Spring 2023

Architecture Tactics and Modifiability Tactics

Seonah Lee

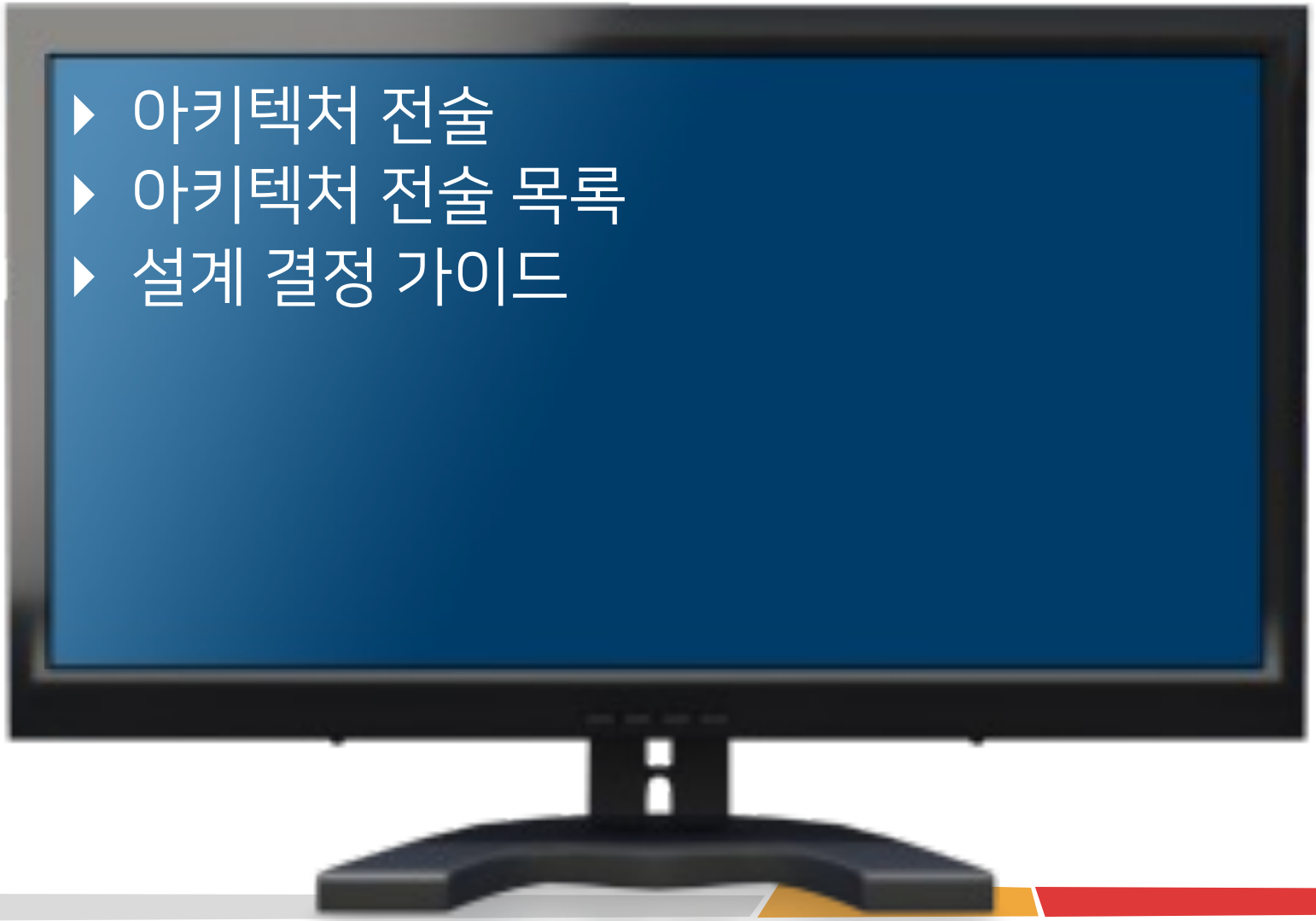
Gyeongsang National University

목 차

- 아키텍처 전술 개념

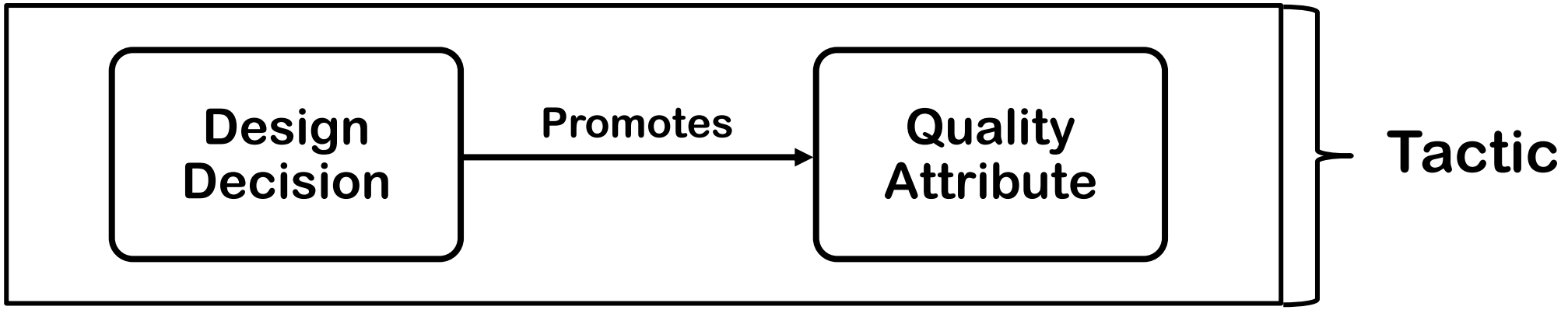
- 변경용이성 품질 시나리오
- 변경용이성 품질 전술
- 변경용이성 관련 토론

