



## Standard, Simplicity, Integration

We are leading the global market by innovating the entire process of product management including process, system and operation system to provide customers and the market with the products they demand in a highly efficient and timely manner.

# SW Architecture in Practice



Start your future with  
**SAMSUNG**

SAMSUNG links people together to create richer and more efficient communication environment.

- Software Engineering Lab.
- 김영기 책임
- [resious@gmail.com](mailto:resious@gmail.com)



# Overview of Part 1 : Introduction

## ❖ **Part 1 tell about ...**

- What is Software Architecture and Why that is import ?
- What effect to the Software Architecture ?

## ❖ **Chapter 1. What Is Software Architecture?**

- Software Architecture Definition
- Software Architecture Views
- Software Architecture Patterns

## ❖ **Chapter 2. Why Is Software Architecture Important?**

- Uses of an Software Architecture
- Roles of an Software Architecture

## ❖ **Chapter 3. The Many Contexts of Software Architecture**

- Contexts in Software Architecture
- Architecture Influence Cycle



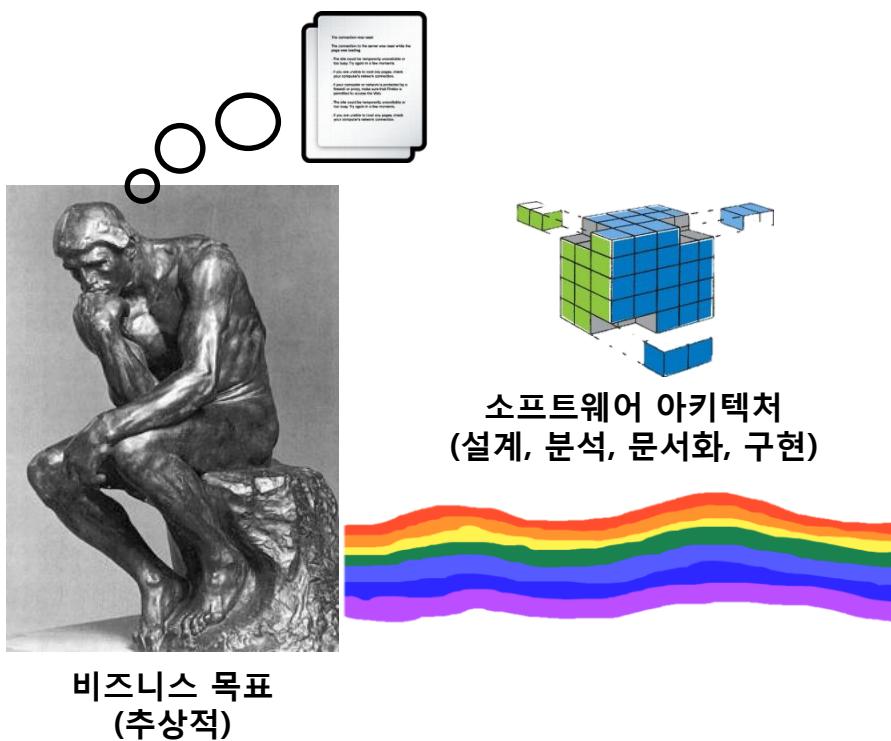
# **Chapter 1. What Is Software Architecture?**



# In this Chapter ...

## ❖ 소프트웨어 아키텍처 책에 대한 2 가지 가정

- 소프트웨어 시스템의 성공적인 개발에 있어서 소프트웨어 아키텍처는 중요하다.
- 아키텍처 책은 충분한 양의, 일반화 된, 지식체계를 포함하고 있다.





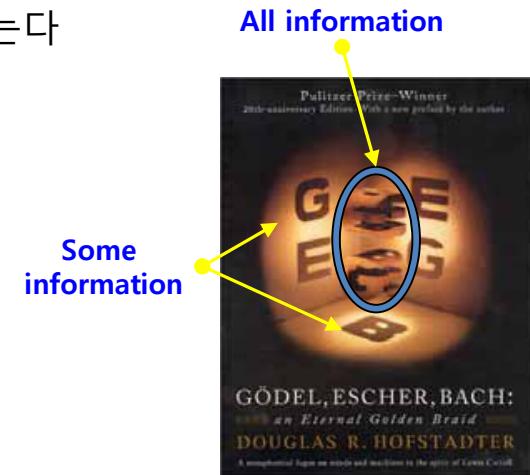
# Chapter 1.1

## ❖ What Software Architecture Is and What It Isn't

- 소프트웨어 아키텍처(Software Architecture) 정의 → 다양하다

The **Software architecture** of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both

- 아키텍처적인 결정은 “프로젝트 초기”에 일어난다
  - 아니다. Agile이나 Spiral의 경우 초기에 결정되지 않는다
- 아키텍처적인 결정은 설계적인 측면의 “중요한” 결정이다
  - 초기의 결정이 모두 아키텍처적인 것은 아니다.
- 소프트웨어 구조 (Software Structure)
  - 소프트웨어는 많은 구조들(Structures)로 이루어진다.
    - Software Architecture ≠ Software Structure
  - Structure는 특정 관점(Perspective)에서 소프트웨어를 정의한 것이다.
  - Structure는 구현 중심으로, Structure의 표현을 View라고 한다.

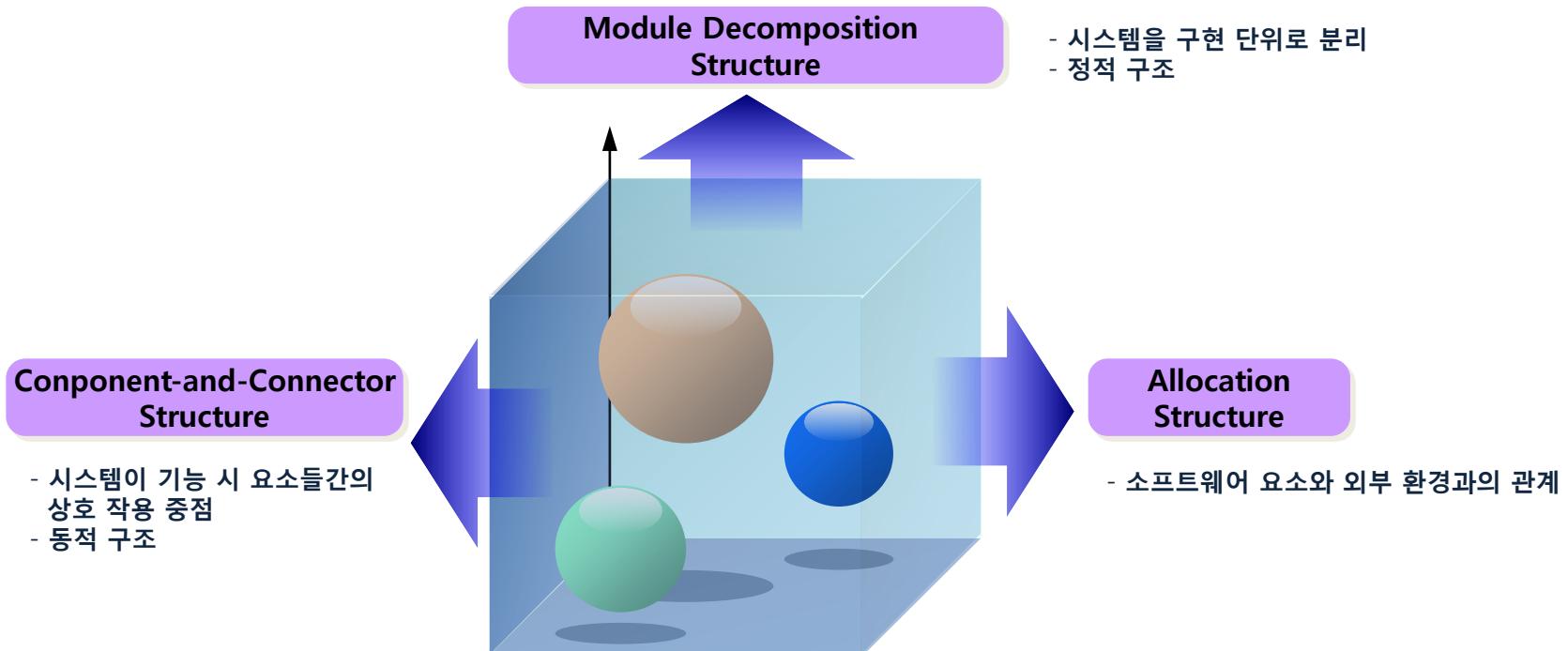




# Chapter 1.1

## ❖ 아키텍처 정의의 의미

- 소프트웨어 아키텍처는 소프트웨어 구조들의 집합이다.
  - 소프트웨어 시스템은 여러 가지 구조로 구성되며, 하나의 구조를 아키텍처라고 할 수 없다.
  - 아키텍처 구조의 3가지 분류





# Chapter 1.1

- 아키텍처는 추상화 수준의 상위 개념이다.
  - 아키텍처는 소프트웨어 요소와 요소들의 관계를 전체적인 관점에서 본다.
  - 아키텍처는 시스템에 유용하지 않은 정보나 세부적인 사항을 생략/압축할 수 있다.
    - 특정 목적과 관계 없는 요소들 간의 관계 정보를 명확하게 한정해서 생략 가능
- 모든 소프트웨어 시스템은 소프트웨어 아키텍처를 가진다.
- 아키텍처는 행위(Behavior)를 포함한다.
- 모든 아키텍처가 좋은 아키텍처는 아니다.
- 시스템 아키텍처와 엔터프라이즈 아키텍처
  - 소프트웨어 아키텍처를 공유한다.

## System Architecture

- 시스템이 어떻게 동작하는지 설명
  - 시스템 구성 및 동작 원리
- 구성 요소 간의 관계 및 시스템 외부 환경과 관계 묘사
- 시스템 설계에 대한 제약 사항

## Enterprise Architecture

- 조직의 프로세스 및 정보 시스템 및 부서의 구조와 기능을 포괄적으로 기술
- 조직이 전략적 목표에 따라 행동하도록 방향 제시



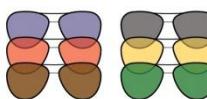
## 1.2

### ❖ 아키텍처 구조와 뷰(View)

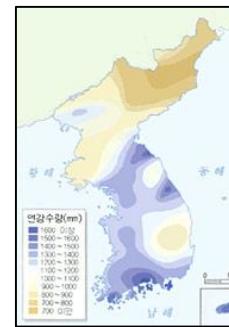
- 구조란 아키텍처적인 요소들의 집합이며, 이러한 구조의 표현이 View이다



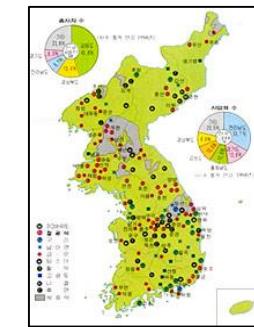
모든 것을 한번에  
표현하기에는 너무 복잡하다



특정 관점의 선택  
(관심의 대상이 무엇인가?)



관심의 대상이 되는  
일부 요소만을 표현



View란 시스템 중에서 관심 있는 시스템 요소와 그들간의 관계를 표현하는 것이다.

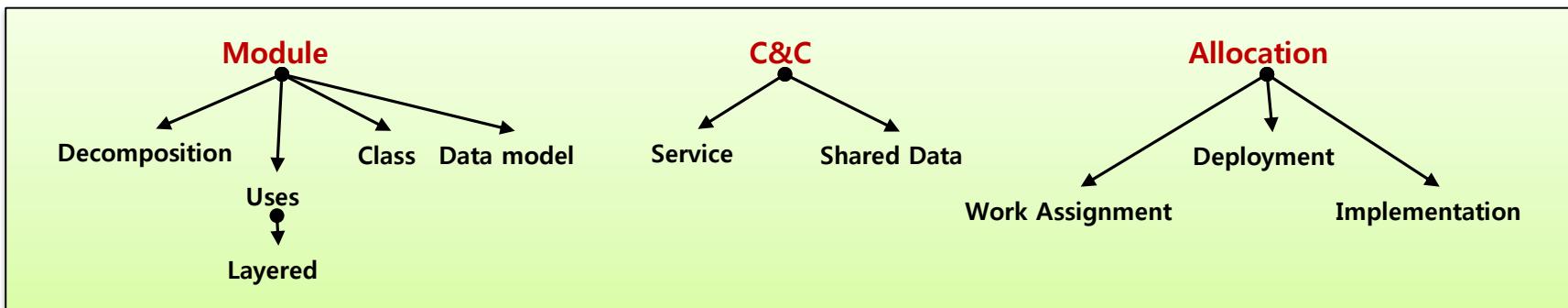
### ❖ 3가지 아키텍처 구조

|  |  |
|--|--|
| 모듈 구조<br>(Module Structures)                         | <ul style="list-style-type: none"><li>구현과 관련된 모듈의 집합으로 시스템을 표현</li><li>How is it structured as a set of code unit?</li></ul>   |
| 컴포넌트와 커넥터 구조<br>(Component-and-Connector Structures) | <ul style="list-style-type: none"><li>런타임 행위와 상호작용을 갖는 요소의 집합으로 시스템을 표현</li><li>How is it structured as a set of elements that have run-time behavior and interacts?</li></ul> |
| 할당 구조<br>(Allocation Structures)                     | <ul style="list-style-type: none"><li>소프트웨어 요소가 환경적인 구조에 Mapping</li><li>How does it relate to non-software structures in its environment?</li></ul>                           |



# 1.2

## ❖ 일반적인 아키텍처 구조



|                   | Software Structure | Element Type  | Relations   | Useful For   | Quality Attributes Affected  |
|-------------------|--------------------|---------------|---|--|------------------------------|
| Module Structures | Decomposition      | Module        | Is a submodule of   | Resource Allocation, Project Structuring/Planning, Information Hiding, Configuration Control | Modifiability                |
|                   | Uses               | Module        | Uses  | Engineering subsets, Engineering extensions  | Subsetability, Extensibility |
|                   | Layers             | Layer         | Requires the correct presence of, Uses the services of, provides abstraction to | Incremental development, Implementing systems on the top of "virtual machines"               | Probatility                  |
|                   | Class              | Class, Object | Is an instance of, Shares access methods of                                     | In OOD System, factoring out commonality, Planning extensions of functionality               | Modifiability, Extensibility |
|                   | Data model         | Data entity   | {one, many}-to-{one, many}, Generalizes, Specializes                            | Engineering global structures for consistency and performance                                | Modifiability, Performance   |



## 1.2

### ❖ 일반적인 아키텍처 구조 (cont.)

|                       | Software Structure     | Element Type                   | Relations   | Useful For  | Quality Attributes Affected         |
|-----------------------|------------------------|--------------------------------|---|---|-------------------------------------|
| C&C Structures        | <b>Service</b>         | Service, ESB, Registry, Others | Runs concurrently with, May run concurrently with, Excludes, precedes, etc. | Scheduling analysis, Performance analysis   | Interoperability, Modifiability     |
|                       | <b>Concurrency</b>     | Processes, Threads             | Can run in parallel   | Identifying locations where resource contention exists, or where threads may fork, join, be created, or be killed | Performance, Availability           |
| Allocation Structures | <b>Deployment</b>      | Components, HW element         | Allocated to, Migrates to   | Performance, availability, security analysis  | Performance, Security, Availability |
|                       | <b>Implementation</b>  | Modules, File Structures       | Stored in   | Configuration control, integration, test activities   | Development Efficiency              |
|                       | <b>Work assignment</b> | Module, Organizational Units   | Assigned to   | Project Management, best use of expertise and available resources, management of commonality                      | Development Efficiency              |

- 위의 구조들은 다양한 관점을 제공하지만, 독립적이지 않다.
- 한 구조의 요소는 다른 구조의 요소와 관련이 있다.
- 참조 가능한 구조는 많지만 모든 구조를 다 참조할 수는 없고, 가능한 적은 수의 구조를 가지는 것이 좋다.

### ❖ 어떤 구조를 선택할 것인가 ?

- 시스템에 있어 가장 중요한 품질 속성에 대하여 통찰력을 주고 이용 가능한 구조를 선택

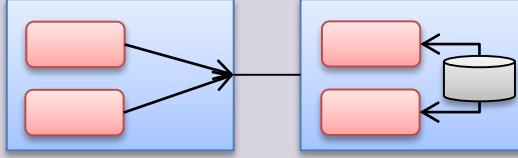
## ❖ 아키텍처 패턴

The ***Software Pattern*** is composition of architectural elements to solve particular problems.

It delineates the element types and their forms of interaction used in solving the problem

- Architectural Patterns

- |                |                      |              |               |
|----------------|----------------------|--------------|---------------|
| ✓ Distributed  | ✓ Pipes and filters  | ✓ Rule-based | ✓ Subsumption |
| ✓ Event-driven | ✓ Repository-centric | ✓ Layered    | ✓ Disposable  |
| ✓ Frame-based  | ✓ Blockboard         | ✓ MVC        |               |
| ✓ Batch        | ✓ Interpreter        | ✓ IR-centric |               |

|  |  |
|--|--|
|   | <ul style="list-style-type: none"> <li>▪ Architectural patterns can be used at the beginning of coarse-grained design addressing system-wide properties</li> </ul> |
|    | <ul style="list-style-type: none"> <li>▪ Design Patterns can be used to refine the architectural elements prior to implementation</li> </ul>                       |
| <pre>Import java.io.IOException; Public class MainThread extends Thread {     private String UserString;     public MyThread(String UserString)     {         this.UserString = UserString;         setDaemon(true);     } }</pre> | <ul style="list-style-type: none"> <li>▪ Idioms are used during detailed design and implementation</li> </ul>  |



# 1.4

## ❖ “좋은” 아키텍처의 요건

- 좋은 아키텍처와 나쁜 아키텍처를 나누는 기준은 없다.
- 특정한 목적에 적합한 아키텍처와 적합하지 않은 아키텍처가 있을 뿐이다.
  - 아키텍처 평가의 목적 : 시스템이 특정 목적에 부합하는지를 확인

## ❖ 권고 사항 (Rule of thumb)

- 아키텍처 수립 시 참고 사항

### 프로세스 측면의 권고 사항

- 아키텍트 팀에 의한 아키텍처 수립
- 요구 사항에 따른 QA 우선 순위화
- 아키텍처 문서화
  - Static view, Dynamic view 포함
- 이해 당사자를 통한 아키텍처 검증
- 성능 품질 → 정량적 수치로 평가
- 골격 시스템은 최소한의 기능만을 포함
- 자원 경쟁 → 정확하고 명확하게 명시 및 관리

### 제품(구조적) 측면의 권고 사항

- 기능 모듈 단위로 구분
- 잘 정의된 인터페이스 제공
- QA 정의 및 QA 목표 달성
- 특정 상용 제품이나 도구에 독립적
- 데이터 생성과 사용 모듈 구분
- 병렬 처리 시스템
  - 모듈을 하나의 컴퓨터나 커넥터로 표시 X
- Task, 프로세스의 문서화
- 특징적인 상호작용 패턴



## **Chapter 2. Why Is Software Architecture Important?**



# In this Chapter ...

## ❖ 기술적 측면에서의 아키텍처의 중요성에 대하여 이야기 한다.

- 시스템 품질 속성의 사용 또는 억제하는 기준이 된다.
- 변경에 대한 원인 파악과 관리를 가능하게 한다.
- 시스템 품질의 예측할 수 있도록 한다.
- 이해당사자와의 의사소통 수단이 된다.
- 시스템 초기의 설계 결정 사항을 기술한다.
- 개발 시 제약사항 정의한다
- 개발 조직 구조를 결정한다
- 점진적인 Prototype 개발을 가능하게 한다
- 비용과 일정을 정확하게 예측하도록 한다.
- 변경가능하고, 재사용 가능한 모델을 제공한다
- 독립적으로 개발된 컴포넌트들의 통합을 가능하게 한다
- 설계 선택 사항을 제한한다.
- 훈련의 기반이 된다.



## 2.1

### ❖ 시스템 품질 속성의 사용 또는 억제하는 기준이 된다.

- 시스템에 요구되는 품질 속성은 아키텍처에 의하여 결정된다.

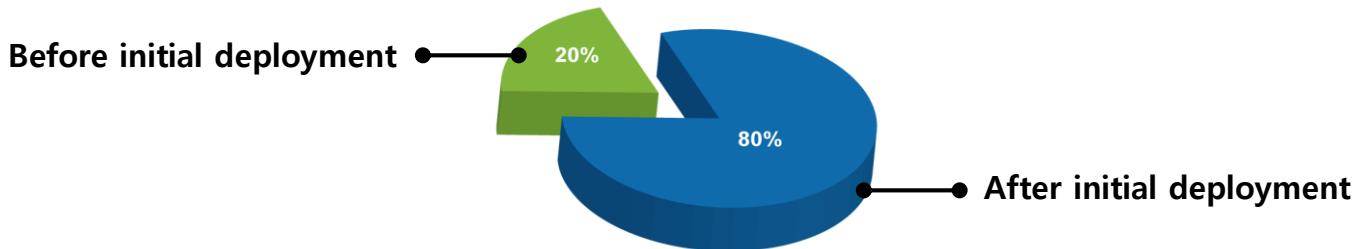


|                              |  |
|------------------------------|--|
| 고성능 (High Performance)       | <ul style="list-style-type: none"><li>▪ 요소들의 시간 기반의 행위</li><li>▪ 공유 자원의 사용</li><li>▪ 내부 요소들 간 커뮤니케이션의 빈도와 크기</li></ul> |
| 변경용이성 (Modifiability)        | <ul style="list-style-type: none"><li>▪ 시스템 요소에 책임 할당 → 변경 범위의 최소화</li></ul>   |
| 보안 (Highly Security)         | <ul style="list-style-type: none"><li>▪ 내부 요소의 외부 연동 관리 또는 금지</li><li>▪ 정보 조회 요소를 한정</li></ul>                         |
| 확장성 (Scalability)            | <ul style="list-style-type: none"><li>▪ 시스템 자원 사용의 내부 한정</li><li>▪ 자원 사용량, 한계를 코드상에 한정하지 말 것</li></ul>                 |
| 점증적 개발 (Deliver Incremental) | <ul style="list-style-type: none"><li>▪ 내부 컴포넌트 사용의 세밀한 관리</li></ul>   |
| 재사용성 (Reusability)           | <ul style="list-style-type: none"><li>▪ 내부 요소간 Coupling 제한</li><li>▪ 시스템 환경과 독립적인 개발</li></ul>                         |

- 아키텍처만으로 시스템에 요구되는 기능과 품질을 보장하지 못한다.
  - 개발주기상의 모든 결정들이 품질에 영향을 미친다.
  - 품질은 설계, 구현, 배포 등 모든 단계에 영향을 받는다.
  - 좋은 품질을 보장하기 위해 좋은 아키텍처가 필요하지만, 아키텍처가 품질을 보장하지는 않는다.

❖ 변경에 대한 원인 파악과 관리를 가능하게 한다.

- 소프트웨어 시스템의 총 비용



- 소프트웨어 시스템은 생명 주기에 따라 계속 변화한다.
  - 시스템 변화의 타입

|                       |  |
|-----------------------|--|
| 지역적 (Local)           | ▪ 하나의 요소만을 수정하는 경우.                                  |
| 전역적 (Nonlocal)        | ▪ 많은 요소를 수정하지만 아키텍처적인 변화는 아니다.                       |
| 아키텍처적 (Architectural) | ▪ 요소들 간의 상호작용에 영향을 준다.<br>▪ 시스템 전반에 대한 변경을 요구하기도 한다. |

- 효과적인 아키텍처는 변화를 쉽게 처리할 수 있도록 한다.
- 시스템 변경에 필요할 때, 어느 부분을 어떤 방법으로 변경하는지를 결정하기 위해서는, 시스템 요소들과 요소의 행위, 요소들 간의 관계에 대한 이해가 필요하다.

## 2.3

### ❖ 시스템 품질의 예측할 수 있도록 한다.

- 아키텍처는 시스템의 품질을 고취하며, 품질을 예측할 수 있게 해준다.



- 아키텍처적인 결정이 어떤 품질 속성에 영향을 주는지 알 수 있다면,
  - 아키텍처적인 결정을 할 수 있다.
  - 아키텍처적인 결정과 관련된 속성의 결과를 올바르게 예측할 수 있다.

정량적인 분석이 어려운 경우라도, 때때로 아키텍처가 규정된 결과를 가져 올 수 있는지 보장할 필요가 있다



## 2.4

### ❖ 이해당사자와의 의사소통 수단이 된다.

- 소프트웨어 아키텍처는 시스템을 추상적으로 표현
  - 대부분의 이해당사자들이 서로 이해하고, 타협하는 동의 수단으로 사용 가능하다
  - 이해당사자는 아키텍처에 영향을 주는 각기 다른 시스템의 특성에 관심을 갖는다.



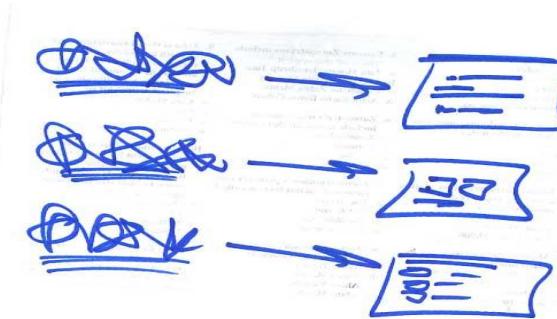
- 소프트웨어 아키텍처는
  - 대부분의 비기술적인 사람들도 이해할 수 있을 정도로 추상화 되어야 한다.
  - 구현, 통합, 테스트, 배포를 가이드 하기 위한 기술적인 명세로 정제할 수 있어야 한다.



## 2.5

### ❖ 시스템 초기의 설계 결정 사항을 기술한다.

- 소프트웨어 아키텍처는 시스템 개발 초기의 설계 결정 사항을 표현
  - 초기의 결정 사항은 최종 산출물에 영향을 준다.
  - 초기의 결정 사항은 나중에 변경하는 것이 힘들다. → 물결 효과 (Ripple effect)



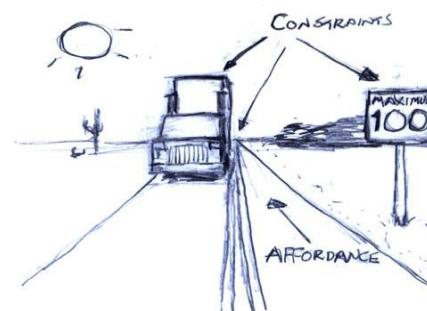
- 초기 결정 사항은 시스템 개발과 유지 보수와 관련된 많은 결정 사항을 해결한다.
- 시스템 개발 초기에 시스템 분석을 가능하게 한다.
- 때때로 아키텍처는 리팩토링 되거나 재설계 되어야 한다. 그러나 이런 작업은 쉽게 수행하기가 어렵다.



