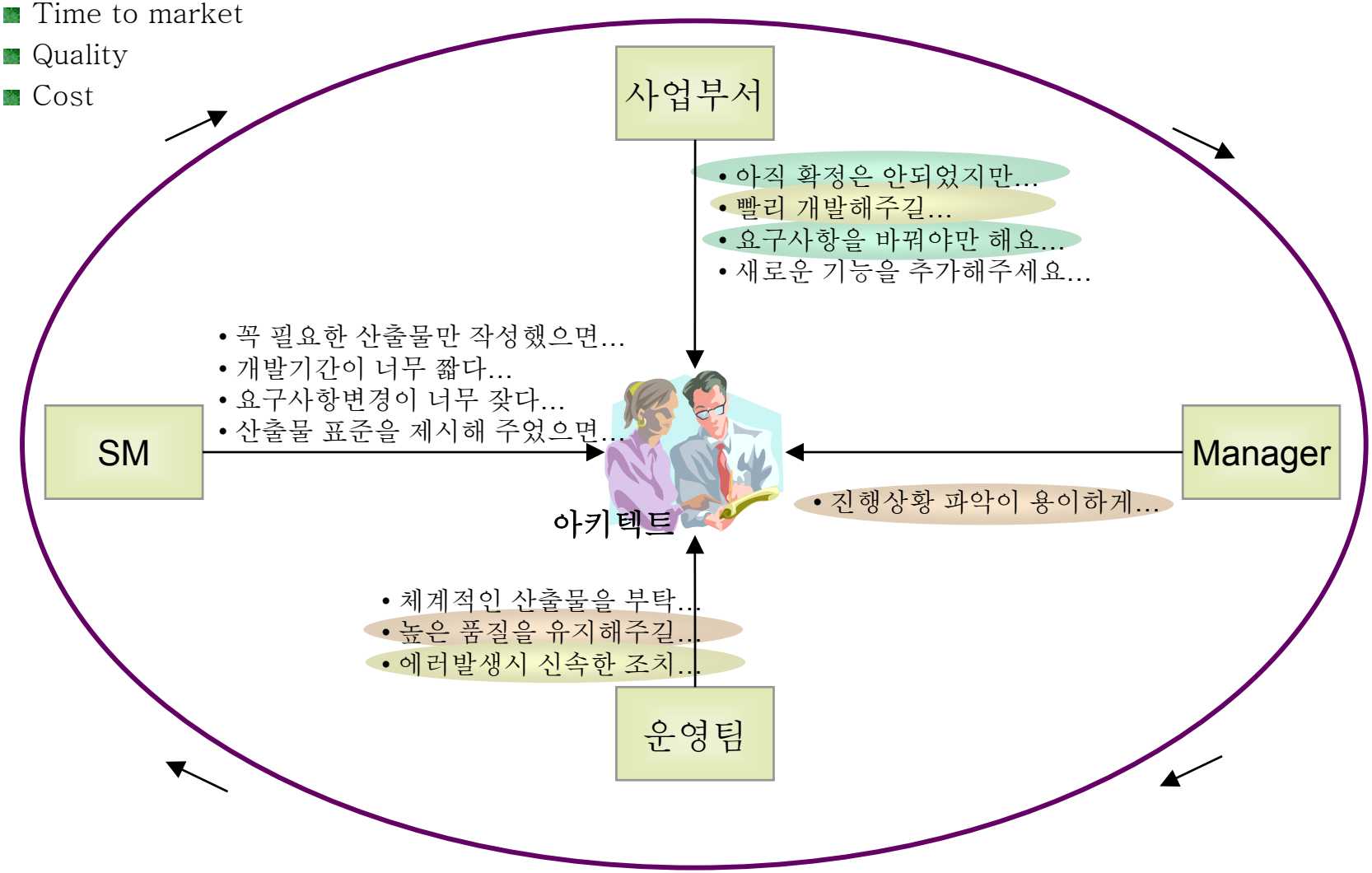
- Architecture는 Stakeholder 사이의 의사소통 수단입니다.
  - Architecture는 시스템의 공통(common)된 abstraction을 나타냄
  - 시스템의 모든 stakeholder들이 상호간의 이해와 조화를 이루고 의사소통을 위한 기본으로 사용됨
- 개발 초기 의사결정에 중요한 역할을 합니다.
  - 구현상의 제약조건을 명시함
  - 조직구조를 반영함
  - 시스템의 품질속성을 명시하고 분석할 수 있게 함
  - 아키텍처를 통해 시스템의 품질을 예측할 수 있게 함
  - 변화를 관리하고 영향을 평가하기 쉬움
  - 진화적 프로토타이핑을 가능하게 함
- Architecture는 초반 디자인 결정사항의 결과입니다.
  - 시스템의 Architecture는 다루어야 할 여러 고려사항을 우선순위를 정할 수 있게 하는 가장 초반 산출물로써 이는 시스템의 품질에 핵심영향을 미친다.
  - 성능과 보안 사이, 유지보수성과 신뢰성 사이, 그리고 현 개발 비용과 향후 개발 비용간의 tradeoff를 명확하게 architecture에 나타낸다.

아키텍처 중요성(2)

- Architecture는 시스템의 추상화로서 재사용할 수 있으며, 재적용 할 수 있습니다.
  - Architecture는 시스템의 구조 및 컴포넌트 간의 연동관계를 비교적 간략하고, 고도화된 모델로 표현한 것이다.
  - 특히 비슷한 요구사항을 갖는 시스템에도 재 적용하기 좋으며 이로써 대단위 재사용 및 software product line 구축에 활용된다.
  - CBD S/W 프로세스와의 논리적인 결합이 가능하다. 분석,설계에서부터 구현 인도, 실행까지 아키텍처가 CBD S/W 프로세스에 녹아있다.
- 비즈니스 전략을 연계하고 재사용 체계를 구축할 수 있게 함
  - 제품계열 조직은 공통 아키텍처를 통해서만 구현할 수 있음
  - 별도로 개발한 컴포넌트 들을 조립하여 규모가 큰 시스템을 구현할 수 있음
  - 가능한 설계 상의 대안들을 명확하게 함
  - 아키텍터를 통해 패턴 등의 템플릿 기반 컴포넌트 개발이 가능함
  - 아키텍처는 조직의 기술성숙도 관리의 근간이 됨
  - Enterprise View를 가질 수 있다. 분할되고 부분적이며 한정된 시각을 가지는 것이 아니라 전체에 대한 View를 가짐으로써 조직의 비전과 전략을 실체화 할 수 있다.

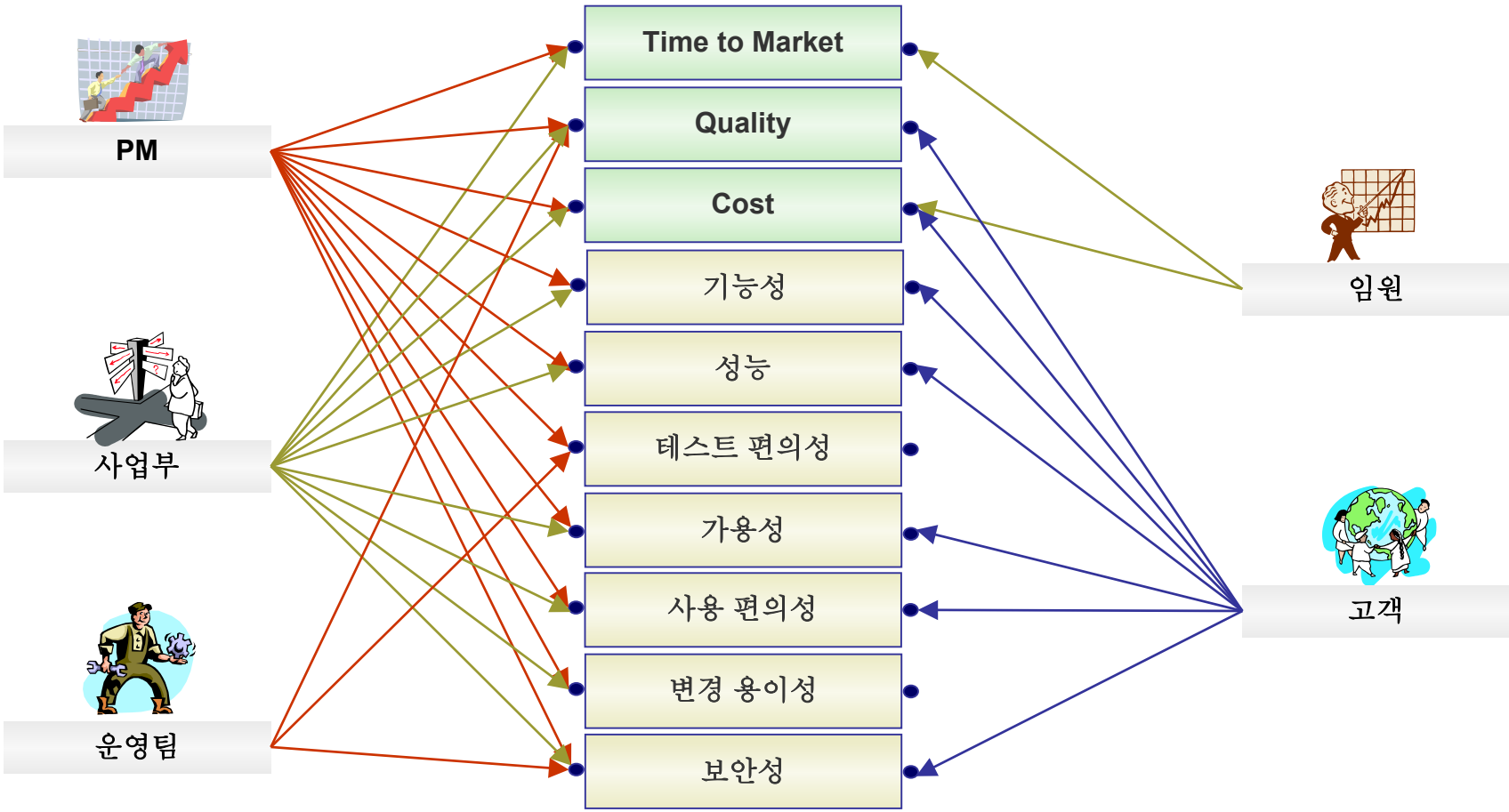
일반적 프로젝트에서 아키텍처 요구 사항

- Overall Business Goal
  - Time to market
  - Quality
  - Cost



일반적 프로젝트에서 아키텍처 요구 사항

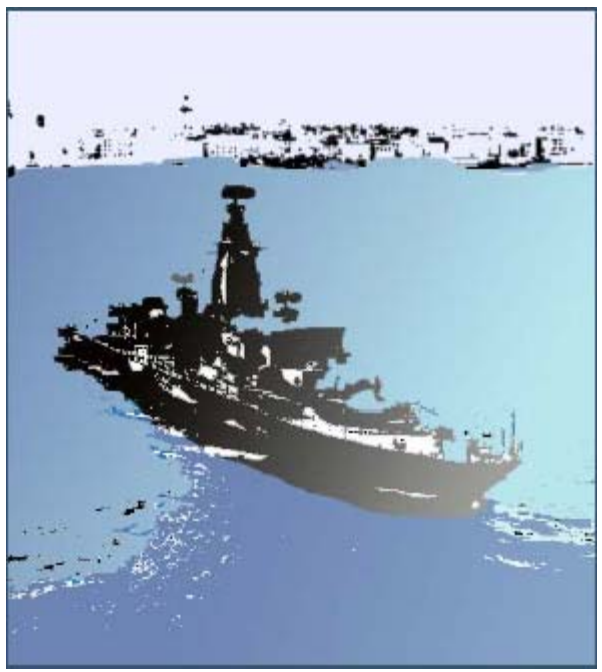
- 여러 이해 당사자들이 가지는 비즈니스 목표 및 품질 요소에 대한 관심은 다음과 같습니다.



Product Line 사례 - CelsiusTech SS2000

- 함선에서의 모든 무기 제어, 명령, 통제, 통신을 처리하는 통합 시스템
- 실 Product의 규모는 해상 및 해저용 함선의 LAN으로 연결된 30~70대의 컴퓨터에서 수행되는 일백만~일백오십만 라인의 Ada 코드로 구성됨.
  - 연안 소형 쾌속 호위함
  - 다목적 순찰선
  - 쾌속 공격선
  - 프리깃함
  - 대양 순시선
  - 잠수함

  - 스웨덴, 덴마크, 오스트리아, 뉴질랜드, 파키스탄, 오만
- 지원 가변성. But... 단일 아키텍처, 단일 Core Asset, 단일 조직으로
  - 각종 무기, 센서
  - 다국어 인터페이스
  - 다양한 하드웨어 : 68020, 68040, RS/6000, ...
  - 다양한 OS : OS2000, IBM AIX, POSIX, Digital Ultrix, ...