소프트웨어 전술 개념



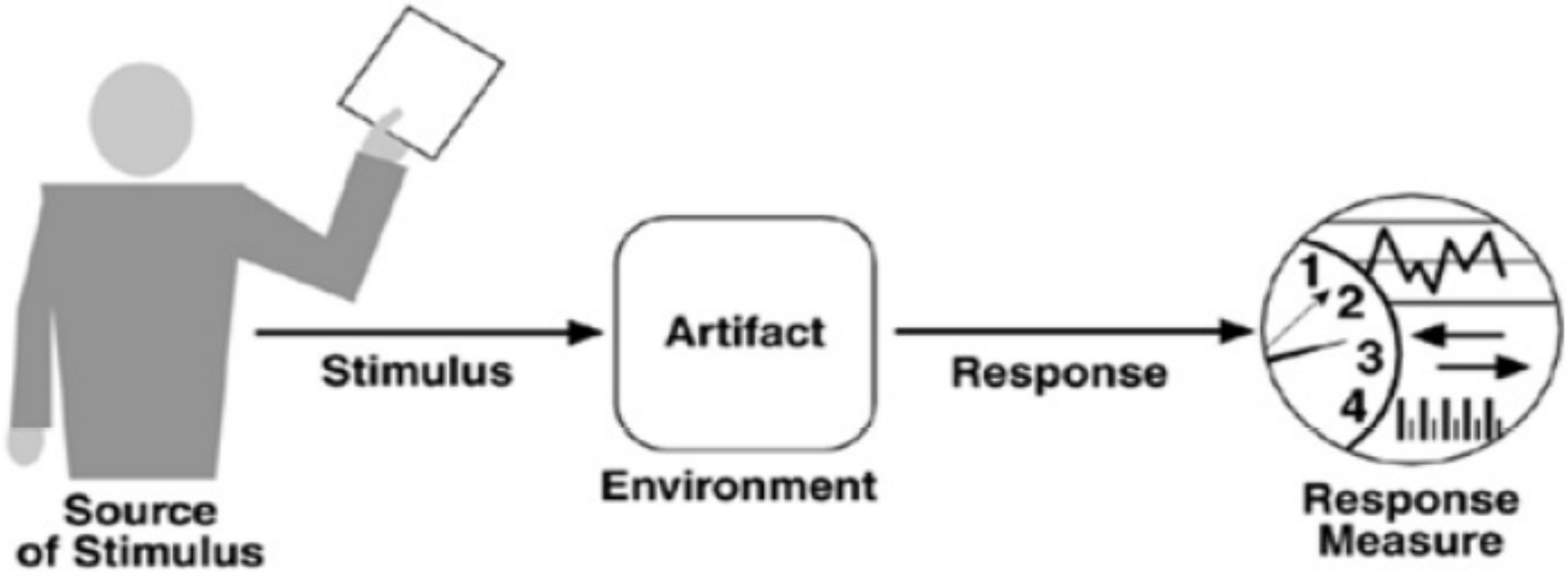
Architecture Tactics

- 아키텍처 전술
  - 품질 속성 반응의 달성에 영향을 주는 설계 결정
  - 품질 수준을 조절할 수 있도록 상위 수준의 패턴을 결정하는 기법



Reminder: Quality Attribute Scenarios

- 품질 속성 시나리오의 주요 구성 컴포넌트는 **자극**과 **응답**



Architecture Tactics

- 전술: 품질 시나리오에서의 **자극(Stimulus)**에 대한 **기대 응답(Response)**를 수행하기 위한 여러 방법 및 패턴을 의미함

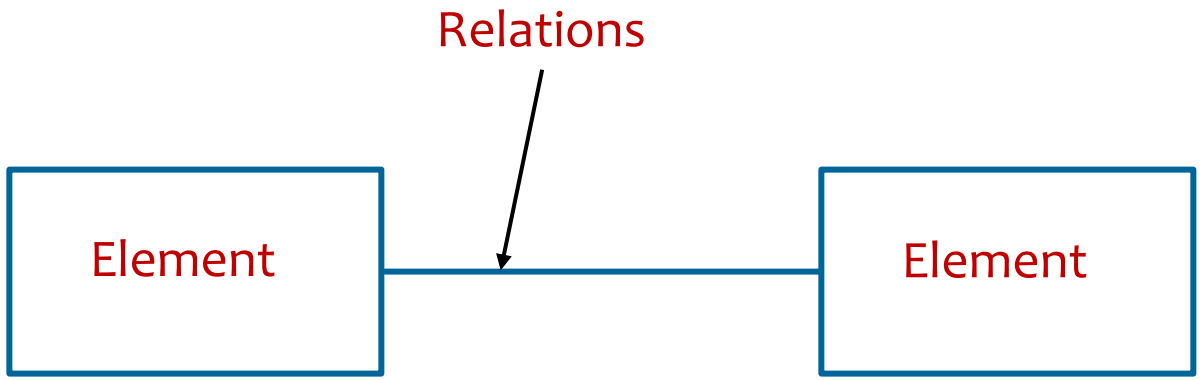
- 전술 목록: 기존에 사용하는 아키텍처 전술을 정리
  - “기본 원칙”으로 부터 설계 단편을 구축할 수 있도록 도움
  - 전술 선택은 다른 품질 속성과 구현 비용의 트레이드 오프 등 고려

Architecture Tactics

- 품질 속성 시나리오에 대해서 다음의 설계 결정을 고려할 수 있음
  - 책임 할당 **allocation of responsibilities**
    - 기본적인 시스템의 중요한 책임을 식별
    - 책임이 런타임 요소(컴포넌트와 커넥터)에 어떻게 할당되는지 결정

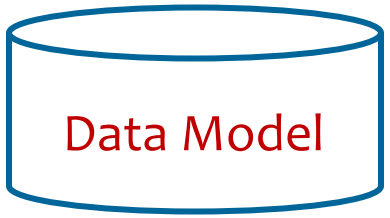
Architecture Tactics

- 품질 속성 시나리오에 대해서 다음의 설계 결정을 고려할 수 있음
  - 조정 모델 **coordination model**
    - 런타임 요소들이 서로 상호작용하는 메커니즘
    - 조정할 시스템 요소 간의 커뮤니케이션 방식 결정