## 2.6

### ❖ 개발 시 제약사항 정의한다

- 시스템 개발 시, 아키텍처에 기술된 설계 구조를 준수 → 아키텍처의 실현
  - 개발 단위 = 아키텍처 구성 요소 단위
  - 아키텍처에 기술된 대로 다른 요소들과 상호작용
  - 아키텍처에 기술된 요소의 기능을 수행



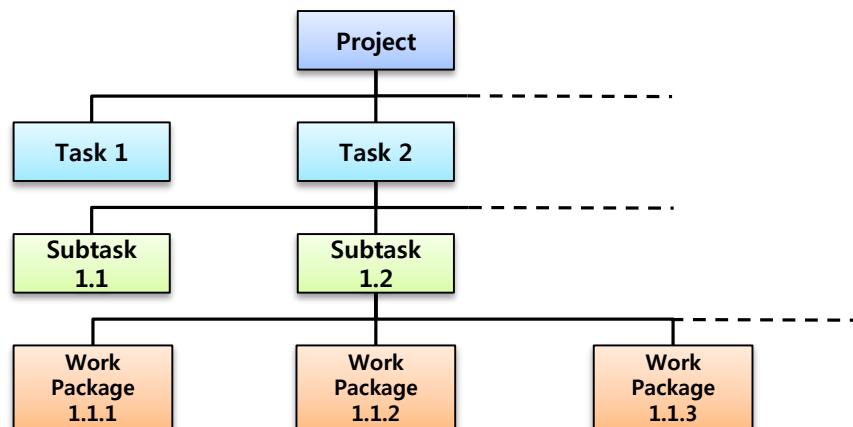
- 구현 시 각 요소의 명세를 지켜야 하지만, 요소들 간의 Trade-off를 고려
- 개발 인력 분배도 제약 사항
  - 개발자는 인식하지 못하나 관리자는 인력과 환경을 최적으로 구성할 수 있는 정보
  - 개발자 → 설계 명세를 이해하고 구현
  - 아키텍트 → 아키텍처의 Trade-off 고려



## 2.7

### ❖ 개발 조직 구조를 결정한다

- 아키텍처는 시스템 구성 구조와 아키텍처 구조를 통한 개발 조직간 관계도 표현
- 대규모 시스템 개발 시 개발 단위 구분 → 개발 부분 할당
  - WBS (Work Breakdown Structure)



- 시스템 아키텍처는 최상위 수준의 시스템 분할 구조를 표현
  - ➔ 작업 분할 구조의 기초로 활용 : 개발 계획, 일정, 비용 계산에 참고
  - 작업 분할 구조 수립은 소프트웨어 아키텍처의 한 측면을 고정 (Side effect)
- 아키텍처가 결정되면 여러 이유로 변경이 어렵다.
  - 대규모 시스템에서 소프트웨어 아키텍처를 확정하려면, 아키텍처 평가가 선행되어야 한다.



## 2.8

### ❖ 점진적인 Prototype 개발을 가능하게 한다

- 아키텍처가 결정되면, 분석 가능한 Prototype 형태의 골격 시스템(Skeletal System)으로 개발이 가능



- 골격 시스템은 최종 시스템이 만들어지기 전 적절한지 여부를 판단하게 한다.
- 실행 가능한 시스템의 경우 초기에 잠재적인 문제를 발견하게 한다.
- 골격 시스템은 실제로 정확함 (Fidelity)가 떨어지기도 한다.
- 프로젝트의 잠재적인 위험 요소(Risk)를 줄인다.
  - 군 제품(Family Product) 개발의 경우, Prototype을 위한 Framework 개발은 비용 분산 효과가 있다.

## ❖ 비용과 일정을 정확하게 예측하도록 한다.

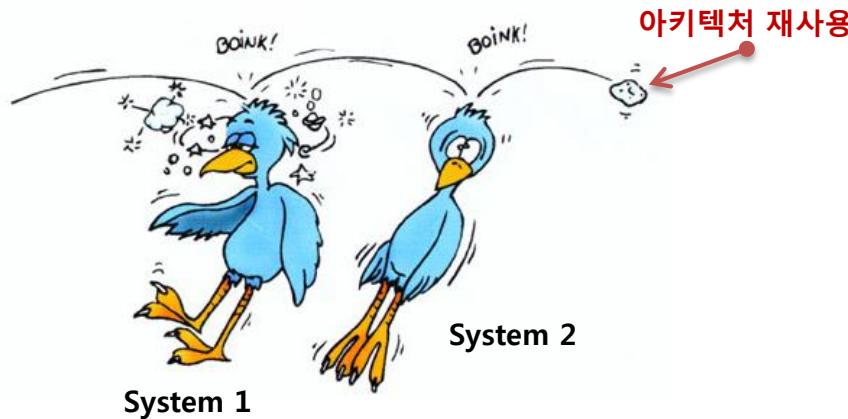
- 비용과 일정의 예측은 프로젝트에 필요한 자원과 문제를 해결하는 관리 도구로 활용 가능하다.
  - 프로젝트 조직 구성은 아키텍처에 의해 결정된다.
  - 비용의 예측은 대략적인 시스템 정보로 도출하는 방식(Top-down)보다, 세부적인 정보를 이해하고 도출하는 방식(Bottom-up)이 정확하다
  - 프로젝트 관리자보다는 프로젝트 내, 팀 단위로 비용과 일정을 예측하는 것이 더 정확하다



시스템 범위에 대한 정보를 더 많이 알수록, 비용과 일정을 좀 더 정확히 예측할 수 있다.

## ❖ 변경가능하고, 재사용 가능한 모델을 제공한다

- 프로젝트 초기에 아키텍처를 재사용하면 많은 혜택을 볼 수 있다.
  - 코드의 재사용뿐 아니라, 아키텍처를 결정하는 요구사항이나 아키텍처, 아키텍처 개발 경험 등 아키텍처 관련된 자료를 여러 시스템에서 재사용 할 수 있다.

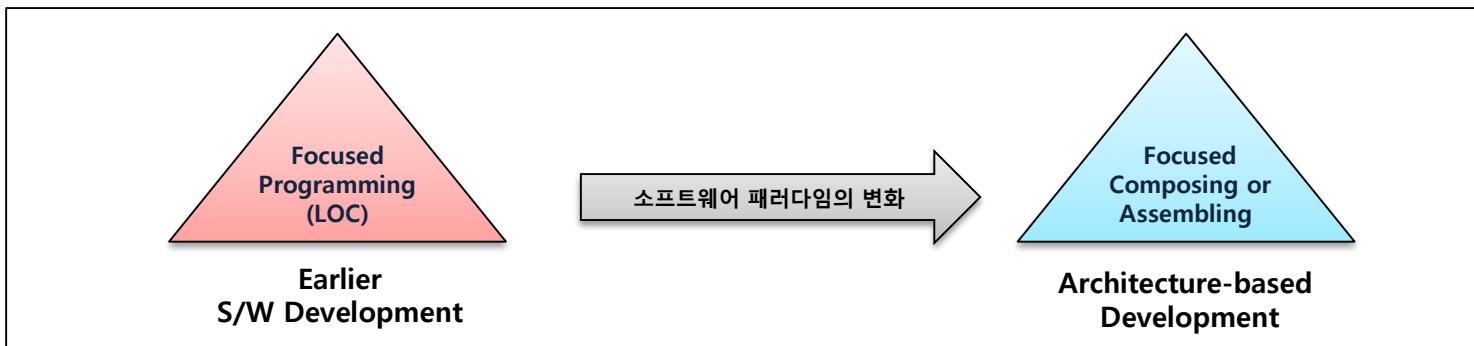


- 소프트웨어 Product Line은 시스템 특징을 공유한다.
  - 핵심 공통 모듈의 개발 → 제품군에서 필요로 하는 것을 기술한 아키텍처가 가장 중요
  - Product Line에서 공유하는 아키텍처는 제품군 내 모든 멤버를 고려하여 선택



## ❖ 독립적으로 개발된 컴포넌트들의 통합을 가능하게 한다

- 소프트웨어 패러다임의 변화

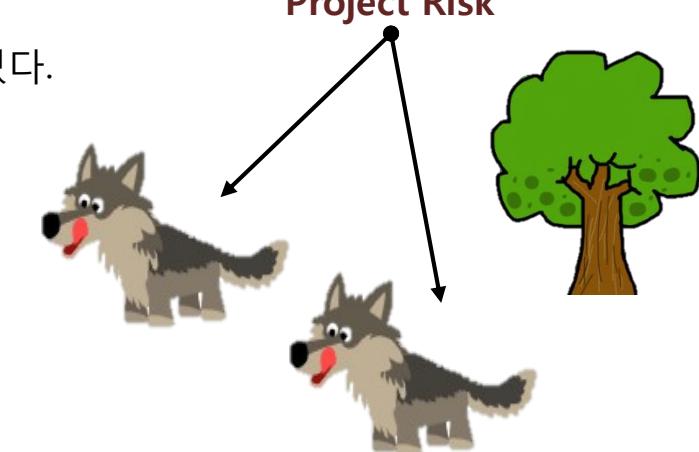
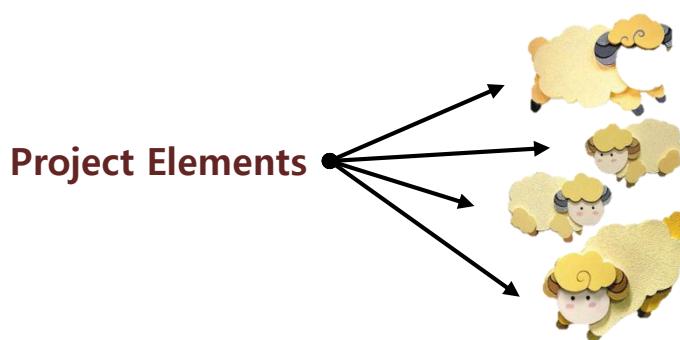


- 아키텍처에는 시스템에 포함될 수 있는 아키텍처 요소들이 정의되어 있다.
  - 아키텍처 요소들은 독립적으로 개발하여 통합할 수 있다.
  - 아키텍처에는 외부 개발 요소를 추가하거나 기존 요소들을 대체하기 위한 조건이 제시될 수 있다.
  - 아키텍처 요소와 인터페이스, 운영 개념의 조합 → 교환용이성 (Interchangeability)
    - 시장 출시 시간의 단축
    - 신뢰성 증가
    - 비용 감소
    - 유연성 증가



## ❖ 설계 선택 사항을 제한한다

- 소프트웨어 요소들은 수많은 방법으로 조합될 수 있다
  - 복잡도가 높아질 가능성이 높다.
- 아키텍처 패턴이나 설계 패턴을 한정하면,
  - 설계 시 복잡도를 낮추고, 재사용을 높일 수 있다.
  - 이해하기 쉽고, 의사소통이 쉬워진다.
  - 분석이 쉬워지고, 개발 시간이 단축된다.
- 아키텍처 패턴을 이용하면,
  - 설계의 이해도를 높일 수 있다.
  - 요구사항의 불일치성을 찾을 수 있다.
  - 충돌이 발생하는 설계 제약 사항을 중재할 수 있다.





# 2.13

## ❖ 훈련의 기반이 된다.

- 아키텍처는 시스템의 행위를 설명한다.
  - 요소와 요소들의 상호작용 방법을 기술
- 아키텍처 문서들은 새로운 프로젝트 멤버의 훈련에 도움을 준다.
  - 아키텍처는 다양한 이해당사자 간의 대화 수단으로 공통적으로 참조할 수 있다.





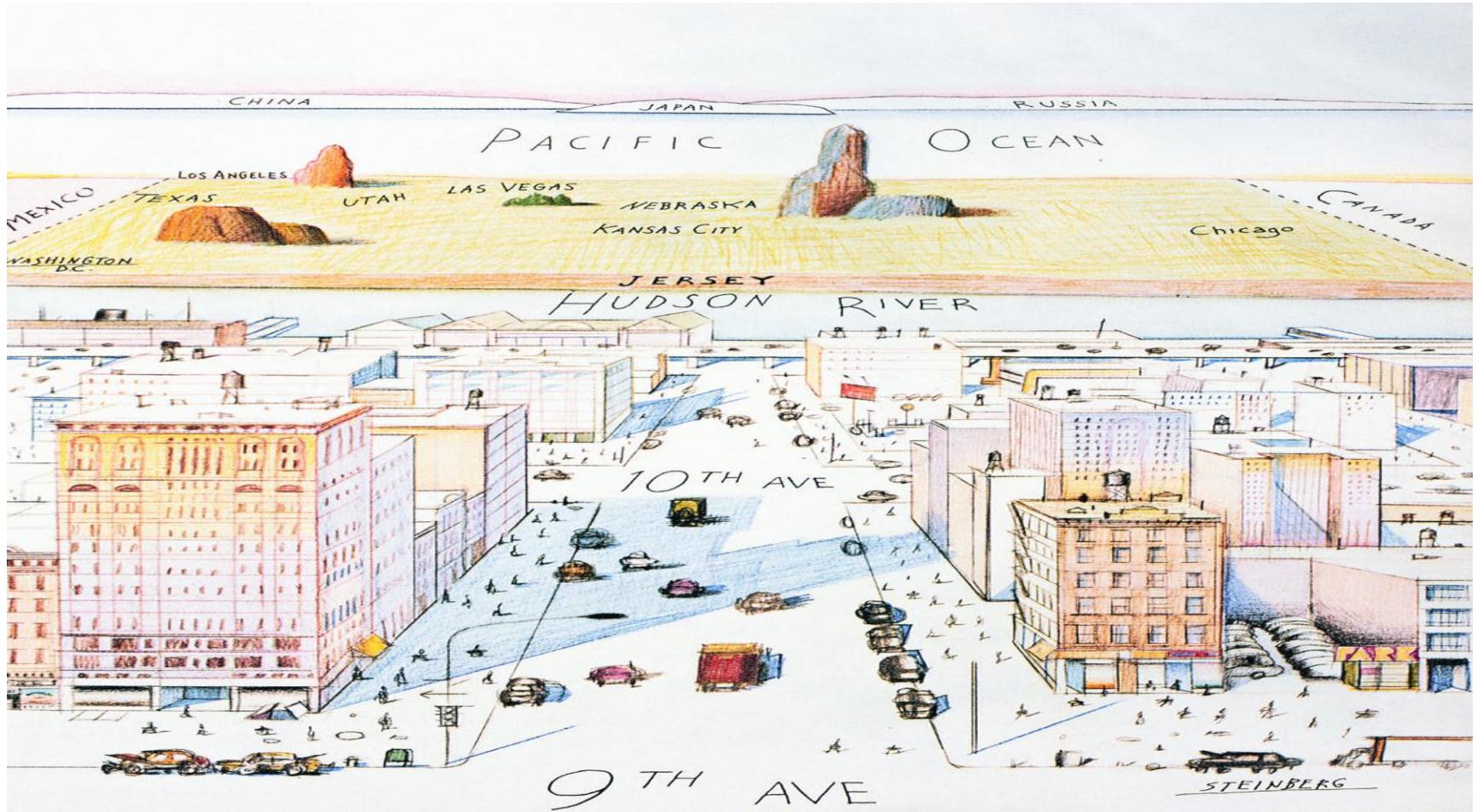
## **Chapter 3. The Many Contexts of Software Architecture**



# In this Chapter ...

## ❖ Starts with Saul Steinberg's cartoon

- View Of The World From 9th Avenue (New Yorker's view of the world)





# In this Chapter ...

- ❖ 다음과 같은 4가지 Context를 소개한다.

Technical

What technical role does the software architecture play in the system or systems of which it's part?

Project Life Cycle

How does a software architecture relate to the other phases of a software development cycle?

Business

How does the presence of a software architecture affect an organization's business environment?

Professional

What is the role of a software architect in an organization or a development project?

- 컨텍스트 자체는 변하지 않지만, 특정 시스템에 적용 시 명확히 할 필요가 있다.
- 아키텍트의 과제 중 하나는 어떤 컨텍스트가 변경 가능한지, 시스템과 개발에 어떻게 적용시켜야 하는지에 대한 것이다.



## 3.1

### ❖ Architecture in a Technical Context

- 아키텍처는 시스템 품질속성에 영향을 미친다. (2장 참조)
  - 아키텍처는 품질속성을 장려하거나 억제한다.
  - 아키텍처를 통하여 시스템 품질의 여러 측면을 예측할 수 있다.
  - 아키텍처는 변경에 대한 원인 파악과 관리를 가능하게 한다.

|                           |  |
|---------------------------|--|
| 고성능 (High Performance)    | <ul style="list-style-type: none"><li>▪ 요소들의 시간 기반의 행위</li><li>▪ 공유 자원의 사용</li><li>▪ 내부 요소들 간 커뮤니케이션의 빈도와 크기</li></ul>   |
| 가용성 (Availability)        | <ul style="list-style-type: none"><li>▪ 이벤트 실패 시, 컴포넌트 상호간 동작 방법</li><li>▪ 오류 발생 시 시스템의 반응</li></ul>   |
| 사용성 (Usability)           | <ul style="list-style-type: none"><li>▪ 사용자 인터페이스와 사용자 경험에 책임이 있는 요소들을 시스템의 다른 부분과 고립 → Tailoring과 향상을 통한 시간 단축</li></ul>  |
| 테스트 가능성 (Testability)     | <ul style="list-style-type: none"><li>▪ 개별 요소에 대한 테스트 가능성<ul style="list-style-type: none"><li>- 각 요소의 상태가 관찰 가능하고, 통제 가능하도록 한다.</li><li>- 요소들이 함께 동작할 때 비정상 행위에 대한 이해</li></ul></li></ul> |
| 안전성 (Safety)              | <ul style="list-style-type: none"><li>▪ 요소들의 행위를 감추고, 비정상 동작에 대한 처리</li></ul>  |
| 상호 운용성 (Interoperability) | <ul style="list-style-type: none"><li>▪ 요소의 외부와의 상호작용과 상호작용에 대한 통제</li></ul>   |

- 아키텍처를 설계할 때의 기술적인 환경은 아키텍처에 영향을 미친다.
  - 기술적 환경이란, 업계 표준적인 사례나 엔지니어링 기법들이다. → 그러나 변한다.

❖ 스웨덴의 전함 “바사(Vasa)호”의 교훈

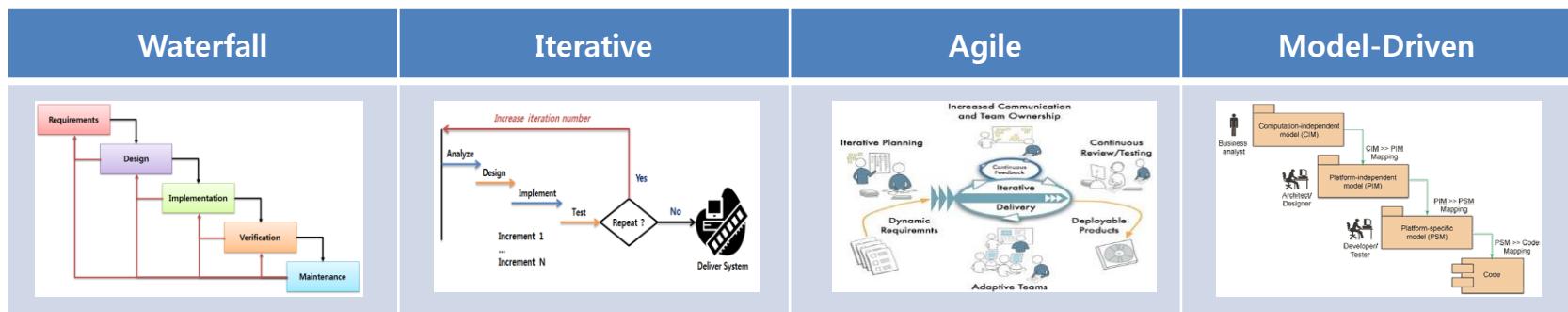


- 요구 사항을 만족시킬 수 있는 성공적인 아키텍처 사례를 통하여 현재에 적용 할 수 있도록 해야 한다.
- 사례가 없는 시스템을 설계할 경우 시스템을 구축하기 전에 아키텍처를 평가 할 수 있는 방법을 통해 위험을 감소시켜야 한다.
- 점진적인 아키텍처 기반의 개발 기법을 통하여 너무 늦지 않도록 설계 오류를 수정할 수 있어야 한다.

## 3.2

### ❖ Architecture in a Project Life-Cycle Context

- 소프트웨어 개발 프로세스는 소프트웨어 개발을 위한 표준적인 접근 방법이다.
  - 개발 프로세스는 소프트웨어 엔지니어와 팀에 대한 원칙을 강제한다.
  - 대표적인 4 가지 소프트웨어 개발 프로세스



- 어떤 개발 프로세스를 사용하던, 아키텍처 관련 활동이 포함되어 있다.
  - 시스템에 대한 비즈니스 사례 만들기
  - 아키텍처적으로 중요한 요구사항 이해하기
  - 아키텍처 생성 또는 선택
  - 아키텍처에 대한 문서화와 커뮤니케이션
  - 아키텍처 분석과 평가
  - 아키텍처에 기반한 구현과 시스템 테스트
  - 아키텍처에 대한 구현 준수에 대한 보장



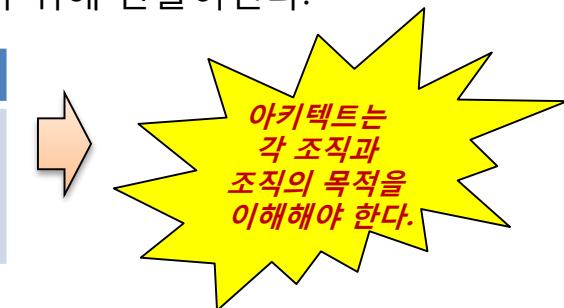
## 3.3

### ❖ Architecture in a Business Context

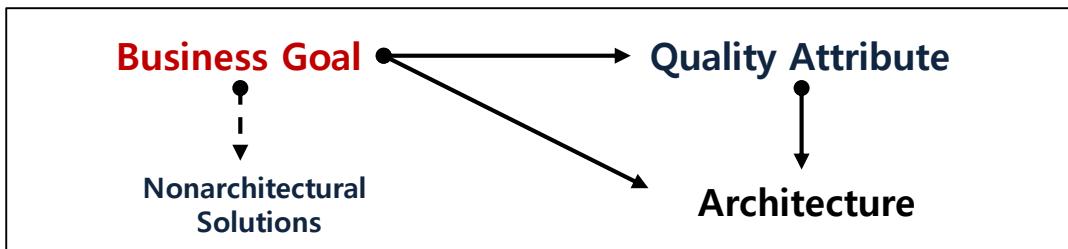
#### ■ 아키텍처와 비즈니스 목표

- 아키텍처와 시스템은 비즈니스 목표를 가지고 구현된다.
- 시스템은 하나 이상의 조직의 비즈니스 목표를 만족시키기 위해 만들어진다.

| 개발 조직  | 고객   | 정부, 하청업체  |
|--|--|---|
| <ul style="list-style-type: none"><li>▪ 수익 창출</li><li>▪ 시장 확보</li><li>▪ 사업 유지</li><li>▪ 고객 편의 증진</li></ul> | <ul style="list-style-type: none"><li>▪ 편의 증진</li><li>▪ 생산성 향상</li></ul> | <ul style="list-style-type: none"><li>▪ 시스템에 대한 규정 준수</li></ul> |



- 비즈니스 목적은 아키텍처에 많은 영향을 준다.



- 비즈니스 목적은 품질속성 목표로 명백해 질 수 있다.
- 어떤 비즈니스 목적은 요구사항 형식으로 보여지지 않는다.
- 어떤 비즈니스 목적은 아키텍처에 영향을 미치지 않는다.

### 3.3

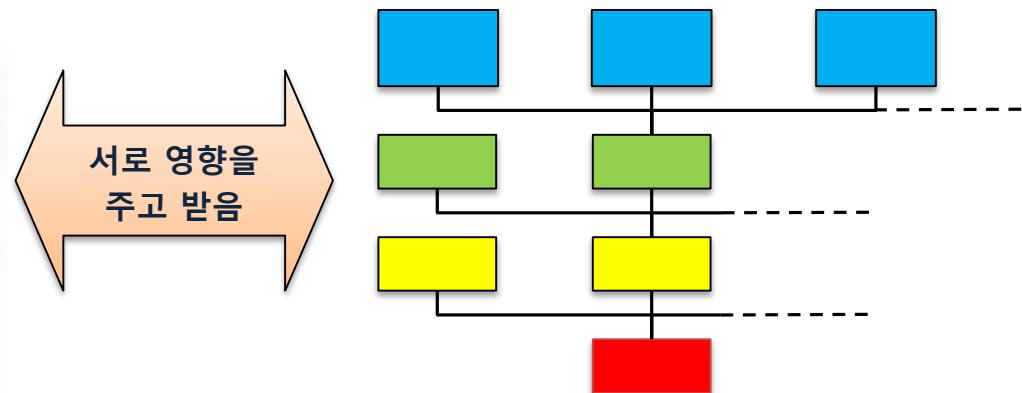
## ❖ Architecture in a Business Context

### ■ 아키텍처와 개발 조직

- **개발 조직의 특성이나 구조는 아키텍처에 영향을 미친다.**
  - 개발자들의 숙련도, 개발 일정, 개발 예산
  - 조직이 가진 기준 아키텍처나 제품 기반의 개발을 할 수 있다. → 재사용 측면의 장점
- 조직은 기반 구조와 같은 장기적인 관점의 투자를 전략적인 목표에 의해 할 수 있다.
  - 기반 구조에 대한 투자는 개발 조직에 영향을 준다.
- 조직인 구조는 소프트웨어 아키텍처의 형태를 결정한다. (역도 성립)
  - 개발 조직은 기술적이고, 적용 개념으로 조직된다.



조직 구조



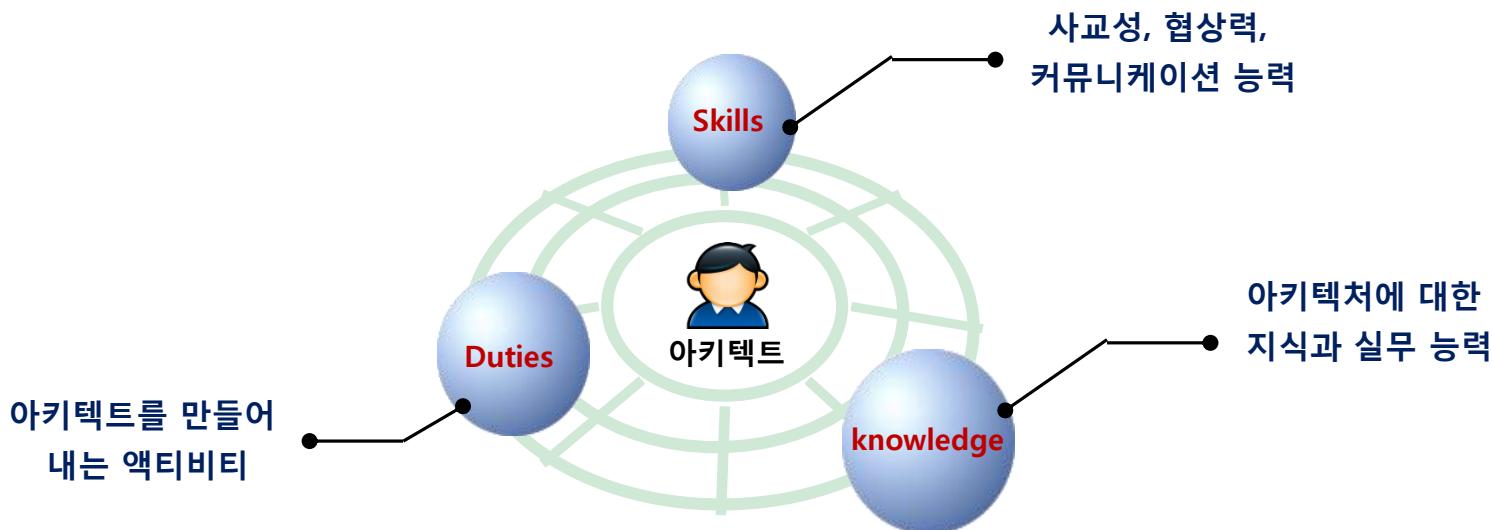
S/W 아키텍처



## 3.4

### ❖ Architecture in a Professional Context

- 아키텍트에게는 기술적인 능력 이외의 것이 필요하다.