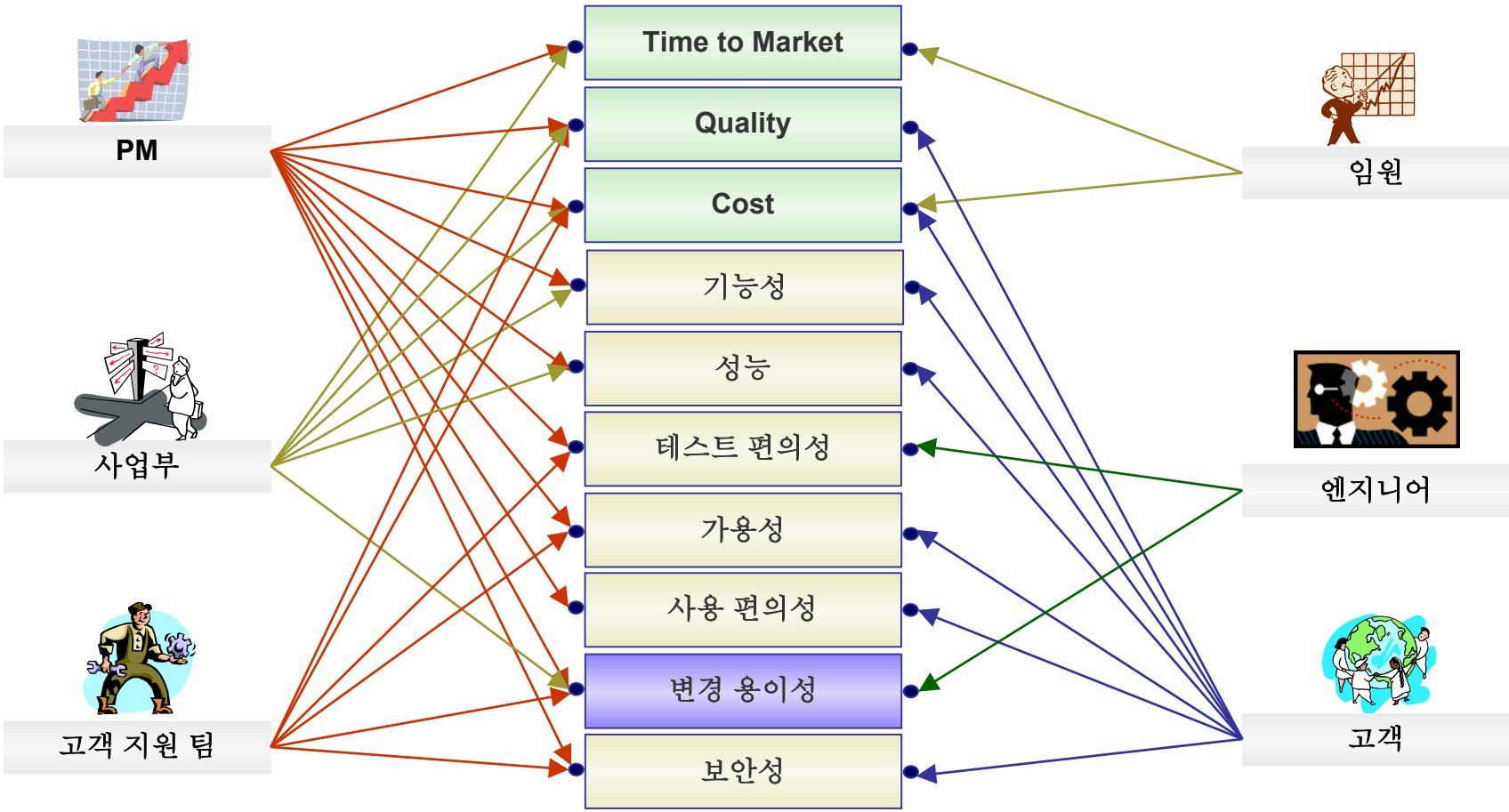


Product Line 사례 - Nokia Mobile Phone

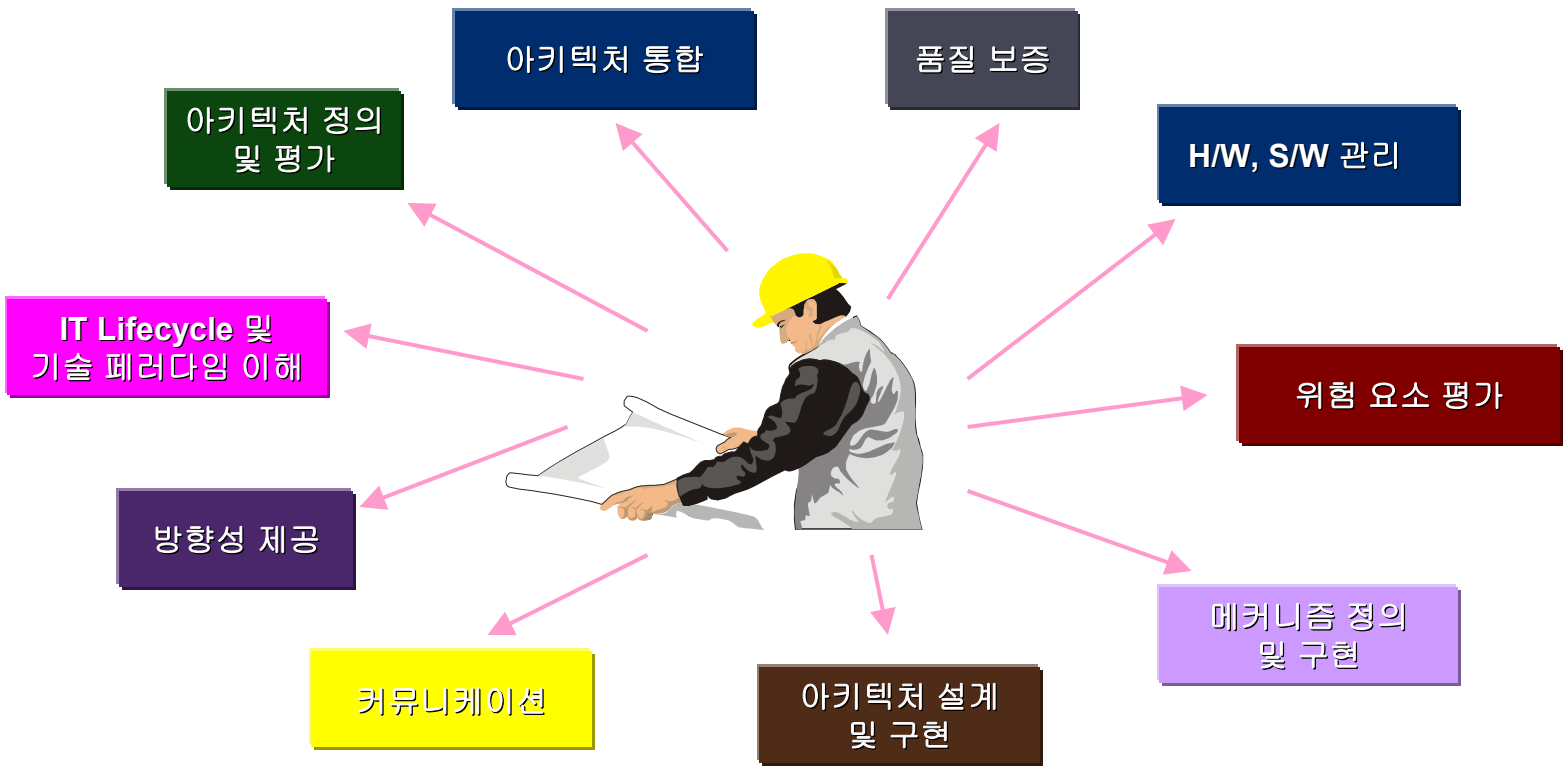
- 연간 약 25~30개의 신제품 출시
- 각 제품별로
  - 다양한 입력 키 구조
  - 다양한 디스플레이
  - 다양한 기능
  - 58개 언어 지원
  - 130개 국에서 사용됨
  - 다양한 통신 프로토콜
  - 옛 버전과의 호환성
  - 기능 설정이 가능해야
  - 출시 후 기능을 변경할 수 있어야

Product Line에서 아키텍처 요구 사항

- 업무 기능, 기술 환경 및 문화적 다양성은 가변성으로 나타나며, 변경 용이성이 핵심 아키텍처 요구사항 임.



- 소프트웨어 아키텍트는 Architecture 정의 및 통합 유지 보수 그리고 프로젝트의 기술적 위험 요소들에 대한 평가와 각 반복 과정의 계획에 따른 연속적 반복과정의 순서와 내용에 대한 정의 등을 포함하고 다양한 설계 구현 통합 그리고 품질 보증 팀(각 팀 내부 QA 및 통제 QA)들에게 위의 결과들을 제공하며 앞으로의 방향성을 제공하는 활동을 함.



아키텍처 설계 프로세스

II

아키텍처 설계 프로세스

품질 속성 식별

품질 시나리오 도출

요구사항 합의 도출

단위 아키텍처 설계 프로세스

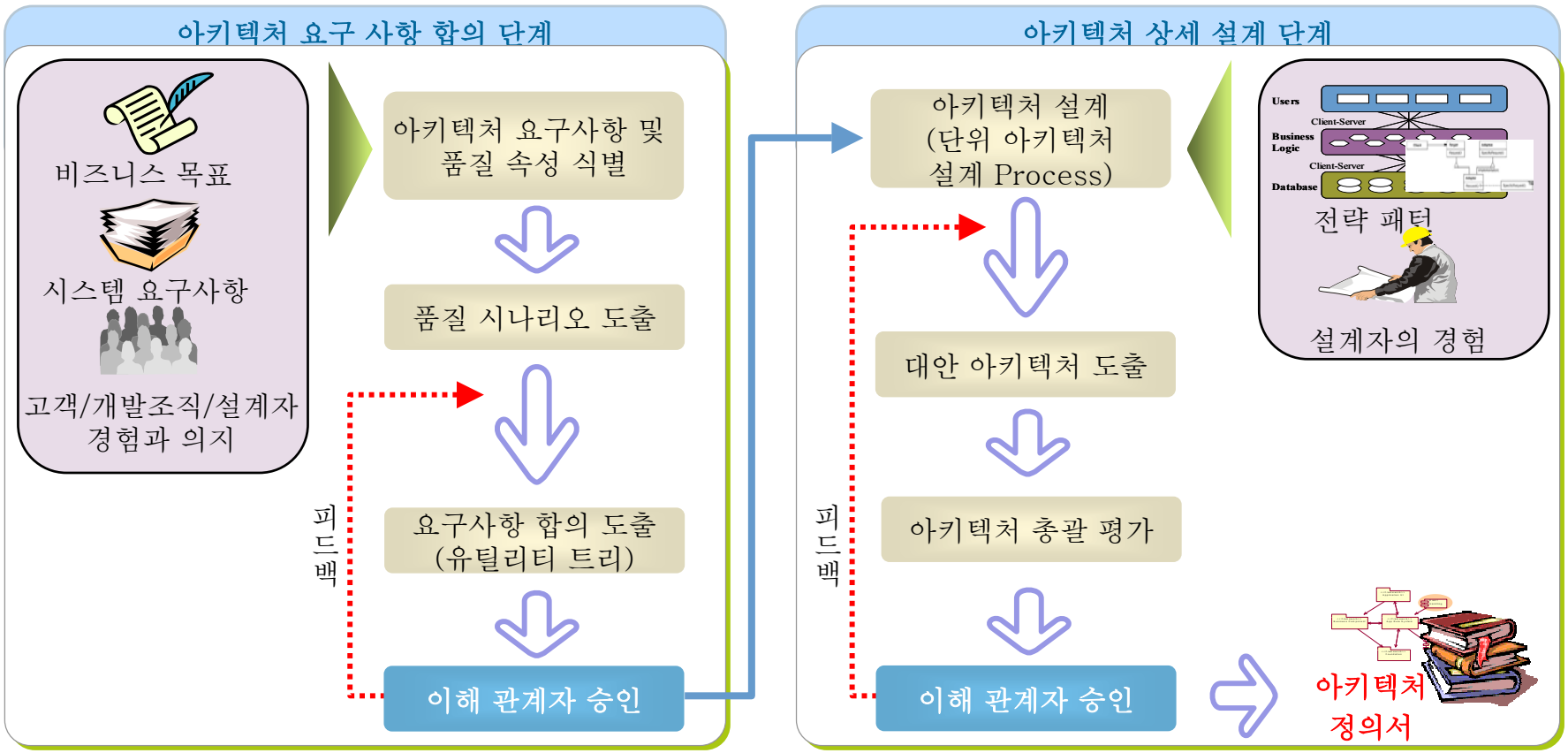
아키텍처 평가

아키텍처 정의서

2e Consulting
Efficiency & Effectiveness

아키텍처 설계 프로세스

- 아키텍처 설계 프로세스는 크게 아키텍처 요구사항 합의 단계 및 아키텍처 상세 설계 단계로 구성합니다.



- 품질 요소로 관리할 항목을 결정합니다.
- 미국 SEI(Software Engineering Institute) 연구소 개발한 ADD(Architecture Driven Design) Process 및 ATAM(Architecture Trade-off Analysis Method) 기법에서 활용하는 다음과 같은 여섯 가지 품질 요소가 대표적입니다.

품질 요소	설명
가용성(Availability)	● 시스템 오류 및 극복 방안과 관련된다. ● 정상가동시간/(정상가동시간 + 복구시간) 으로 평가한다.
변경 용이성(Modifiability)	● 수정 비용과 관련된다. 수정을 요하는 원인에는 시스템 기능 변경, 플랫폼 변경, 운용 환경의 변경, 프로토콜 변경 등 다양하며, 이러한 경우 얼마나 빨리, 저 비용으로 변경 할 수 있는가와 관련된다.
성능(Performance)	● 시스템 이벤트 발생 시 이를 처리하여 반응하기 까지 시간과 관련된다.
보안성(Security)	● 불법적 시스템 활용을 방비할 수 있는 능력과 관련된다.
테스트 용이성(Testability)	● 테스트 시 시스템 결함을 얼마나 효과적으로 나타낼 수 있는가와 관련된다.
사용 편이성(Usability)	● 사용자가 원하는 결과를 얼마나 쉽게 얻을 수 있는가와 관련된다.

품질 시나리오 도출

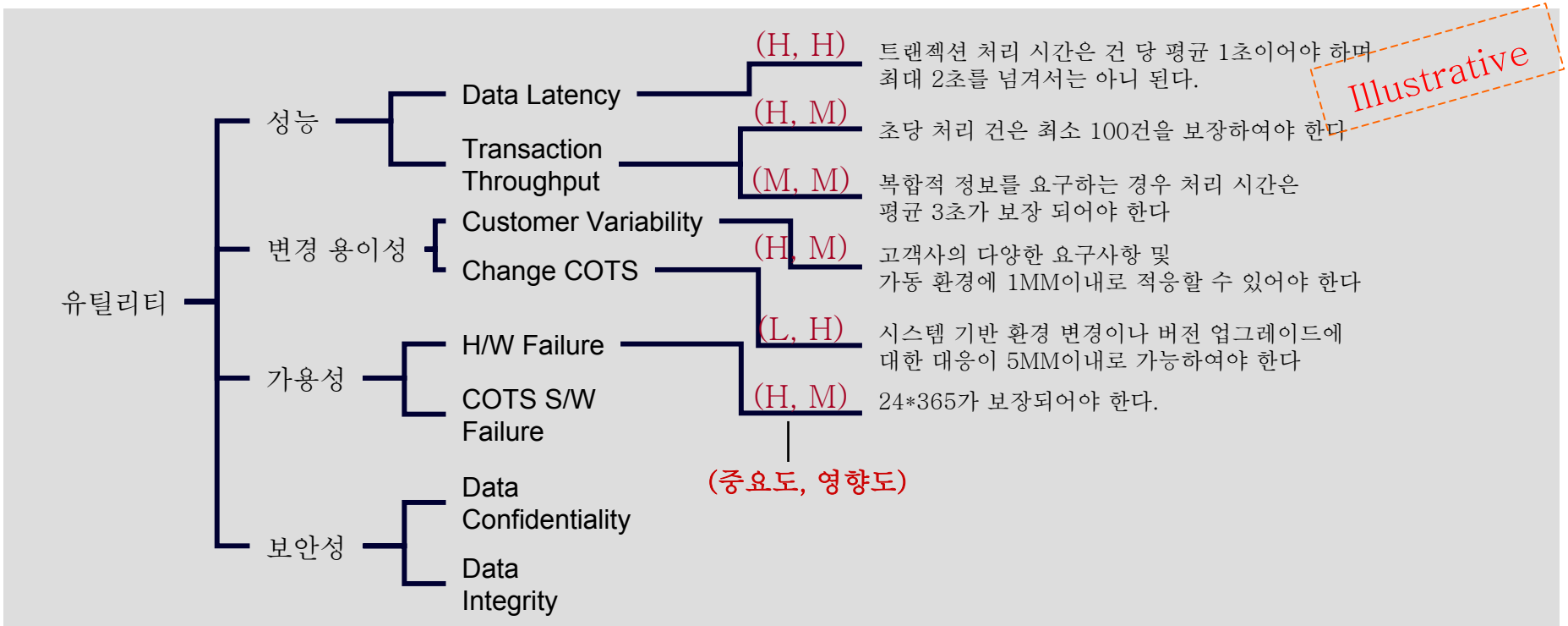
- 각 품질 요소는 품질 시나리오로 정리하며 품질 시나리오는 다음과 같은 방법으로 도출합니다.
  - 시스템 설계, 유지보수, 운영 요원의 노하우
  - 유즈케이스 기술서 작성 시 규정한 비기능 요구사항
- 미리 예상할 수 있는 품질 시나리오에는 다음과 같은 것들이 있습니다.