Architecture Tactics

- 품질 속성 시나리오에 대해서 다음의 설계 결정을 고려할 수 있음
  - 데이터 모델 **data model**
    - 시스템 관점의 데이터를 표현하고 해석하는 방법
    - 데이터를 구성, 데이터의 특성과 함수를 결정

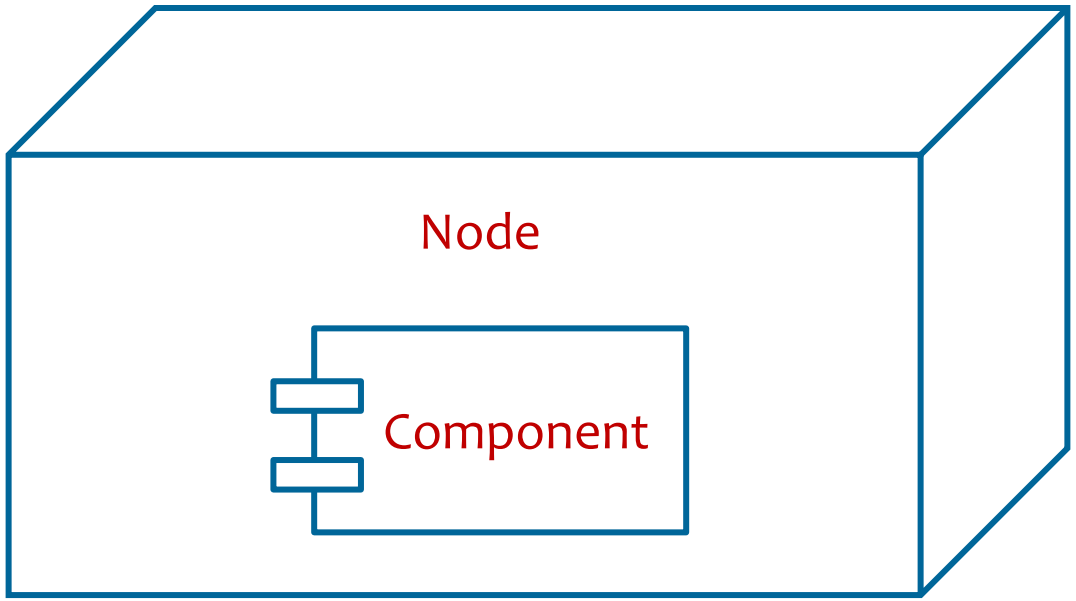


Architecture Tactics

- 품질 속성 시나리오에 대해서 다음의 설계 결정을 고려할 수 있음
  - 리소스 관리 **resource management**
    - 아키텍처 안에서 공유되는 리소스의 사용을 중재
    - 하드웨어 리소스와 소프트웨어 리소스를 포함

Architecture Tactics

- 품질 속성 시나리오에 대해서 다음의 설계 결정을 고려할 수 있음
  - 아키텍처 요소 매핑 **mapping architectural elements**
    - 서로 다른 유형의 아키텍처 구조에 있는 요소들 사이의 매핑
    - 소프트웨어 요소와 환경 요소 사이의 매핑



Architecture Tactics

- 품질 속성 시나리오에 대해서 다음의 설계 결정을 고려할 수 있음
  - 바인딩 시간 결정 **binding time decisions**
    - 범위, 라이프 사이클의 바인딩 시점, 가변 달성 메커니즘 수립
    - 상기 부류를 위한 바인딩 결정 중 구축 비용과 변경 비용 검토
  - 기술 선택 **choice of technology**
    - 모든 아키텍처 결정은 궁극적으로 특정한 기술로 실현
    - 상기 부류를 위한 결정을 실현한 적당한 기술을 선택

Questions

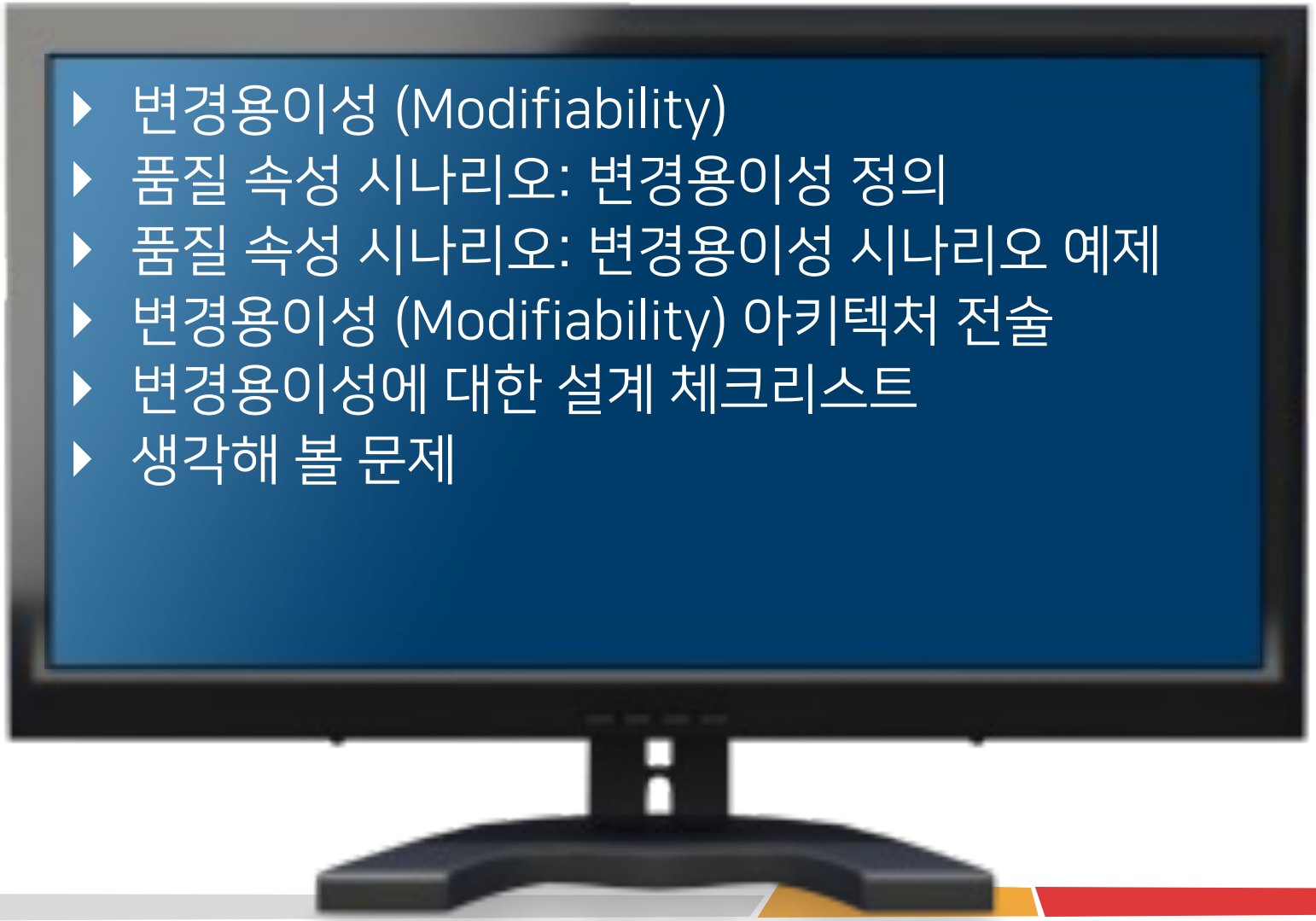
Q1. 유스케이스와 품질 속성 시나리오와의 관계는 무엇인가?