- 아키텍처의 배경과 경험도 아키텍처에 영향을 준다.
  - 아키텍트가 경험한 프로젝트에서의 아키텍처 적용 사례 (성공과 실패 사례 모두)
  - (아키텍처 패턴이나 기술에 대한) 교육과 훈련에 의한 아키텍처적인 결정
  - 아키텍트의 능력

# 3.5

## ❖ Stakeholders

- S/W 시스템에 직간접적으로 이해 관계를 가지는(관심이 있는) 사람 또는 조직
  - 고객, 사용자, 개발자, 프로젝트 관리자, 유지 보수자, 마케팅 책임자 ...
- 이해 당사자는 각자 시스템에 대한 다양한 요구사항을 가진다
  - 성능, 신뢰성, 가용성, 변경 용이성, 사용편의성, 상호 운용성, 호환성, 효율 ...
  - 이해 관계자의 목적에 따라 요구사항이 충돌 → 우선 순위를 가능한 빨리 알아야 한다
  - 아키텍처 평가, 반복적인 Prototype 을 통해 사용자 참여를 유도

시스템 동작, 성능,  
보안, 신뢰성, 사용  
편의성까지..



사용자

적은 비용,  
고용 기술자 유지...



개발조직  
관리자

깔끔한 UI, Time-  
to-Market, 적은  
비용, 경쟁력



마케팅  
담당자

적은 비용, 일정  
준수, 시스템 개발  
후 변경 않됨...



고객

변경 용이성,  
시스템 이해도가  
높게...



유지보수  
담당자



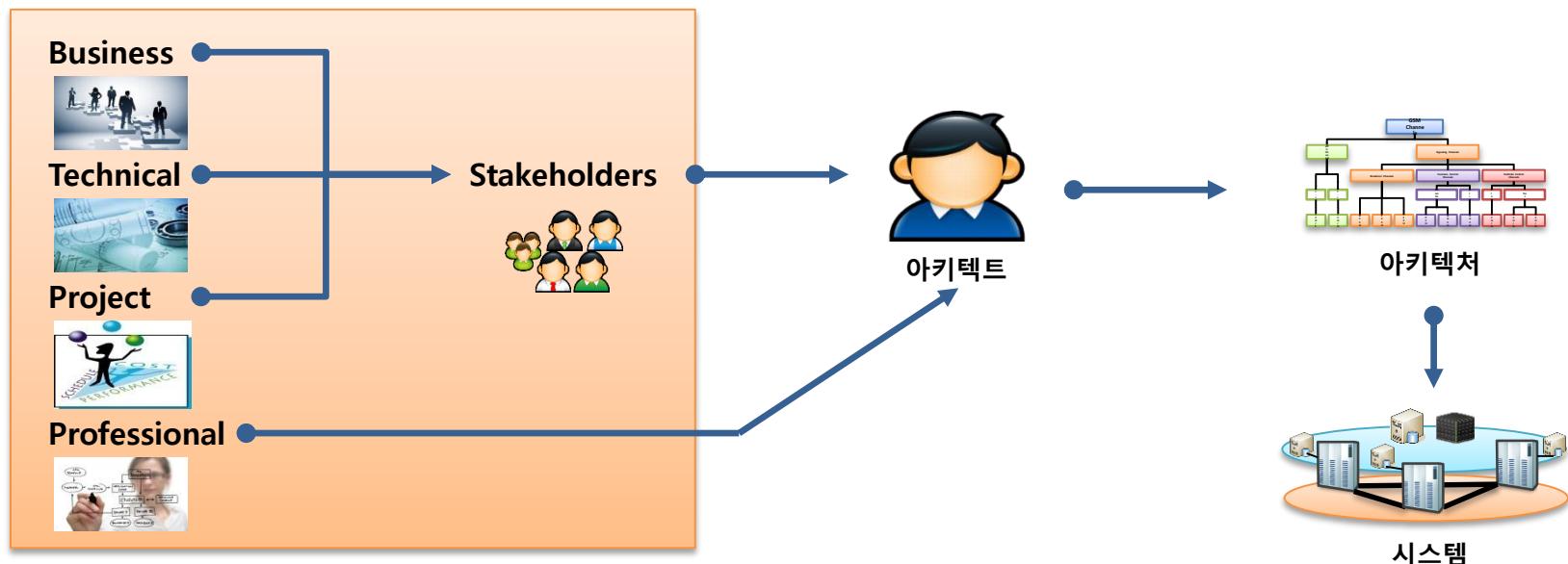
아키텍트

아키텍트는 시스템 특성을  
모두 기록하여 테스트가 가능한 수준까지  
작성할 수 있어야 한다.

## ❖ How Is Architecture Influenced ?

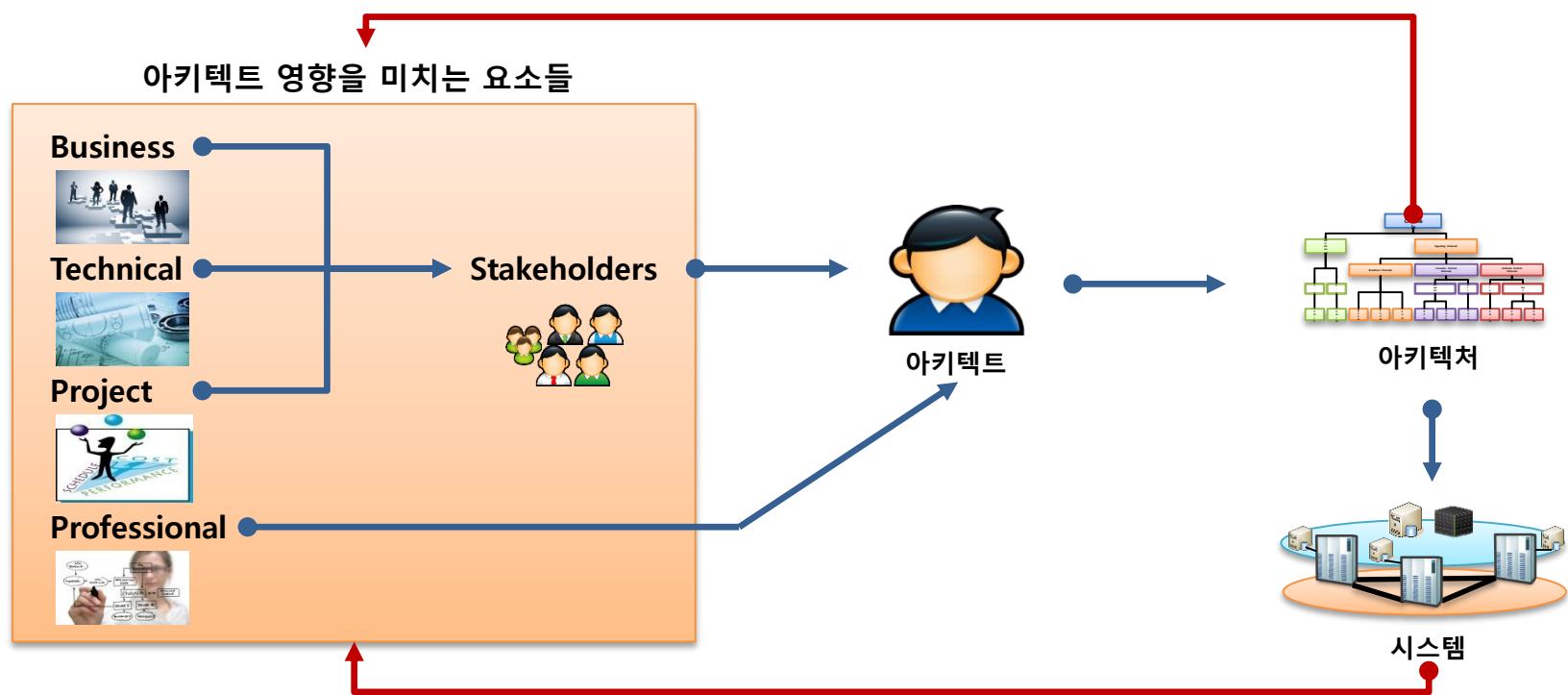
- 아키텍처는 비즈니스, 기술, 프로젝트, 아키텍트의 경험 등에 영향을 받는다.
- 아키텍트는 아키텍처를 만들고, 이를 기반으로 시스템이 구축된다.

아키텍트 영향을 미치는 요소들



## ❖ What Do Architectures Influence?

- 아키텍처는 비즈니스, 기술, 프로젝트, 아키텍트의 경험 등에 영향을 받는다.
- 아키텍트는 아키텍처를 만들고, 이를 기반으로 시스템이 구축된다.
- **아키텍처는 비즈니스, 기술, 프로젝트, 아키텍트의 경험에 영향을 준다.**
- **아키텍트의 환경적인 요소들은 미래의 아키텍처에 영향을 준다.**





# Overview of Part 2 : Quality Attributes

## ❖ Part 2 Provide technical foundation about ...

- Design or analyze an architecture to achieve particular quality attribute
- Not discuss design or analysis processes

## ❖ Chapter 4. Understanding Quality Attributes

- Specify a quality attribute, requirement, and tactics
- 7 categories for design decisions

## ❖ Chapter 5 ~ 12. Particular quality Attributes

- Availability, Interoperability, Modifiability, Performance, Security, Testability
- Additional Quality Attributes

## ❖ Chapter 13. Architectural Tactics and Patterns

- Some of most important patterns and tactics and their relationship

## ❖ Chapter 14. Quality Attribute Modeling and Analysis

- Modeling techniques for some of quality attributes



## **Chapter 4. Understanding Quality Attributes**



# In this Chapter ...

## ❖ 왜 품질속성(Quality Attribute)에 대하여 논의하는가?

- 품질 속성들은 시스템의 기능(Functionality)과 더불어 아키텍처에 영향을 준다.
- 시스템이 재설계되는 이유는 대부분 기능 부족 때문이 아니다.
  - 유지보수, 포팅, 확장성, 성능 등의 이슈 → 품질 속성과 관련된다.

## ❖ 품질 속성의 정의

품질 속성은 이해 당사자의 요구를 시스템이 어떻게 잘 만족시키는가를 나타내는,  
측정 가능하고 (Measurable) 테스트 가능한 (Testable) 시스템의 특성이다.

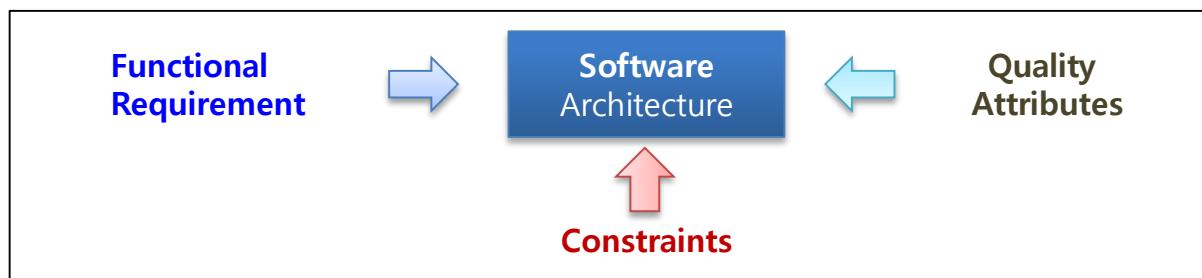
## ❖ 이번 장에서는 다음의 사항에 중점을 둔다.

- 구축하고자 하는 시스템이나 시스템들에 제공되는 아키텍처에 원하는 품질 속성을 어떻게 표현 할 것인가?
- 어떻게 품질 속성(의 수준)을 만족시킬 것인가?
- 각 품질 속성에 대한 설계적인 결정은 어떻게 내릴 것인가?



## ❖ Architecture and Requirement

- 시스템에 대한 요구사항은 다양한 형태를 가진다.
  - 요구사항 문장, 실물 모형(mockup), 기존 시스템, Use case, 사용자 스토리 ...
- 요구사항의 3가지 분류
  - 기능 요구사항 (Functional Requirements)
    - 시스템이 반드시 제공해야 하는 기능을 의미한다.
    - 설계 전반에 걸쳐 적절한 순서로 책임(Responsibility)을 할당한다.
  - 품질 요구사항 (Quality Attribute Requirements)
    - 시스템이나 기능 요구사항 전반에 걸쳐 요구되는 능력을 의미한다.
    - 요소들간 동작과 상호작용을 정의한 다양한 구조들(Structure)를 아키텍처에 적용한다.
  - 제약사항 (Constraints)
    - 선택의 여지가 없는 설계 결정 사항이다.
    - 설계 결정 사항을 받아 들이고, 다른 설계 결정 사항과 조화를 이루게 해야 한다.





## 4.2

### ❖ **Functionality**

기능성(Functionality)이란 의도한 작업을 수행할 수 있는 시스템의 능력이다.

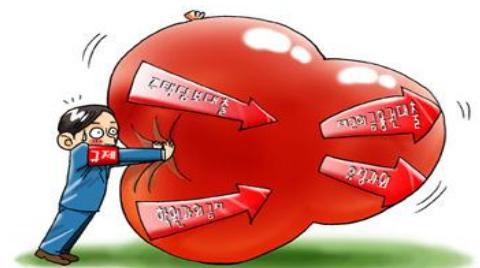
- 기능성은 아키텍처를 결정하지 않는다.
  - 주어진 기능을 만족시키는 아키텍처는 수없이 많다.
- 기능성은 임의의 특정 구조에 독립적이다.
  - 기능성은 아키텍처 요소에 책임을 할당함으로써 달성할 수 있다.
  - 책임은 임의의 모듈에 할당 될 수 있다.
  - 다른 품질 속성이 중요한 경우, 아키텍처는 책임의 할당을 제한 할 수 있다.



## ❖ Quality Attribute Considerations

- 시스템의 기능은 품질속성에 고려 없이 표현될 수 없다. (역도 성립)
- 기존 품질 속성에 대한 논의는 아키텍처 관점에서 다음과 같은 문제가 있다.
  - 기존 속성에 대한 정의들은 테스트가 가능하지 않다.
  - 종종 논의가 어느 품질의 관점에서 볼 것인가에 중점을 둔다.
  - 각 커뮤니티는 각자 자신들만의 용어를 발전시켜 왔다.
    - Event ? Or Attacks ?
    - Failure ? Or User Input?
- 품질에 대한 2가지 관점
  - 시스템 실행 시의 특성 : Availability, Performance, Usability
  - 시스템 개발 시의 특성 : Modifiability, Testability
- 복잡한 시스템에서, 품질속성은 고립되어 성취할 수 없다.
  - 여러 품질속성들간에 Trade-off가 발생한다.
  - ➔ 설계에 대한 결정이 필요 (17장에서..)

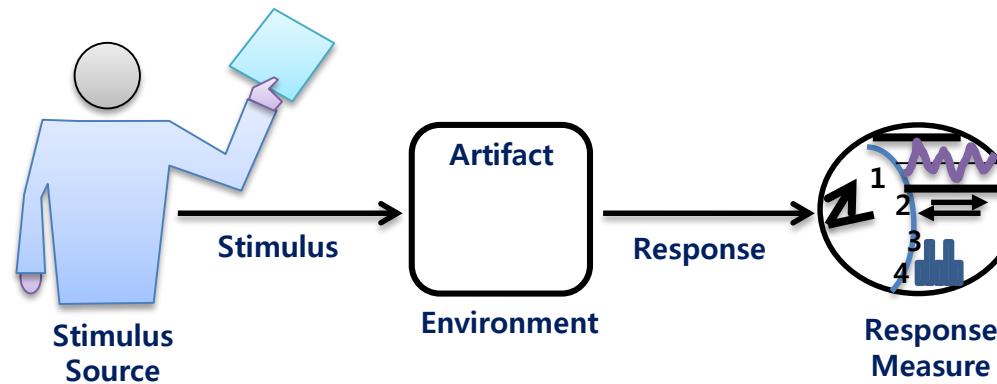
품질속성 시나리오로  
품질속성을 구체화  
각 품질 속성에 대한  
기본 관심 사항에 대한 토의



## ❖ Specify Quality Attribute Requirement

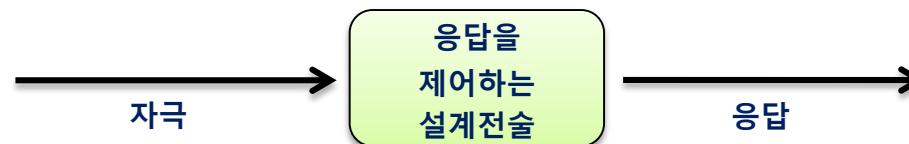
- 품질속성 요구사항은 모호하지 않고, 테스트 가능해야 한다.  
**→ 모든 품질속성 요구사항을 명세하기 위한 공통된 형식을 사용 !!!**
- 품질속성을 표현은 품질속성 시나리오를 이용한다.
  - 품질속성 시나리오의 6 가지 요소

|                          |   |
|--------------------------|---|
| 자극 유발원 (Stimulus Source) | <ul style="list-style-type: none"> <li>• 자극을 만들어내는 대상체</li> </ul>                       |
| 자극 (Stimulus)            | <ul style="list-style-type: none"> <li>• 이벤트가 시스템에 도착했을 경우</li> </ul>                   |
| 응답 (Response)            | <ul style="list-style-type: none"> <li>• 자극에 대하여 시스템이 취하는 동작 또는 행위</li> </ul>           |
| 응답 측정 (Response Measure) | <ul style="list-style-type: none"> <li>• 응답이 발생하면, 요구사항의 만족여부를 응답 측정을 통해서 검증</li> </ul> |
| 환경 (Environment)         | <ul style="list-style-type: none"> <li>• 시나리오가 발생하는 특정 환경의 집합</li> </ul>                |
| 대상체 (Artifact)           | <ul style="list-style-type: none"> <li>• 자극을 받는 대상</li> </ul>                           |



## ❖ Achieving Quality Attributes through Tactics

- 아키텍트 전술 (Architectural Tactics)
  - 요구되는 품질속성을 성취하기 위해 아키텍트가 사용 할 수 있는 기법
  - 품질속성의 응답을 제어하는데 영향을 주는 설계 결정 사항
  - 하나의 품질속성의 응답에 대해서 초점을 둔다. →
    - No Trade-off. : 아키텍처 패턴과의 차이점
  - 디자인 패턴과 같이 오랫동안 사용해 온 설계 기법들이다.



- 새로운 전술을 고안하기 보다, 적용할 수 있는 전술에 기술한다.
  - 디자인 패턴들은 복잡하다, 또한 때때로 적용하기가 어려운 경우가 있다.
  - 설계 목적을 실현시킬 패턴이 없는 경우, 아키텍트가 디자인 분할을 구성할 수 있도록 한다.
  - 일부 제한 사항에 대한 좀더 시스템적인 결정을 하도록 한다.
- 기존의 전술은 개선(refine)될 수 있다.

## ❖ Guiding Quality Design Decisions

- 아키텍처는 설계 결정의 집합을 적용한 결과이다.
  - 설계 결정은 설계 관점을 가장 문제가 있을 것 같은 것에 아키텍처가 초점을 갖게 하며 시스템적으로 분류할 수 있다.
  - 설계 결정은 다음과 같이 7가지로 분류 된다.

|   |  |
|---|--|
| <b>책임 할당</b><br>(Allocation of responsibilities)                | <ul style="list-style-type: none"> <li>• 기본적인 시스템 기능, 아키텍처 인프라, 품질속성을 만족시키기 위한 중요한 책임의 정의</li> <li>• 책임들의 정적(non-runtime), 동적(runtime) 할당</li> </ul>   |
| <b>조정 모델</b><br>(Coordination model)                            | <ul style="list-style-type: none"> <li>• 조정되거나 금지해야 하는 시스템 요소의 정의</li> <li>• 조정해야 할 특성 결정 (시간 적절성, 동시성, 완전성 ..)</li> <li>• 특성을 구현하기 위한 커뮤니케이션 메커니즘</li> </ul>  |
| <b>데이터 모델</b><br>(Data model)                                   | <ul style="list-style-type: none"> <li>• 주요 데이터의 추상화, 오퍼레이션, 데이터 특성 선택</li> <li>• 데이터의 일관성 있는 해석을 위한 메타데이터 편집</li> <li>• 데이터 조직화</li> </ul>  |
| <b>자원 관리</b><br>(Management of resources)                       | <ul style="list-style-type: none"> <li>• 관리되어야 하는 자원과 각각의 대한 한계 설정</li> <li>• 자원을 관리하기 위한 시스템 요소 맵핑</li> <li>• 자원의 공유 방법</li> <li>• 여러 자원들의 포화 상태에 대한 영향 결정</li> </ul>                               |
| <b>아키텍처 요소들 간의 맵핑</b><br>(Mapping among architectural elements) | <ul style="list-style-type: none"> <li>• 모듈과 실행 요소간의 맵핑</li> <li>• 실행 요소에 대한 프로세서 할당</li> <li>• 데이터 저장소에 저장할 데이터 모델 상의 아이템 할당</li> </ul>   |
| <b>바인딩 시간 결정</b><br>(Binding time decisions)                    | <ul style="list-style-type: none"> <li>• 시점에 따라 다른 엔터티와 연동 → 범위 내 변동을 가능하게 한다.</li> <li>• 책임의 할당 → 파라미터화 된 Makefile → build 시 선택 가능</li> <li>• 조정 모델 → 프로토콜 선택, 자원 관리 → 드라이버 선택, 사용 기술 선택</li> </ul> |
| <b>기술 선택</b><br>(Choice of technology)                          | <ul style="list-style-type: none"> <li>• 사용할 기술과 해당 기술을 지원하는 도구 선택</li> <li>• 기술의 내부적인 유사성과 외부적인 지원 정도</li> <li>• 선택된 기술의 부작용 확인과 신기술에 대한 호환성</li> </ul>   |



## **Chapter 5. Availability**



# In this Chapter ...

## ❖ 가용성(Availability)이란 ...

- 가용성이란 Task의 수행이 필요 시 수행 준비가 되어있음을 나타내는 S/W의 특성이다
  - 사용자 입장에서 시스템을 얼마나 사용할 수 있는가를 정도를 의미한다.
- 가용성은 신뢰성(Reliability)를 확장한 개념이다
  - Availability = reliability + recovery notion
  - Dependability = 허용 가능한 것보다 더 자주 일어나고, 더 심각한 실패를 회피할 수 있는 능력
- 가용성은 누적 서비스 중단 기간이 명시된 시간 동안에 요구되는 값을 초과하지 않도록 오류(faults)를 가리거나 회복할 수 있는 시스템의 능력을 의미한다.
- 가용성은 다른 속성들과 관련이 있다
  - 보안(Security)
  - 성능(Performance)
  - 안전(Safety)
- 가용성은 시스템의 오류를 완화시킴으로써 서비스 중단 시간을 최소화하는 것이다.





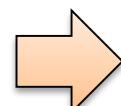
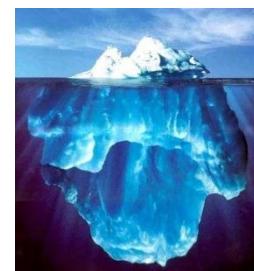
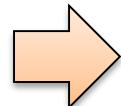
# In this Chapter ...

## ❖ Fault, Failure, Error

- 고 가용성(High-Availability)의 Fault-tolerant 한 시스템을 위해서는 운영 중 발생되는 Failure(실패)의 본질을 이해해야 한다.

- ✓ Fault – 시스템 실패의 원인
- ✓ Failure – 시스템이 정상적으로 동작하지 않는 상태로 사용자 등이 관찰 가능
- ✓ Error – 시스템 실패의 원인을 가지고 있지만 Failure가 발생하지 않은 상태

- Fault와 Failure의 구분은 자동화 된 회복 전략(Automated repair strategies)에 대한 논의를 가능하게 하다.
- Fault는 방지(prevented), 처리(tolerated), 제거(removed), 예측(forecast) 될 수 있다.