Illustrative

품질 요소	예상 품질 시나리오	
가용성	● 24*365가 보장되어야 한다.	
변경용이성	● 여러 국내 고객사의 다양한 요구사항 및 가동 환경에 적응할 수 있어야 한다. ● 관련 법 규정의 변경 시 대처할 수 있어야 한다.	● 고객의 다양한 접근 채널에 대한 지원이 가능하여야 한다. ● 향후 기술의 발전 시 나타날 신규 채널에 대한 대처가 가능하여야 한다. ● 시스템 기반 환경 변경이나 버전 업그레이드에 대한 대응이 가능하여야 한다.
성능	● 트랜젝션 처리 시간은 건 당 평균 1초이어야 하며 최대 2초를 넘겨서는 아니 된다. 초당 처리 건은 최소 100건을 보장하여야 한다. ● 복합적 정보를 요구하는 경우 처리 시간은 평균 3초가 보장 되어야 한다.	
보안성	● 외부 네트웍 망을 통한 어떠한 불법적 접근도 차단하여야 한다. ● 특히 내부 직원의 이상 접근이나 해킹 시도도 방지 되어야 한다.	
테스트용이성	● 단위 테스트, 통합 테스트, 시스템 테스트, 병행 가동 테스트, 시스템 운영 각 단계별 테스트 방안이 구분 지원되어야 한다. ● 외부 시스템 테스트 도구와의 연동이 지원되어야 한다.	
사용편이성	● 일반 거래 고객, 고객 사 직원, 내부 시스템 운영자 등 다양한 시스템 고객 정보 요구 특성 및 고객의 연령, IT 활용 수준, 개인별 특성, 취향에 대한 대응 방안이 있어야 한다.	

요구사항 합의 도출

- 개별 품질 시나리오의 중요도 및 아키텍처 영향도를 프로젝트 관련자가 개별적으로 평가합니다.
- 개별적 평가를 취합하여 전체 이해 당사자들이 합의합니다.
- 합의된 사항은 다음 그림의 유틸리티 트리로 문서화하고 이를 공유합니다.
- 합의된 사항은 중요도에 따라 단위 아키텍처 설계 시 우선적으로 반영합니다.

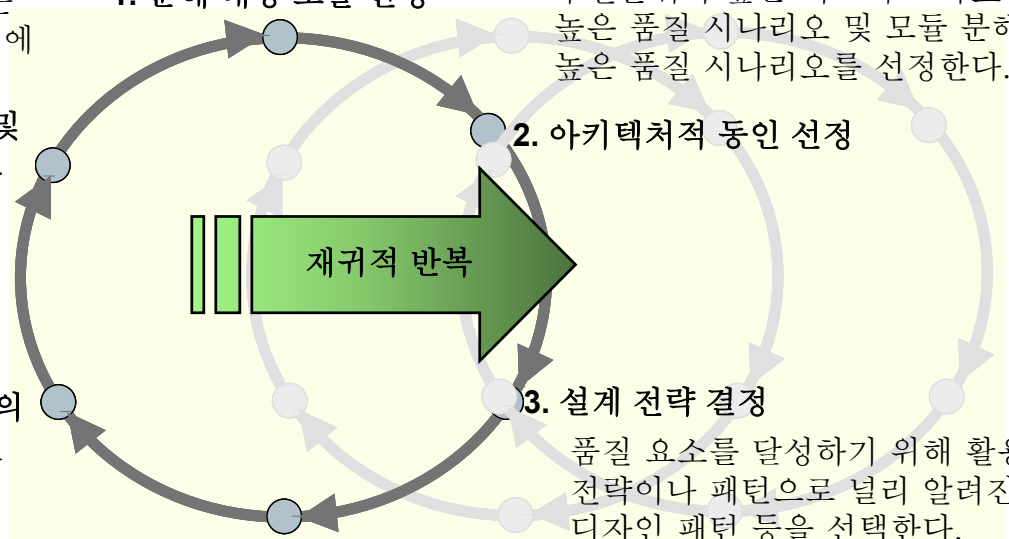


단위 아키텍처 설계 프로세스

- 단위 아키텍처 설계 프로세스는 다음 그림과 같이 전체 여섯 가지 절차로 이루어지며 반복 적용합니다.

분해 대상 모듈은 우선 시스템 단위에서 시작하여
서브시스템, 모듈, 서브 모듈 순으로 내려간다.
각 대상 모듈은 기능 요구사항, 품질요소 요구사항,
제약 조건과 관련 있는 것을 선정한다.

모듈 분해의 대상이었던 품질 시나리오는
분해된 하위 모듈 및 그들 간의 연관관계에
의하여 해결되었는지 검증한다.



우선순위가 높은 비즈니스 목표와 관련성이
높은 품질 시나리오 및 모듈 분해와 연관성이
높은 품질 시나리오를 선정한다.

품질 요소를 달성하기 위해 활용할 수 있는
전략이나 패턴으로 널리 알려진 아키텍처 패턴,
디자인 패턴 등을 선택한다.

품질요소 달성 전략이 결정되면 해당 전략에
따른 분해 전형이 주어진다. 이에 따라 분해대상
모듈을 분해한다.

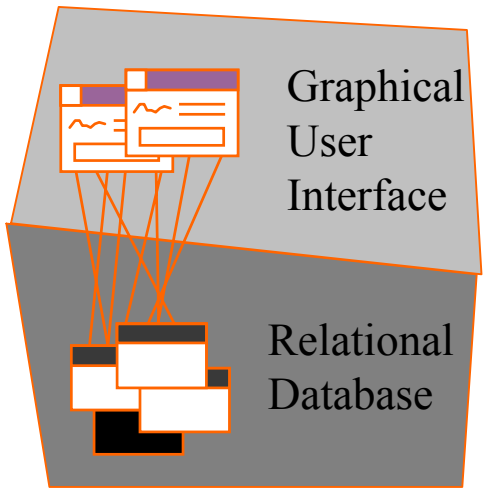
모듈이 제공하거나 요구하는 서비스나
특성을 인터페이스로 정의한다.

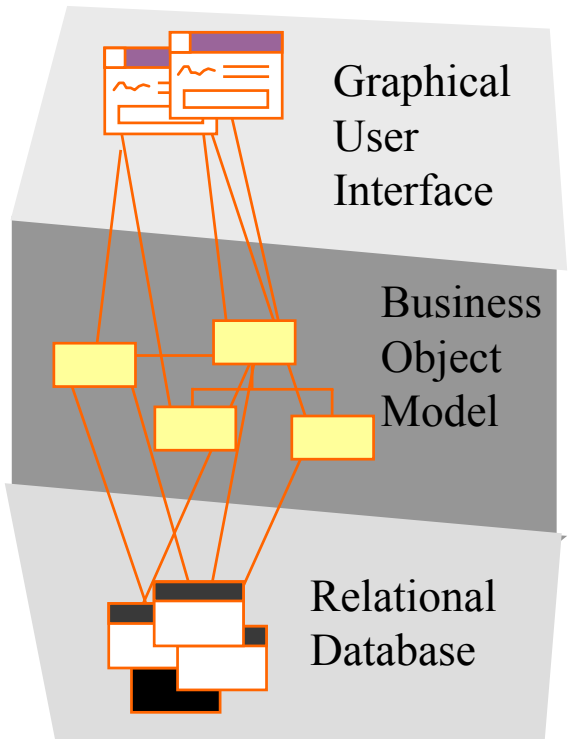
아키텍처 평가

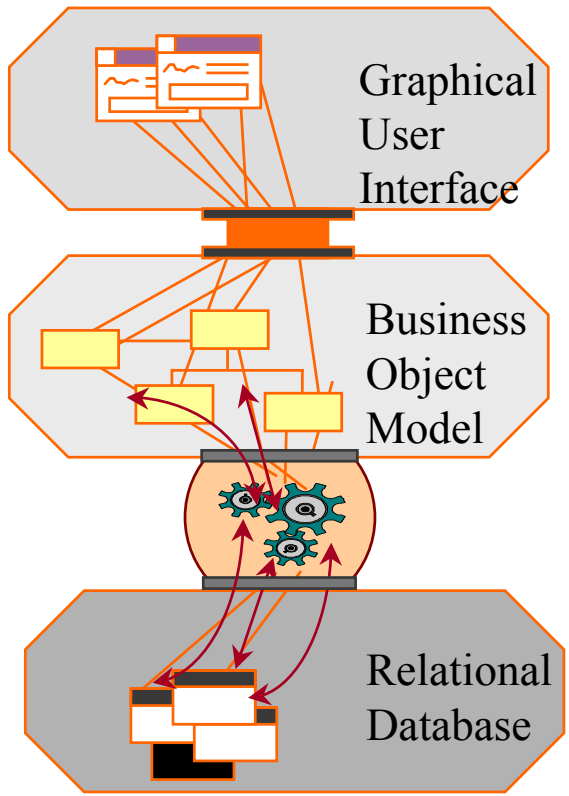
- 모든 디자인에는 trade-off 관계가 있다.
  - 당구 시뮬레이션 정도라면 이해하기 쉽고, 구현하기 쉬우면서도, 정확한 계산이 가능한 뉴튼 방정식을 활용
  - 그러나 이를 중성자 실험 시뮬레이션에서도 사용 가능할까?
  - 상대성 이론은 이해하기 어렵고, 구현하기 어렵지만, 중성자 실험 시뮬레이션에는 적합하다.
  - 당구 시뮬레이션에 상대성 이론을 적용한다면?
- 품질 요소간에도 이러한 Trade-off 관계가 있다.
  - 변경 용이성 vs 성능
  - 가용성 vs 비용
  - 가용성 vs 성능
  - 성능 vs 비용
  - …

각 아키텍처에 대하여 평가해 봅시다

- 성능, 가용성, 변경 용이성, 보안성, 테스트 용이성, 사용 편의성 측면에서 평가합니다.

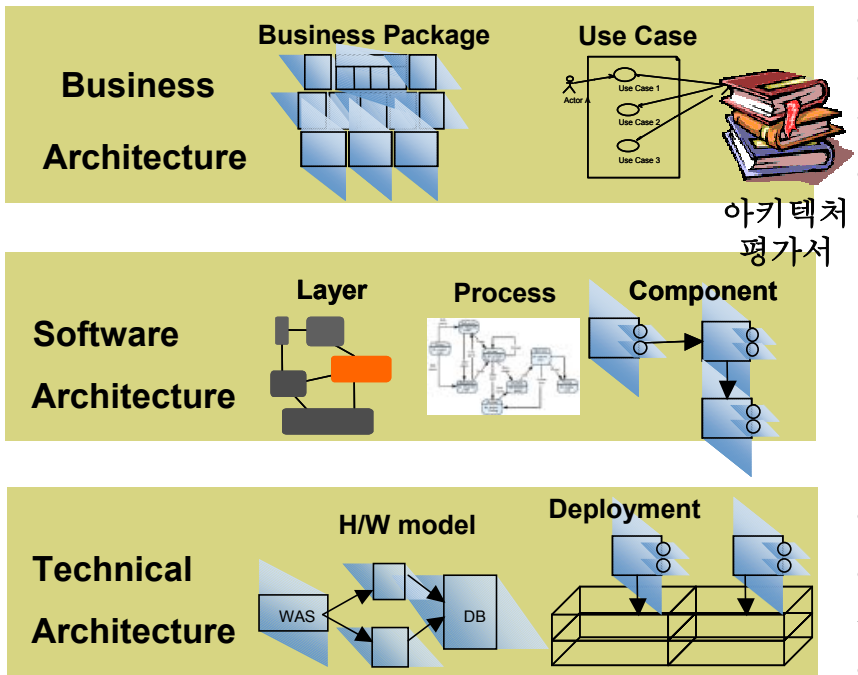






아키텍처 정의서

- 설계한 아키텍처는 업무 관점, 구현 전략 관점, 기반 기술 환경 관점에서 추상화 하여야 합니다.
- 각 관점 별로 분리 문서화하여 프로젝트 관계자의 이해력을 높여야 합니다.



아키텍처 평가서

- 시스템의 기능 관점 및 관련 비즈니스 패키지들로 구성됨
- 시스템 비전 및 요구사항 정의/분석에서 나타나는 아키텍처 관련 상위 수준 품질요소를 수용할 수 있도록 설계됨
- Package View, Use Case View로 구성됨

- 재사용성과 요구변화에 대한 견고성을 위주로 설계됨
- 아키텍처 품질요소 달성 전략 관련 요소로 구성되어 시스템 프레임워크 요소들로 채워짐
- Layer View, Process View, Component View로 구성됨

- 컴포넌트 실행 환경과 컴포넌트간 배치 구조임
- 시스템 용량 산정, 네트웍 용량 산정 등을 바탕으로 도출된 노드 및 토폴리지를 H/W & N/W 아키텍처로 도식화
- Deployment View, Network View 및 Hardware View로 구성됨

아키텍처 설계 패턴

IV

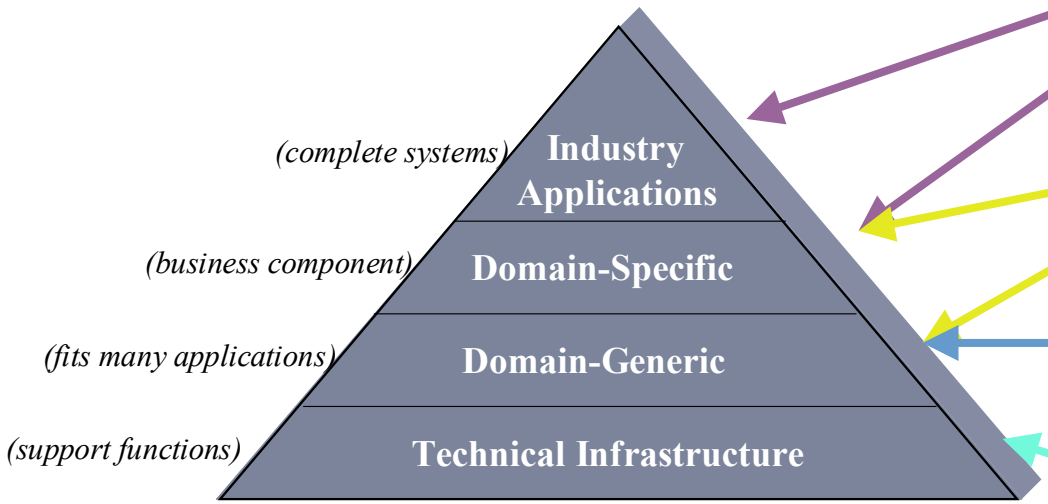
아키텍처 설계 영역

영역별 주요 설계 전략

아키텍처 설계 전략

아키텍처 설계 영역

- 현 아키텍처 평가에서 파악한 위험 요인에 대하여 합의된 아키텍처 요구사항을 바탕으로 최우선적으로 재설계 하여야 합니다.
- 이전 시스템 구축 시점의 제반 환경(WAS 버전, 국내 가용 기술진 수준)의 변화에 발맞추어, 이를 최적으로 활용할 수 있는 아키텍처를 설계하여야 합니다.
- 설계 방안은 크게 컴포넌트 유형 및 품질 속성에 따라 제시 됩니다.



제어 구조의 유사성

상위 제어구조 유사성을 중심으로 변경 용이성, 사용 편의성 확보에 치중합니다.

정보적 유사성

정보 구조 유사성을 중심으로 변경 용이성 달성에 치중합니다.