- 유스케이스에 품질 속성 정보를 추가하고 싶다면 어떻게 해야 하는가?

Q2. 품질 속성에 대한 전술 집합이 무한하다고 생각하는가?

- 혹은 유한하다고 생각하는가?
- 왜 그런가?

변경용이성 (Modifiability) 아키텍처 전술

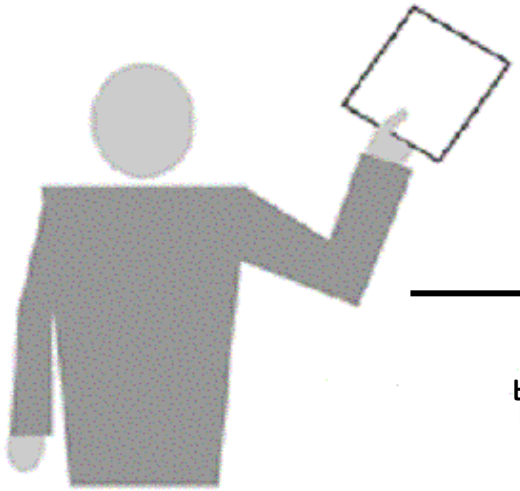


Modifiability

- 변경 용이성 **(Modifiability)**
  - 일반적으로 소프트웨어 시스템의 대부분의 비용은 **1**차 릴리즈 이후에 발생함
    - 새로운 기능을 추가할 때, 오래된 시스템을 대체할 때도 변경이 발생
    - 결함을 수정하거나, 보안을 강화할 때, 성능을 개선할 때도 변경 발생
    - 사용자 경험을 개선할 때도 변경 발생
    - 새로운 기술, 플랫폼, 프로토콜, 표준을 도입할 때도 변경 발생
    - 시스템들을 함께 동작할 수 있도록 하는데도 변경 발생
  - 변경 용이성은 변경을 수행하기 위한 시간과 비용에 대한 것

