P08. *Architecture Design Procedure*

2014

Sungwon Kang

8. 아키텍처 설계 절차

8.1 요구사항에서 개념적 아키텍처뷰로

- How do we design software architecture?
  - ☛ By Weaving Together Requirements and Architecture

The next slide shows why the boundary between problem and solution is fuzzy.

Weaving Together Requirements and Architecture

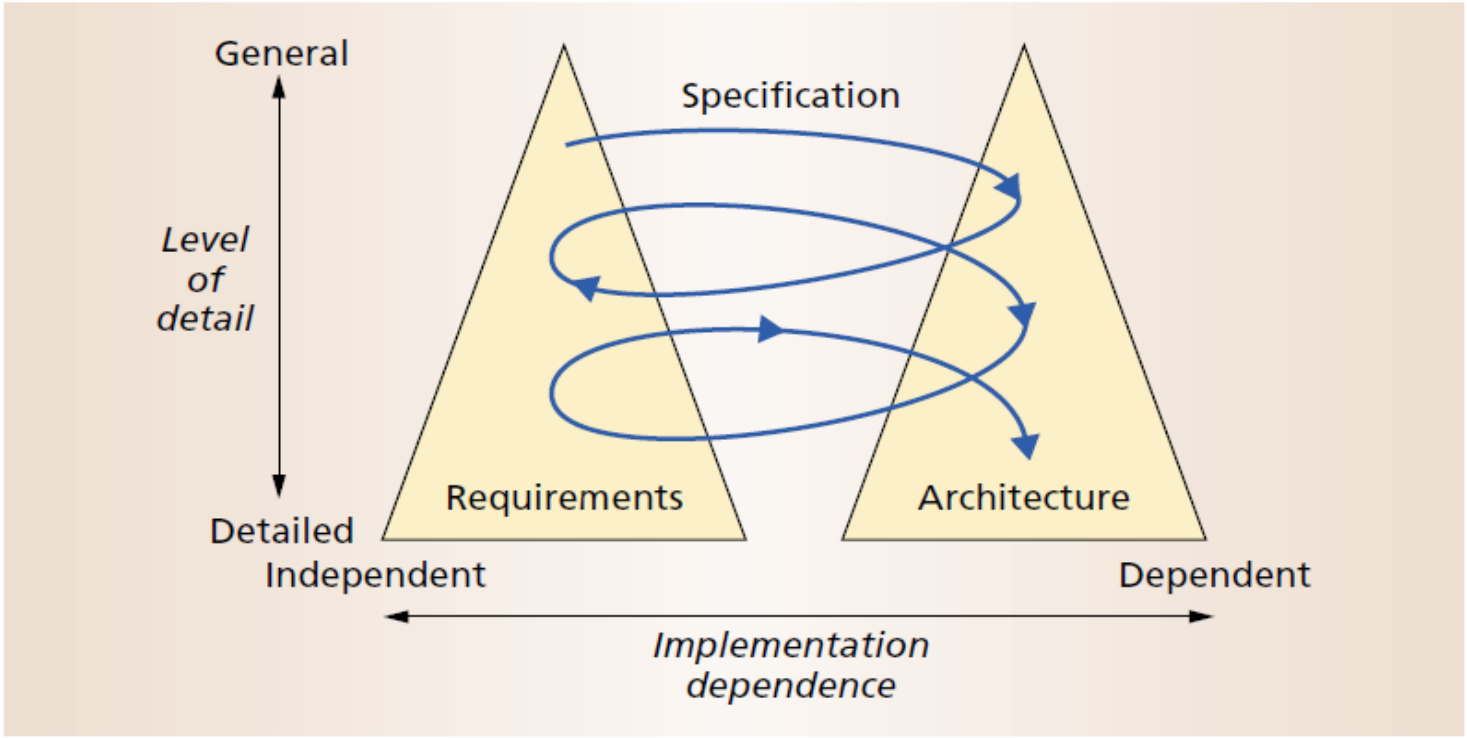


Figure 1. The Twin Peaks model develops progressively more detailed requirements and architectural specifications concurrently. This is an adaptation of the model first published in Paul Ward and Stephen Mellor's* Structured Development for Real-Time Systems: Introduction and Tools, *vol. 1, Prentice Hall, Upper Saddle River, N.J., 1985, and subsequently adapted by Andrew Vickers in his student lecture notes at the University of York, UK.