하부 단위 기능 공통화

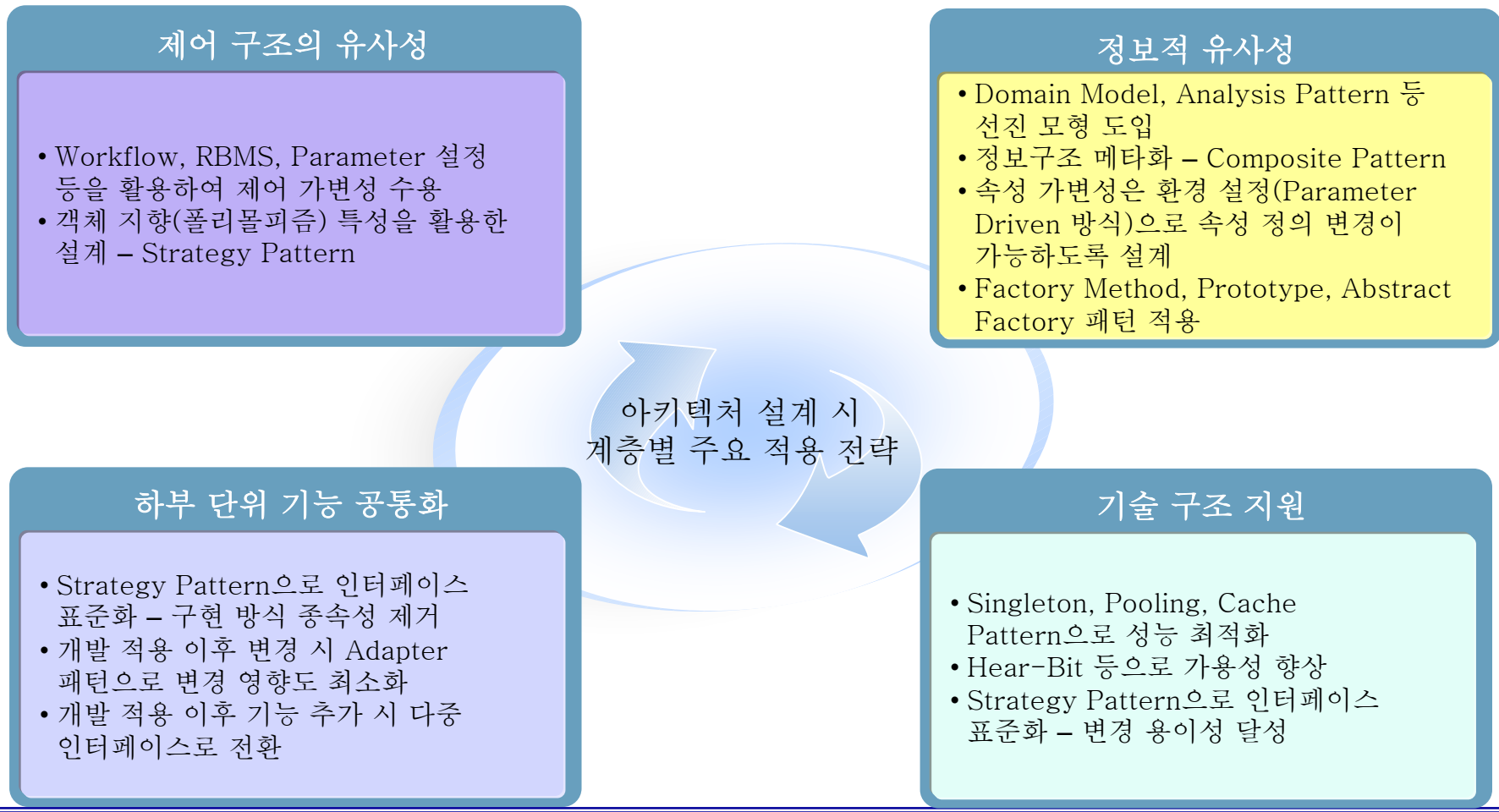
공통 기능 함수 설계 시에는 성능, 변경 용이성, 테스트 편의성 확보에 치중합니다.

기술 구조 지원

플랫폼, 기반 기술을 제어하며, 성능, 가용성, 변경용이성, 테스트 용이성 향상에 치중합니다.

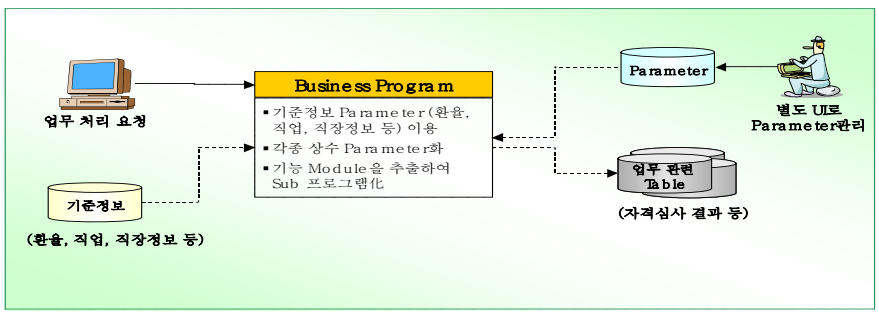
영역별 주요 설계 전략

- 계층별 권고 안(설계 전략)은 다음과 같습니다.

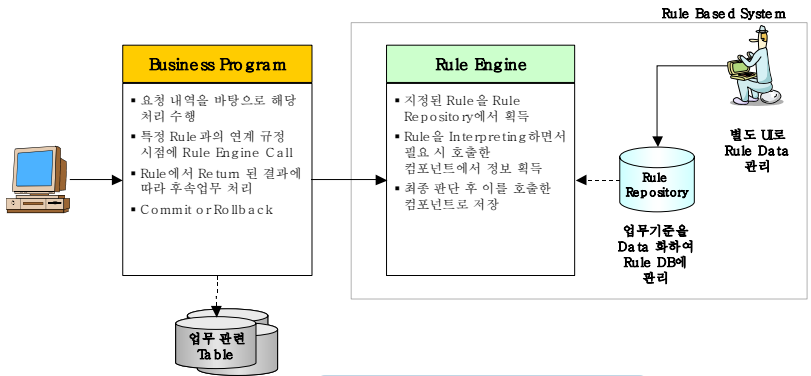


아키텍처 설계 전략 - 제어 가변성 수용

- 컴포넌트 제어 기능을 외부 환경 설정으로 분리, 구축합니다. 주로 Parameter Driven & RBMS 연계와 관련 있습니다.
- Parameter Driven 및 RBMS는 기술 프레임워크 요소이며, 업무 컴포넌트와의 연동 부분에 대한 인터페이스는 변경되지 않아야 합니다.



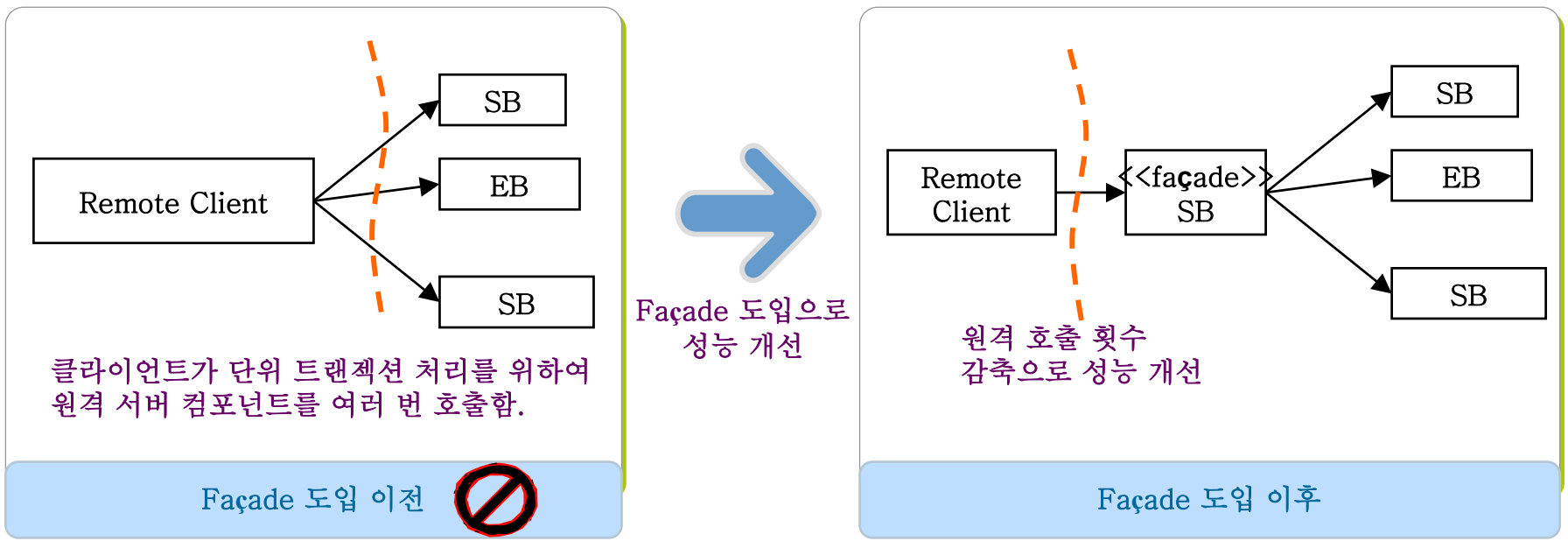
Parameter Driven



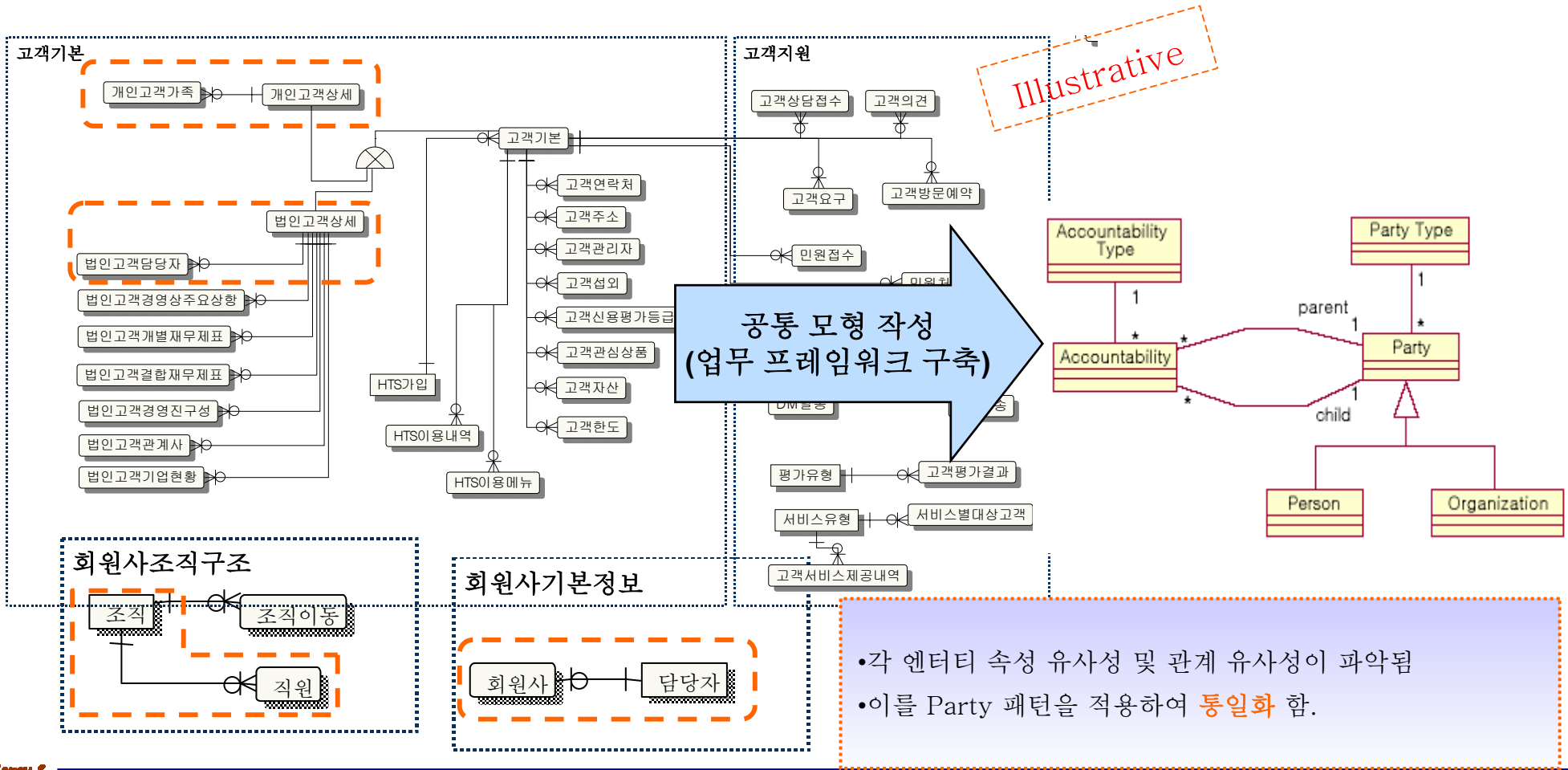
RBMS

아키텍처 설계 전략 - Façade Pattern으로 성능 향상

- 어플리케이션 서버와 클라이언트 사이의 원격(remote) 호출은 상당한 오버헤드(overhead)를 가집니다. 따라서 원격 호출 횟수를 줄여 성능을 향상하여야 합니다.
- 어플리케이션 서버 컴포넌트들의 대표주자(Façade)를 수립하여 대표 컴포넌트를 한번만 호출하도록 설계합니다. 이는 일반적인 코딩 스타일 검사 항목입니다.

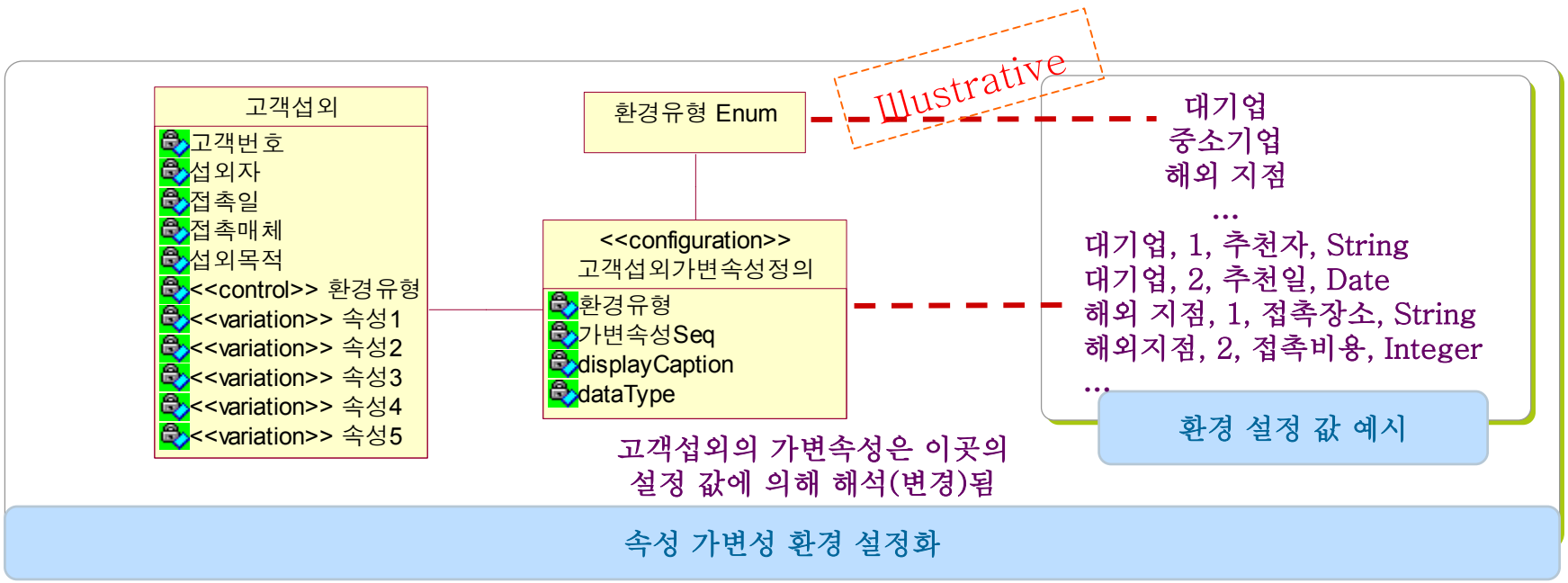


- 각 업무 영역의 개별 설계 모형에서 공통성을 추출하여 추상화/일반화를 기반으로 공통 모형으로 작성(프레임워크로 구축)한 예입니다.