Quality Attribute Scenario for Modifiability

- 변경용이성(**Modifiability**): 소프트웨어 시스템이 변화하려고 할 때 발생하는 비용과 변화 용이성을 의미



자극의 근원:
개발자

자극:
변경 원함

대상(Artifact):
코드, 사용FAQ

환경:
설계 시점

응답:
변경으로 인한 부작용 없음

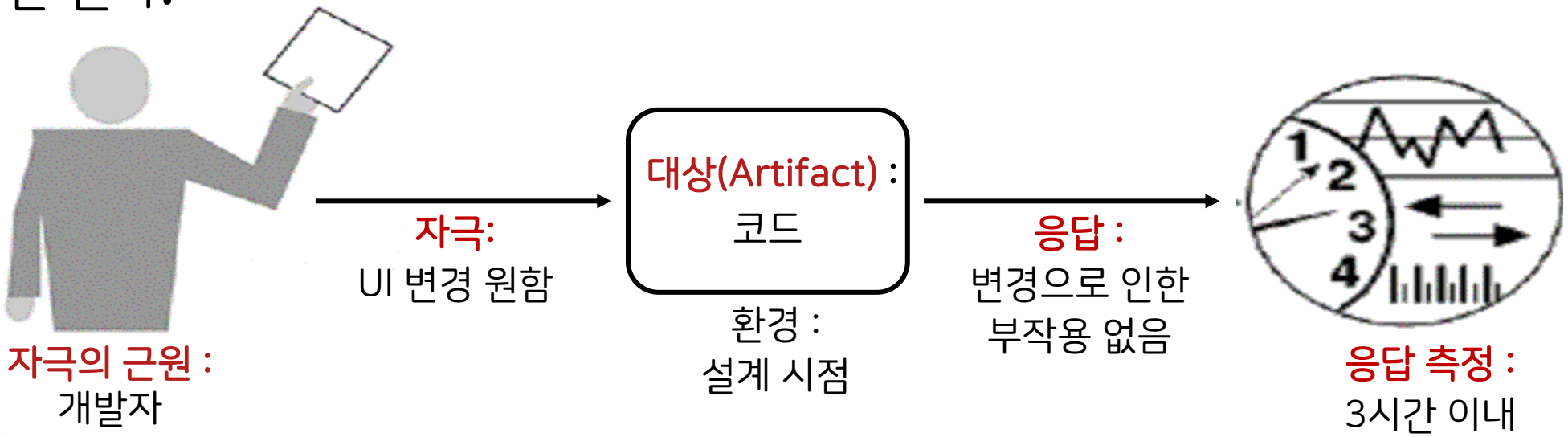
응답 측정:
기간, 수정 범위

Quality Attribute Scenario for Modifiability

Component	Description
자극의 근원 (Source)	변경을 일으키는 주체 • 개발자 | 시스템 관리자 | 최종 사용자
자극 (Stimulus)	변경이 원인이 되는 요구 • 새로운 기능의 추가, 기존 기능의 수정, 기능의 삭제 • 동시 사용자의 수의 증가 • 품질 속성의 변경
환경 (Environment)	변경이 발생되는 시기 • 설계 시간 | 컴파일 시간 | 빌드 시간, 초기화 | 런타임
대상 (Artifact)	변경되는 대상 • 시스템의 기능성, 시스템의 플랫폼 • 사용자 인터페이스 • 환경(혹은 상호 운용되는 타 시스템)
응답 (Response)	변경에 대한 내용 • 다른 기능에 영향을 주지 않고 수정함 • 변경을 시험하고 변경된 결과를 배치함
응답 측정 (Response Measure)	• 변경에 대한 수정을 수행하기 위해 소모되는 시간과 비용

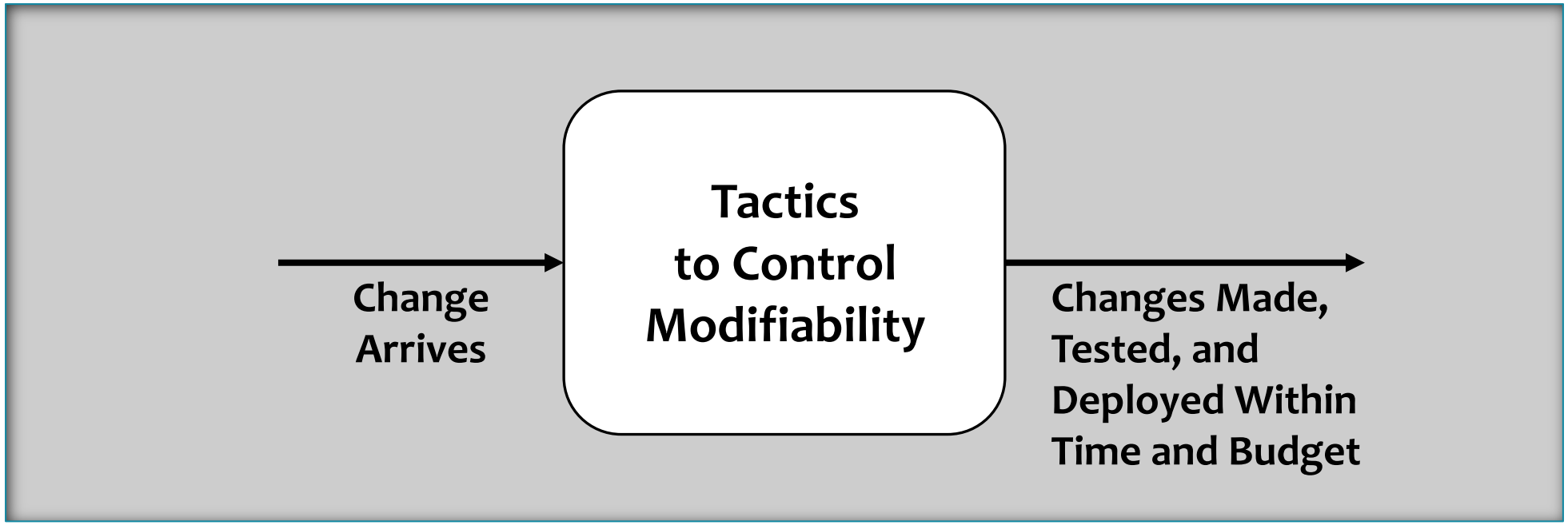
Quality Attribute Scenario Example for Modifiability

- 변경용이성 (**Modifiability**)의 시나리오 예
  - 개발자가 시스템의 배경화면을 파란색으로 바꾸기 위해 사용자 인터페이스를 변경하려고 한다. 이에 따라 개발자는 설계 시점에 코드를 변경할 것이다.
  - **3**시간 이내에 변경과 시험이 이루어져야 하고 이 행위가 시스템에 다른 영향을 주면 안 된다.