# In this Chapter ...

## ❖ 가용성(Availability)이란 ...

- 가용성은 확률적인 계산이 가능하다.

$$\text{Steady-state availability} = \frac{\text{MTBF}}{(\text{MTBF} + \text{MTTR})}$$

- MTBF = mean time to between failure
- MTTR = mean time to repair

## ❖ Planning for Failure

- 실패는 조건적인 것이 아니라, 대부분 불가피한 것이다.
- 시스템이 어떠한 실패의 경향이 있는지, 각각의 유형이 어떤 결과를 가져오는지에 대한 이해가 필요
- 실패를 처리하기 위한 3가지 기법
  - Hazard Analysis
  - Fault Tree Analysis
  - Failure Mode, Effect, and Criticality Analysis (FMECA)





# In this Chapter ...

## ❖ Hazard Analysis

- 시스템 운영 중 발생할 수 있는 위험들에 대하여 목록화
- 위험의 심각도에 따라 분류, 위험의 정의와 분류는 도메인마다 다름
- DO-178B Standard (항공기 시스템과 장비 인증에 관한 소프트웨어 고려사항)
  - Software Level and Structural Coverage Requirement

| Software Criticality Level | Failure Definition  | Associated Structure Coverage Level  |
|----------------------------|---|--|
| Level A                    | Software resulting in a <b>catastrophic</b> failure condition for the system              | Modified condition/Decision Coverage, Decision Coverage & Statement Coverage |
| Level B                    | Software resulting in a <b>hazardous</b> or severe-major failure condition for the system | Decision Coverage & Statement Coverage                                       |
| Level C                    | Software resulting in a <b>major</b> failure condition for the system                     | Statement Coverage   |
| Level D                    | Software resulting in a <b>minor</b> failure condition for the system                     | None Required  |
| Level E                    | Software resulting in <b>no effect</b> for the system                                     | None Required  |

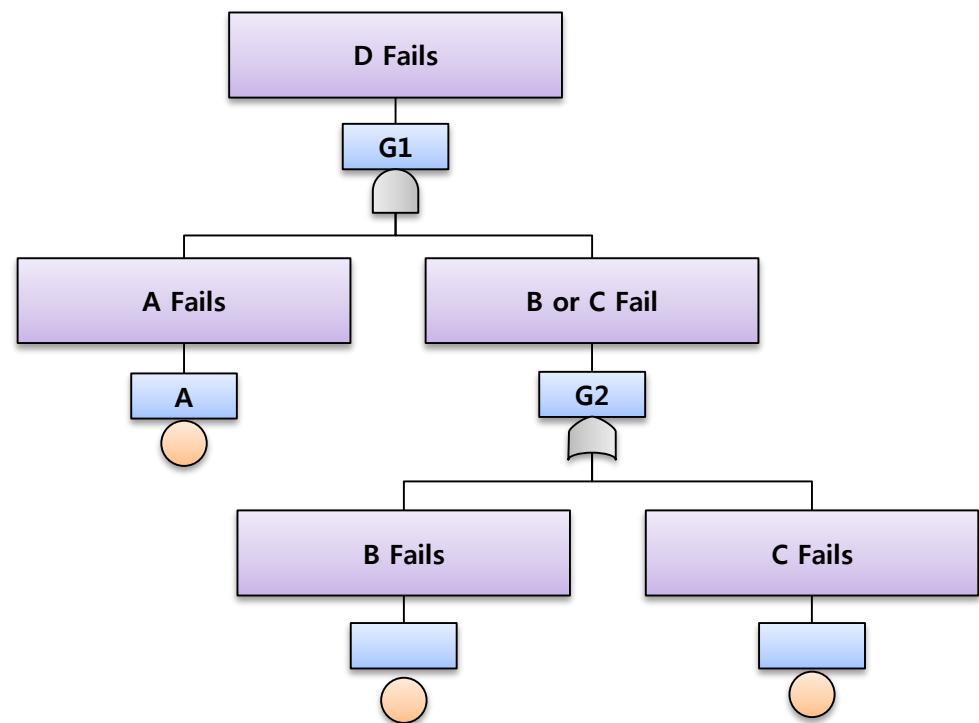


# In this Chapter ...

## ❖ Fault Tree Analysis

- 시스템의 안전과 신뢰성에 부정적인 영향을 주는 시스템의 상태를 명세하기 위한 분석적인 기법으로,
- 시스템의 바람직하지 않은 상태를 발견하기 위해 시스템의 컨텍스트와 작동방식을 분석

| Gate | Meaning      |
|------|--------------|
|      | AND          |
|      | OR           |
|      | COMBINATION  |
|      | EXCLUSIVE OR |
|      | PRIORITY AND |
|      | INHIBIT      |





# In this Chapter ...

## ❖ Failure Mode, Effect, and Criticality Analysis (FMECA)

- 시스템 실패의 종류를 실패가 갖는 심각도와 함께 목록화
- 과거의 유사 시스템의 실패 이력에 의존

| Component | Failure Probability | Failure Mode | %Failure by Mode | Effects           |             |
|-----------|---------------------|--------------|------------------|-------------------|-------------|
|           |                     |              |                  | Critical          | Noncritical |
| A         | $1*10^{-3}$         | Open         | 90               |                   | X           |
|           |                     | Short        | 5                | X ( $5*10^{-5}$ ) |             |
|           |                     | Other        | 5                | X ( $5*10^{-5}$ ) |             |
| B         | $1*10^{-3}$         | Open         | 90               |                   | X           |
|           |                     | Short        | 5                | X ( $5*10^{-5}$ ) |             |
|           |                     | Other        | 5                | X ( $5*10^{-5}$ ) |             |

- Probability of a critical system failure
  - $(5*10^{-5}) + (5*10^{-5}) + (5*10^{-5}) + (5*10^{-5}) = 2*10^{-4}$



# 5.1

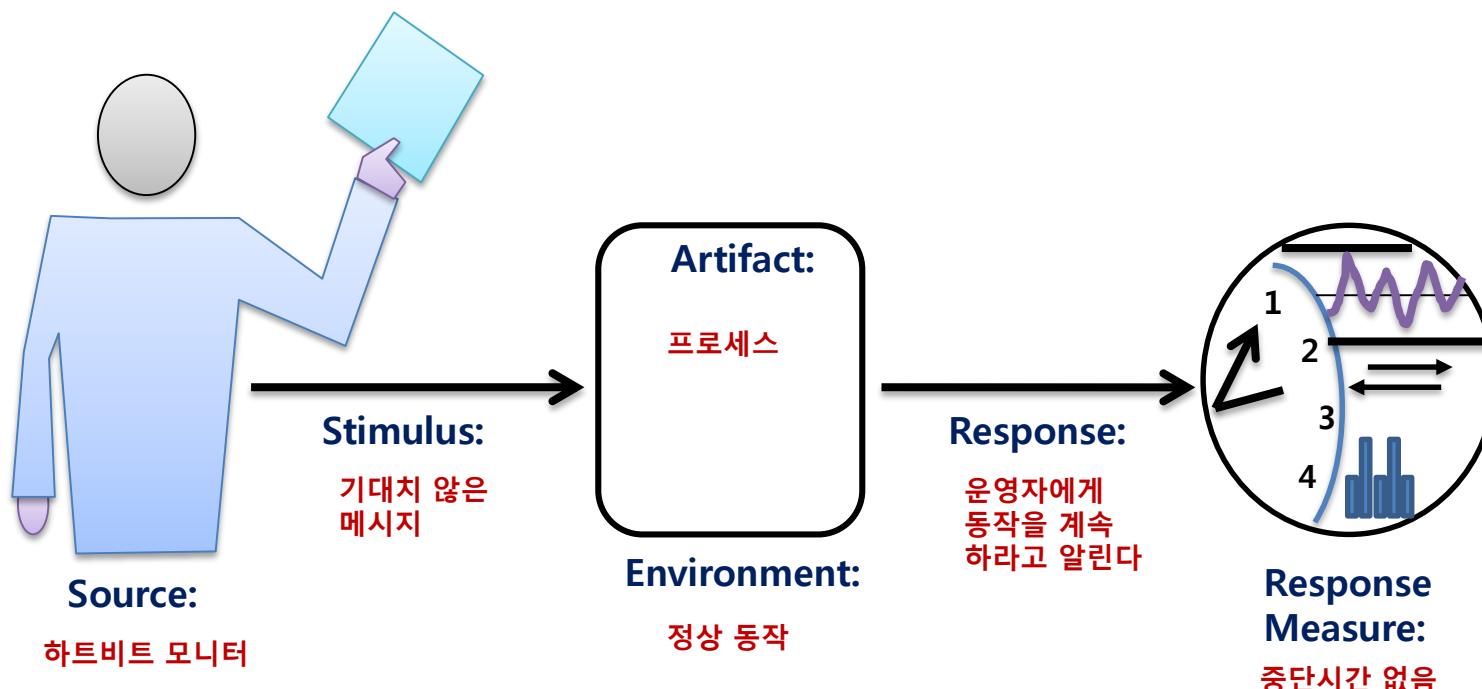
## ❖ Availability General Scenario

| 시나리오 항목          | 입력 가능한 값  |
|------------------|---|
| Source           | 시스템 내부/시스템 외부 : 사람, 하드웨어, 소프트웨어, 물리적인 하부구조, 물리적 환경  |
| Stimulus         | 결함 : 누락, 정지, 부정확한 타이밍, 부정확한 응답  |
| Environment      | 정상 동작, 시작, 종료, 수리모드, 저하된 동작, 가중된 동작   |
| Artifact         | 프로세서, 통신 채널, 영구 저장 장치, 프로세스   |
| Response         | <p>실패가 될 수 있는 결함을 방지한다.</p> <p>결함의 발견</p> <ul style="list-style-type: none"><li>▪ 결함의 기록</li><li>▪ 적절한 실체에 통보 (사람 또는 시스템)</li></ul> <p>결함을 복구한다.</p> <ul style="list-style-type: none"><li>▪ 결함의 원인이 되는 이벤트의 소스를 사용하지 못하게 한다.</li><li>▪ 수리 동안에 일시적으로 이용하지 못하도록 한다.</li><li>▪ 결함/실패 또는 손상의 원인을 포함하는 것을 고치거나 감춘다.</li><li>▪ 수리되는 동안 저하모드로 동작한다.</li></ul> |
| Response Measure | <p>가용성 백분율 (예를 들어 99.999)</p> <p>결함 감지 시간</p> <p>결함 수리 시간</p> <p>시스템이 비정상 모드에서 동작할 수 있는 시간 또는 시간 간격</p> <p>시스템이 방지하거나 실패없이 처리한 결함이 임의의 결함 분포나 비율</p>  |

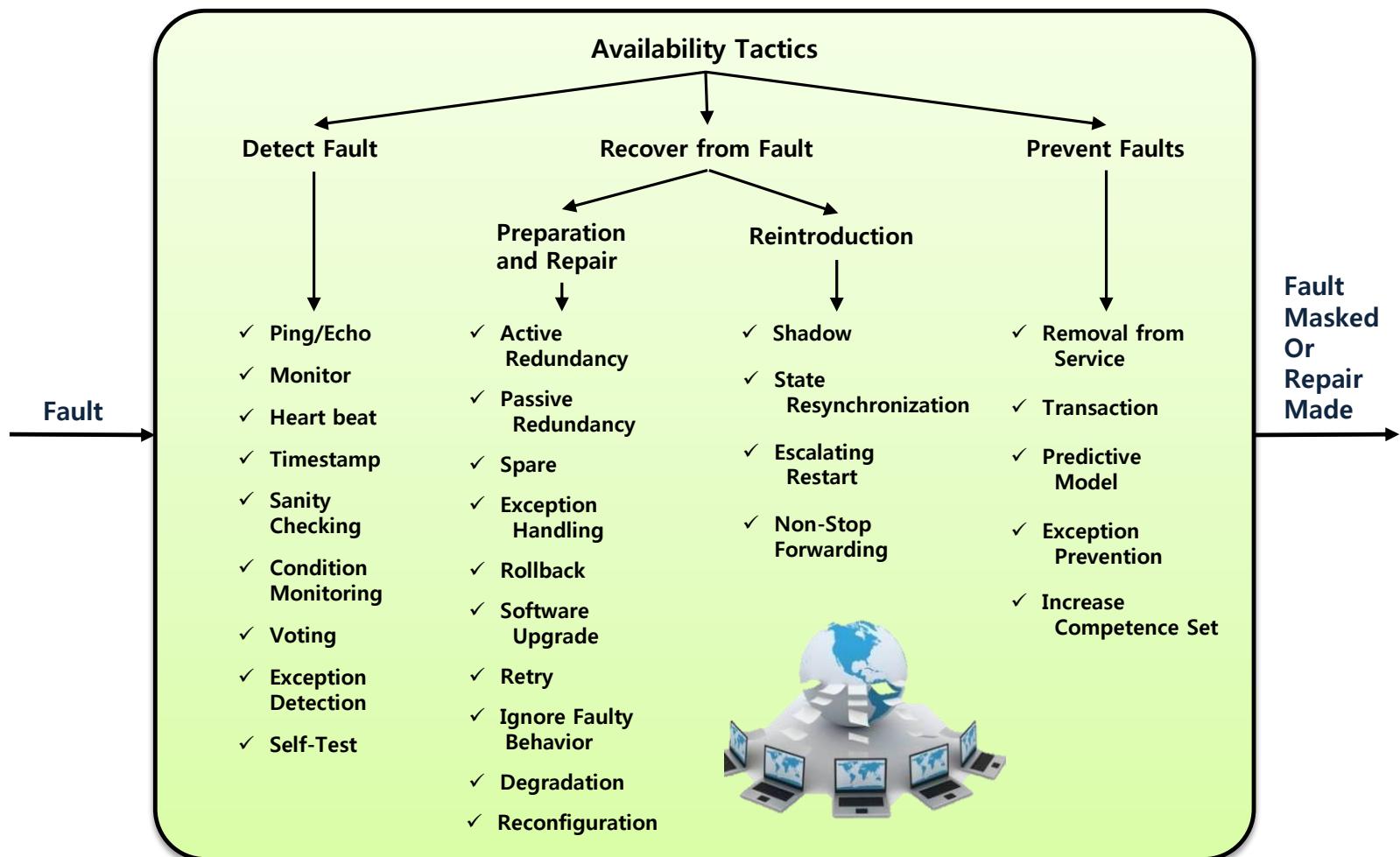
# 5.1

## ❖ Availability General Scenario

- Sample concrete availability scenario



## ❖ Tactics for Availability



## ❖ Availability Tactics

|                     |                             |  |
|---------------------|-----------------------------|--|
| <b>Detect Fault</b> | <b>Ping/Echo</b>            | <ul style="list-style-type: none"> <li>한 컴포넌트가 신호를 보내고 다른 컴포넌트로부터 정해진 시간 내에 응답이 오는지를 확인</li> </ul>                                     |
|                     | <b>Monitor</b>              | <ul style="list-style-type: none"> <li>시스템의 다양한 부분의 상태를 감시하는 컴포넌트(monitor)의 사용</li> </ul>  |
|                     | <b>Heartbeat</b>            | <ul style="list-style-type: none"> <li>한 컴포넌트로부터 주기적 신호/메시지를 받아 상태를 결함 여부를 파악</li> </ul>   |
|                     | <b>Timestamp</b>            | <ul style="list-style-type: none"> <li>분산 메시지 처리 시스템에서 잘못된 이벤트의 순서를 감지하기 위해 사용</li> </ul>  |
|                     | <b>Sanity Checking</b>      | <ul style="list-style-type: none"> <li>특정 동작이나 컴포넌트의 출력의 적절성/유효성을 체크</li> </ul>  |
|                     | <b>Condition Monitoring</b> | <ul style="list-style-type: none"> <li>프로세스나 디바이스의 상태를 체크</li> <li>설계 시 정의된 가정을 검증</li> </ul>  |
|                     | <b>Voting</b>               | <ul style="list-style-type: none"> <li>중복된 프로세스에서 실행되는 프로세스들이 같은 입력을 받고 투표자에게 값을 보내 다르게 동작하는 프로세스 감지 – 다수결 원칙</li> </ul>               |
|                     | <b>Exception Detection</b>  | <ul style="list-style-type: none"> <li>정상적인 시스템 흐름에서 변경된 시스템 상태의 감지<br/>- System Exception, Parameter fence/typing, Timeout</li> </ul> |
|                     | <b>Self-Test</b>            | <ul style="list-style-type: none"> <li>컴포넌트나 서브시스템이 올바른 동작을 위하여 자체 테스트를 수행</li> </ul>  |



## 5.2

### ❖ Availability Tactics

|                    |                        |                         |  |
|--------------------|------------------------|-------------------------|--|
| Recover from Fault | Preparation and Repair | Active Redundancy       | ▪ 중복된 컴포넌트들이 동시에 이벤트에 응답하여, 결과적으로 모두 동일 상태를 갖도록 한다.                                      |
|                    |                        | Passive Redundancy      | ▪ 한 컴포넌트가 이벤트에 응답하고, 대기 컴포넌트들에게 상태를 통보하면 대기 컴포넌트들도 업데이트를 수행                              |
|                    |                        | Spare                   | ▪ 다양한 종류의 컴포넌트 실패에 대하려는 목적으로 예비 컴포넌트 준비  |
|                    |                        | Exception Handling      | ▪ 예외 발생 시 시스템이 해당 예외 처리를 하여 시스템 동작이 계속 진행될 수 있도록 처리                                      |
|                    |                        | Rollback                | ▪ 실패를 감지한 경우 이전의 알려졌던 정상 상태로 되돌림   |
|                    |                        | Software Upgrade        | ▪ 서비스에 영향을 주지 않는 방법으로 실행 코드 이미지를 업그레이드   |
|                    |                        | Retry                   | ▪ 실패의 원인이 일시적인 경우, 재시도를 통하여 시스템 동작을 정상적으로 수행   |
|                    |                        | Ignore Faulty Behavior  | ▪ 특정 소스로부터 오는 메시지가 의미 없다고 판단한 경우 해당 소스의 메시지를 무시  |
|                    |                        | Degradation             | ▪ 시스템 실패 시 덜 중요한 컴포넌트의 기능은 내리고 가장 중요한 시스템 기능만 유지   |
|                    |                        | Reconfiguration         | ▪ 가능한 기능은 유지하면서, 자원에 대하여 책임을 재할당함으로 컴포넌트의 실패로부터 복구를 시도                                   |
| Reintroduction     | Reintroduction         | Shadow                  | ▪ 실패한 컴포넌트를 "Shadow Mode"로 작동시켜 정상 상태가 되면 다시 복구 시킴                                       |
|                    |                        | State Resynchronization | ▪ Active Redundancy나 Passive Redundancy에서 파트너에게 현재 상태를 통보                                |
|                    |                        | Escalating Restart      | ▪ 다양한 컴포넌트들의 재 시작 단위를 통한 시스템의 결함으로 복구를 지원하며, 이에 영향을 받는 서비스의 수준을 최소화함                     |
|                    |                        | Non-Stop Forwarding     | ▪ 라우터 설계에서 고안된 개념, Control Plane과 Data Plane으로 구분, Control Plane 상태에 관계없이 Data Plane은 동작 |



## 5.2

### ❖ Availability Tactics

|               |                         |   |
|---------------|-------------------------|---|
| Prevent Fault | Removal from Service    | <ul style="list-style-type: none"><li>실패를 회피하기 위해 현재 진행 중인 동작에서 특정 컴포넌트를 제거하는 것</li></ul>                             |
|               | Transactions            | <ul style="list-style-type: none"><li>일련의 순차적인 절차를 한꺼번에 원상태로 복구할 수 있도록 그 절차들을 묶어 놓는 경우 (ACID Properties)</li></ul>    |
|               | Predictive Model        | <ul style="list-style-type: none"><li>명목상의 운영 파라미터를 가지고 정상 행위를 하는지를 보장하기 위해 시스템의 상태를 감시하기 위하여 사용</li></ul>            |
|               | Exception Prevention    | <ul style="list-style-type: none"><li>시스템 예외 발생을 방지하기 위한 기법 – 예외 클래스 등</li><li>시스템 예외 발생 시 투명한 방식으로 시스템을 복구</li></ul> |
|               | Increase Competence Set | <ul style="list-style-type: none"><li>프로그램의 동작 시 더 많은 오류 경우에 대하여 처리할 수 있도록 설계하는 것을 의미</li></ul>                       |

# 5.3

## ❖ A Design Checklist for Availability

| 분류                                   | 체크리스트  |
|--------------------------------------|--|
| Allocation of Responsibilities       | <ul style="list-style-type: none"><li>✓ 고 가용성을 위한 시스템의 책임을 결정<ul style="list-style-type: none"><li>- 누락, 충돌, 잘못된 타이밍, 또는 잘못된 응답을 감지하기 위한 추가적인 책임의 할당을 포함</li><li>- 로깅, 통지, 이벤트 비활성화, 일시 정지, 결함/실패의 수정 및 감춤, 저하 모드 수행</li></ul></li></ul>             |
| Coordination Model                   | <ul style="list-style-type: none"><li>✓ 조정 메커니즘이 누락, 충돌, 잘못된 타이밍, 또는 잘못된 응답을 감지할 수 있는지 확인</li><li>✓ 조정 모델이 로깅, 통지, 이벤트 비활성화, 일시 정지, 결함/실패의 수정 및 감춤, 저하 모드 수행을 확인</li><li>✓ 조정 모델이 사용되는 대상체의 교체를 지원하는지 확인</li><li>✓ 조정 모델이 저하 모드에서 수행 여부 결정</li></ul> |
| Data Model                           | <ul style="list-style-type: none"><li>✓ 시스템의 어떤 부분이 고 가용성이 필요한지 결정<ul style="list-style-type: none"><li>- 데이터 추상화와 데이터와 관련된 동작 및 특성</li></ul></li></ul>  |
| Mapping among Architectural Elements | <ul style="list-style-type: none"><li>✓ 대상체와 발생되는 결함의 Mapping</li><li>✓ 아키텍처 요소들의 Mapping이 결함 복구 시 충분히 유연한지 여부</li></ul>   |
| Resource Management                  | <ul style="list-style-type: none"><li>✓ 결함이 발생한 경우 시스템이 계속 동작하기 위해 중요한 자원의 결정</li><li>✓ 중요 자원의 가용 시간 결정</li></ul>  |
| Binding Time                         | <ul style="list-style-type: none"><li>✓ 아키텍처 요소들이 언제, 어떤 방법으로 바인딩 될 것인지를 결정<ul style="list-style-type: none"><li>- 선택된 가용 전략이 알려진 결함을 대비 가능한지 확인</li></ul></li></ul>   |
| Choice of Technology                 | <ul style="list-style-type: none"><li>✓ 시스템 결함을 감지할 수 있는 기술의 선택</li><li>✓ 발생되는 결함과 선택된 기술의 Mapping 결정</li><li>✓ 선택된 기술 자체의 가용 특성 결정</li></ul>  |



## **Chapter 6. Interoperability**



# In this Chapter ...

## ❖ 상호 운용성(Interoperability)이란 ...

- 상호 운용성이란 둘 이상의 시스템들이 의미 있는 정보를 특정 컨텍스트 내에서 인터페이스를 통하여 교환할 수 있는 정도를 의미한다.
- 상호 운영성은 다음의 2가지 의미를 가진다.
  - Syntactic interoperability : 데이터를 교환 할 수 있는 능력
  - Semantic interoperability : 교환된 데이터를 올바르게 해석할 수 있는 능력



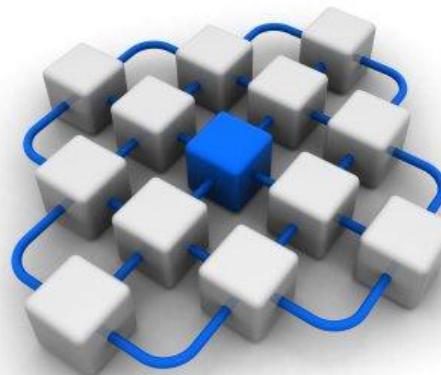
- 상호 운용성은 시스템들이 예상하는 운용 방식에 영향을 받는다.
  - 시스템이 상호 운영할 외부 시스템의 인터페이스를 미리 알고 있다면,
    - 해당 정보가 시스템에 포함되도록 설계가 가능하다.
    - 시스템을 좀 더 일반화 된 형식으로 상호 운영도록 설계할 수 있다.
- 상호 운영성은 “된다, 안 된다”의 명제가 아니라 여러 의미(shades of meaning)를 가진다.



# In this Chapter ...

## ❖ Systems of Systems

- 시스템들이 그룹으로 하나의 공통된 목적을 위해 상호 운영하는 경우
  - 각각의 독립적이고 유용한 시스템들을 더 큰 규모의 시스템으로 통합



- System of System 분류

|               |  |
|---------------|--|
| Directed      | <ul style="list-style-type: none"><li>• SoS objectives, centralized management, funding, and authority for the overall SoS are in place. Systems are subordinated to the SoS</li></ul>   |
| Acknowledged  | <ul style="list-style-type: none"><li>• SoS objectives, centralized management, funding, and authority in place, However, systems retain their own management, funding, and authority in parallel with the SoS</li></ul>                   |
| Collaborative | <ul style="list-style-type: none"><li>• There are no overall objectives, centralized management, authority, responsibility, or funding at the SoS level, Systems voluntarily work together to address shared or common interests</li></ul> |
| Virtual       | <ul style="list-style-type: none"><li>• Like collaborative, but systems don't know about each other</li></ul>  |



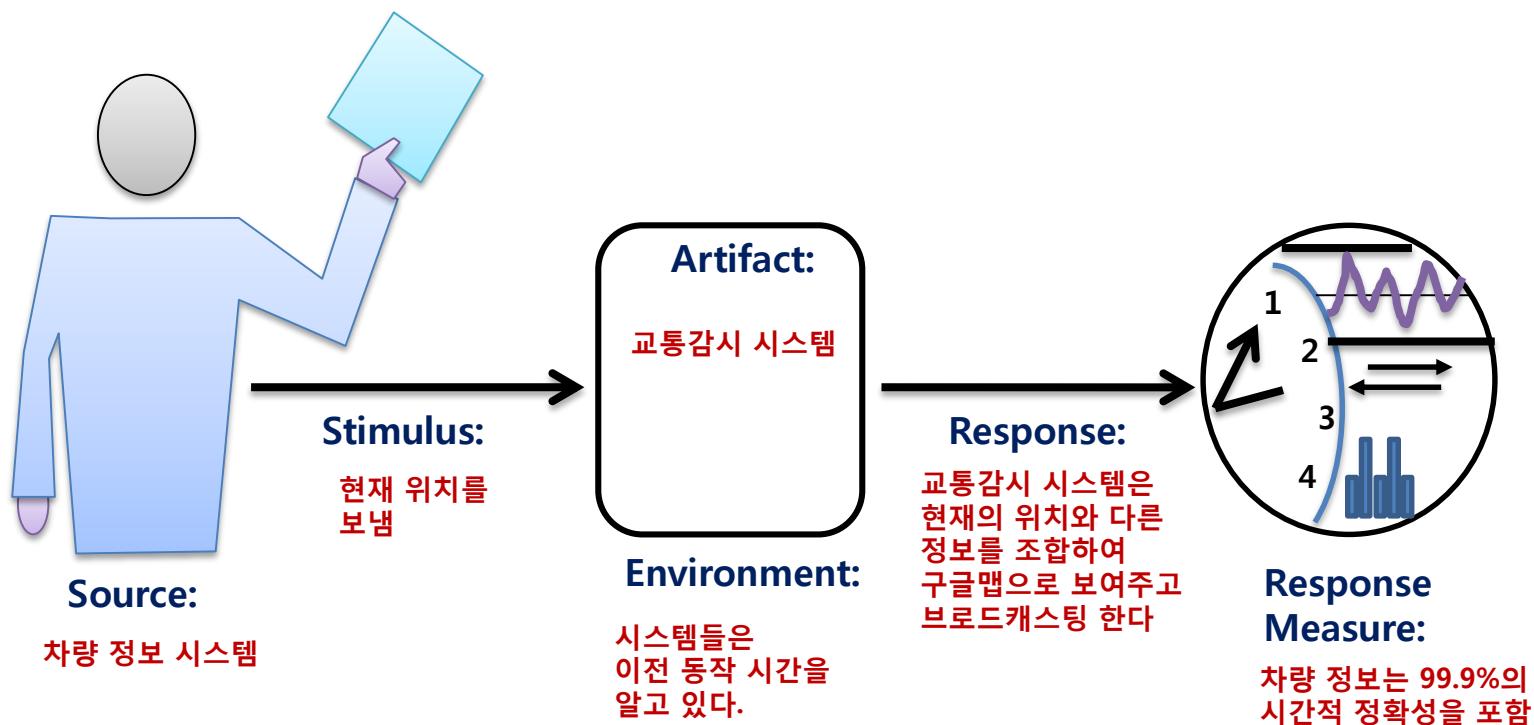
## 6.1

### ❖ Interoperability General Scenario

| 시나리오 항목          | 입력 가능한 값  |
|------------------|---|
| Source           | 시스템은 다른 시스템과 상호 운용에 대한 요청을 보낸다.   |
| Stimulus         | 시스템 사이의 정보를 교환하기 위한 요청  |
| Environment      | 상호 운영하고자 하는 시스템은 실행 시 발견되거나 실행 전 미리 알려져 있다.   |
| Artifact         | 상호 운용하고자 하는 시스템   |
| Response         | 다음 중 하나 또는 그 이상이 된다. <ul style="list-style-type: none"><li>▪ 요청이 (적절하게) 거부되거나 적당한 엔터티(사람 또는 시스템)에 통보된다.</li><li>▪ 요청이 (적절하게) 승인되고, 정보가 성공적으로 교환된다.</li><li>▪ 요청이 하나 또는 그 이상의 포함된 시스템들에 의해 기록된다.</li></ul> |
| Response Measure | 다음 중 하나 또는 그 이상이 된다. <ul style="list-style-type: none"><li>▪ 적절하게 처리된, 교환된 정보의 비율</li><li>▪ 적절하게 거부된, 교환 정보의 비율</li></ul>   |