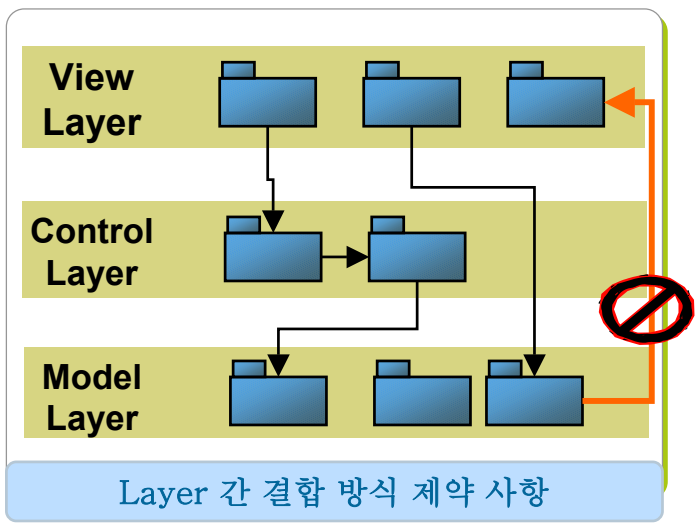
아키텍처 설계 전략 – 속성 가변 설계로 변경용이성 향상

- 여러 시스템 사용자들이 요구하는 기능의 가변성은 특정 컴포넌트의 속성 가변성으로 나타날 수 있습니다.
  - 특히 마케팅적인 관리 특성은 각 부서, 고객 마다 특이성이 있음
  - 신규 기능 수용의 경우 설계 변경을 방지할 수 있음
- 확정 속성과 가변 속성으로 구분하고 가변 속성의 정의 부분을 환경 설정으로 분리합니다.
- 이는 성능 향상과도 관계됩니다.
  - 모든 속성을 단순 나열하면 DBMS의 Page에 담을 수 있는 레코드 수가 감소하여 빈번한 Cache swapping을 유발하고, 따라서 성능 저하를 초래함. (1Page 당 레코드 수= Page Size / Record Size)
  - 속성 정의 부분은 시스템 기동 시 메모리로 캐시 시키고, 프로그램적으로 결합하여 불필요한 Table Join(성능 저하)을 방지합니다.

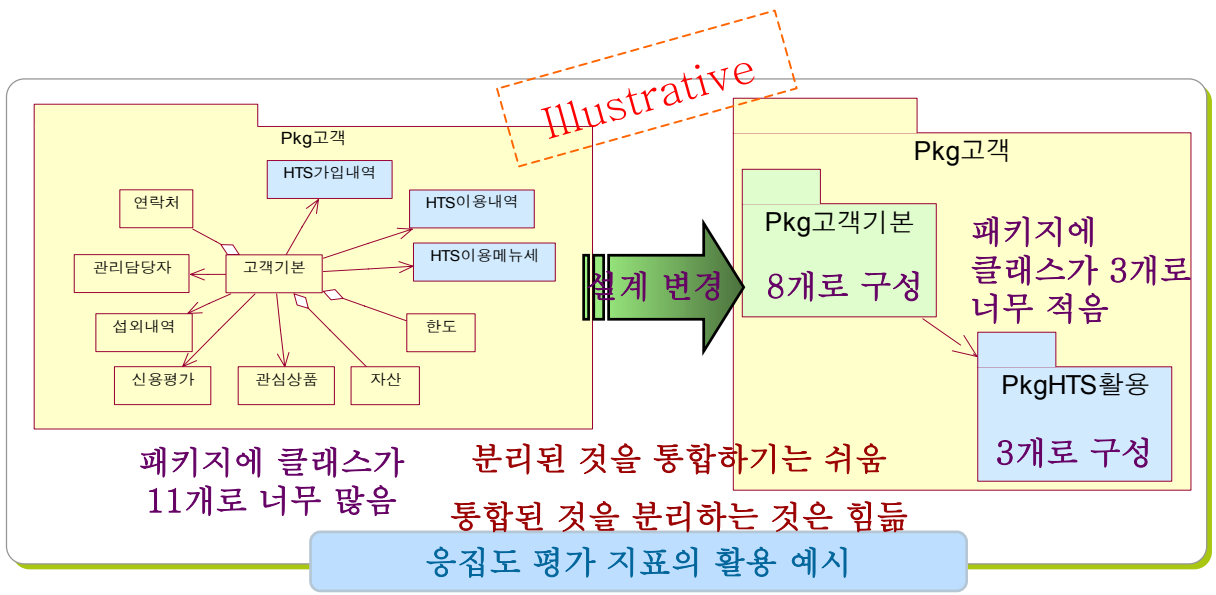


아키텍처 설계 전략 - 변경용이성 향상 일반 (모듈화)

- 모듈화를 통한 응집도 제고, 결합도 제거를 추구합니다.
  - 티어(단말, 접속, 업무, 대외), 레이어(MVC(Model-View-Control) Pattern 위주), 프레임워크(업무, 업무 기능, 기술), 업무 구분, 주제 영역(고객, 상품, 계좌, …) 등으로 분할 정복하여 응집도를 높입니다.
  - 티어 간 연계 방식, 레이어 간 연계 제약 사항을 준수하여 결합도를 낮추어야 합니다.
    - 상위 레이어 요소는 하위 레이어 요소를 활용할 수 있지만 그 역 방향은 허용하지 않는다.
    - 레이어간 의존은 중간 레이어를 뛰어넘어 차 하위 레이어 요소를 활용하는 것은 지양한다.
  - 표준 평가 지표를 초기에 확정 짓고 이를 통해 측정, 활용되어야 합니다.
    - 응집도 평가 지표 : 7 ± 2 Magic Number
    - 결합도 평가 지표 : McCabe's Cyclomatic Metric(= Link – Node + 2)



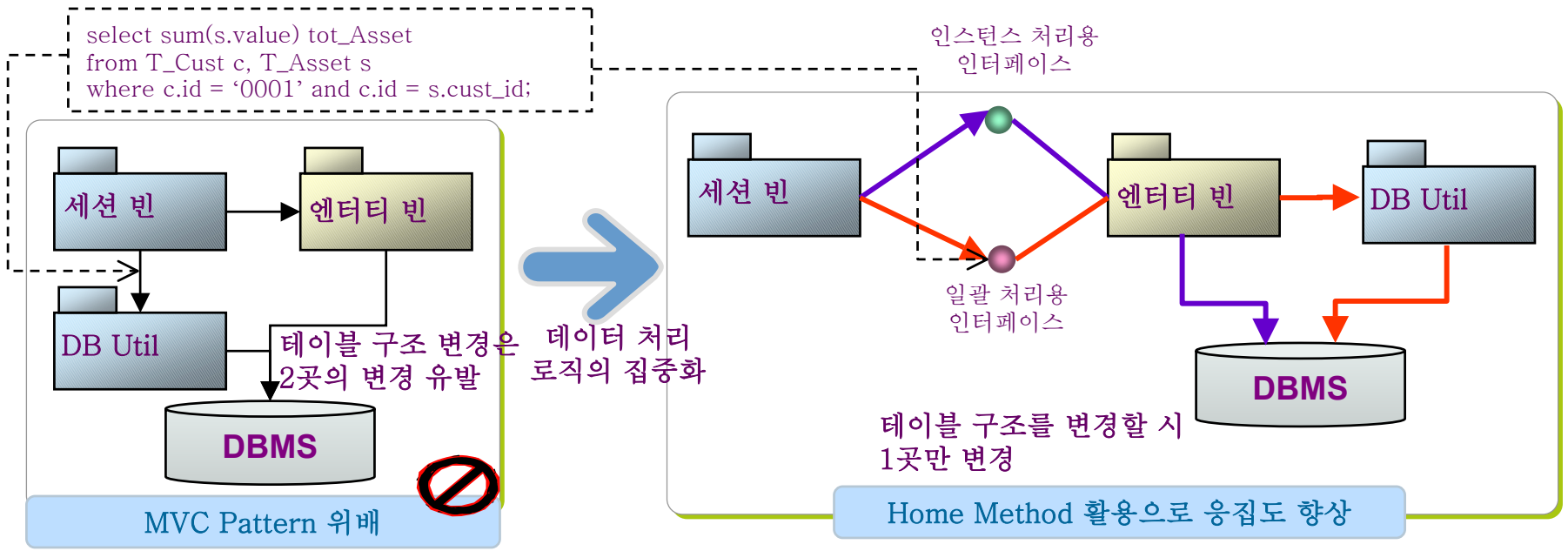
Layer 간 결합 방식 제약 사항



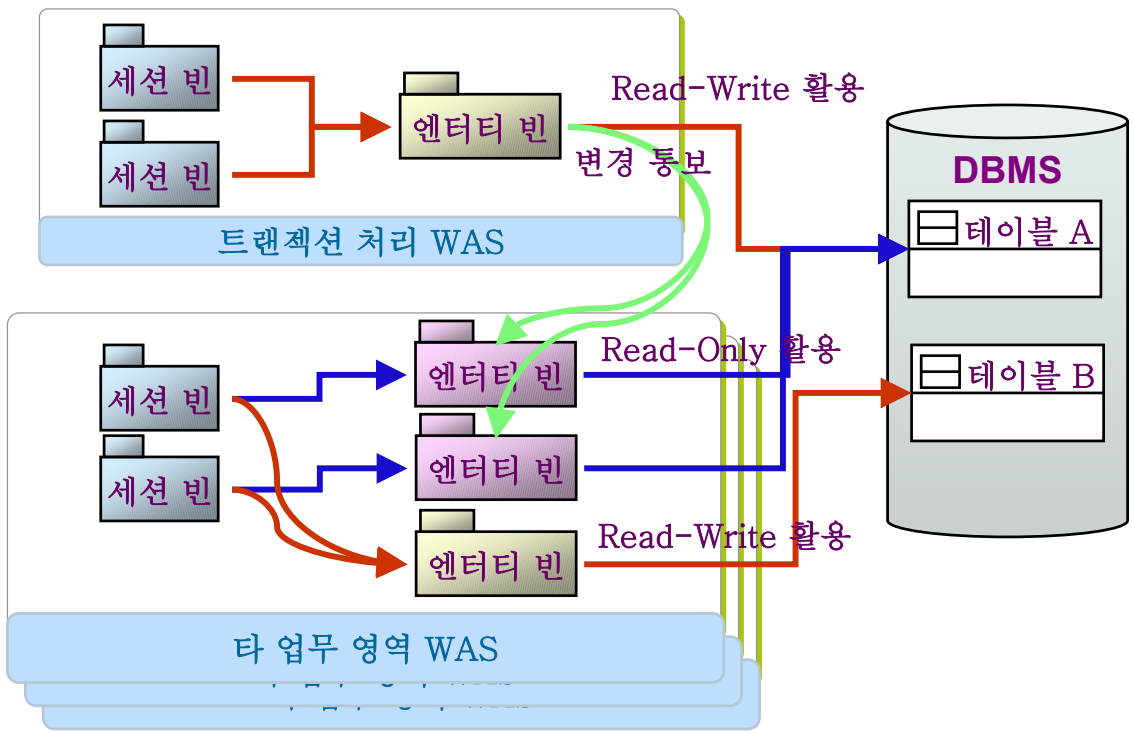
응집도 평가 지표의 활용 예시

아키텍처 설계 전략 - 정보와 제어의 분리 (모듈화)

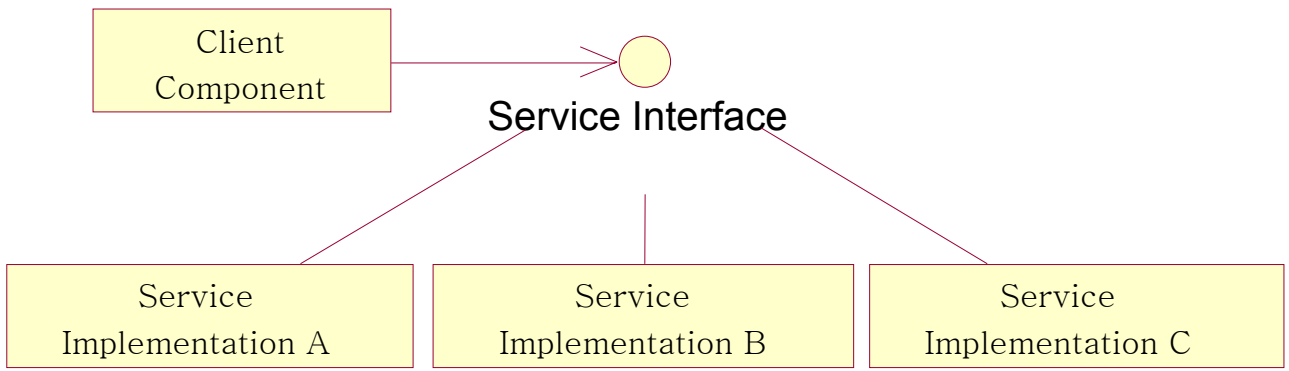
- J2EE에서는 MVC Pattern을 지원하기 위하여 세션 빈(Session Bean)과 엔터티 빈(Entity Bean) 컴포넌트를 활용합니다. 세션 빈은 제어적(Control) 측면에서, 엔터티 빈은 정보적(Model) 측면에서 활용합니다.
- 제어 기능을 담당하도록 구축한 세션 빈에서 정보를 획득하는 기능을 복합적으로 구현하는 것은 정보와 제어의 분할 정복이라는 원칙 위배되며, 유지 보수성에 나쁜 영향을 미칩니다.
- 엔터티 빈 인스턴스(instance)는 테이블의 개별 레코드에 해당합니다. 따라서 개별 인스턴스 처리 로직용 인터페이스와 일괄 처리용 인터페이스를 분리 설계하고 일괄 처리용 인터페이스에는 Home Method용 기능만을 정의합니다.