B.A.Nuseibeh, "Weaving Together Requirements and Architecture", IEEE Computer, 34(3):115-117, March 2001

Why?

Barry Boehm, “Requirements that Handle IKIWISI, COTS, and Rapid Change,” Computer, July 2000, pp. 99-102.

- I’ll Know It When I See It (IKIWISI).
  - Requirements often emerge only after users have had an opportunity to view and provide feedback on models or prototypes.
- Rapid change.
  - Managing change continues to be a fundamental problem in software development and project management.
- Commercial off-the-shelf software(COTS).
  - Increasingly, software development is actually a process of identifying and selecting desirable requirements from existing commercially available software packages.

☛ Requirements are discovered rather than given !

구축성	목적	논리적 아키텍처관점에 있어서 가장 중요한 품질속성으로 시스템을 서브시스템으로 효율적으로 분해하여 시스템의 실현가능성과 구현성을 높인다.
	전략	– 아키텍처 패턴들을 이용하여 시스템을 서브시스템들로 분배한다. – 기능적 분배를 통하여 시스템을 서브시스템들로 분해한다.
	전략 가이드 라인	– 명료한 아키텍처로 분해한다. – 문제 도메인(application domain)을 반영한다. – 낮은 커플링과 높은 응집도를 고려한다.
재사용성	목적	시스템을 서브시스템으로 분해 시에 기존의 컴포넌트들을 최대한 재사용을 하여 개발이 효율적이 될 수 있도록 한다.
	전략	– 시스템을 분해할 때 기존의 컴포넌트를 재사용한다. – 공통된 기능적 요구사항이 있으면 독립적인 컴포넌트로 분해한다.
	전략 가이드 라인	– 기능적 요구사항을 변경을 고려하여 재사용성을 높이도록 고려한다. – 인터페이스의 수리 및 복구를 통해 인터페이스 불일치를 최소화 한다. – 분해 시에 직접적인 두 개의 컴포넌트를 대신하여 공통적인 요구사항을 담고 있는 중간의 컴포넌트의 도입을 고려한다.

그림 8-1. 논리적 아키텍처 설계를 위한 구축성과 재사용성을 위한 설계 가이드라인

8.2 아키텍처 설계 절차의 개요

- **Guideline**
  - 설계의 일반원리 **GP1-GP3**를 적용한 아키텍처 설계 절차
    (상하향식이 적절한 경우는 **GP4** 까지 포함**)**
  - 아키텍처 스타일에 맞는 아키텍처 설계 절차
  - 다관점**(Multiple Architecture Viewpoints)**을 적용한 아키텍처 설계절차

설계의 일반원리 GP1-GP3를 적용한 아키텍처 설계 절차

- 설계의 일반원리(7장):
  - **GP1**. 큰 규모의 문제의 해결을 위해서는 분할정복의 접근방법을 적용한다.
  - **GP2**. 설계는 합성과 분석으로 반복으로 구성된다.
  - **GP3**. 합성을 위하여 경험적 방법 혹은 창의적 방법이 사용된다.

설계의 일반원리 **GP1-GP3**를 적용한 아키텍처 설계 절차

- Reqs: 요구사항의 집합
- QADs: 아키텍처 드라이버 품질속성의 집합
- CADs: 아키텍처 드라이버 제약사항의 집합
- ADs: 아키텍처드라이버의 집합
- QSADs: 품질속성시나리오의 집합
  - ADs = QADs ∪ CADs
  - ADs = QSADs ∪ CADs