## ❖ Interoperability General Scenario

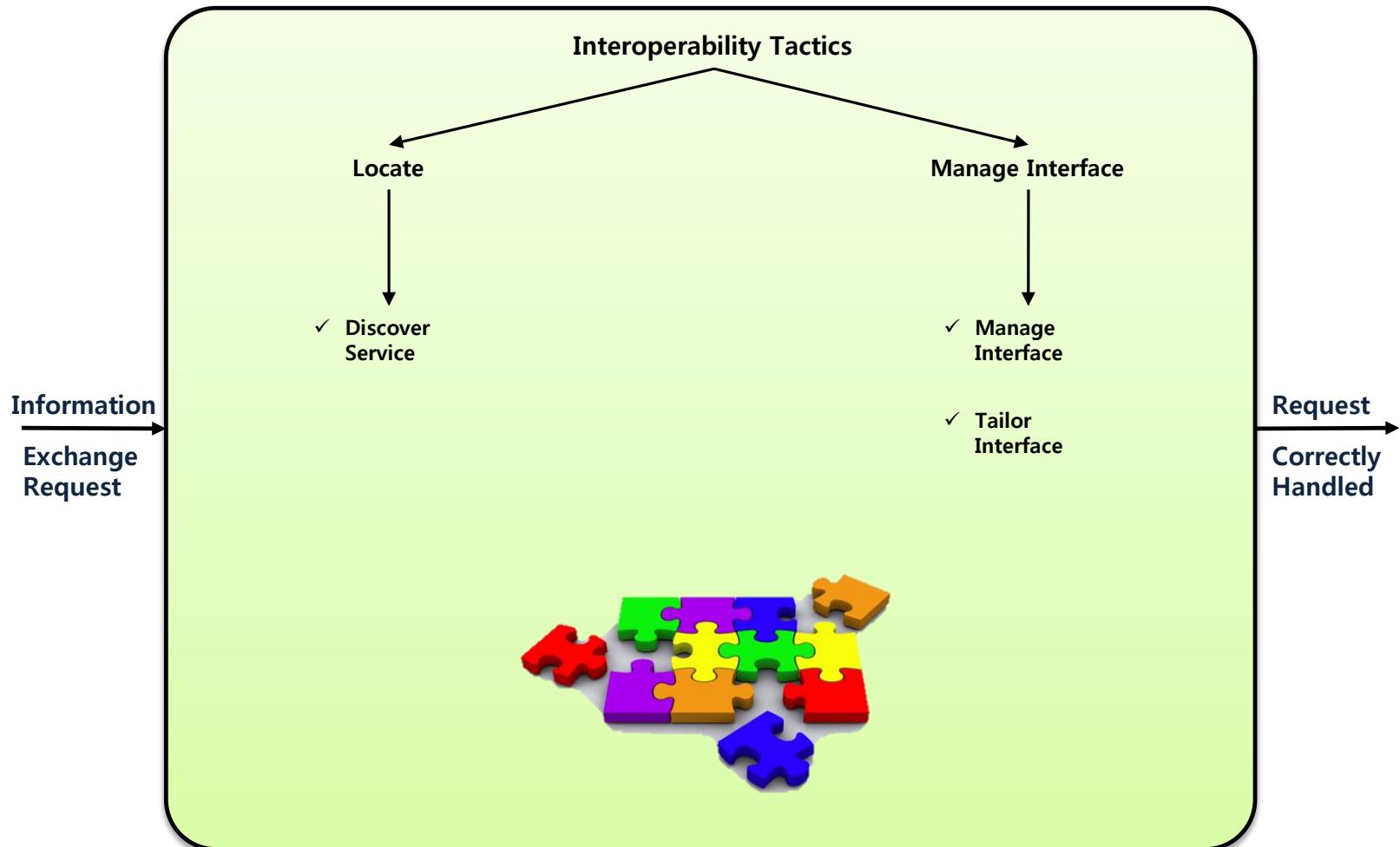
- Sample concrete interoperability scenario





6.2

## ❖ **Tactics for Interoperability**





## 6.2

### ❖ Interoperability Tactics

|                  |                  |  |
|------------------|------------------|--|
| Locate           | Discover Service | <ul style="list-style-type: none"><li>알려진 디렉터리 서비스 검색을 통해 서비스를 찾는다.</li></ul>      |
| Manage Interface | Orchestrate      | <ul style="list-style-type: none"><li>특정 서비스들의 호출 순서를 조정하고, 관리하는 통제 메커니즘</li></ul> |
|                  | Tailor Interface | <ul style="list-style-type: none"><li>인터페이스 새로운 기능을 추가하거나 제거</li></ul>             |

## 6.3

### ❖ A Design Checklist for Interoperability

| 분류                                   | 체크리스트   |
|--------------------------------------|---|
| Allocation of Responsibilities       | <ul style="list-style-type: none"><li>✓ 다른 시스템과의 상호 운용을 위한 시스템의 책임을 결정</li><li>✓ 외부 시스템과 상호 운영을 위한 요구를 감지하기 위한 책임이 할당되었는가의 확인<ul style="list-style-type: none"><li>- 요청 수락, 정보 교환, 요청 거부, 적절한 통보, 신뢰할 수 없는 환경에서의 요청 기록</li></ul></li></ul>  |
| Coordination Model                   | <ul style="list-style-type: none"><li>✓ 중요 품질 속성 요구사항을 조정 모델이 만족하는지 보장<ul style="list-style-type: none"><li>- 네트워크 상의 트래픽 양, 시스템에 보내는 시간에 관계없는 메시지, 메시지를 보내는 비용, 메시지 도착시의 잡음 등</li></ul></li></ul>  |
| Data Model                           | <ul style="list-style-type: none"><li>✓ 주요 데이터 추상화의 구문과 의미 결정</li><li>✓ 사용 운영중인 시스템 사이의 데이터 일관성을 위한 주된 데이터 추상화의 확인</li></ul>  |
| Mapping among Architectural Elements | <ul style="list-style-type: none"><li>✓ 컴포넌트와 프로세서의 Mapping<ul style="list-style-type: none"><li>- 보안, 가용성, 그리고 성능 요구사항을 고려</li></ul></li></ul>   |
| Resource Management                  | <ul style="list-style-type: none"><li>✓ 다른 시스템과의 상호 운영을 위한 요구/거절 시 중요 시스템 자원이 고갈되지 않는지 여부를 확인</li><li>✓ 상호 운영의 가능한 통신 요구사항에 의해 내포된 자원의 사용 정도에 대한 확인</li><li>✓ 상호 운영 시 시스템들 간의 자원의 공유가 요구된다면, 적당한 분배 정책이 있는지 확인</li></ul>  |
| Binding Time                         | <ul style="list-style-type: none"><li>✓ 시스템이 상호 운영한다면, 언제 시스템들이 서로를 알게 되는지를 결정<ul style="list-style-type: none"><li>- 알려지거나 알려지지 않은 외부 시스템의 바인딩을 처리하기 위한 정책이 있는지 확인</li><li>- 승인되지 않은 바인딩을 거절하고, 기록하기 위한 메커니즘을 확인</li><li>- 늦은 바인딩 시점의 경우 관련된 새로운 서비스나 프로토콜을 찾기 위한 메커니즘을 확인</li></ul></li></ul> |
| Choice of Technology                 | <ul style="list-style-type: none"><li>✓ 시스템의 인터페이스가 “보여지는지”, 그들이 상호 운용에 영향을 주는지</li><li>✓ 선택된 기술들이 상호 운용성을 지원하도록 설계되었는지</li></ul>   |



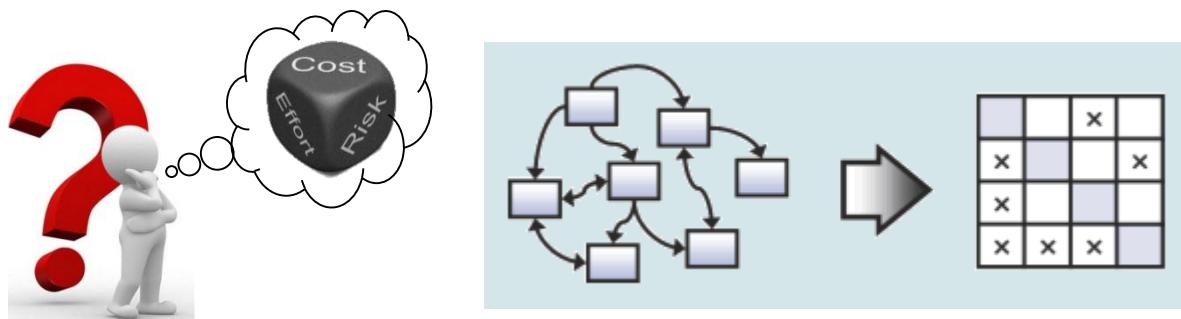
## **Chapter 7. Modifiability**



# In this Chapter ...

## ❖ 변경용이성(Modifiability)이란 ...

- 변화와 변화가 만들어내는 비용과 위험에 관심을 둔다.



- 변경용이성에 대한 계획을 위해서는 다음 사항에 관심을 둔다.
  - 무엇이 변경될 수 있는가?
  - 변화의 가능성 있는 것은 무엇인가?
  - 변화가 일어나는 시기와 누가 변화는 만드는가?
  - 변화의 비용은 얼마나 되는가?
    - 시스템에 적용할 새로운 메커니즘을 소개하는 비용
    - 새로운 메커니즘을 시스템에 적용하는 비용





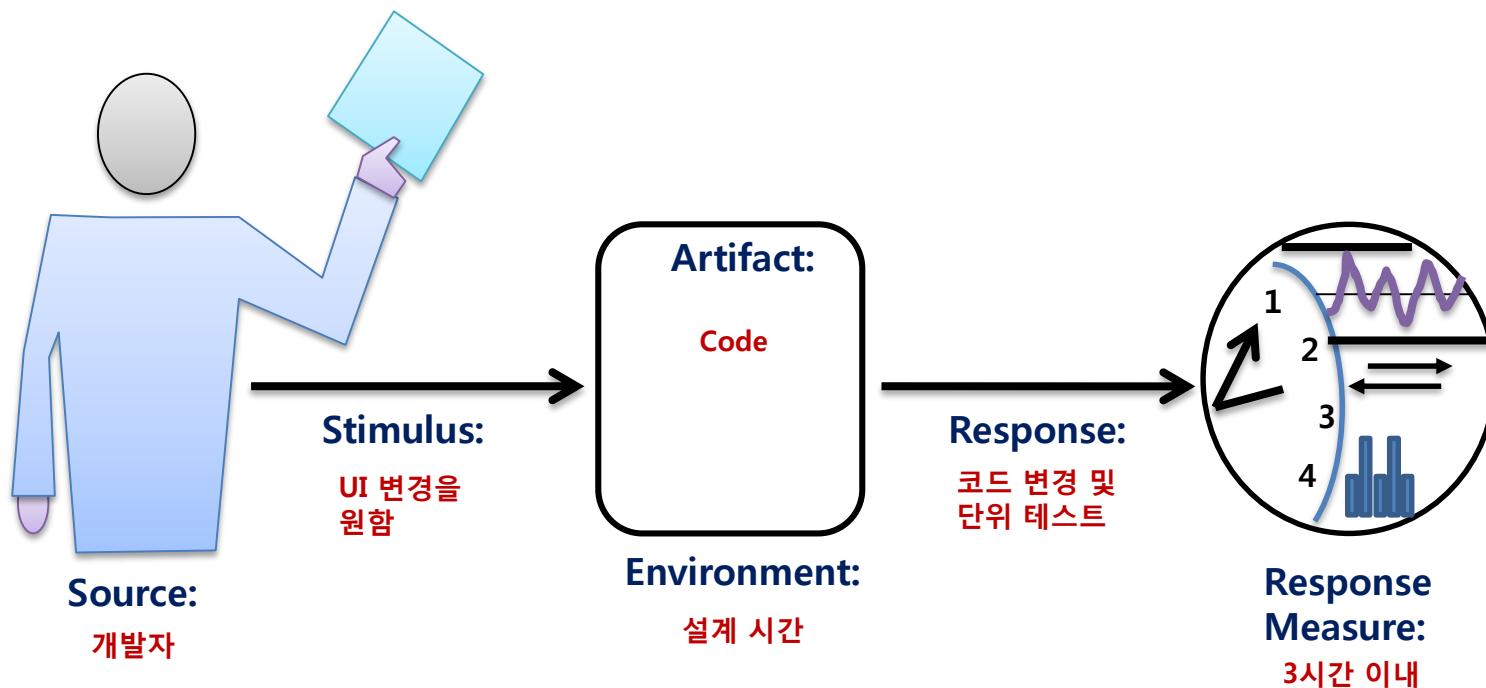
# 7.1

## ❖ Modifiability General Scenario

| 시나리오 항목          | 입력 가능한 값  |
|------------------|---|
| Source           | 최종 사용자, 개발자, 시스템 관리자  |
| Stimulus         | 기능의 추가/삭제/변경 지시, 또는 품질속성, 용량, 기술에 대한 변경   |
| Environment      | 실행 시점, 컴파일 시점, 빌드 시점, 초기화 시점, 설계 시간   |
| Artifact         | 코드, 데이터, 인터페이스, 컴포넌트, 자원, 환경 구성   |
| Response         | 다음의 하나 이상을 따른다. <ul style="list-style-type: none"><li>▪ 변경을 한다.</li><li>▪ 변경 결과를 테스트한다.</li><li>▪ 변경 결과를 배포한다.</li></ul>  |
| Response Measure | 다음 관점의 비용이 따른다. <ul style="list-style-type: none"><li>▪ 영향을 받는 요소의 개수, 크기, 복잡도</li><li>▪ 노력</li><li>▪ 일정</li><li>▪ 비용 (직접 비용 또는 간접 비용)</li><li>▪ 해당 변경이 확장되어 다른 기능이나 품질속성에 영향을 준다.</li><li>▪ 새로운 결함의 발견</li></ul> |

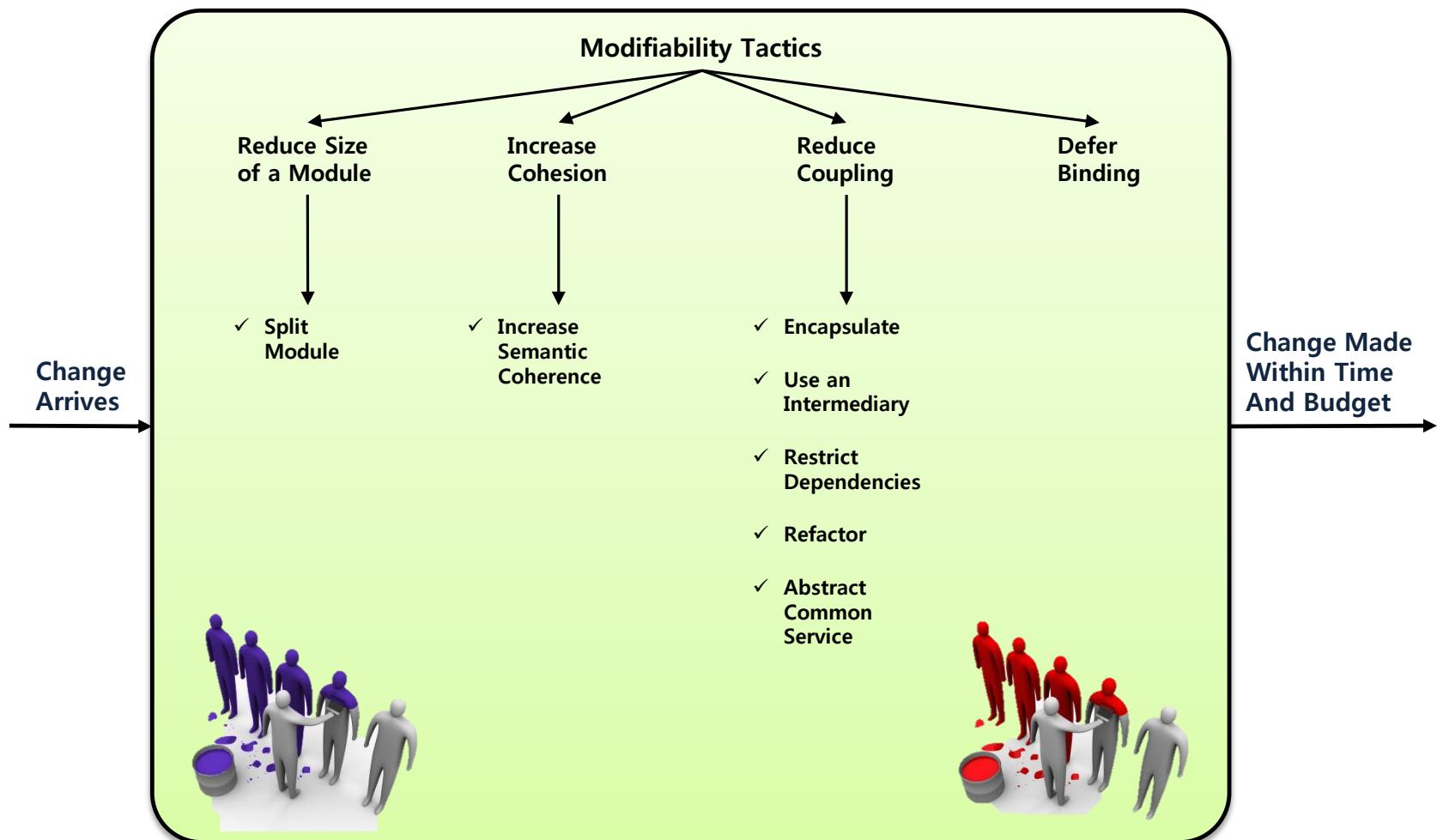
## ❖ Modifiability General Scenario

- Sample concrete modifiability scenario



# 7.2

## ❖ Tactics for Modifiability



## ❖ Modifiability Tactics

|                         |                             |   |
|-------------------------|-----------------------------|---|
| Reduce Size of a Module | Split Module                | <ul style="list-style-type: none"> <li>평균적인 변경 비용이 감소하도록 모듈을 여러 개의 작은 모듈로 나눈다.</li> </ul>                         |
| Increase Cohesion       | Increase Semantic Coherence | <ul style="list-style-type: none"> <li>하나의 모듈에 있는 다른 책임들이 동일 목적을 위해 제공되지 않아야 한다. (서로 다른 모듈에 위치시켜야 함)</li> </ul>   |
| Reduce Coupling         | Encapsulate                 | <ul style="list-style-type: none"> <li>모듈에 대한 명확한 인터페이스 정의</li> <li>한 모듈의 변경이 다른 모듈에 전파 될 확률을 줄인다.</li> </ul>     |
|                         | Use an Intermediary         | <ul style="list-style-type: none"> <li>한 모듈이 다른 모듈에 의존한다면, 의존성과 관련된 행위를 제어하는 중개자를 중간에 삽입</li> </ul>               |
|                         | Restrict Dependencies       | <ul style="list-style-type: none"> <li>주어진 모듈과 상호작용하거나 종속성을 갖는 모듈을 제한 (모듈에 대한 가시성을 제한)</li> </ul>                 |
|                         | Refactor                    | <ul style="list-style-type: none"> <li>두 모듈이 서로 종복되어 있어 동일한 변경에 (잠재적) 영향을 받는 경우</li> </ul>                        |
|                         | Abstract Common Services    | <ul style="list-style-type: none"> <li>두 모듈이 상당히 동일하지는 않지만 유사한 서비스를 제공하는 경우, 서비스를 일반화된 형식으로 구현 (비용 효과)</li> </ul> |
| Defer Binding           |                             | <ul style="list-style-type: none"> <li>배치 시점과 비 개발자들에 의한 변경을 허용</li> <li>추가적인 내부 구조가 필요</li> </ul>                |



## 7.3

### ❖ A Design Checklist for Interoperability

| 분류                                   | 체크리스트  |
|--------------------------------------|--|
| Allocation of Responsibilities       | <ul style="list-style-type: none"><li>✓ 기술적, 법적, 사회적, 업무적, 고객에 의한 변화의 고려를 통해 일어날 만한 변화와 변화의 분류 결정<ul style="list-style-type: none"><li>- 변경을 위해 추가, 변경, 삭제가 필요한 책임 결정, 변화에 영향을 받는 책임 결정</li><li>- 모듈에 대한 책임의 결정 (동일 모듈에서 동시에 변경 / 다른 모듈에서 다른 시점에 변경)</li></ul></li></ul> |
| Coordination Model                   | <ul style="list-style-type: none"><li>✓ 어떤 기능과 품질 속성이 실행 시 변경 되는지 여부와 그것이 조정 모델에 얼마나 영향을 주는지 결정</li><li>✓ 변화에 대한 조정을 위해 어떤 디바이스, 프로토콜, 통신 경로가 사용되는지 결정</li></ul>   |
| Data Model                           | <ul style="list-style-type: none"><li>✓ 데이터의 추상화, 동작, 특성에 어떤 변경이 있는지 결정 (생성, 초기화, 지속, 처리, 변환, 파기 포함)</li><li>✓ 최종 사용자, 시스템 관리자, 또는 개발자에 의해 변경이 일어날지 여부 결정</li></ul>  |
| Mapping among Architectural Elements | <ul style="list-style-type: none"><li>✓ 실행, 컴파일, 설계 또는 빌드 시의 기능과 계산요소의 맵핑이 바람직한 변경방식인지를 결정</li><li>✓ 기능과 품질 속성의 추가, 삭제, 변경을 수용하기 위해 필요한 변경의 범위를 결정<ul style="list-style-type: none"><li>- 실행 종속성 / 데이터 베이스에 데이터 할당 / 프로세스, 쓰레드, 프로세서 실행 요소 할당</li></ul></li></ul>        |
| Resource Management                  | <ul style="list-style-type: none"><li>✓ 자원 사용에 영향을 주는 책임이나 품질 속성을 어떻게 추가, 삭제, 변경할 것인지를 결정</li><li>✓ 변경 후의 자원이 시스템 요구 사항을 만족하는지 확인</li><li>✓ 모든 자원 관리자를 캡슐화하고, 이러한 자원 관리자에 의해 구현된 정책들이 캡슐화되고 바인딩이 가능한 범위까지 지연되는지를 확인</li></ul>  |
| Binding Time                         | <ul style="list-style-type: none"><li>✓ 변경이 필요할 최종 시간을 결정</li><li>✓ 선택된 시간에 적당한 기능을 제공하기 위한 지역 바인딩 메커니즘을 선택</li><li>✓ 메커니즘 소개 비용과 선택된 메커니즘을 사용하여 만들어지는 변경을 결정</li><li>✓ 변화가 내포하는 종속성 사이의 선택은 복잡하고 알려져 있지 않기 때문에 너무 많은 바인딩 선택 사항을 갖지 말 것</li></ul>                        |
| Choice of Technology                 | <ul style="list-style-type: none"><li>✓ 선택된 기술에 의하여 어떠한 변경이 더 쉬워지는지 또는 어려워지는지 결정한다.</li><li>✓ 가장 있음직한 변경을 지원하기 기술의 선택</li></ul>  |



# Appendix : Coupling and Cohesion

## ❖ Coupling

- External interaction of the module with other modules

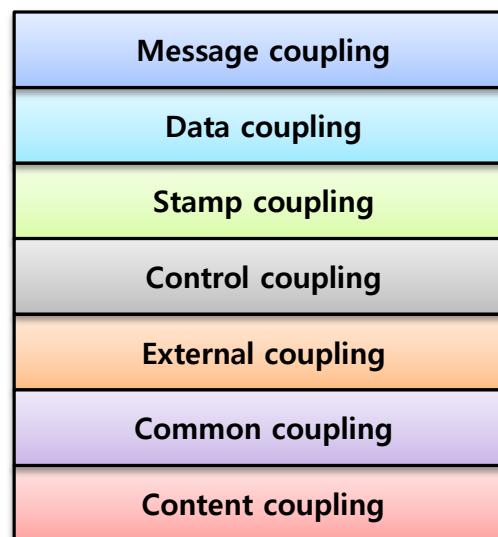
## ❖ Cohesion

- Internal interaction of the module (Crisp abstraction of purpose)

## ❖ For component independence

- High cohesion, Low coupling

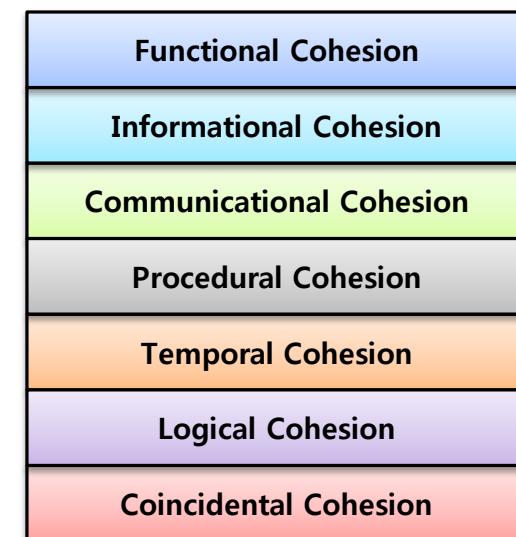
- ✓ Less interdependency
- ✓ Less coordination
- ✓ Less information flow



Good



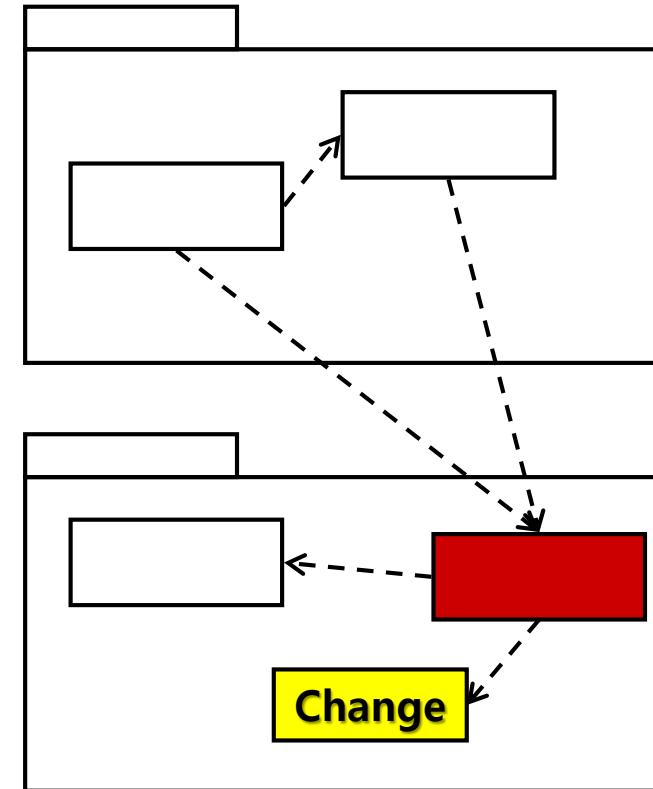
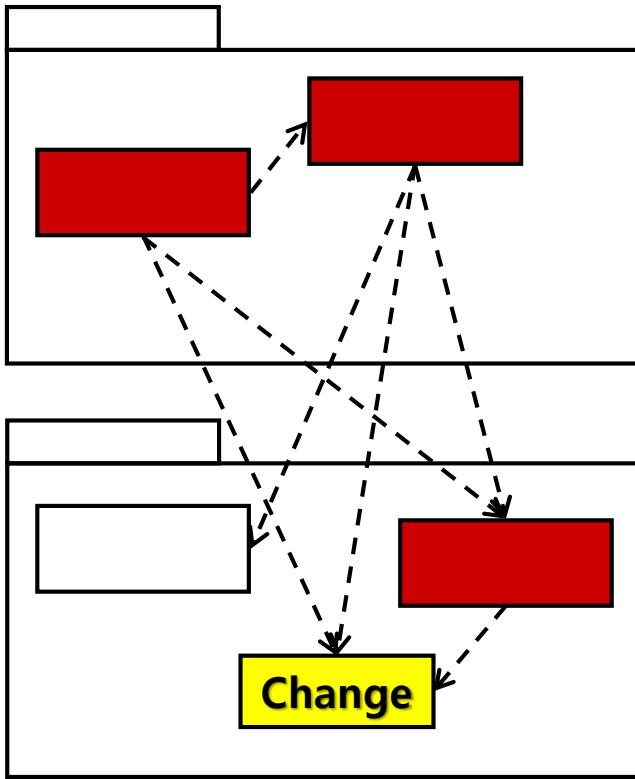
Bad