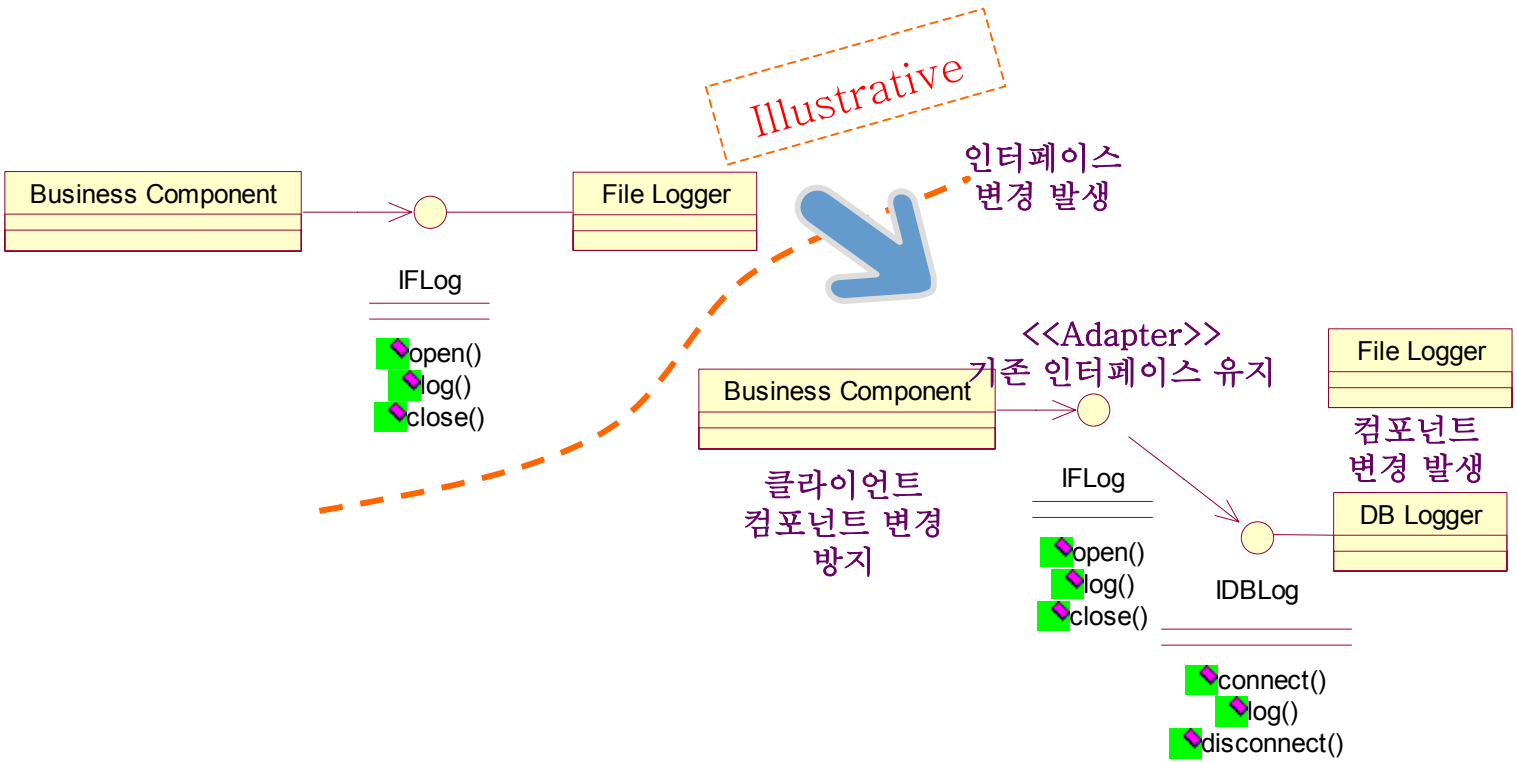
- 가끔 정보 수정이 발생하는 엔터티에 대하여 트랜젝션에 참여하는 Read-Write 컴포넌트 및 병행 접근을 허용하는 Read-Only 컴포넌트로 구분하여 구축함으로써 전체적인 성능을 향상 시킵니다.
- Read-Only 엔터티는 read-timeout-seconds 속성을 조정하여 데이터 변경 빈도와 조율하며, Read-Write 엔터티에서는 변경이 발생한 후 invalidate를 통하여 Read-Only 엔터티를 재 로드 하도록 하여 정보의 일관성을 보장합니다.



- 다양한 구현 방식이 있고, 상황에 따라 여러 방식을 활용하여야 할 경우 이들의 인터페이스를 통일하여 이를 활용하는 클라이언트 컴포넌트에서 발생할 수 있는 구현 방식 종속성을 제거합니다.

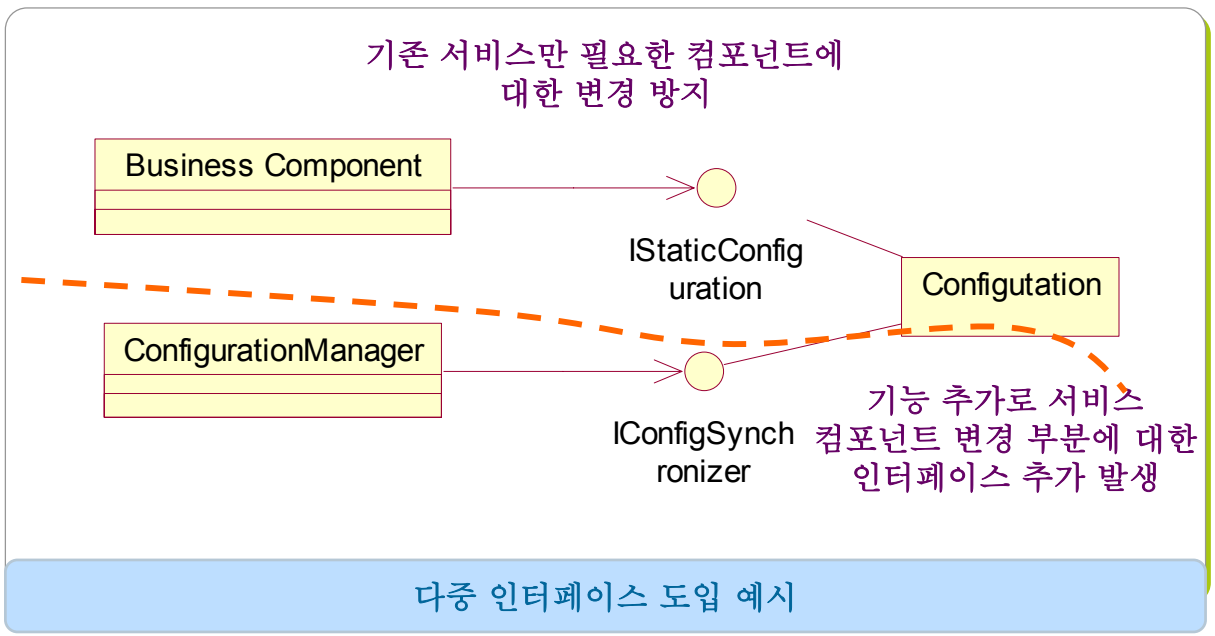


- 아키텍처 요소 컴포넌트의 인터페이스는 조기 확정 지어야 합니다. 이의 변경은 최소화 되어야 합니다.
- 변경된 컴포넌트의 인터페이스에 영향 받지 않는(기 인터페이스를 그대로 유지하여도 문제가 발생하지 않는) 컴포넌트에 대한 변경을 방지하기 위하여 어댑터를 추가할 수 있습니다. 어댑터는 클라이언트가 원하는 인터페이스(기존 인터페이스)이며, 변경된 인터페이스로의 통로 역할을 수행합니다.

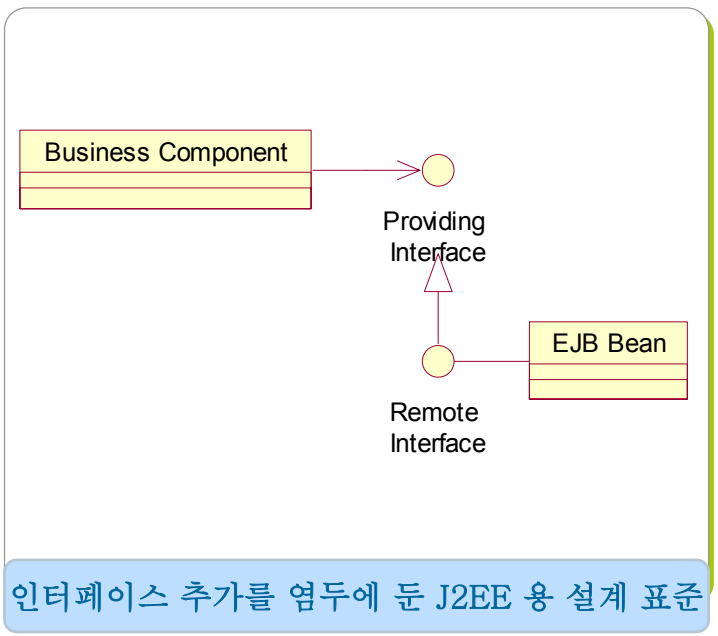


아키텍처 설계 전략 - 변경 용이성 달성 전략 일반(Multiple Interface Pattern)

- 서비스 컴포넌트에 새로운 종류의 기능 추가 시 신규 기능 전용 인터페이스를 추가하여 기존 기능만을 활용하여도 충분한 클라이언트 컴포넌트의 변경을 방지합니다.
- 다중 인터페이스 지원 여부는 활용 언어마다 상이합니다. J2EE에서 EJB(Enterprise Java Bean)는 Remote Interface를 하나만 가질 수 있습니다. 따라서 클라이언트가 활용하는 인터페이스는 다중 인터페이스 도입을 염두에 두어 Remote Interface와 별도로 상속 구조를 활용하여 구축하여야 합니다.



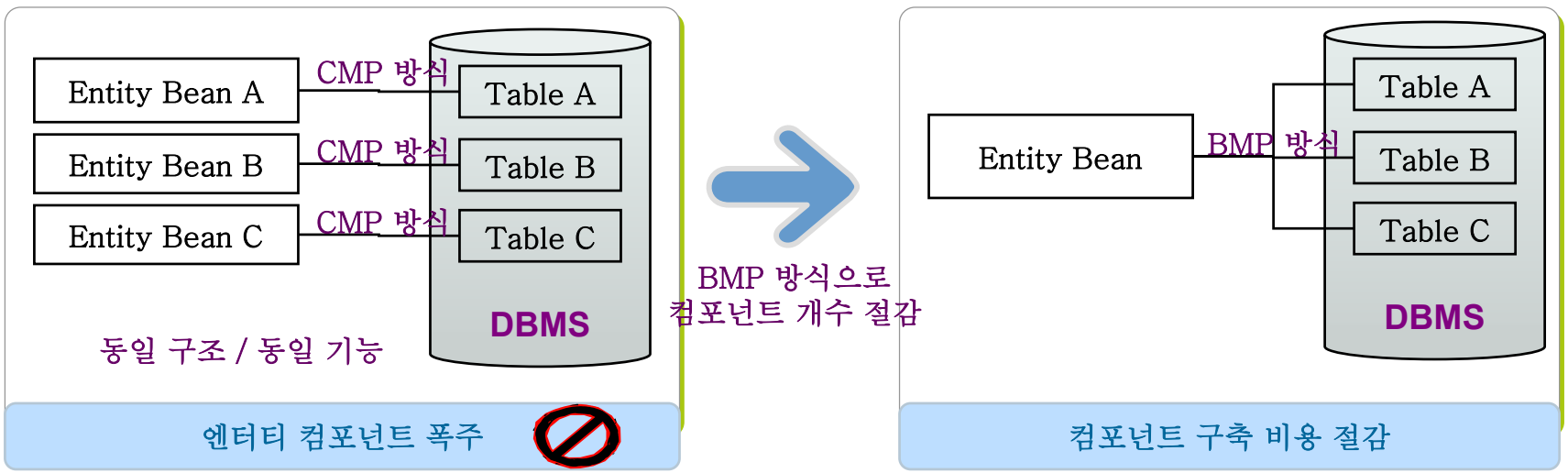
다중 인터페이스 도입 예시



인터페이스 추가를 염두에 둔 J2EE 용 설계 표준

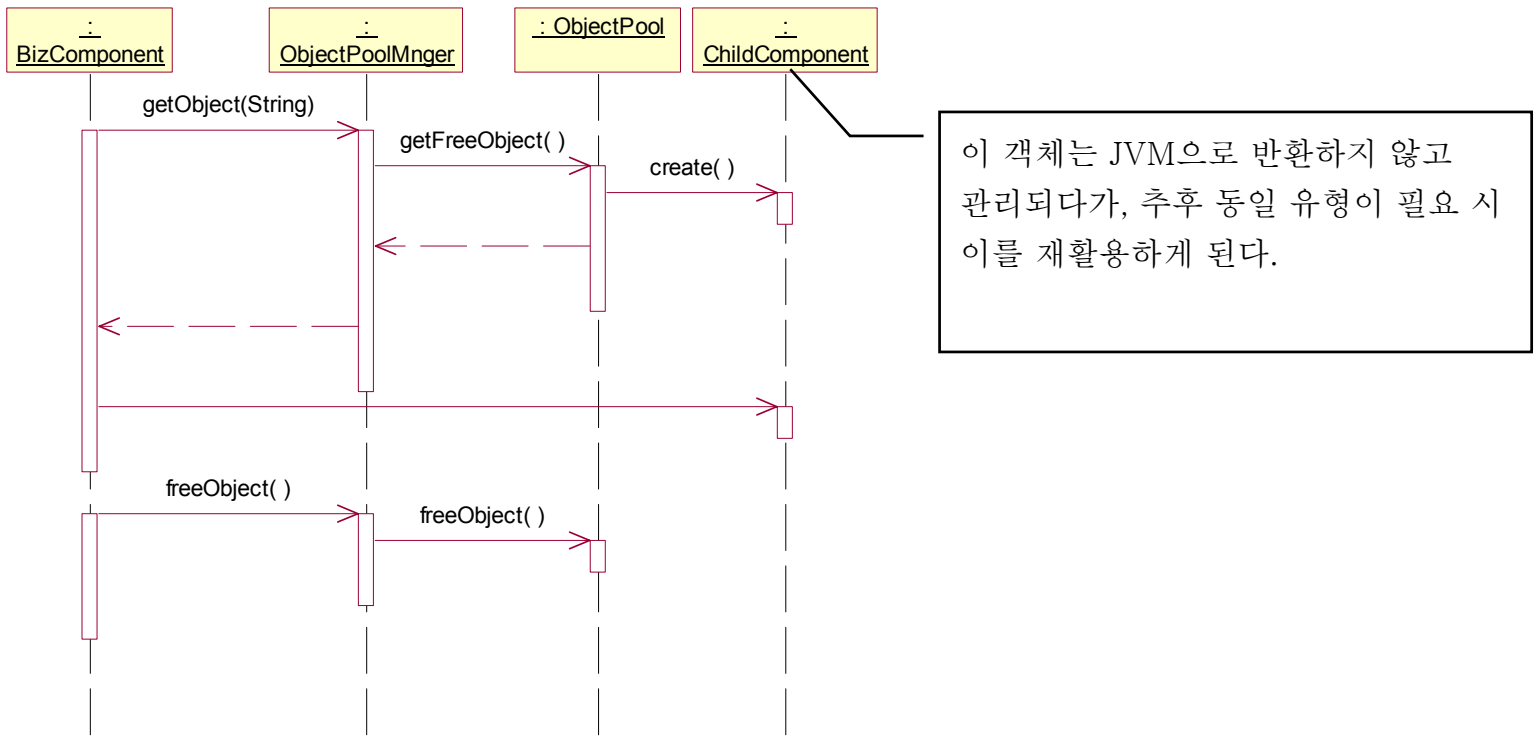
아키텍처 설계 전략 - 개발 비용 절감

- 정보적 기능을 처리하는 J2EE의 엔터티 빈(Entity Bean)은 DBMS Table과 1:1로 작성되어야 합니다.
- 만약 Table의 구조가 동일하며 처리 기능도 유사한 경우(예 : 코드 테이블) 엔터티 빈의 BMT 방식을 활용하여 컴포넌트 개수를 획기적으로 줄일 수 있습니다.
- 이는 개발 기간의 단축 뿐만 아니라 향후 유지 보수 시점의 효율성을 제고할 수 있는 J2EE용 구현 기법입니다.