설계의 일반원리 **GP1-GP3**를 적용한 아키텍처 설계 절차

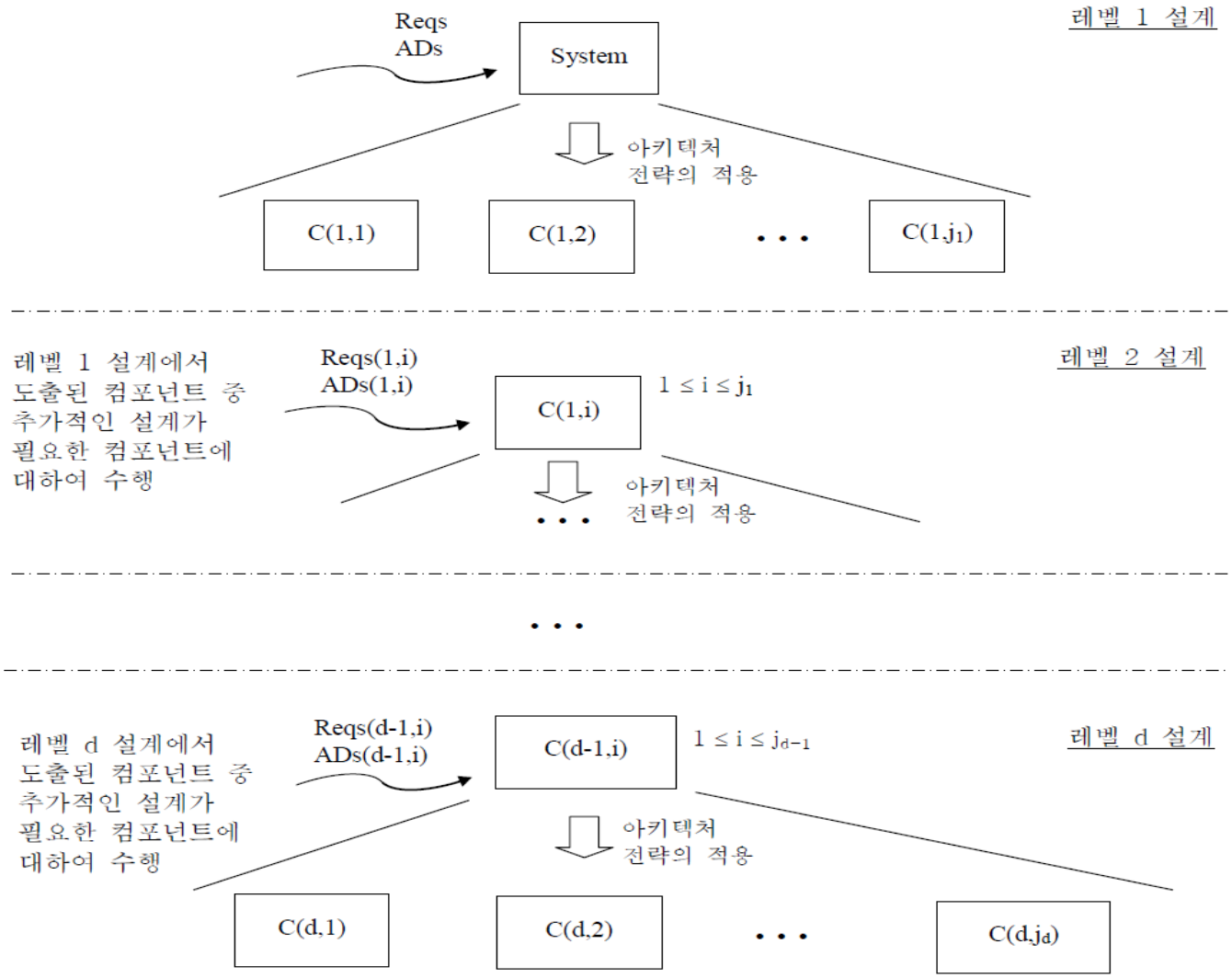


그림 8-3. 설계의 일반원리 GDP1-GDP3를 적용한 아키텍처 설계 절차

Your project may require top-down bottom-up approach (GP4).