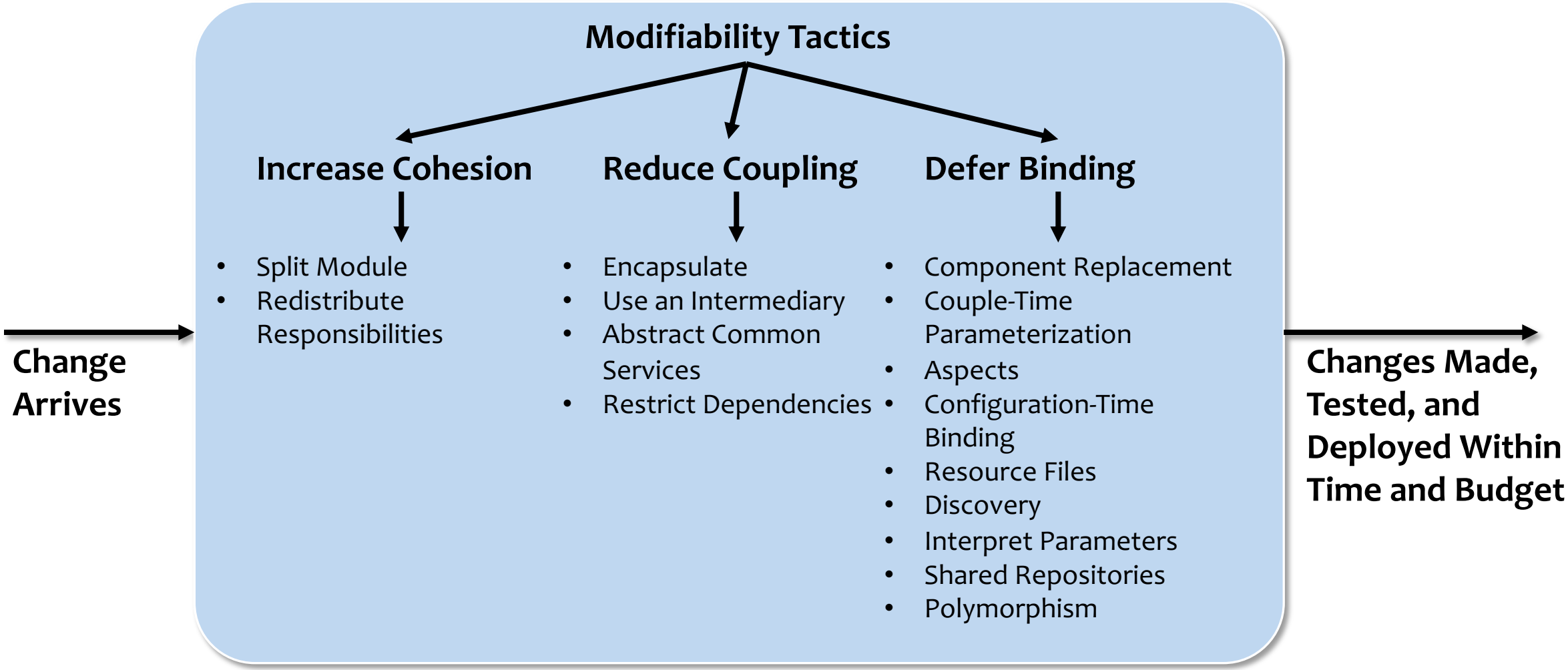


Modifiability Tactics

- 변경 용이성은 변경이 필요할 때, 변경을 구현, 시험, 적용하는데 걸리는 시간과 비용이다.



Modifiability Tactics

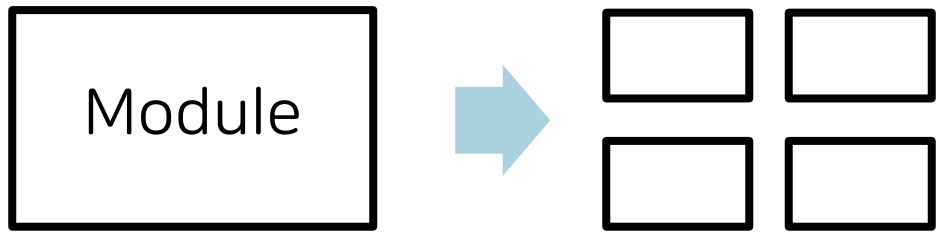


Modifiability Tactics

- 하나의 변경이 가급적 적은 수의 모듈에 영향을 미치도록 함
  - **Loose Coupling & High Cohesion**
- 응집성 증가(**Increase Cohesion**)
  - 응집성이란?
    - 하나의 모듈의 책임들이 얼마나 강력하게 연결되는지 측정
      - 모듈 목적의 단일성을 측정
        - 하나의 책임에 영향을 미치는 변경 시나리오가 다른 책임에도 영향을 미칠 확률
        - 응집성이 높을 수록, 주어진 변경이 여러 모듈에 미칠 영향의 확률이 낮음
      - 하나의 모듈이 응집성이 낮다면 예측되는 변경에 영향을 받지 않는 책임을 제거하여 높임

Modifiability Tactics

- 모듈 분할 **(Split Module)**
  - 만약 모듈의 크기가 크다면, 수정 비용도 높아짐
    - 모듈을 작게 분할하는 것이 향후 평균적인 변경 비용을 줄임
  - 모듈이 응집성 없는 책임을 가지고 있다면, 변경 비용이 높아짐
    - 하나의 모듈을 여러 개의 응집성 있는 모듈로 나누어 변경 비용을 줄일 수 있음
      - 모듈의 코드를 반으로 나누어 분할한다는 것이 아님
      - 응집성 있는 하위 모듈들로 분할을 하는 결과를 내야 함

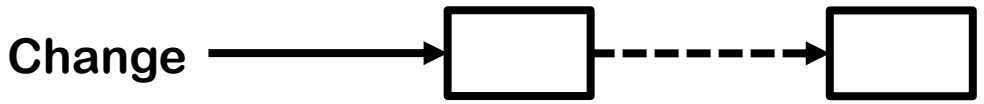


Modifiability Tactics

- 책임 재분배 (**Redistribute Responsibilities**)
  - 유사한 책임이 여러 모듈로 나누어져 있으면, 합쳐야 함
    - 책임 **A, A', A''** ➔ 하나의 모듈
  - 이동해야 할 책임을 명시하는 방법
    - 변경 시나리오의 집합을 가정해 보는 것
      - 변경 시나리오가 모듈의 일정 부분에만 영향을 미치는 경우
        - 다른 부분들은 별도의 책임을 가지고 있다고 볼 수 있음, 따라서 이동해야 함
      - 변경 시나리오가 여러 모듈의 변경을 필요로 하는 경우
        - 영향을 받는 책임들을 묶어 새로운 모듈로 그룹핑

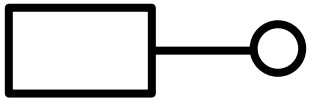
Modifiability Tactics

- 결합성 감소(Reduce Coupling)
  - 결합성이란?
    - 두 개의 모듈의 책임이 어떤 측면에서 중복이 된다면, 하나의 변경이 양 모듈에 영향을 미침
    - 이러한 중복은 하나의 모듈에서의 변경이 다른 모듈에 어떻게 영향을 미치는지의 확률로 평가
    - 이러한 두 개의 모듈 사이의 관계를 커플링이라고 함
  - 두 개의 모듈의 결합도를 낮추면 변경용이성의 비용을 줄임
    - 결합도를 줄이는 전술은 높게 결합될 모듈 사이에 다양한 중재자를 두는 방식