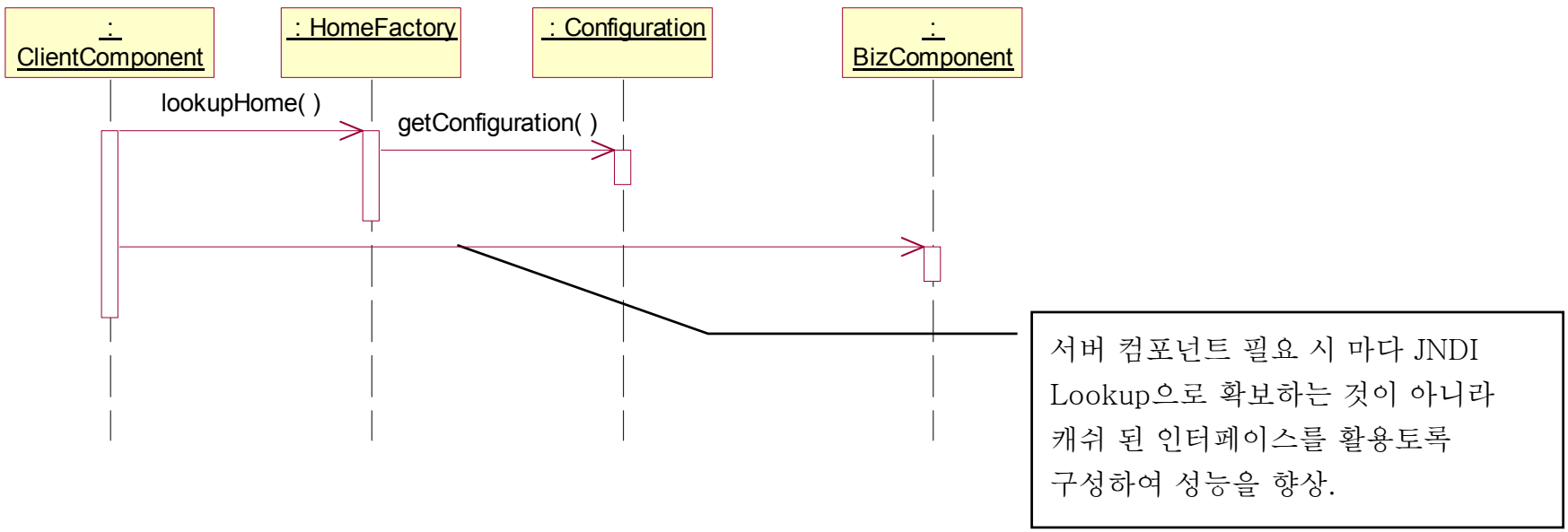
아키텍처 설계 전략 - Memory Pooling(성능 향상)

- JVM에서 발생하는 가비지 컬렉션(GC)의 크기를 줄이기 위해 생성된 객체를 JVM으로 반환하지 않고 ObjectPoolManager를 통해 관리하게 한다.
- 추후 동일 유형의 객체를 필요로 할 시 이전 생성 객체를 ObjectPooler에서 할당 받는다.



아키텍처 설계 전략 - HomeInterface Cache(성능 향상)

- WAS의 EJB 컴포넌트를 찾아주는 표준 서비스로 JNDI가 있습니다. 이를 클라이언트가 필요 시 마다 Lookup하게 되면 성능적 손해가 상당합니다.
- 따라서 이를 캐쉬로 관리하고 있다가, 요청 시 캐쉬로 관리 중이던 인터페이스를 반환, 활용하도록 구성하여야 합니다.
- 또한 서비스 컴포넌트는 여러 사유에 의하여 변경이 발생할 수 있으므로, 이로 인한 소스 코드의 변경이 있어서는 아니 됩니다.



사례로 본 Product Line의 효과

V

CelsiusTech Systems 회사 소개

SS2000 Product Line

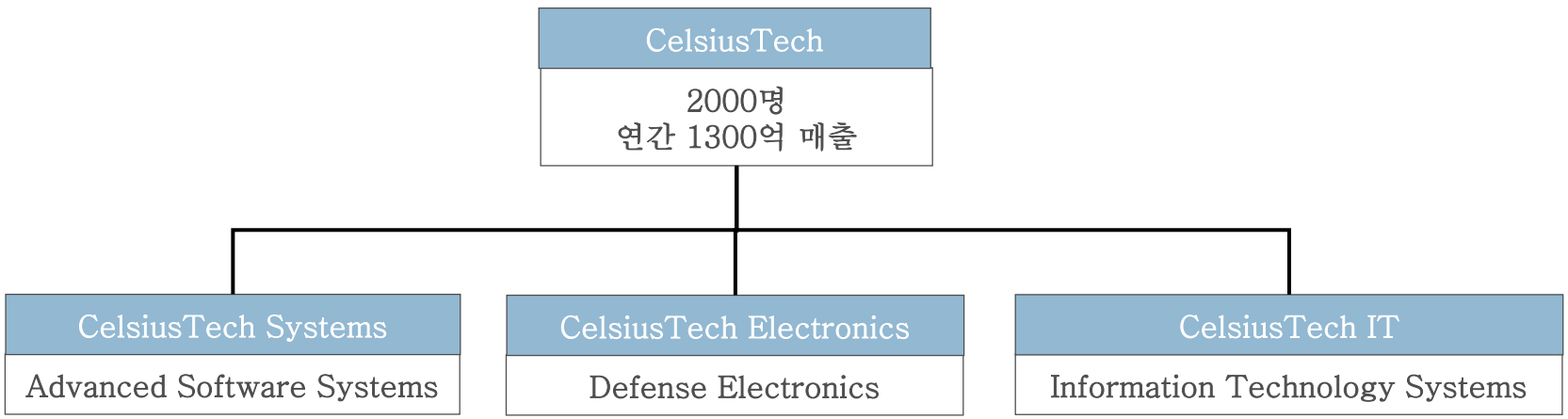
Product Line의 경제적 효과

조직 구조

인원 구성의 변화

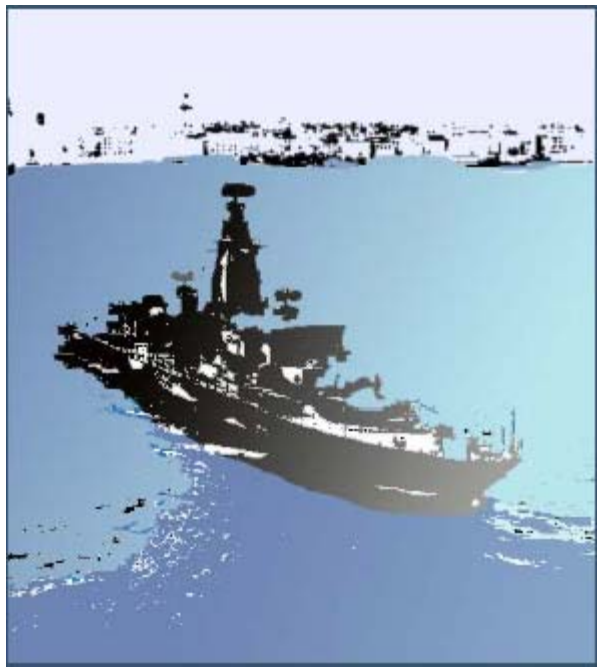
CelsiusTech Systems 회사 소개

- 회사 소개
  - 스웨덴 해군 계약자
  - Product Line의 창조자, 얼리어댑터
  - 보유 Product Line : SS200(Ship System 200) - 스칸디나비아, 중동, 남 태평양 함대용 군함 명령•통제 시스템
  - 3번의 인수합병 – 경영진의 교체는 있었으나 중간 관리자 및 기술진의 변화는 적었음. 따라서 연속성, 안정성 확보가 가능 했음
- Product Line Architecture를 통해 새로운 사업 기회를 창출함



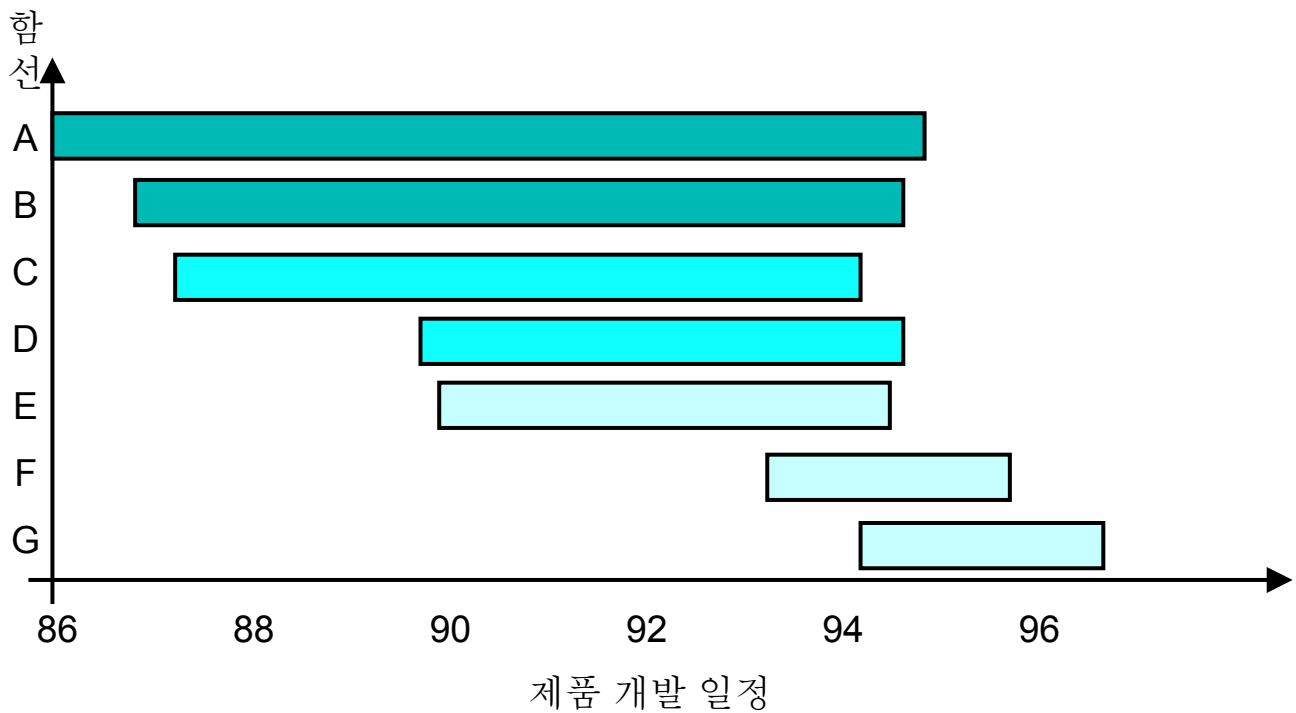
- 함선에서의 모든 무기 제어, 명령, 통제, 통신을 처리하는 통합 시스템
- 실 Product의 규모는 해상 및 해저용 함선의 LAN으로 연결된 30~70대의 컴퓨터에서 수행되는 일백만~일백오십만 라인의 Ada 코드로 구성됨.
  - 연안 소형 쾌속 호위함
  - 다목적 순찰선
  - 쾌속 공격선
  - 프리깃함
  - 대양 순시선
  - 잠수함

  - 스웨덴, 덴마크, 오스트리아, 뉴질랜드, 파키스탄, 오만
- 지원 가변성. But... 단일 아키텍처, 단일 Core Asset, 단일 조직으로
  - 각종 무기, 센서
  - 다국어 인터페이스
  - 다양한 하드웨어 : 68020, 68040, RS/6000, ...
  - 다양한 OS : OS2000, IBM AIX, POSIX, Digital Ultrix, ...



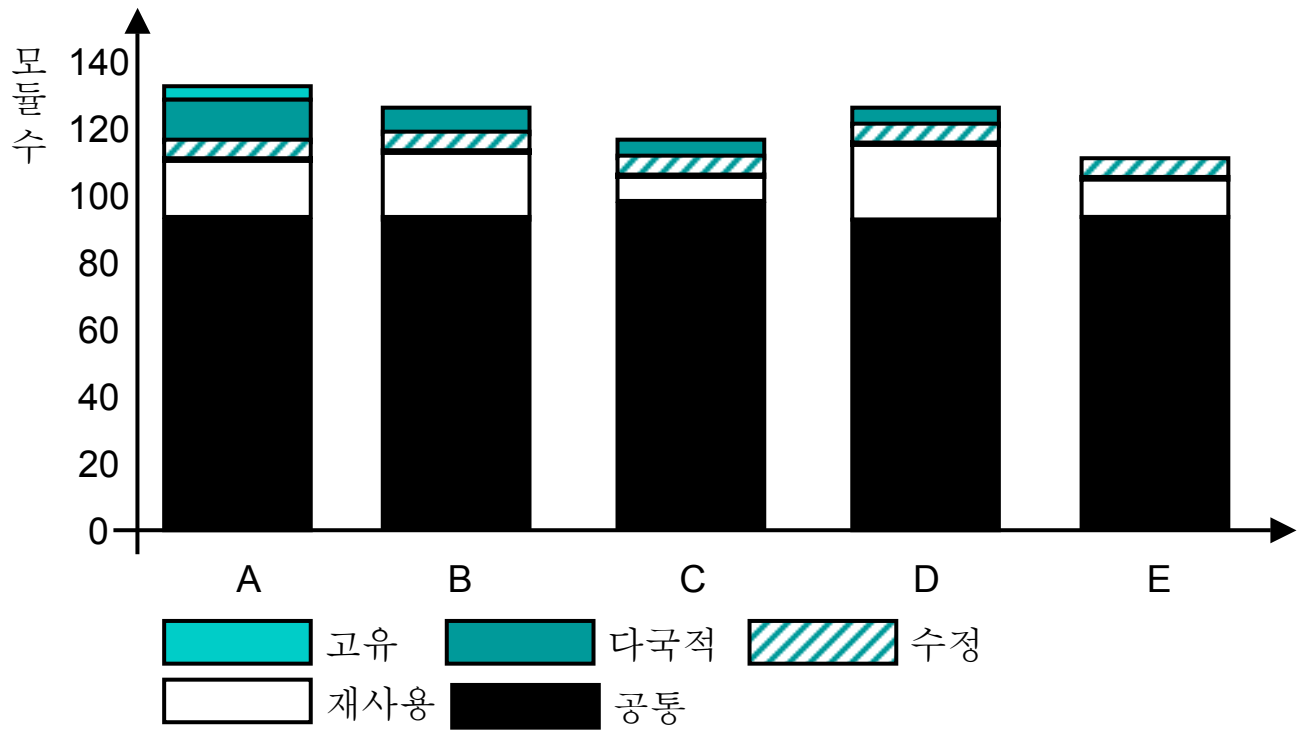
Product Line의 경제적 효과 – 일정 단축

- 동시에 계약된 A, B로 인해 Product Line 접근법을 선택
- A, B 개별적으로는 일정 효과가 없었으나, 한 프로젝트로 구축함.
- C, D는 Product Line을 활용하여 만들어짐. 일정 단축 효과가 상당했음
- E, F, G 일정 예상과 합치됨