8.3 다관점체계를 위한 아키텍처 설계 절차

8.3.1 아키텍처 뷰의 설계순서

8.3.2 다관점체계의 아키텍처 설계를 위한 두 가지 접근방법

8.3.1 아키텍처 뷰의 설계순서

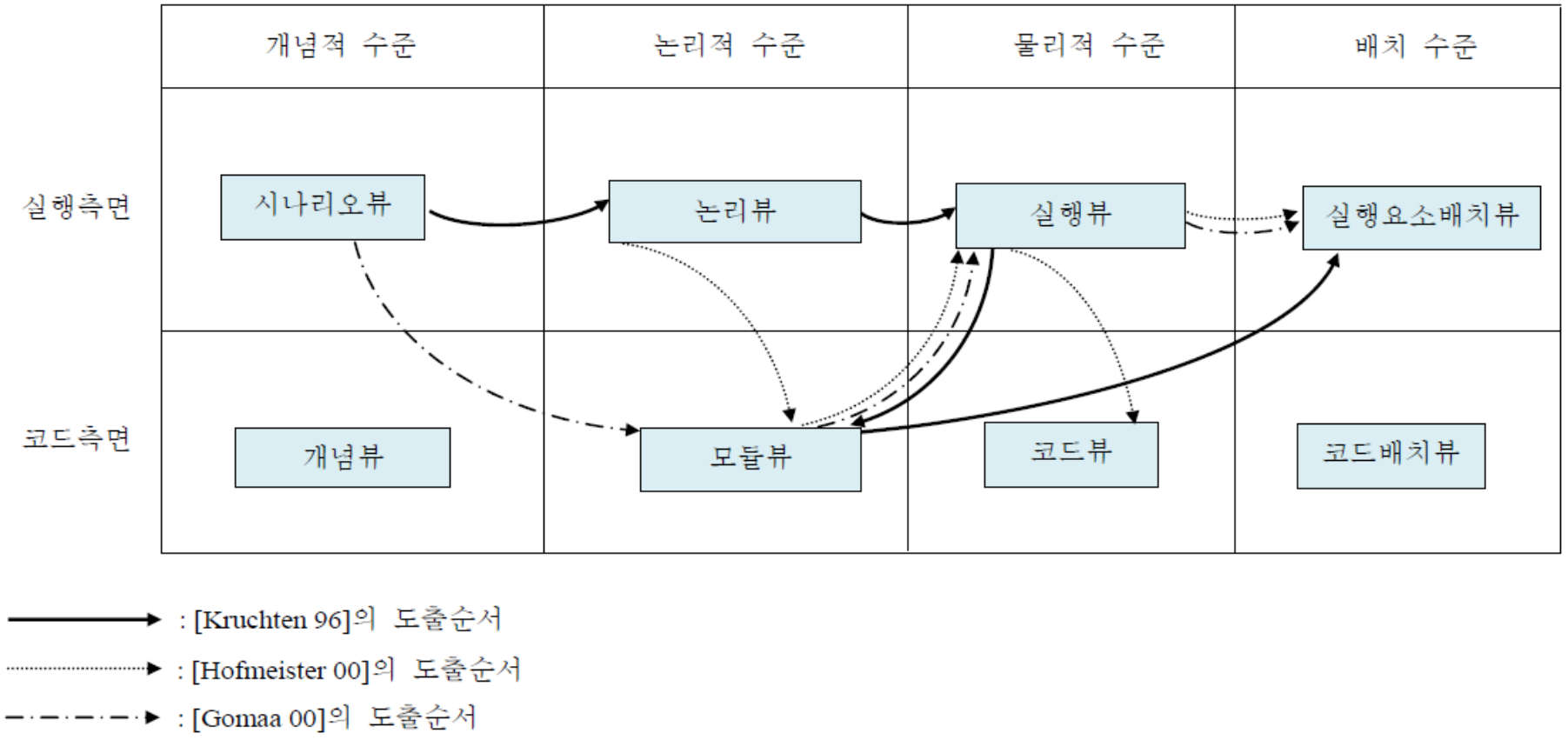
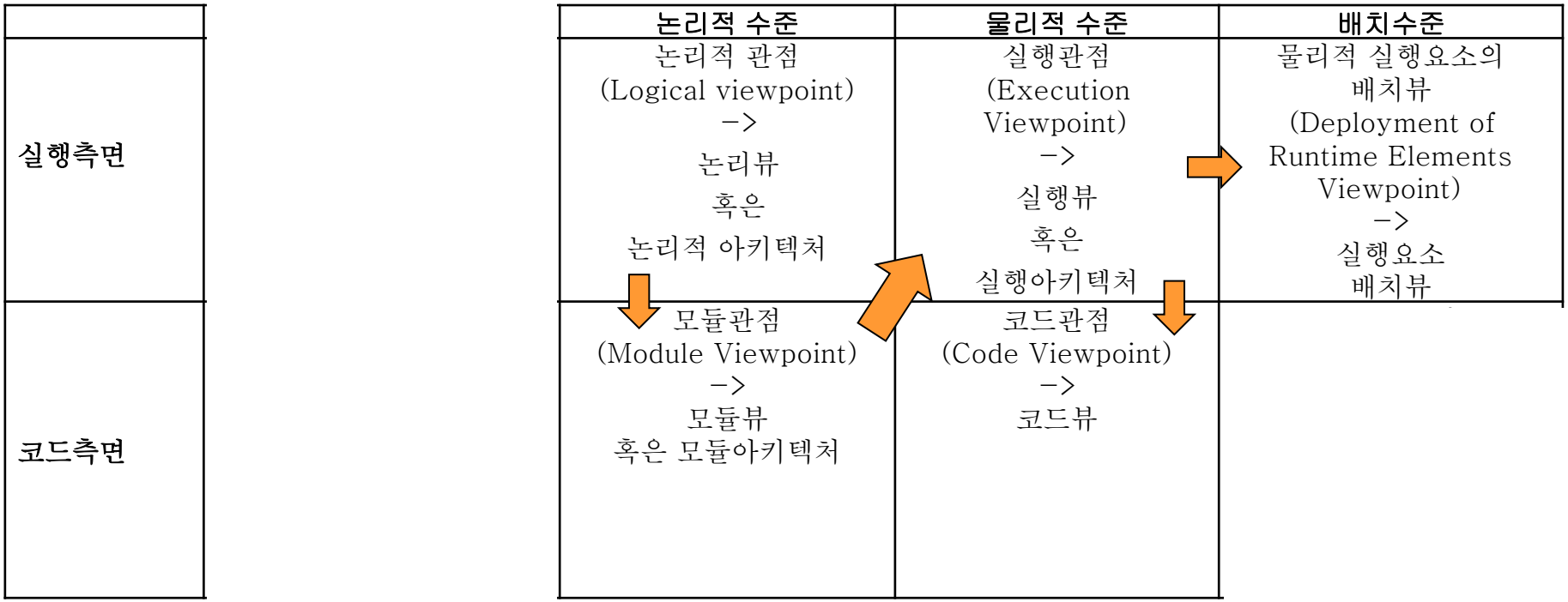
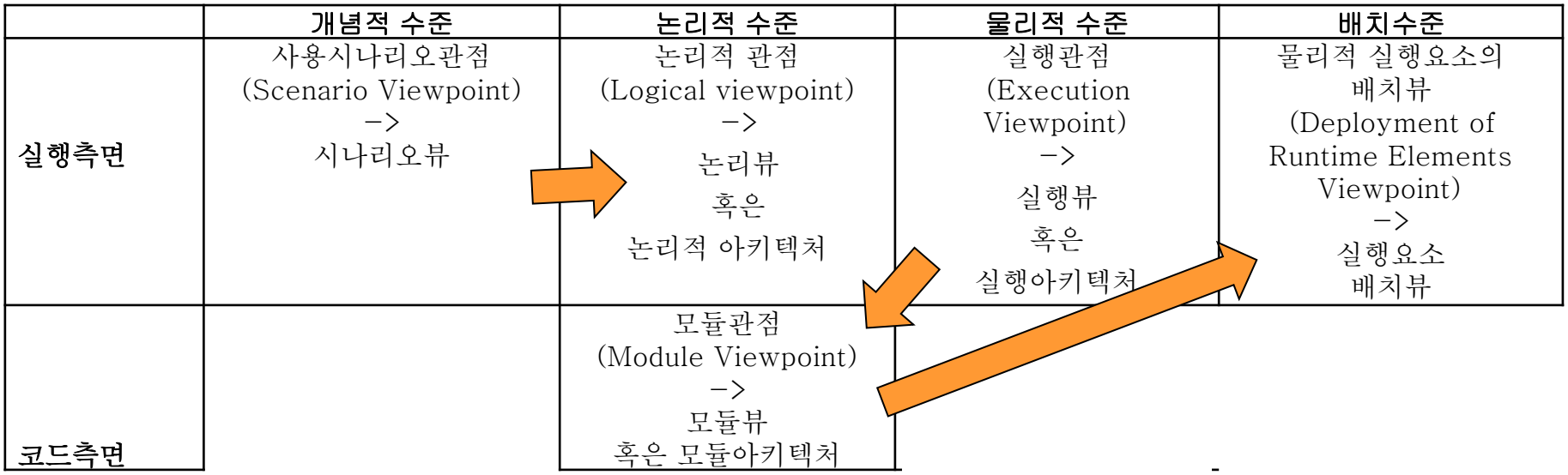