- ✓ More interdependency
- ✓ More coordination
- ✓ More information flow



# Appendix : Coupling Example

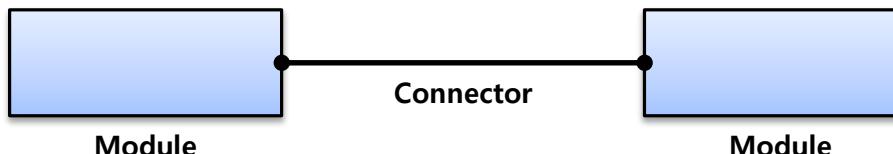


High coupling makes modifying parts of the system difficult,  
e.g., modifying a component affects all the components to which the component is connected



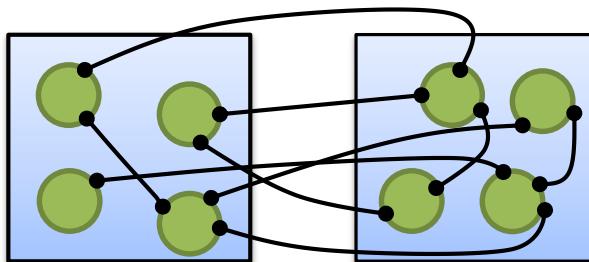
# Appendix : Coupling and Cohesion

## ❖ Architectural Building Blocks

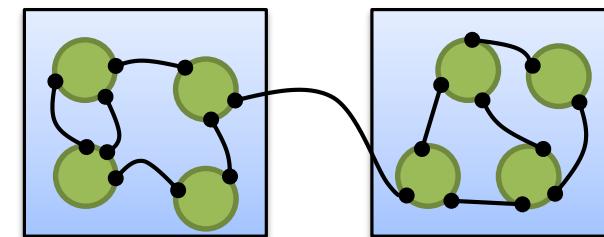


## ❖ A good architecture

- Minimizes coupling between modules
  - Goal : modules don't need to know much about one another to interact
  - Low coupling makes future change easier
- Maximizes the cohesion of each module
  - Goal : the contents of each module are strongly inter-related
  - High cohesion makes a module easier to understand



**Bad Case**



**Good Case**



# Appendix : Type of Cohesion

| Cohesion Type                        | Description  |
|--------------------------------------|--|
| <b>Functional cohesion (best)</b>    | <ul style="list-style-type: none"><li>Functional cohesion is when parts of a module are grouped because they all contribute to a single well-defined task of the module (e.g. <a href="#">tokenizing</a> a string of XML)</li></ul>  |
| <b>Sequential cohesion</b>           | <ul style="list-style-type: none"><li>Sequential cohesion is when parts of a module are grouped because the output from one part is the input to another part like an assembly line (e.g. a function which reads data from a file and processes the data)</li></ul>  |
| <b>Communicational cohesion</b>      | <ul style="list-style-type: none"><li>Communicational cohesion is when parts of a module are grouped because they operate on the same data (e.g. a module which operates on the same record of information)</li></ul>  |
| <b>Procedural cohesion</b>           | <ul style="list-style-type: none"><li>Procedural cohesion is when parts of a module are grouped because they always follow a certain sequence of execution (e.g. a function which checks file permissions and then opens the file).</li></ul>  |
| <b>Temporal cohesion</b>             | <ul style="list-style-type: none"><li>Temporal cohesion is when parts of a module are grouped by when they are processed - the parts are processed at a particular time in program execution (e.g. a function which is called after catching an exception which closes open files, creates an error log, and notifies the user).</li></ul> |
| <b>Logical cohesion</b>              | <ul style="list-style-type: none"><li>Logical cohesion is when parts of a module are grouped because they logically are categorized to do the same thing, even if they are different by nature (e.g. grouping all mouse and keyboard input handling routines).</li></ul>   |
| <b>Coincidental cohesion (worst)</b> | <ul style="list-style-type: none"><li>Coincidental cohesion is when parts of a module are grouped arbitrarily; the only relationship between the parts is that they have been grouped together (e.g. a "Utilities" class).</li></ul>   |

Refer to : [http://en.wikipedia.org/wiki/Cohesion\\_\(computer\\_science\)](http://en.wikipedia.org/wiki/Cohesion_(computer_science))



# Appendix : Type of Coupling

| Coupling Type  | Description  |
|--|--|
| <b>No coupling</b>                                   | <ul style="list-style-type: none"><li>▪ Modules do not communicate at all with one another.</li></ul>  |
| <b>Message coupling (low)</b>                        | <ul style="list-style-type: none"><li>▪ This is the loosest type of coupling. It can be achieved by state decentralization (as in objects) and component communication is done via parameters or message passing</li></ul>   |
| <b>Data coupling</b>                                 | <ul style="list-style-type: none"><li>▪ Data coupling is when modules share data through, for example, parameters. Each datum is an elementary piece, and these are the only data shared (e.g., passing an integer to a function that computes a square root).</li></ul>   |
| <b>Stamp coupling<br/>(Data-structured coupling)</b> | <ul style="list-style-type: none"><li>▪ Stamp coupling is when modules share a composite data structure and use only a part of it, possibly a different part (e.g., passing a whole record to a function that only needs one field of it).</li></ul>   |
| <b>Control coupling</b>                              | <ul style="list-style-type: none"><li>▪ Control coupling is one module controlling the flow of another, by passing it information on what to do (e.g., passing a what-to-do flag).</li></ul>   |
| <b>External coupling</b>                             | <ul style="list-style-type: none"><li>▪ External coupling occurs when two modules share an externally imposed data format, communication protocol, or device interface. This is basically related to the communication to external tools and devices.</li></ul>  |
| <b>Common coupling</b>                               | <ul style="list-style-type: none"><li>▪ Common coupling (also known as <b>Global coupling</b>) is when two modules share the same global data (e.g., a global variable). Changing the shared resource implies changing all the modules using it.</li></ul>   |
| <b>Content coupling (high)</b>                       | <ul style="list-style-type: none"><li>▪ Content coupling (also known as <b>Pathological coupling</b>) is when one module modifies or relies on the internal workings of another module (e.g., accessing local data of another module). Therefore changing the way the second module produces data (location, type, timing) will lead to changing the dependent module.</li></ul> |

Refer to : [http://en.wikipedia.org/wiki/Coupling\\_\(computer\\_programming\)](http://en.wikipedia.org/wiki/Coupling_(computer_programming))



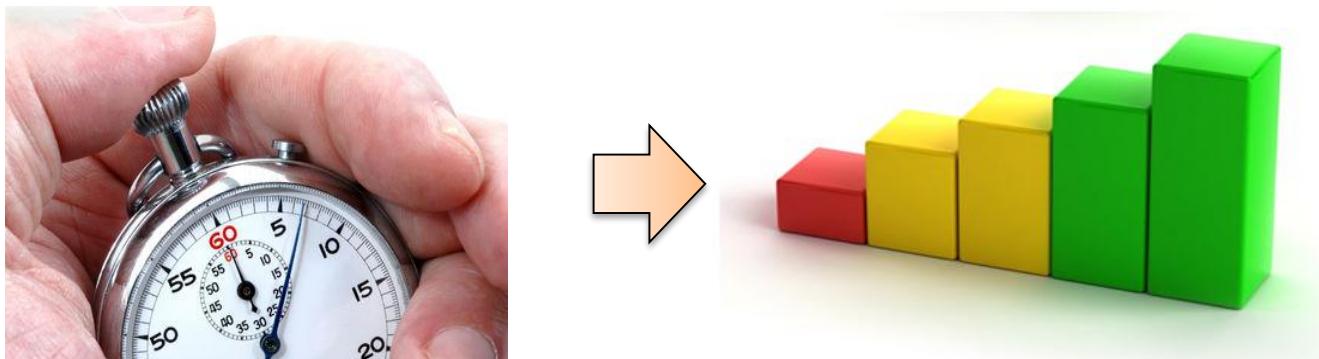
## **Chapter 8. Performance**



# In this Chapter ...

## ❖ 성능(Performance)이란 ...

- 시간과 시간적인 요구사항을 만족시키는 S/W 시스템의 능력이다.
  - 시스템이나 시스템의 일부 요소에서 이벤트가 발생하면, 적절한 시간 내에 응답이 되어야 한다.
  - 이벤트 패턴과 응답 패턴은 특성화 할 수 있다.



- 모든 시스템은 성능 요구사항을 갖는다.
- 성능을 때때로 확장성(Scalability)와 관련을 갖는다.

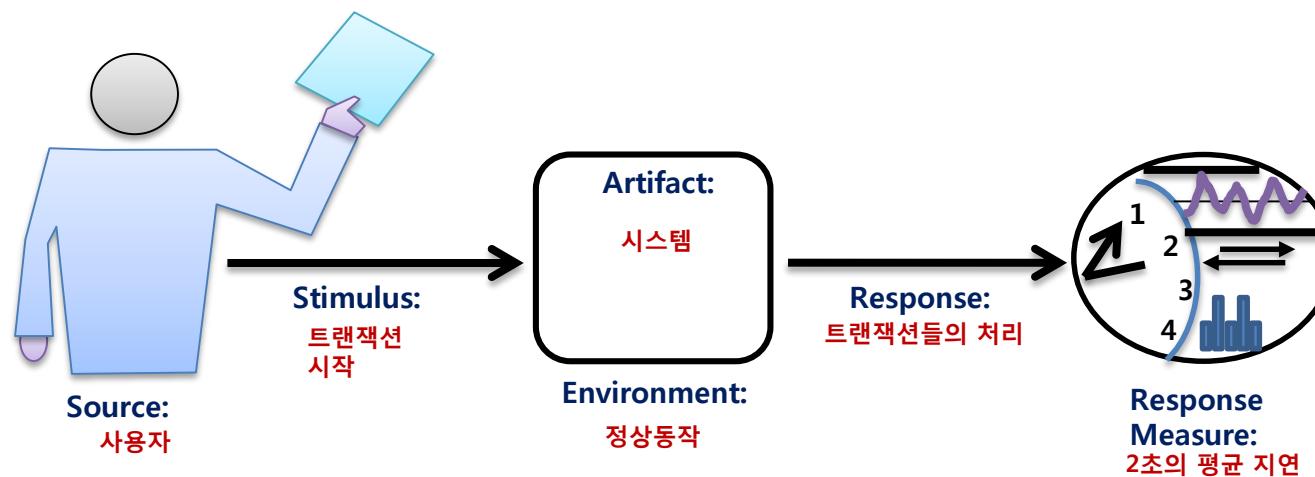
# 8.1

## ❖ Performance General Scenario

| 시나리오 항목          | 입력 가능한 값                      |
|------------------|-------------------------------|
| Source           | 시스템의 내부 또는 외부                 |
| Stimulus         | 이벤트의 주기적, 산발적, 또는 확률적인 도착     |
| Environment      | 운영모드 : 정상, 응급, 최대 부하, 과부화     |
| Artifact         | 시스템 또는 시스템 내의 하나 이상의 컴포넌트     |
| Response         | 프로세스 이벤트, 서비스 레벨의 변화          |
| Response Measure | 대기시간, 마감시간, 처리량, 지연시간, 데이터 손실 |

## ❖ Performance General Scenario

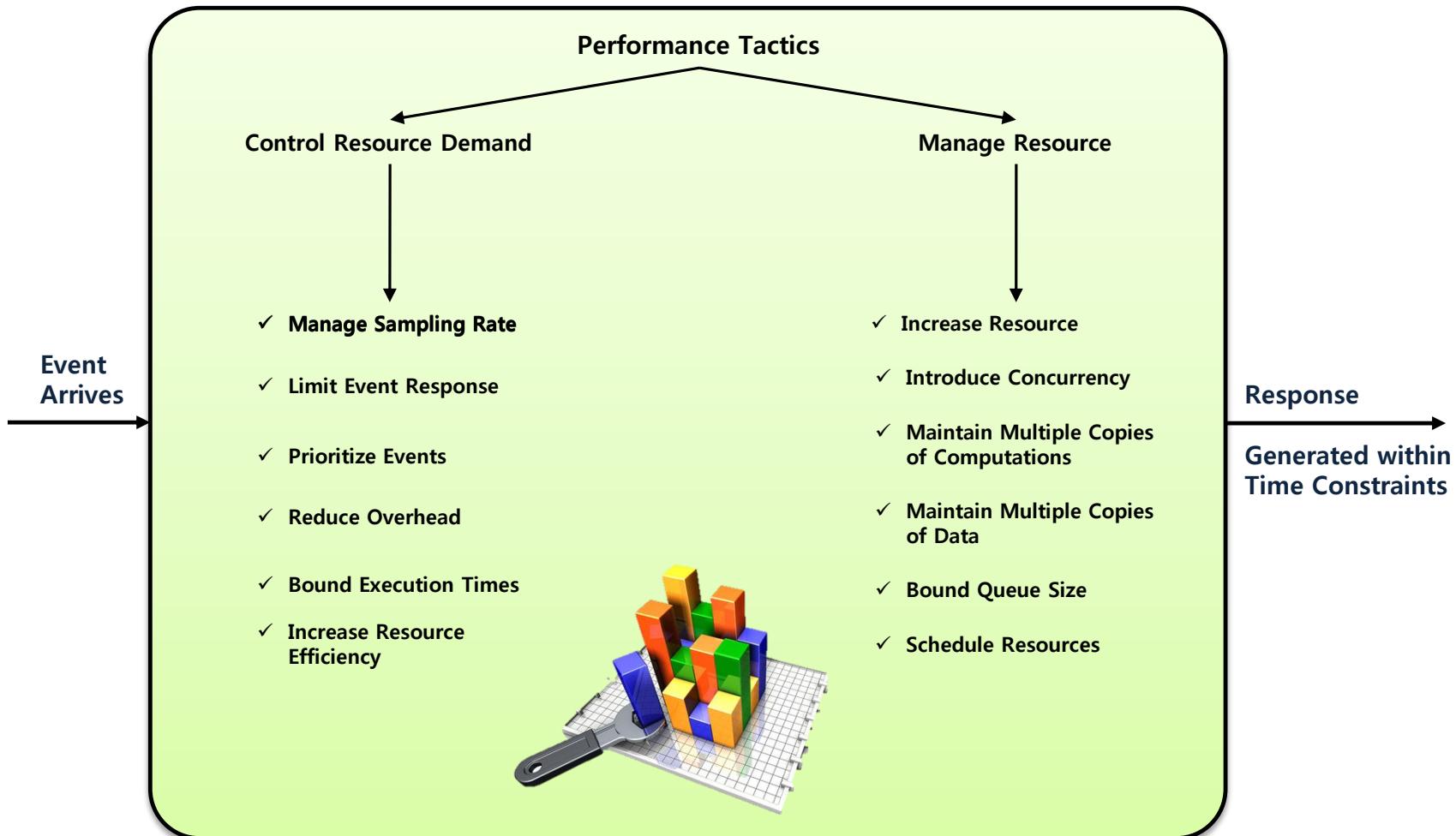
- Sample concrete performance scenario





## 8.2

### ❖ Tactics for Performance





## 8.2

### ❖ Performance Tactics

|                         |  |  |
|-------------------------|--|--|
| Control Resource Demand | Manage Sampling Rate                     | ▪ 외부에서 만들어지는 이벤트 도착에 대한 제어   |
|                         | Limit Event Response                     | ▪ 이산적인 이벤트들의 도착이 처리하기에는 너무 빠른 경우 이벤트가 처리될 때까지 큐에 저장                              |
|                         | Prioritize Events                        | ▪ 이벤트에 우선순위화를 통하여 중요한 이벤트 순서에 따라 처리  |
|                         | Reduce Overhead                          | ▪ 중개자의 사용은 이벤트 처리에 자원을 더 요구하지만 지연을 제거함으로 대기시간을 줄인다.                              |
|                         | Bound Execution Times                    | ▪ 이벤트에 응답하는데 사용되는 실행 시간을 제한<br>▪ 정확한 계산을 경우보다 비용이 적다.                            |
|                         | Increase Resource Efficiency             | ▪ 중요 지점에 대한 알고리즘의 향상을 통하여 대기시간을 단축   |
| Manage Resources        | Increase Resources                       | ▪ 더 많은 자원의 추가를 통한 대기 시간의 감소 (비용 고려)  |
|                         | Introduce Concurrency                    | ▪ 요청에 대한 병렬 처리는 자원 대기 시간을 줄임   |
|                         | Maintain Multiple Copies of Computations | ▪ 동일한 복사본을 여러 곳에 분산하여 경쟁을 줄임<br>▪ Load balancer → Round-robin, Least busy server |
|                         | Maintain Multiple Copies of Data         | ▪ 접근 속도가 다른 저장 장치 상의 데이터 복사본을 유지   |
|                         | Bound Queue Sizes                        | ▪ 이벤트 도착을 처리하는데 사용하는 자원과 대기중인 도착 이벤트의 최대치를 제어                                    |
|                         | Schedule Resources                       | ▪ 항상 자원에 대한 경쟁이 있다면, 자원 사용에 대해서 스케줄링이 되어야 한다.                                    |



## 8.3

### ❖ A Design Checklist for Performance

| 분류                                   | 체크리스트  |
|--------------------------------------|--|
| Allocation of Responsibilities       | <ul style="list-style-type: none"><li>✓ 많은 자원, 시간에 민감한 응답 요구사항을 포함한 시스템의 책임과 자원을 많이 사용하거나 시간에 민감한 이벤트가 발생하는 영향을 받는 시스템의 부분을 결정</li><li>✓ 각각의 책임에 대한 처리 요구사항을 확인하고, 병목현상의 원인이 되는지 결정</li><li>✓ 정의된 책임과 자원에 대하여 요구되는 성능 응답을 만족시키는지 확인</li></ul>                                    |
| Coordination Model                   | <ul style="list-style-type: none"><li>✓ (직접 또는 간접적으로) 서로 조정되어야 하는 시스템의 요소 결정</li><li>✓ 커뮤니케이션과 조정 메커니즘 선택<ul style="list-style-type: none"><li>- 동시성, 이벤트 우선순위, 스케줄링 전략 / 요구된 성능 응답의 제공 보장</li><li>- 주기적, 확률적, 산발적인 이벤트 도착의 포착 / 적절한 통신 메커니즘의 특성을 가지고 있는지 여부</li></ul></li></ul> |
| Data Model                           | <ul style="list-style-type: none"><li>✓ 많은 자원, 시간에 민감한 응답 요구사항을 포함한 시스템의 책임과 자원을 많이 사용하거나 시간에 민감한 이벤트가 발생하는 영향을 받는 데이터 모델의 부분을 결정<ul style="list-style-type: none"><li>- 핵심 데이터의 다중 복사본 유지 / 성능을 위한 데이터 분리 / 데이터 처리 요구사항 축소 / 자원 추가</li></ul></li></ul>                          |
| Mapping among Architectural Elements | <ul style="list-style-type: none"><li>✓ 네트워크 상에 부하가 일어날 수 있는 부분과 성능을 위해 일부 컴포넌트를 동시에 어디에 위치 시킬지 결정</li><li>✓ 많은 계산 요구사항을 가진 컴포넌트를 가장 처리 능력이 많은 프로세스에 할당했는지 확인</li><li>✓ 동시성을 어디에 적용할지 여부와 성능에 긍정적인지를 확인</li><li>✓ 제어 쓰레드의 선택 여부와 병목에 유발하는 책임들의 연관 관계를 결정</li></ul>               |
| Resource Management                  | <ul style="list-style-type: none"><li>✓ 시스템의 어떤 자원이 성능에 심각한 영향을 주는지 결정<ul style="list-style-type: none"><li>- 시간과 성능에 민감한 자원을 관리하기 위한 시스템 요소, 프로세스/쓰레드 모델, 자원과 자원의 접근에 대한 우선 순위화, 스케줄링과 락킹 전략, 증가되는 부하를 만족시키기 위한 자원의 추가 배분</li></ul></li></ul>                                   |
| Binding Time                         | <ul style="list-style-type: none"><li>✓ 완전한 바인딩에 필요한 시간</li><li>✓ 늦은 바인딩 메커니즘에 의한 추가적인 오버헤드</li><li>✓ 해당 값들이 시스템에 허용되지 않는 성능 저하를 일으키지 않는지를 확인</li></ul>  |
| Choice of Technology                 | <ul style="list-style-type: none"><li>✓ 선택한 기술이 hard/real-time deadline을 만족할 수 있는가?<ul style="list-style-type: none"><li>- 스케줄링 정책, 우선 순위, 요구 감소를 위한 정책, 프로세서에 대한 기술 분야 할당, 다른 성능 관련 파라미터</li></ul></li></ul>  |



## **Chapter 9. Security**



# In this Chapter ...

## ❖ 보안(Security)이란 ...

- 인증된 사람과 시스템에게는 접근을 허용하는 반면, 인증되지 않은 접근으로부터 데이터와 정보를 보호하는 시스템의 능력이다.
- 보안의 3가지 특징(CIA)

|                           |  |
|---------------------------|--|
| 비밀보장<br>(Confidentiality) | <ul style="list-style-type: none"><li>• 인증되지 않은 접근으로부터 데이터 또는 서비스가 보호되어야 한다.</li></ul>     |
| 무결성<br>(Integrity)        | <ul style="list-style-type: none"><li>• 인증되지 않은 처리로부터 데이터 또는 서비스가 영향을 받지 않아야 한다.</li></ul> |
| 가용성<br>(Availability)     | <ul style="list-style-type: none"><li>• 시스템은 정당한 사용이 가능해야 한다.</li></ul>                    |



- CIA를 지원하기 위한 다른 특성들

|                          |  |
|--------------------------|--|
| 인증<br>(Authentication)   | <ul style="list-style-type: none"><li>• 처리를 위한 참가자들의 신분을 검증하고, 그들이 정당한 사용자인지를 확인한다.</li></ul>            |
| 부인방지<br>(Nonrepudiation) | <ul style="list-style-type: none"><li>• 송신측이 메시지를 보낸 사실과 수신측이 메시지를 수신한 사실을 나중에 부인하지 못하도록 보장한다.</li></ul> |
| 권한부여<br>(Authorization)  | <ul style="list-style-type: none"><li>• 사용자가 테스트를 수행할 수 있는 권한을 부여한다.</li></ul>                           |