Modifiability Tactics

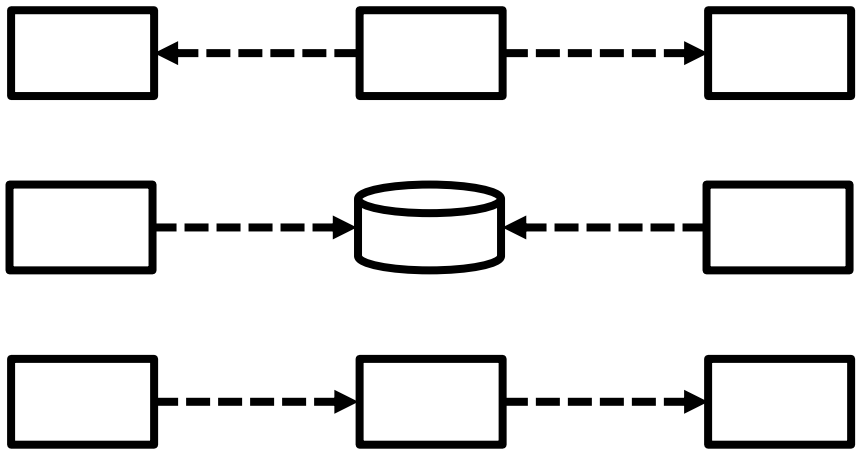
- 캡슐화 (**Encapsulate**)
  - 캡슐화는 구성요소에 명시적인 인터페이스를 도입



    - 요소 내부에 대한 의존은 제거됨
  - 구성요소에 접근할 때 인터페이스를 통해서 접근하는지 확인
    - 모든 의존성은 인터페이스를 통해 흐름 진행
  - 캡슐화는 하나의 구성요소의 변경이 다른 구성요소에 영향을 미치는 확률 줄임
    - 의존의 수를 줄임
    - 의존의 거리를 줄임
  - 약점: 외부의 책임있는 시스템/구성요소가 구성 요소와 상호작용하는 정도를 제약
    - 인터페이스를 통한 직접 상호작용 지원, 간접 상호작용(예: 서비스 품질)을 바꾸기는 힘듦

Modifiability Tactics

- 중재자 사용 (**Use an Intermediary**)
  - 시스템에 있는 두 컴포넌트의 의존성을 제거하기 위해 사용
    - 책임 **A**와 **B** 간에 의존성이 있다면, 이는 중재자를 사용하여 없앨 수 있음
  - 중재자는 서로 다른 타입의 의존성을 해결하는데 활용될 수 있음
    - **Publish-subscribe bus**
    - **Shared data repository**
    - **Dynamic service discovery**
    - **Data transformers**
    - **Protocol translators**



Modifiability Tactics

- 공통 추상 서비스 (**Abstract Common Services**)
  - 두 개의 모듈이 완전히 같지는 않으나 비슷한 기능을 제공할 때 좀 더 일반적인 서비스를 위해서 일반적 추상화 뒤에 구체적인 요소를 숨기는 것이 유용
    - 두 개의 모듈이 구현하는 공통 인터페이스로 구현
    - 중재자를 활용: 추상적 서비스를 위한 요청을 추상화 뒤의 요소를 위한 구체적 요청으로 번역
  - 공통 추상 서비스 **+** 중재자(**wrapper or adapter**)
    - 특정 요소의 형식적 내용적 변이를 정규화할 수 있음
      - 예: 서로 다른 제조사의 같은 타입의 센서 사용 **=>** 일반적인 인터페이스 제공
  - 공통 인프라스트럭처의 관심 사항을 처리할 때의 일관성을 지원
    - 번역, 보안, 로깅 메커니즘 들

Modifiability Tactics

- 의존성 제약 (**Restrict Dependencies**)
  - 주어진 모듈에 상호작용하거나 의존하는 모듈들을 제약하는 방법
    - 모듈의 가시성을 제약
      - 개발자가 인터페이스를 보지 못하게 함으로써, 해당 모듈을 활용하는 선택을 제약
    - 사용 승인
      - 승인 받은 모듈만 접근할 수 있도록 제약
  - **Layered Architecture**에서 이러한 의존성 제약을 볼 수 있음
    - 레이어는 오직 하위 레이어만 사용할 수 있음
  - **Wrapper**를 사용할 수 있음
    - 외부 **Entity**는 오직 **Wrapper**만 볼 수 있고 내부 기능을 직접 볼 수 없음

Modifiability Tactics

- 바인딩 지연 **(Defer Binding)**
  - 바인딩 지연의 이유
    - 사람의 작업은 거의 언제나 컴퓨터의 작업보다 비쌈
    - 따라서 컴퓨터가 가능한 변경을 수행하는 것이 변경 비용을 줄임
- 바인딩 지연 시간
  - 빌드 시점: 컴파일 및 빌드 시점의 바인딩
  - 배포 시점: 환경 설정 시점의 바인딩
  - 초기화 시점: 리소스 파일을 활용하여 바인딩
  - 실행 시점: 실행 시간에 바인딩

Modifiability Tactics

- 컴파일 및 빌드 시점
  - 컴포넌트 대체
    - **build script**
    - **makefile**
  - 컴파일 시점의 파라미터
    - **Compile-time parameterization**
  - **Aspects** 적용
- 배포 시점 및 초기화 시점
  - **Configuration-time binding**
  - **Resource files**
- 런타임 시점
  - **Discovery**
  - **Interpret parameters**
  - **Shared repositories**
  - **Polymorphism**