그림 8-5. 아키텍처 뷰의 설계 순서

Siemens

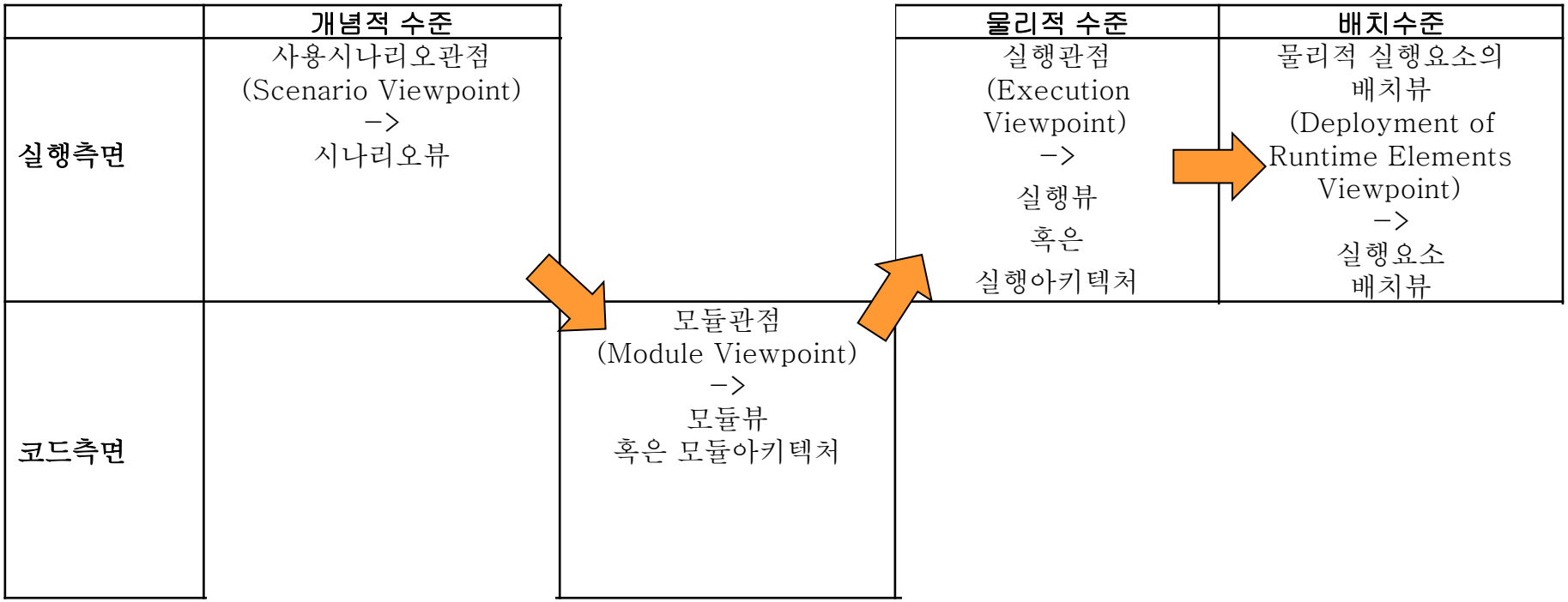

실행측면
코드측면
논리적 수준
논리적 관점
(Logical viewpoint)
->
논리뷰
혹은
논리적 아키텍처
물리적 수준
실행관점
(Execution
Viewpoint)
->
실행뷰
혹은
실행아키텍처
배치수준
물리적 실행요소의
배치뷰
(Deployment of
Runtime Elements
Viewpoint)
->
실행요소
배치뷰
모듈관점
(Module Viewpoint)
->
모듈뷰
혹은 모듈아키텍처
코드관점
(Code Viewpoint)
->
코드뷰

Kruchten



Execution viewpoint -> Process View
Module viewpoint -> Development View

Gomaa


개념적 수준
물리적 수준
배치수준
실행측면
사용시나리오관점
(Scenario Viewpoint)
->
시나리오뷰
실행관점
(Execution
Viewpoint)
->
실행뷰
혹은
실행아키텍처
물리적 실행요소의
배치뷰
(Deployment of
Runtime Elements
Viewpoint)
->
실행요소
배치뷰
코드측면
모듈관점
(Module Viewpoint)
->
모듈뷰
혹은 모듈아키텍처