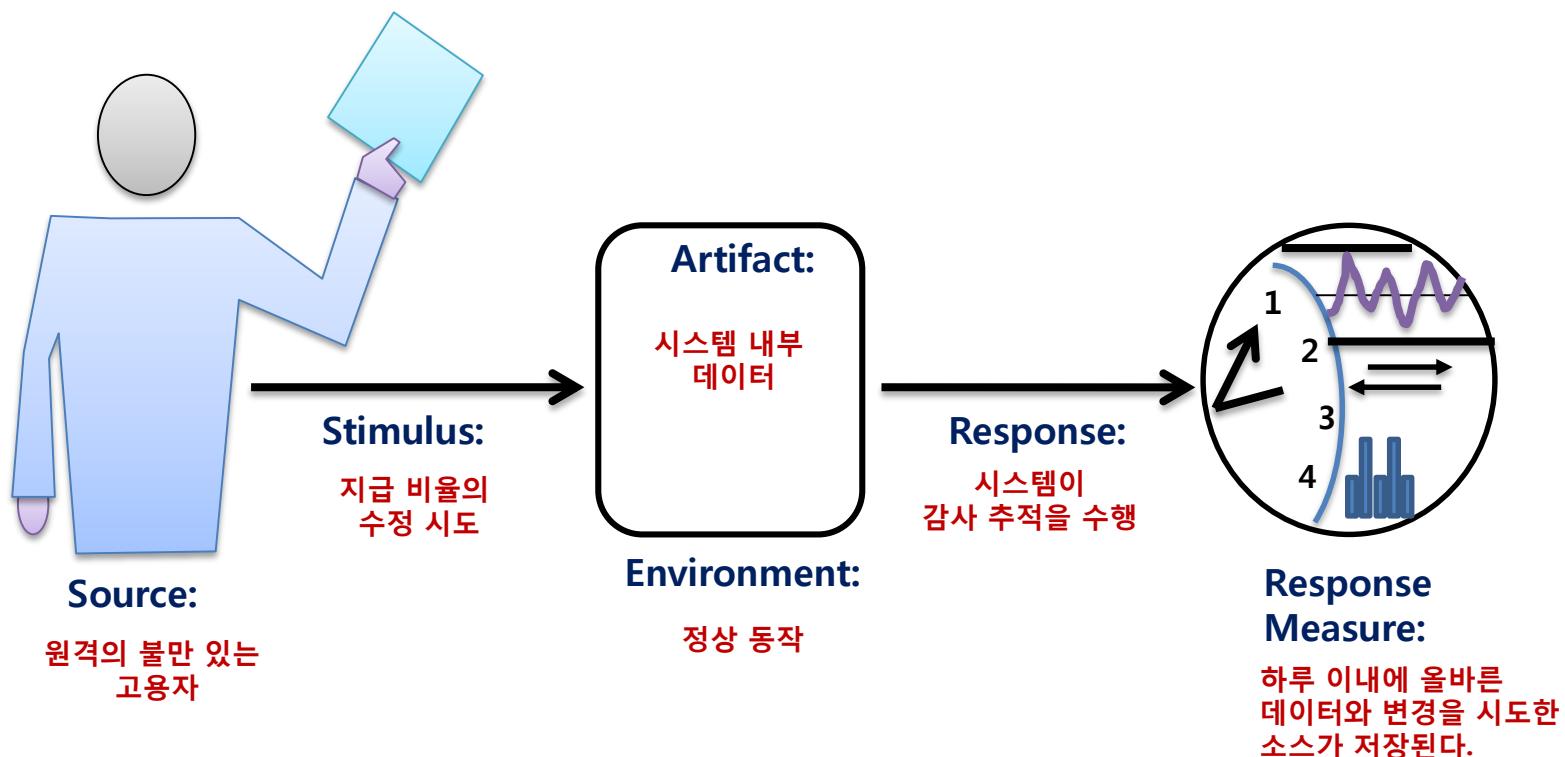


## ❖ Security General Scenario

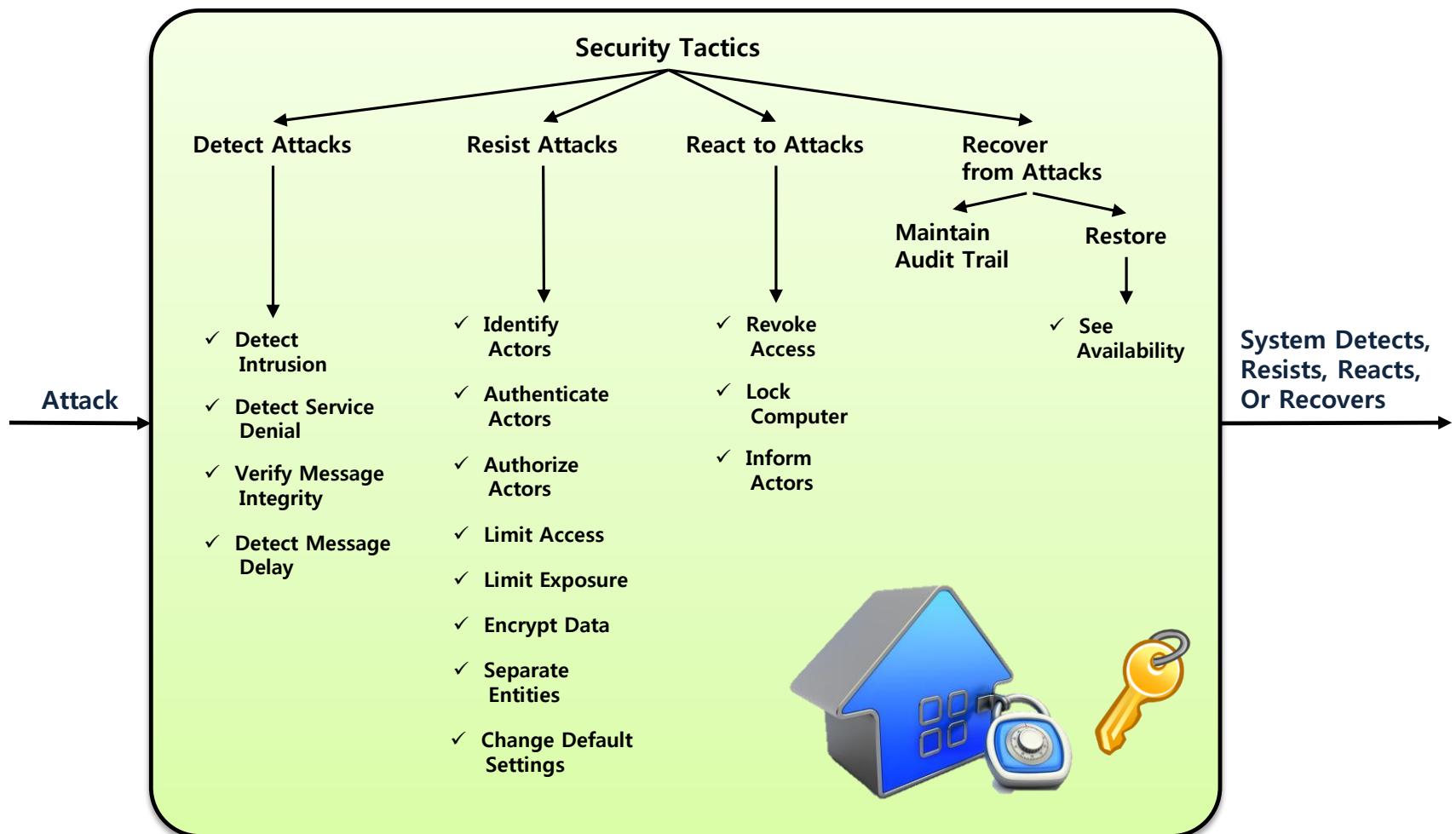
| 시나리오 항목          | 입력 가능한 값   |
|------------------|--|
| Source           | 사람 또는 미리 정의된/알려지지 않은 시스템으로부터의 공격<br>조직 내부 또는 외부로부터의 공격   |
| Stimulus         | 데이터를 표시, 변경 또는 삭제하려는 인증되지 않은 시도, 시스템 서비스로의 접근,<br>시스템 동작의 변경, 또는 시스템 가용성의 축소하려는 시도   |
| Environment      | 온라인 또는 오프라인, 네트워크의 연결 또는 비연결, 방화벽 또는 공개,<br>완전 작동, 부분적 작동, 미작동   |
| Artifact         | 시스템 서비스, 시스템 내의 데이터, 시스템의 컴포넌트 또는 자원,<br>시스템에 의해 생산되거나 소비된 자원  |
| Response         | 트랜잭션은 다음과 같은 방식으로 수행된다. <ul style="list-style-type: none"><li>▪ 인증되지 않은 접근으로부터 데이터 또는 서비스가 보호된다.</li><li>▪ 데이터나 서비스가 인증 없이는 처리되지 않는다.</li><li>▪ 트랜잭션의 일부가 보장된다.</li><li>▪ 트랜잭션의 일부가 그들의 포함을 거부할 수 없다.</li><li>▪ 데이터, 자원 그리고 시스템 서비스를 합법적으로 사용할 수 있다.</li></ul> 시스템은 다음에 의하여 활동을 추적 <ul style="list-style-type: none"><li>▪ 접근과 변경을 기록</li><li>▪ 데이터, 자원, 그리고 서비스에 대한 접근 시도를 기록</li><li>▪ 공격이 감지된 경우 적절한 엔티티(사람 또는 시스템)에 통보</li></ul> |
| Response Measure | 다음의 하나 이상을 따른다. <ul style="list-style-type: none"><li>▪ 특정 컴포넌트나 데이터가 손상되었을 때 시스템이 얼마나 손상 정도</li><li>▪ 공격을 감지하기 까지 걸린 시간</li><li>▪ 얼마나 많은 공격을 견디었는지 여부</li><li>▪ 성공적인 공격으로 회복하는데 걸린 시간</li><li>▪ 특정 공격에 대한 데이터의 취약성 정도</li></ul>   |

## ❖ Security General Scenario

- Sample concrete security scenario



## ❖ Tactics for Security





## 9.2

### ❖ Security Tactics

|                |                          |  |
|----------------|--------------------------|--|
| Detect Attacks | Detect Intrusion         | <ul style="list-style-type: none"><li>▪ DB에 저장된 알려진 악의적인 행동 패턴과 네트워크 트래픽 또는 서비스 요청 패턴과 비교</li></ul>    |
|                | Detect Service Denial    | <ul style="list-style-type: none"><li>▪ 알려진 DoS 공격 패턴을 시스템에 들어오는 네트워크 트래픽 신호나 패턴과 비교</li></ul>         |
|                | Verify Message Integrity | <ul style="list-style-type: none"><li>▪ 메시지와 파일의 무결성을 검증하기 위해 checksum 또는 Hash 값을 사용</li></ul>         |
|                | Detect Message Delay     | <ul style="list-style-type: none"><li>▪ 악의적인 사람이 메시지를 가로채는 사람이 중간에 개입하는 잠재적인 공격을 감지</li></ul>          |
| Resist Attacks | Identify Actors          | <ul style="list-style-type: none"><li>▪ 시스템의 모든 외부 입력에 대한 소스를 확인</li></ul>                             |
|                | Authenticate Actors      | <ul style="list-style-type: none"><li>▪ 사용자나 원격 컴퓨터가 실제로 누구인지 보장</li></ul>                             |
|                | Authorize Actors         | <ul style="list-style-type: none"><li>▪ 인증된 사용자에게 데이터나 서비스의 접근과 변경에 대한 올바른 권한이 있는지 확인</li></ul>        |
|                | Limit Access             | <ul style="list-style-type: none"><li>▪ 메모리, 네트워크 접속 등에 대한 자원에 대한 접근을 포함한 컴퓨팅 자원에 대한 접근 제어</li></ul>   |
|                | Limit Exposure           | <ul style="list-style-type: none"><li>▪ 시스템의 외부의 공격을 최소화하는 전략의 노출을 제한</li></ul>                        |
|                | Encrypt Data             | <ul style="list-style-type: none"><li>▪ 인증되지 않은 접근으로부터의 데이터 보호</li></ul>                               |
|                | Separate Entities        | <ul style="list-style-type: none"><li>▪ 중요한 데이터와 중요하지 않은 데이터의 분리는 공격의 가능성을 줄인다. (물리적인 분리 포함)</li></ul> |
|                | Change Default Settings  | <ul style="list-style-type: none"><li>▪ 일반적이고 공통적으로 이용 가능한 시스템 설정을 통한 잠재적인 공격 방지</li></ul>             |

## ❖ Security Tactics

|                      |                      |   |
|----------------------|----------------------|---|
| React to Attacks     | Revoke Access        | <ul style="list-style-type: none"><li>▪ 시스템이나 시스템 관리자가 시스템이 공격을 받는다고 믿는다면, 정상적인 사용자나 사용에 대해서도 민감한 자원에 대한 접근을 심각하게 제한할 수 있다.</li></ul> |
|                      | Lock Computer        | <ul style="list-style-type: none"><li>▪ 반복된 로그인 실패는 잠재적인 공격 시도가 될 수 있다. 이런 경우 특정 컴퓨터로부터의 로그인을 제한할 수 있다.</li></ul>                     |
|                      | Inform Actors        | <ul style="list-style-type: none"><li>▪ 시스템에 대한 공격이 감지되었을 경우, 이에 대응하기 위한 담당자에게 통보가 필요</li></ul>                                       |
| Recover from Attacks | Maintain Audit Trail | <ul style="list-style-type: none"><li>▪ 시스템 내부 데이터에 적용된 각 트랜잭션을 식별 정보와 함께 복사해 놓는다 → 감사, 부인 방지, 시스템 복구 지원</li></ul>                    |
|                      | Restore              | <ul style="list-style-type: none"><li>▪ 가용성(Availability) 참조</li></ul>  |



## 9.3

### ❖ A Design Checklist for Security

| 분류                                   | 체크리스트  |
|--------------------------------------|--|
| Allocation of Responsibilities       | <ul style="list-style-type: none"><li>✓ 안전을 위해 필요한 시스템의 책임 결정<ul style="list-style-type: none"><li>- 액터 식별, 액터 인증, 액터 권한 부여, 데이터나 서비스에 접근 허용 또는 거부, 데이터나 서비스에 접근 또는 변경에 대한 기록, 데이터 암호화, 공격으로부터의 복구, Checksum과 Hash 값 검증</li></ul></li></ul>  |
| Coordination Model                   | <ul style="list-style-type: none"><li>✓ 다른 시스템이나 사람들과의 커뮤니케이션과 조정을 위한 메커니즘 결정<ul style="list-style-type: none"><li>- 액터와 시스템에 대한 인증 및 권한 부여, 전송을 위한 데이터 암호화 메커니즘 확인</li><li>- 예상치 못한 자원과 서비스의 요구를 감시 및 인식/접속과 해제를 제한하기 위한 메커니즘 확인</li></ul></li></ul>  |
| Data Model                           | <ul style="list-style-type: none"><li>✓ 여러 데이터 필드에 대한 민감도 결정<ul style="list-style-type: none"><li>- 데이터에 따른 민감도가 분리되었는지, 민감도에 따라 다른 접근 권한과 접속 전 접근 권한을 확인하는지 체크,</li><li>- 민감한 데이터 접근에 대한 기록과 기록이 적절히 보호되는지 확인, 데이터 변경 시 적절히 저장되는지 확인</li><li>- 데이터가 적절하게 암호화되고 암호화 키가 데이터와 분리되어 있는지 확인,</li></ul></li></ul> |
| Mapping among Architectural Elements | <ul style="list-style-type: none"><li>✓ 시스템 서비스와 자원에 접근이나 데이터를 읽고, 쓰고 수정 방법의 변경을 고려할 때 아키텍처 요소들에 대한 대안적인 맵핑을 결정</li><li>✓ 대안적인 맵핑이 데이터, 서비스, 자원 접근에 대한 기록과 예상치 못한 많은 자원 요청을 인지하는데 어떻게 영향을 줄 것인지를 결정</li></ul>  |
| Resource Management                  | <ul style="list-style-type: none"><li>✓ 요구되는 시스템 자원과 내/외부의 개인과 시스템의 인증 여부 감시를 결정</li><li>✓ 보안 관련 동작을 위해 필요한 자원을 결정, 감염된 요소가 다른 요소를 감염시키지 않는지 확인</li><li>✓ 민감한 데이터의 전달에 공유 자원이 사용되지 않는지를 보장</li></ul>   |
| Binding Time                         | <ul style="list-style-type: none"><li>✓ 늦은 컴포넌트의 바인딩의 인스턴스가 신뢰할 수 없는 경우를 결정<ul style="list-style-type: none"><li>- 해당 경우 각각의 컴포넌트는 검증되어야 한다.</li></ul></li></ul>   |
| Choice of Technology                 | <ul style="list-style-type: none"><li>✓ 사용자 인증, 데이터 접근 권한, 자원 보호, 데이터 암호화에 사용 가능한 기술 결정</li><li>✓ 선택된 기술이 보안 요구에 관련된 전술을 지원하는지 확인</li></ul>  |



## **Chapter 10. Testability**



# 10.1

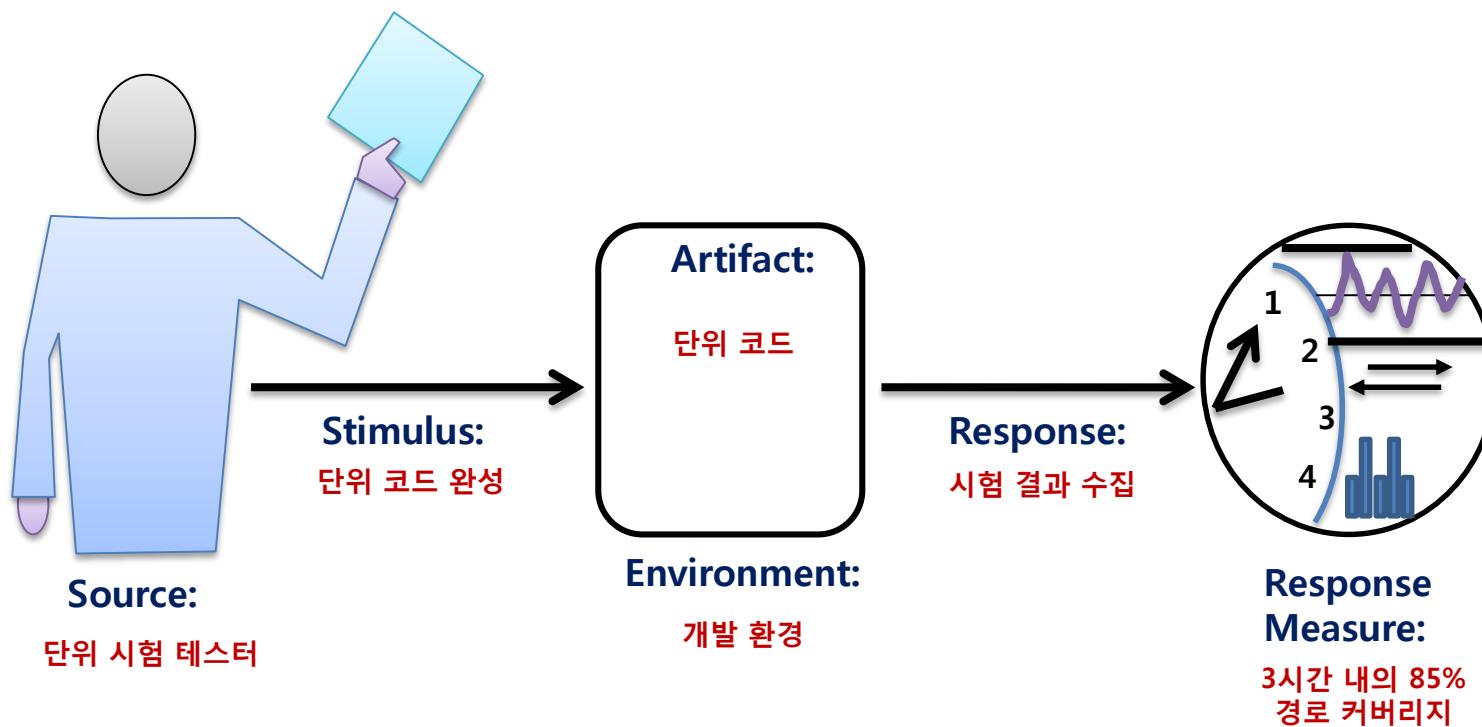
## ❖ Testability General Scenario

| 시나리오 항목          | 입력 가능한 값   |
|------------------|--|
| Source           | 단위 테스터, 통합 시험 테스터, 시스템 테스터, 고객 인증 테스터, 최종 사용자<br>수동 테스트 수행 또는 자동 테스팅 툴 수행  |
| Stimulus         | 클래스 계층이나 서비스와 같은 코딩 추가 부분의 구현 완료에 따른 테스트 셋의 수행,<br>서브시스템의 통합전체 시스템의 구현, 또는 고객에게 시스템 전달   |
| Environment      | 설계 시간, 개발 시간, 컴파일 시간, 통합 시간, 배포 시간, 수행 시간  |
| Artifact         | 테스트 중인 시스템의 일부   |
| Response         | 다음의 하나 이상을 따른다. <ul style="list-style-type: none"><li>▪ 테스트 셋의 수행과 결과의 확보</li><li>▪ 결함의 결과 행위 기록</li><li>▪ 시스템 상태의 통제와 감시</li></ul>  |
| Response Measure | 다음의 하나 이상을 따른다. <ul style="list-style-type: none"><li>▪ 결함과 결함의 분류를 구분하기 위한 노력</li><li>▪ 상태 공간 커버리지의 지정된 비율을 달성하기 위한 노력</li><li>▪ 다음 테스트에 의해 결함이 나타날 확률</li><li>▪ 테스트 수행 시간</li><li>▪ 결함을 감지하기 위한 노력</li><li>▪ 테스트 상의 가장 긴 종속성 체인의 길이</li><li>▪ 테스트 환경 준비 시간</li><li>▪ 노출된 위험의 감소(크기 * 확률)</li></ul> |

# 10.1

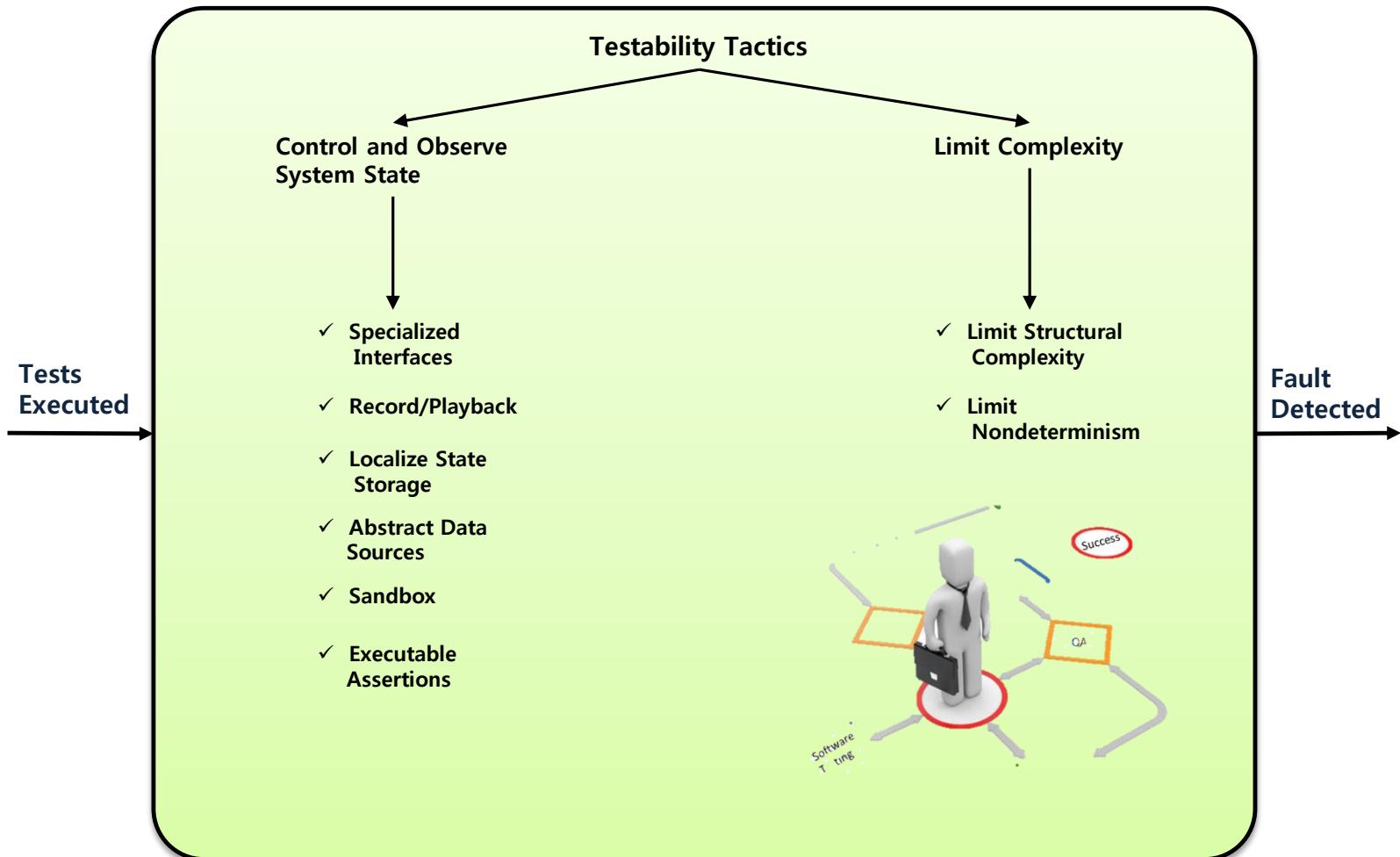
## ❖ Testability General Scenario

- Sample concrete testability scenario



# 10.2

## ❖ Tactics for Testability





## 10.2

### ❖ Testability Tactics

|                                  |                             |  |
|----------------------------------|-----------------------------|--|
| Control and Observe System State | Specialized Interfaces      | ▪ 특화된 시험 인터페이스는 개별적인 정상동작뿐 아니라 테스트 장비를 통한 변수 값을 기록/명세하게 한다.    |
|                                  | Record/Playback             | ▪ 인터페이스를 통과하는 정보를 모으고,<br>▪ 이를 통한 시험 입력 생성 및 시험 출력 저장          |
|                                  | Localize State Storage      | ▪ 현재 상태를 추적하고 보고하는 메커니즘으로 상태 저장을 외부화하는 State Machine 사용        |
|                                  | Abstract Data Sources       | ▪ 인터페이스 추상화를 통하여 테스트 데이터를 쉽게 변경 가능                             |
|                                  | Sandbox                     | ▪ 실험이 가능하도록 시스템의 인스턴스를 실세계로부터 격리                               |
|                                  | Executable Assertions       | ▪ 데이터 값이 지정된 제약을 만족하는지 확인                                      |
| Limit Complexity                 | Limit Structural Complexity | ▪ 순환 종속성을 회피 또는 해결, 외부 환경에 대한 종속성을 고립시키거나 캡슐화, 컴포넌트 사이의 종속성 감소 |
|                                  | Limit Nondeterminism        | ▪ 결정되지 않은 행동에 대한 모든 소스를 발견                                     |



## 10.3

### ❖ A Design Checklist for Testability

| 분류                                   | 체크리스트  |
|--------------------------------------|--|
| Allocation of Responsibilities       | <ul style="list-style-type: none"><li>✓ 어떤 시스템 책임이 가장 중요하고, 완전한 테스트가 필요한지 결정</li><li>✓ 테스트 실행과 결과 저장, 예상치 못한 행위에 대한 기록, 연관 시스템 상태의 통제 및 관찰을 위한 추가적인 시스템 책임들의 할당을 확인</li><li>✓ 높은 응집도, 낮은 결합도, 관심의 분리와 낮은 구조적 복잡성을 제공하는 기능의 할당을 보장</li></ul>                |
| Coordination Model                   | <ul style="list-style-type: none"><li>✓ 시스템의 조정 및 커뮤니케이션 메커니즘을 확인<ul style="list-style-type: none"><li>- 시스템이나 시스템 사이의 테스트의 수행과 결과 기록, 결함의 결과를 기록하는 액티비티 지원</li><li>- 시스템이나 시스템 사이의 테스팅을 위해 사용되는 통신 채널로 상태의 주입 및 감시 지원</li></ul></li></ul>                 |
| Data Model                           | <ul style="list-style-type: none"><li>✓ 시스템의 정상 동작을 보장하기 위해 반드시 테스트 되어야 하는 주요 데이터 추상화 결정<ul style="list-style-type: none"><li>- 추상화 데이터 인스턴스의 가능한 결과값, 값 설정 가능 여부 확인</li><li>- 추상화 데이터 인스턴스의 생성, 초기화, 저장, 처리, 변환, 파기가 수행 및 기록이 가능한지 확인</li></ul></li></ul> |
| Mapping among Architectural Elements | <ul style="list-style-type: none"><li>✓ 테스트를 위해 아키텍처 요소들을 어떻게 맵핑 할 것인지를 결정<ul style="list-style-type: none"><li>- 프로세스와 프로세서, 쓰레드와 프로세스, 모듈과 컴포넌트</li><li>- 원하는 테스트 결과를 받고, 잠재적인 경쟁 상태를 확인</li></ul></li></ul>   |
| Resource Management                  | <ul style="list-style-type: none"><li>✓ 테스트 실행과 결과의 저장을 위한 충분한 자원 이용이 가능한지 확인<ul style="list-style-type: none"><li>- 테스트 자원 한도, 이벤트 실패 분석을 위한 자원 할당, 테스트 목적의 새로운 자원 주입 한계, 테스팅 가상 자원</li></ul></li><li>✓ 테스트 환경이 실제 수행 환경을 대표하는지 확인</li></ul>              |
| Binding Time                         | <ul style="list-style-type: none"><li>✓ 컴파일 이후 바인드 된 컴포넌트가 늦은 바운드 컨텍스트로 테스트 되는지 확인</li><li>✓ 실패 시 해당 상황의 재현이 가능한지 확인</li><li>✓ 전체 범위의 바인딩 가능성에 대해서 테스트가 가능한지 확인</li></ul>  |
| Choice of Technology                 | <ul style="list-style-type: none"><li>✓ 아키텍처에 적용할 테스트 용이성 시나리오를 가능하게 하는데 이용 가능한 기술의 결정</li><li>✓ 선택한 기술이 얼마나 테스트 가능한지와 선택한 기법이 시스템에 적당한 테스트 수준을 제공하는지 확인</li></ul>   |