Questions

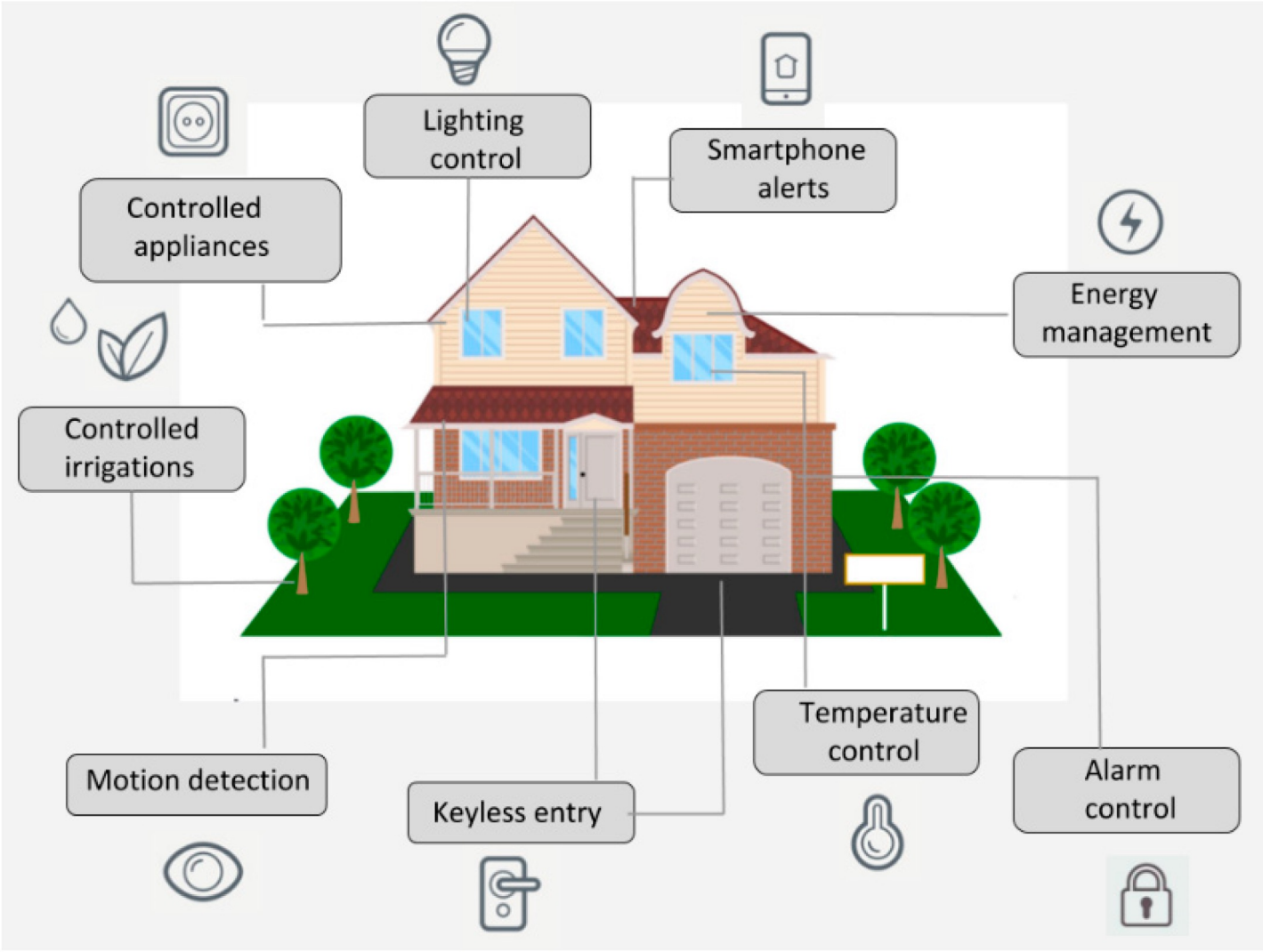
Q1. 어떤 프로젝트에서 배포용이성(**Deployability**)는 얼마나 손쉽게 시스템의 새로운 버전을 제공하는가를 측정하는 중요한 품질 속성이다.

- 이것은 인터넷을 통해서 업데이트를 전송하는 것을 의미한다.
- 또한 일단 도착하면 업데이트를 설치하는 데 걸리는 시간을 포함한다.
- 배포용이성을 별도로 측정하는 프로젝트에서 새로운 버전이 출시할 준비가 되었을 때 변경 비용 측정이 중단되어야 하는가?
- 대답의 이유를 설명하자.

Q2. 추상적인 공통서비스 전략은 결합성과 응집성을 모두 감소시킨다. 해당 전략이 바람직한지 논의한다.

고려 상황: 홈 자동화 시스템

- 조명 켜고 끄기, 문 잠금, 문 잠금 해제, 원격 카메라 관찰 등 홈 자동화 시스템
  - 각 시스템은 소비자(소규모 가족)에게 판매
  - 회사는 처음 **3**년 동안 이러한 장치를 수천 대 판매 예상



https://nealford.com/katas/list.html; https://www.mdpi.com/1424-8220/21/11/3784

고려 상황: 홈 자동화 시스템

- 요구 사항
  - 시스템은 가능한 한 턴키 방식이어야 하지만 쉽게 구매할 수 있도록 모듈식 장치(카메라, 잠금 장치, 온도 조절기 등)로 판매되어야 함
  - 장치는 인터넷을 통해 액세스할 수 있어야 함(원격 모니터링 및 액세스용)
    - 사용자가 사용할 기존 **WiFi** 설정(라우터 및 연결)이 있다고 가정
  - 고객은 자신의 필요에 따라 다양한 모듈을 제어하도록 시스템을 프로그래밍할 수 있음
  - 장치의 전기 엔지니어링은 다른 그룹에서 처리
    - 모듈 제어를 위한 소프트웨어 프로토콜은 아키텍처의 요구/설계에 따라 유연
      - 당신이 프로토콜을 모듈에 지정하면 다른 그룹은 프로토콜의 모듈 측 구현을 처리할 것임
- 추가 컨텍스트
  - 국제기업: 새로운 비즈니스 라인을 순조롭게 시작하기 위해 많은 금액을 투자할 의향있음
  - 더 광범위한 통계 수집을 선택한 고객으로부터 데이터를 수집

Question?

Seonah Lee
saleese@gmail.com