8.3.2 다관점체계의 아키텍처 설계를 위한 두 가지 접근방법

- 순차적인 다관점 설계절차
- 병행적인 다관점 설계절차

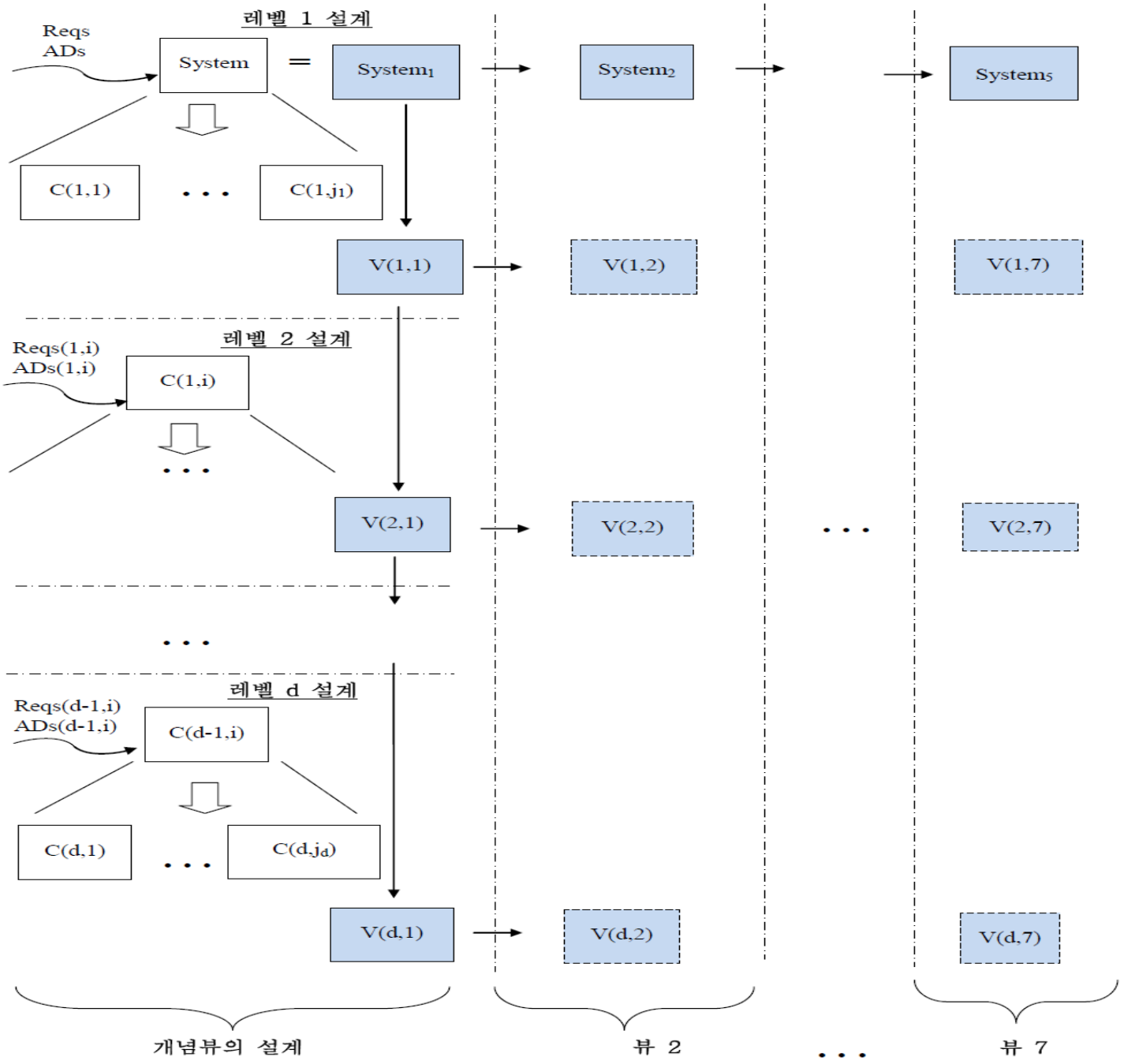


그림 8-6. 순차적인 다관점 설계절차

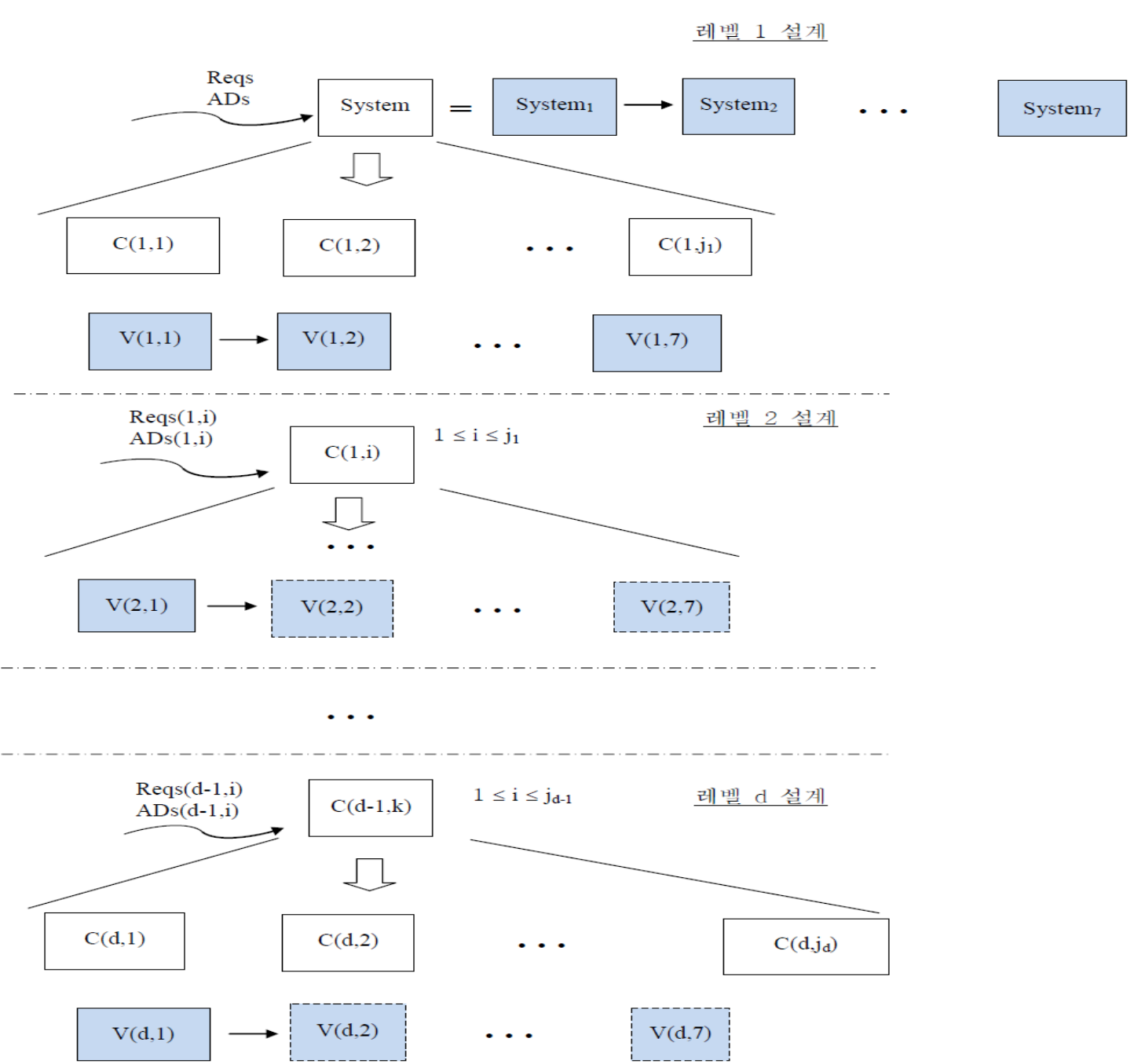


그림 8-7. 병행적인 다관점 설계절차

아키텍처 설계 절차의 정의

- What architecture style was selected?

What viewpoints were selected?

What is your architecture design process?

☛ Part of your project work !

- Define the architecture design procedure you will use.
- There is no completely correct or completely wrong procedure.

8.4 아키텍처 설계 결정표

설계결정	해결조건/방법	적용점
ADS1 (클라이언트-서버 아키텍처 스타일의 적용)	중앙의 데이터를 많은 사용자가 공유해야 한다.	E.1절의 개념뷰
ADS2 (load balancing)	J2EE는 서버의 수를 늘리고 서버들 사이에 부하를 균형 있게 만들 수 있는 메커니즘을 갖고 있다.	config7.xml 의 변수 _noservers ... 등
. . .	. . .	. . .

Questions?