## **Chapter 11. Usability**



# 11.1

## ❖ Usability General Scenario

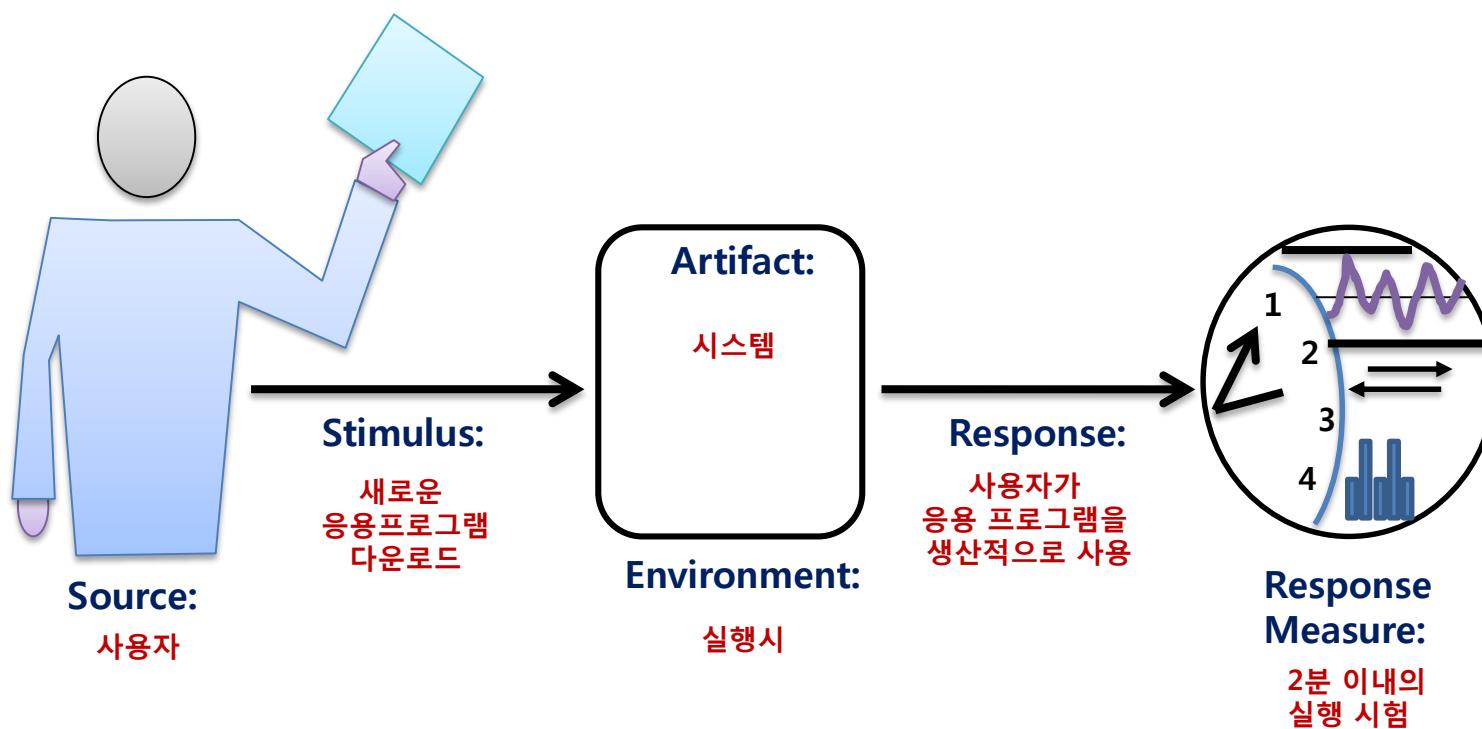
| 시나리오 항목          | 입력 가능한 값  |
|------------------|---|
| Source           | 최종 사용자, 가능한 전문 역할   |
| Stimulus         | 최종 사용자가 시스템을 효율적으로 사용하려고 시도 / 시스템 사용법을 배우려고 한다/ 에러의 영향을 최소화하려고 한다/ 시스템을 변경하려고 한다/ 시스템 환경을 구성하려고 한다  |
| Environment      | 실행시간 또는 환경 구성 시간  |
| Artifact         | 시스템 또는 사용자와 상호작용하는 시스템의 특정 부분   |
| Response         | 시스템은 사용자가 필요로 하는 기능을 제공하거나, 사용자의 요구를 예상할 수 있어야 한다.  |
| Response Measure | 다음의 하나 이상을 따른다. <ul style="list-style-type: none"><li>▪ 작업 시간, 에러의 수, 완료된 작업의 수</li><li>▪ 사용자 만족도, 사용자 지식의 습득</li><li>▪ 전제 동작 중 성공한 동작의 비율</li><li>▪ 총 에러 발생 시간 또는 에러가 발생한 경우 분실된 데이터</li></ul> |



## 11.1

### ❖ Usability General Scenario

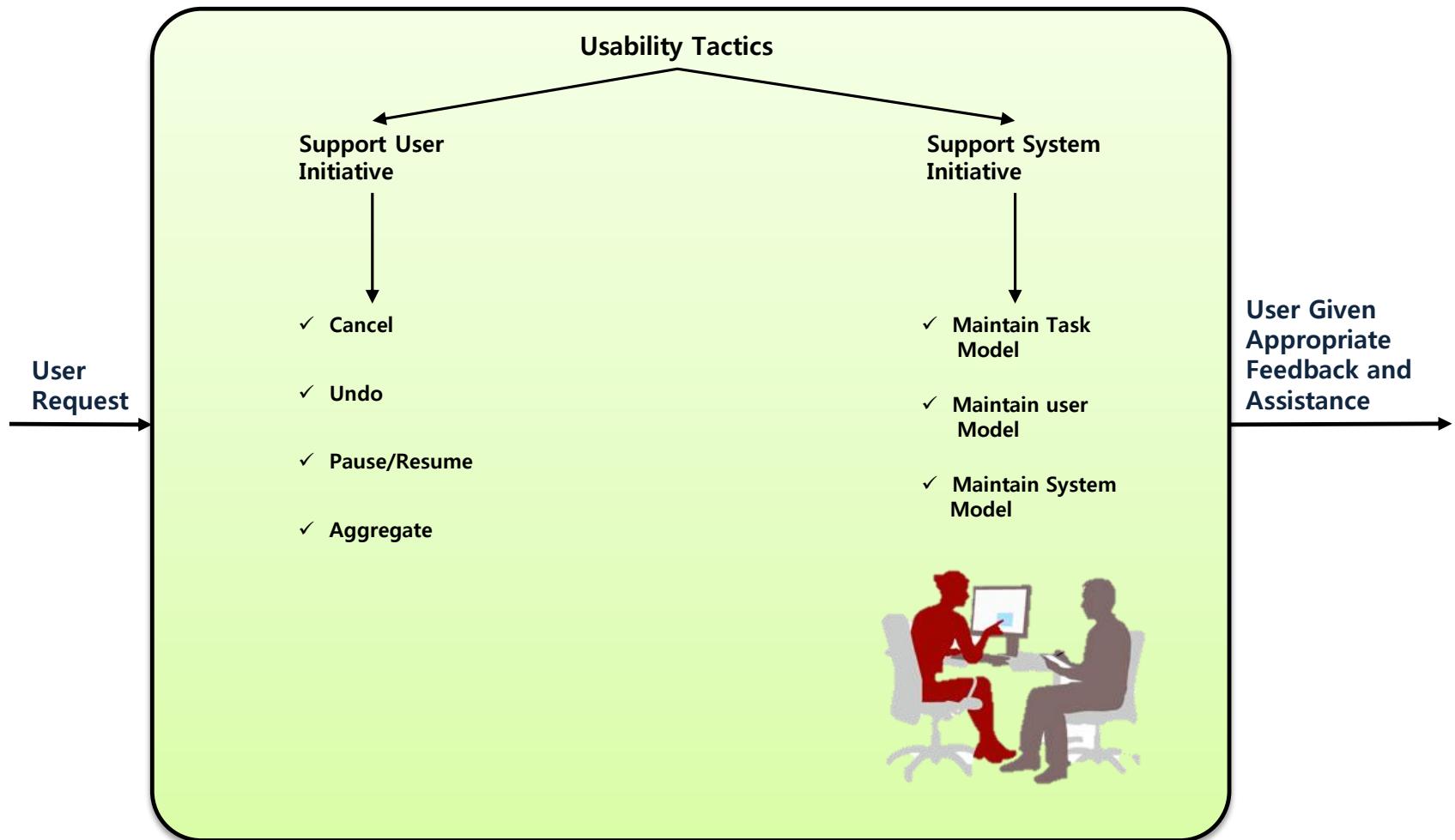
- Sample concrete usability scenario





# 11.2

## ❖ Tactics for Usability





## 11.2

### ❖ Usability Tactics

|                                  |                              |   |
|----------------------------------|------------------------------|---|
| Control and Observe System State | <b>Cancel</b>                | <ul style="list-style-type: none"><li>▪ 사용자가 취소 명령을 내리면 시스템은 해당 명령에 반응해야 하고, 취소되는 명령은 반드시 종료되어야 한다.</li></ul>           |
|                                  | <b>Undo</b>                  | <ul style="list-style-type: none"><li>▪ 이전 상태로 되돌리기 위해 시스템은 시스템의 상태에 대한 충분한 양의 정보를 유지해야 한다.</li></ul>                   |
|                                  | <b>Pause/Resume</b>          | <ul style="list-style-type: none"><li>▪ 사용자가 오랜 시간 동작하는 명령을 수행 시, 다른 태스크에 자원을 할당할 수 있도록 임시로 자원을 해제할 수 있도록 한다</li></ul>  |
|                                  | <b>Aggregate</b>             | <ul style="list-style-type: none"><li>▪ 사용자가 반복적인 명령을 수행 시, 또는 명령이 많은 수의 객체에 동일한 방식으로 적용 시, 명령을 그룹에 대하여 적용한다.</li></ul> |
| Limit Complexity                 | <b>Maintain Task Model</b>   | <ul style="list-style-type: none"><li>▪ 작업 모델은 컨텍스트를 이해하고, 사용자에게 어떤 지원을 할지를 결정하는데 사용된다.</li></ul>                       |
|                                  | <b>Maintain User Model</b>   | <ul style="list-style-type: none"><li>▪ 예상 응답 시간이나 다른 항목을 이용해 시스템에 대한 사용자의 지식과 행동을 명시적으로 나타낸다.</li></ul>                |
|                                  | <b>Maintain System Model</b> | <ul style="list-style-type: none"><li>▪ 사용자에게 적절한 피드백을 제공하기 위해 예상되는 시스템의 동작을 결정한다.</li></ul>                            |



# 11.3

## ❖ A Design Checklist for Usability

| 분류                                   | 체크리스트  |
|--------------------------------------|--|
| Allocation of Responsibilities       | <ul style="list-style-type: none"><li>✓ 사용자를 지원하기 위해 필요한 추가적인 시스템 책임이 할당되었는지 확인<ul style="list-style-type: none"><li>- 시스템 사용 방법 학습, 수동으로 효과적인 태스크 수행, 시스템의 적용 및 구성</li><li>- 사용자 및 시스템 오류로 부터의 복구</li></ul></li></ul> |
| Coordination Model                   | <ul style="list-style-type: none"><li>✓ 시스템 요소의 특성 조정- 적절성, 동시성, 완전성, 정확성, 일관성-이 사용자의 사용에 어떻게 영향을 주는지 결정</li></ul>   |
| Data Model                           | <ul style="list-style-type: none"><li>✓ 사용자가 인지할 수 있는 행위를 포함한 주요 데이터 추상화를 결정<ul style="list-style-type: none"><li>- 이러한 데이터의 추상화가 사용자가 태스크를 완수하는데 지원하기 위해 고안된 것인지를 확인</li></ul></li></ul>                              |
| Mapping among Architectural Elements | <ul style="list-style-type: none"><li>✓ 아키텍처 요소들 사이의 어떤 맵핑이 최종 사용자에게 보여지는지 확인<ul style="list-style-type: none"><li>- 이러한 맵핑이 어떻게 사용자에게 영향을 주는지 확인</li></ul></li></ul>  |
| Resource Management                  | <ul style="list-style-type: none"><li>✓ 사용자가 시스템의 사용 자원을 어떻게 적용하고 구성하는지 결정</li><li>✓ 모든 사용자 구성이 통제된 환경하에서 자원 제한이 사용자의 태스크 수행을 저하시키는지 확인</li><li>✓ 자원의 사용 수준이 사용자의 시스템 사용 능력에 영향을 주지 않는지 확인</li></ul>                   |
| Binding Time                         | <ul style="list-style-type: none"><li>✓ 사용자 통제하의 바인딩 시간 결정을 결정하고 사용자가 사용성 지원을 위한 결정을 내릴 수 있는지를 확인</li></ul>  |
| Choice of Technology                 | <ul style="list-style-type: none"><li>✓ 선택된 기술이 시스템에 적용할 사용성 시나리오를 성취하는데 도움을 주는지 확인</li><li>✓ 선택된 기술이 시스템의 사용성에 악영향을 주지 않는지 확인</li></ul>   |



## **Chapter 12. Other Quality Attributes**



# 12.1

## ❖ Other Important Quality Attributes

|  |   |
|--|---|
| 가변성<br>(Variability)                       | <ul style="list-style-type: none"><li>▪ 변경용이성의 특별한 경우</li><li>▪ 미리 계획된 방식에 의하여 변형된 산출물 세트를 지원하는 능력</li><li>▪ Product Line에서 중요한 특성</li></ul>  |
| 이식성<br>(Portability)                       | <ul style="list-style-type: none"><li>▪ 변경용이성의 특별한 경우</li><li>▪ 한 플랫폼에서 구현된 S/W가 다른 플랫폼에서도 동작하도록 변경될 수 있는 특성</li><li>▪ 플랫폼 종속성 小, 종속성의 격리, 가상 머신(Virtual Machine)</li></ul>   |
| 개발 분산용이성<br>(Development Distributability) | <ul style="list-style-type: none"><li>▪ 소프트웨어의 분산 개발을 지원하기 위한 설계 품질</li><li>▪ (코드와 데이터 모델에 대한) 개발팀 간의 조정을 최소화</li></ul>   |
| 확장성<br>(Scalability)                       | <ul style="list-style-type: none"><li>▪ 수직적 확장성 (논리적 단위에 자원 추가), 수평적 확장성 (물리적 단위 자원 추가)</li><li>▪ 추가된 자원을 얼마나 효과적으로 활용하는가에 대한 문제가 생김</li><li>▪ Effective : 추가적인 자원→측정 가능한 시스템 품질 향상, 추가 노력이 없어야 하며, 시스템 동작을 방해하지 말아야 한다.</li></ul>  |
| 배치성<br>(Deployability)                     | <ul style="list-style-type: none"><li>▪ 실행 단위가 어떻게 호스트 플랫폼에 도착하고 나중에 호출되는가에 대한 특성</li></ul>   |
| 이동성<br>(Mobility)                          | <ul style="list-style-type: none"><li>▪ 플랫폼의 이동과 제공에 관한 문제</li><li>▪ 배터리 관리, 단절 후 재 접속 기간, 여러 플랫폼 지원을 위한 다른 사용자 인터페이스 지원 等</li></ul>  |
| 관찰 용이성<br>(Monitorability)                 | <ul style="list-style-type: none"><li>▪ 시스템에 실행 시 운영 상태를 모니터링 할 수 있는 능력</li></ul>   |
| 안전성<br>(Safety)                            | <ul style="list-style-type: none"><li>▪ 소프트웨어 환경 내에서 손실, 부상, 사용자의 사망 등의 상태의 원인이 되거나 유도할 수 있는 상태로 들어가는 것을 회피하고, 나쁜 상태로 들어 갔을 경우, 손실을 제한하고 회복하는 소프트웨어의 능력</li><li>▪ 위험한 실패에 대한 방지 및 회복</li><li>▪ 가용성(Availability)과 관련이 많다.</li></ul> |



## ❖ Other Categories of Quality Attributes

- 최종 제품의 “이로운 점(Goodness)”를 측정하는 품질속성 부류
- 아키텍처의 개념적인 무결성(Conceptual Integrity)
  - 아키텍처 설계상의 일관성 → 아키텍처에 대한 이해도 ↑, 혼동에 따른 에러 ↓
  - 같은 것을 같은 방식(the same thing is done in the same way)으로,
  - 그러나 간결하게 (Less is more)
- 사용 품질 (Quality in use)
  - 유효성(Effectiveness)
    - 시스템이 올바르게 구축되었는가? (요구사항을 만족하는가?)
    - 올바른 시스템이 구축되었는가? (사용자 의도대로 동작하는가?)
  - 효율성(Efficiency)
    - 비용과 노력이 얼마나 드는가?
  - 위험으로부터의 자유(Freedom from risk)
    - 제품이나 시스템이 경제적인 상태, 인간의 삶, 건강, 또는 환경에 영향을 주는 정도
- 시장성 (Marketability)
  - 시장 경쟁에 대한 시스템의 효용성 (알려진 아키텍처의 경우 다른 속성보다 우선시 함)



# 12.3

## ❖ Software Quality Attributes and System Quality Attributes

- 물리적인 시스템은 품질 속성을 만족시키기 위해 내장된 S/W에 의존한다.
  - 때때로 소프트웨어 아키텍처는 시스템의 품질 속성에 많은 영향을 줄 수 있다.



- 대규모 시스템의 경우

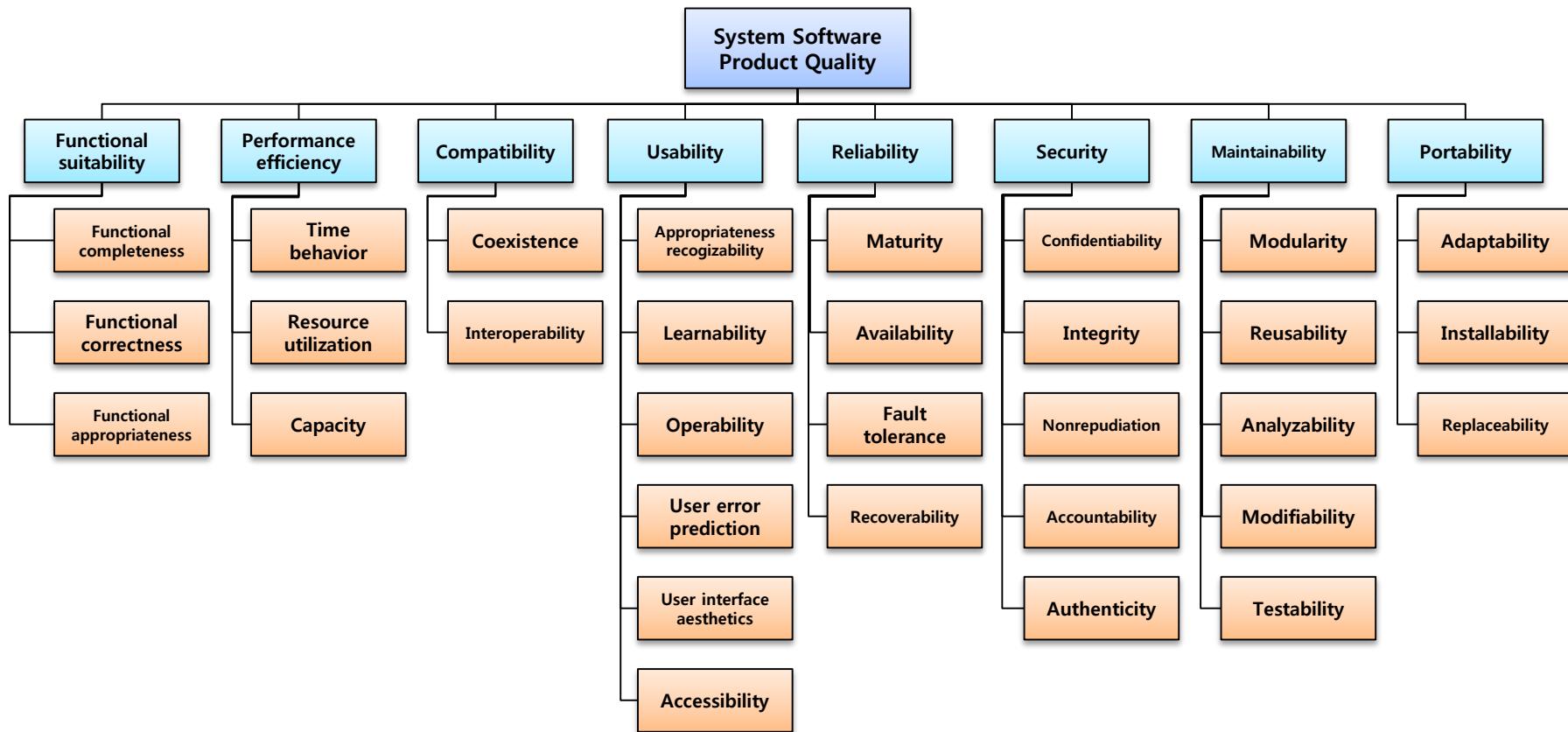


# 12.4

## ❖ Using Standard Lists of Quality Attributes – or Not

- ISO/IEC FCD 25010

- System and software product **Quality Requirements and Evaluation** (SQuaRE)
- 품질 속성을 사용 품질(Quality in Use)와 제품 품질(Product Quality)로 구분

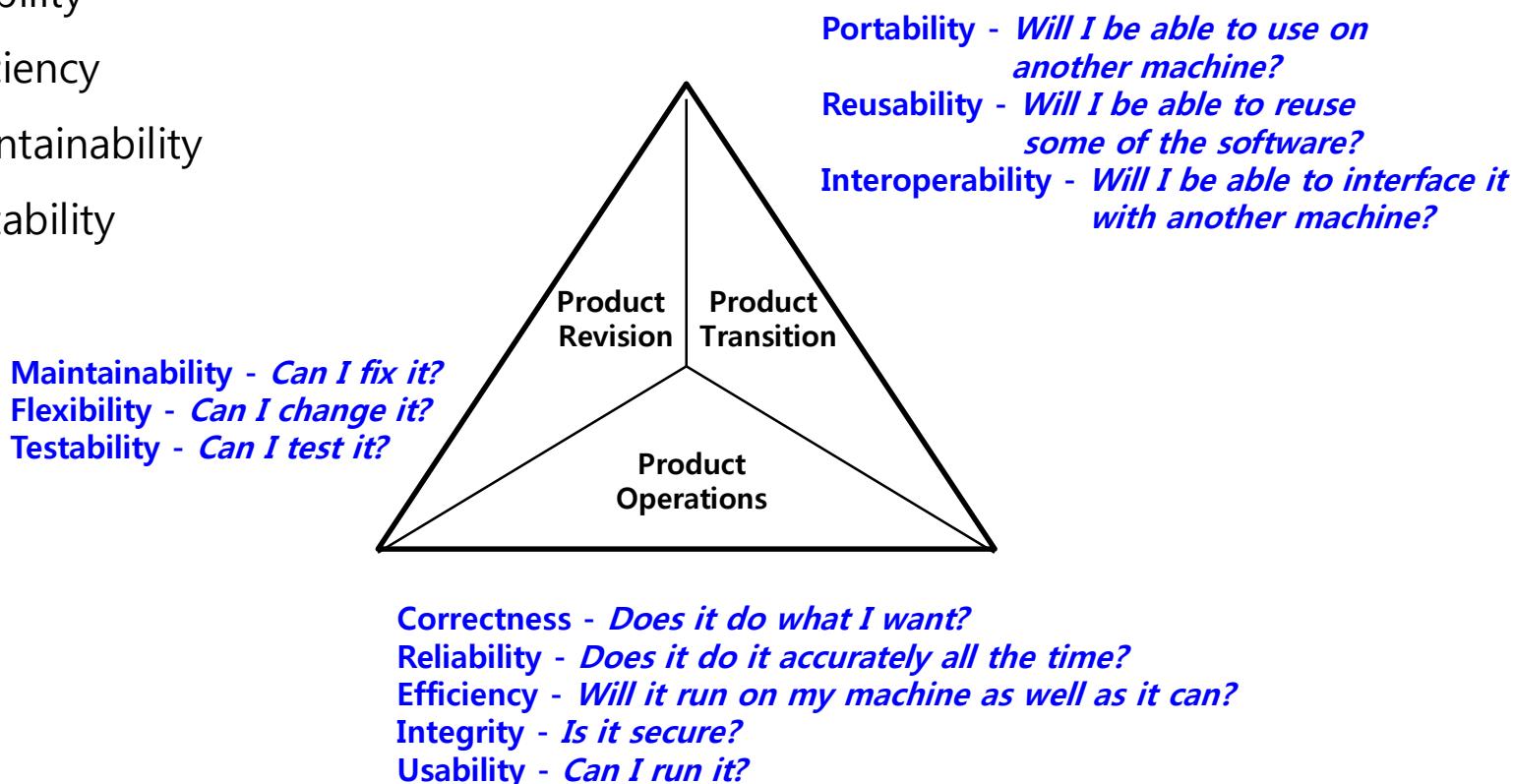




# Appendix : Quality Attributes in ISO 9126

## ❖ ISO 9126 defines 6 attributes

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability





## 12.4

### ❖ Using Standard Lists of Quality Attributes – or Not

- 표준 품질 속성 리스트의 장점과 단점

#### 장 점

- 요구사항(Requirements) 수집 시
  - 요구사항에 대한 체크리스트 작성에 도움을 준다
  - 중요한 요구(Needs)가 빠졌는지 확인하게 한다
- 관심을 가지는 분야의 품질 속성에 대한 체크리스트 작성의 기초가 된다.
- 도메인, 산업, 조직, 제품

#### 단 점

- 완벽한 리스트가 될 수 없다
- 이해보단 논란을 만드는 경우가 있다
- 때때로 분류에 중점을 둔다
- 아키텍트에게 모든 품질 속성에 관심을 두도록 강요한다
  - 시스템에 불필요한 품질 속성에 대해서도

- 표준 품질 속성 리스트의 사용은

- 체크리스트로 도움을 줄 수 있다. 그러나 용어에 너무 집착하는 것은 좋지 않다.
- 품질 속성과 그 하위 품질 속성이 무엇인지에 논하는 것은 대부분 의미 없는 일이다.
- 품질 속성 시나리오는 품질 속성을 논할 때, 가정 정확하게 이야기 할 수 있는 방법이다.



# 12.5

## ❖ Dealing with “X-ability”: Bring a New Quality Attribute into the Fold

### 1 단계 : 새로운 품질 속성에 대한 시나리오 작성

- 1.1 이해 당사자를 통하여 새로운 품질 속성에 대한 필요성 확인
- 1.2 해당 품질 속성의 특징을 묘사하는 시나리오들 작성
- 1.3 수집된 시나리오들을 일반화 한 시나리오 작성

### 2 단계 : 새로운 품질 속성을 위한 설계 방법 만들기

- 2.1 기존 패턴을 조사하고, 각 패턴이 새 품질 속성에 어떻게 영향을 주는지 확인
- 2.2 품질 속성을 다루기 위한 설계 기법 조사
- 2.3 전문가와 면담 및 조언 듣기
- 2.4 일반화 시나리오를 통하여 올바른 응답을 만들기 위한 설계 방법의 리스트 목록화
- 2.5 일반화 시나리오를 통하여 문제 아키텍처가 요구된 응답에 실패하는 경우에 대한 리스트 목록화와 이러한 경우를 제거하기 위한 설계 방법의 고안

### 3 단계 : 새로운 품질 속성 모델링

- 3.1 설계 방법(패턴과 전술)의 셋을 만드는데 도움을 줄 수 있는 개념 모델 생성
- 3.2 모델을 통한 품질 속성에 영향을 주는 파라미터 세의 이해

### 4 단계 : 새로운 품질 속성에 대한 설계 전략 세트 만들기

- 4.1 품질 속성에 대한 전술의 소스(모델과 전문가) 파악
- 4.2 모델에 기반한 전술 생성

- 4.2.1 모델의 파라미터 열거
- 4.2.2 파라미터 별로 영향을 주는 아키텍처적인 결정들 열거

- 디자인 문제를 추적 가능하게 한다
- 일부 속성은 모델을 만들 수 경우가 있다. 이 경우 ...
  - 해당 분야의 전문가와 인터뷰
  - 시스템에 대한 검사
  - 관련된 디자인 검색
  - 문서화된 아키텍처 패턴 검사

### 5 단계 : 새로운 품질 속성에 대한 설계 체크리스트 작성

- 5.1 설계 결정에 대한 7가지 분류에 대하여 시험한다.
- 5.2 소프트웨어 아키텍처에 대한 리뷰
  - 어떻게 새로운 품질을 만족할 것인지 이해