Product Line의 경제적 효과 – 코드 재사용

- 공통 : 70~80% – 완전 컴포넌트 호출형 재사용 부분
- 재사용 : 10~15% - 소스 코드 수준의 재사용, 즉 재컴파일 필요
- 수정 : 5 ~ 10% - 프로젝트 특성에 맞게 인터페이스는 유지하되 구현 상세는 변경해야 하는 부분이 존재함
- 고유 : 첫 프로젝트에서만 존재, A 프로젝트 기간의 대부분은 고유 모듈 특성에 의해 발생
- 다국적 : 경험의 축적을 통하여 복잡한 기능 영역도 통일화 될 수 있음을 보여줌



공통성 구축 변화도

Product Line의 경제적 효과

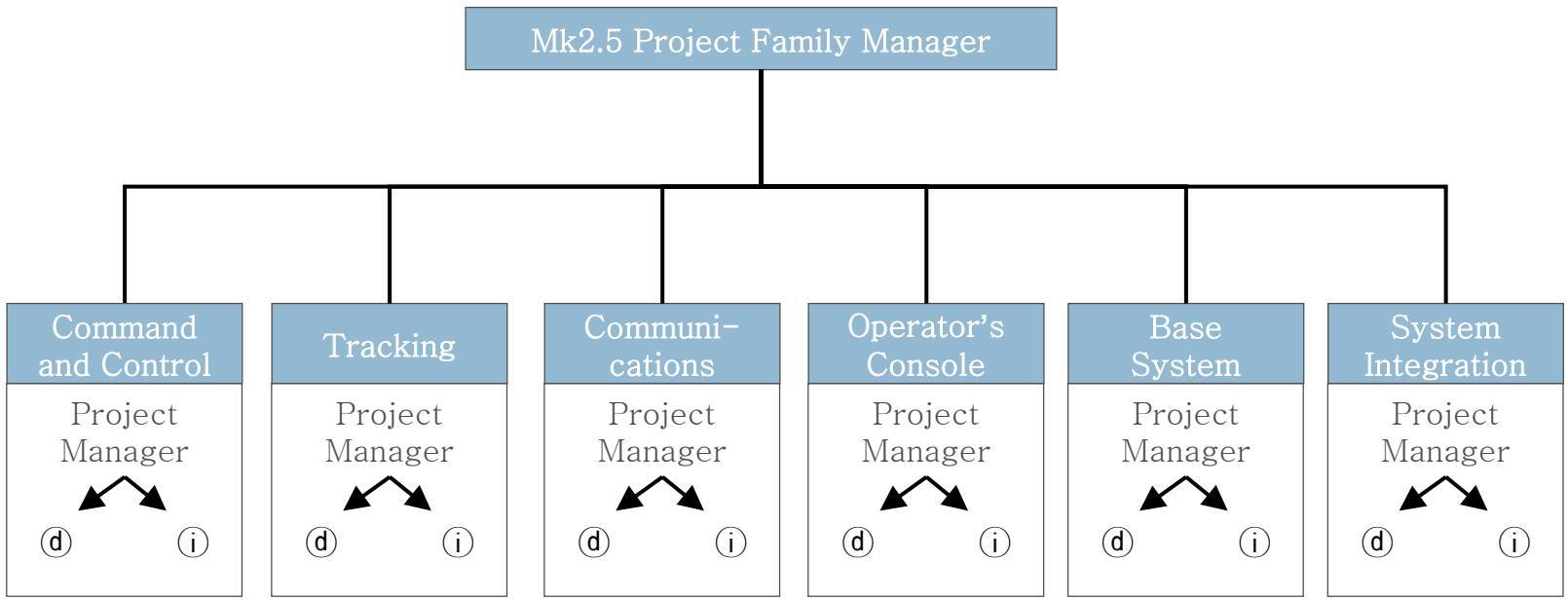
- 일백만~일백오십만 줄 규모의 시스템 통합 테스팅 : 1~2명으로 수행
- 완전 새 OS에 탑재 : 3개월
- 비용, 일정 : 예측 범위 내에서
- 성능, 분산 기능 : 사전 파악됨
- 고객 만족도 : 상
- H/w : S/W 비용 구조 : 35:65에서 80:20으로 – S/W 구축 생산성 8배

Product Line의 경제적 효과 – 신규 사업 영역 창출

- STRIC : 스웨덴 공군의 대공 방어 시스템
- 시스템 요소의 40%를 SS2000 Product Line에서 활용
  - 좌우로 흔들리지 않는 배 위에 탑재된 무기

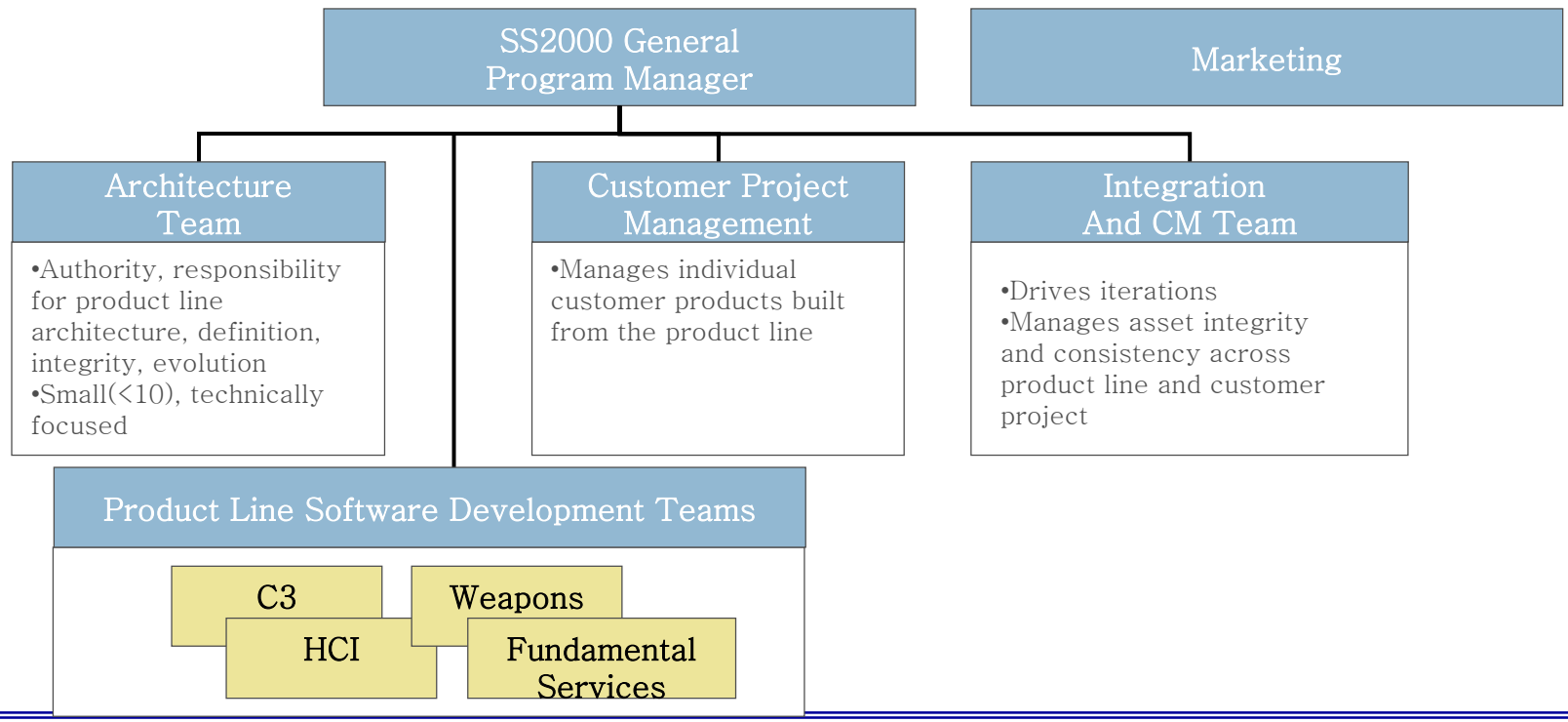
86년 이전의 조직 구조

- 발생 현상
  - 조직 구도가 시스템 분석 결과에 기능적 구도로 반영.
  - 기능 영역 경계간 요구사항이 정확히 도출되기 힘듦. 설계, 구현, 테스트 까지 이런 상태로 유지
  - 인터페이스 불일치를 시스템 통합 이전에 발견하기 어려움.
  - 타 영역 기능에 대한 이해 수준 저하
  - 영역별 관리자는 프로그래밍 수준의 이슈를 해결하려고만 함.



86-91년 사이의 조직 구조

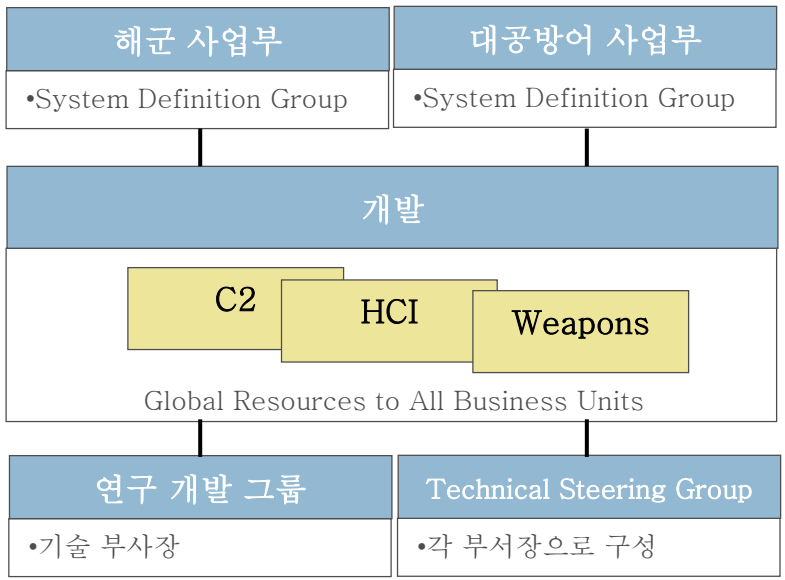
- 강력한 관리 체계
- 아키텍처 – 자산화, 설계 일관성
  - Product line 원칙 관리
  - 계층 구조 도출 및 예상 인터페이스 관리
  - 기능 할당
  - 공통 메커니즘 식별, 프로토타이핑(에러핸들링, interprocess communication protocol,...)
- Integration and CM
  - ❑ 테스트 전략, 계획, 테스트 케이스 개발
  - ❑ 점진적 개발 일정계획 작성
  - ❑ 형상 관리



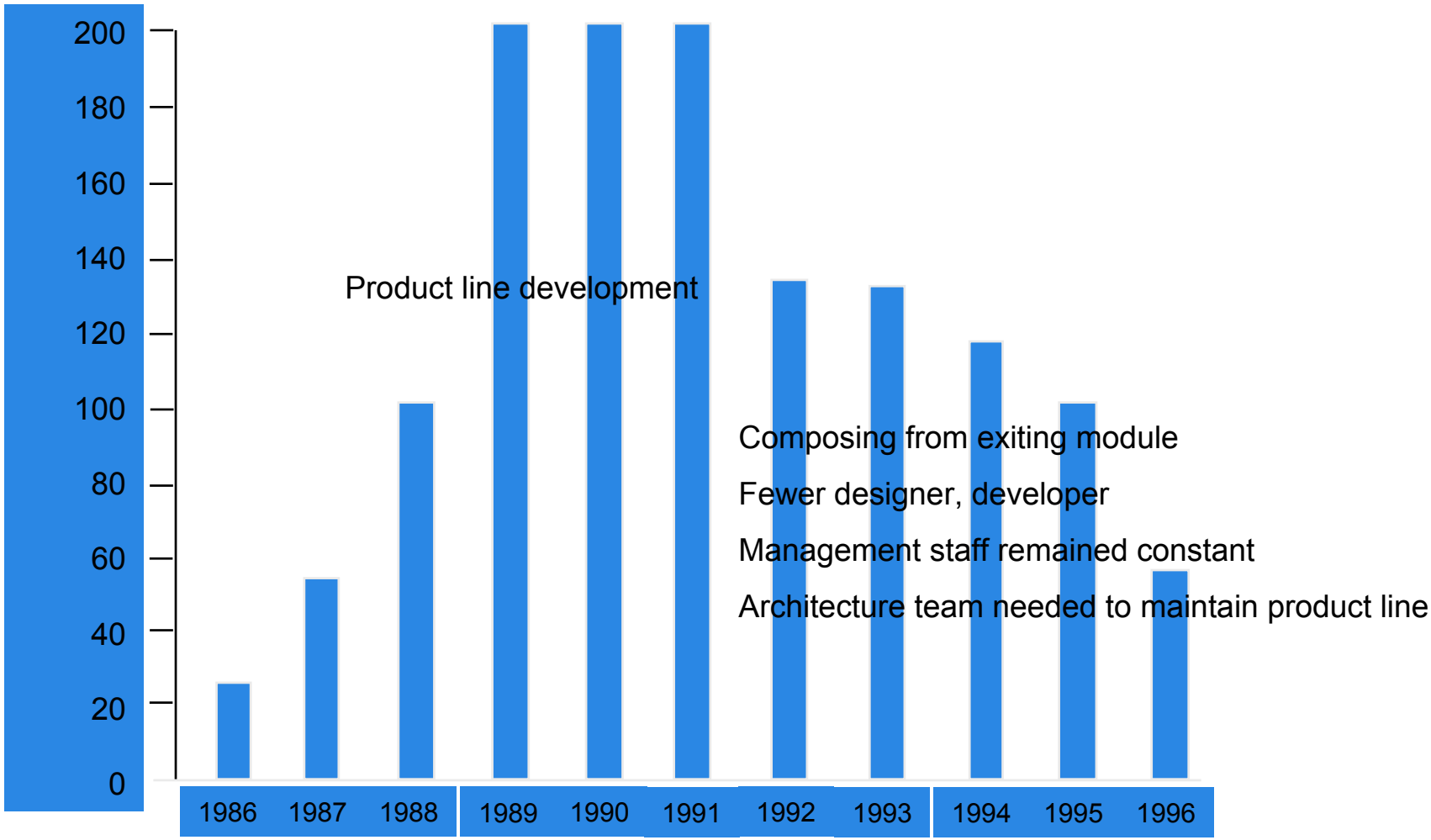
92년 이후의 조직 구조 - from product line element development to Composition

- 사업부
  - 고객과의 협상
  - 요구사항 분석, 시스템 통합/테스트/인도
  - 프로젝트 자금 관리, 일정 계획 관리

- TSG
  - ❑ 전 조직 차원에서 잠재적 신기술 파악, 파일로팅



인원 구성의 변화



Product line benefits of reuse

- Defect reduction: defect fixed in one product automatically fixed for all future products
- Performance: performance issues addressed for all products (e.g., schedulability, deadlock, distributed system issues)
- Planning: more accurate because all products are produced in the same way
- Reduction in
  - time to market
  - staffing

Q & A