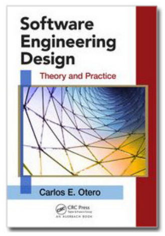
# Chapter 3: Principles of Software Architecture

## Session III: Designing the Architecture

*Software Engineering Design: Theory and Practice*
by Carlos E. Otero



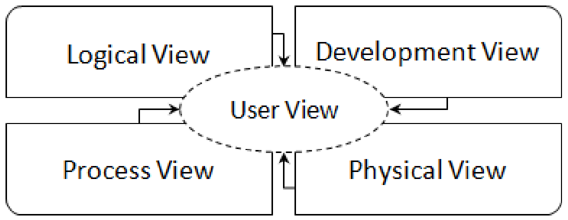
## SESSION'S AGENDA

- Designing the Architecture with the 4+1view model
- Designing the logical architecture
  - ✓ Example using a component diagram
- Designing the process architecture
  - ✓ Example using a communication diagram
- Designing the development architecture
  - ✓ Example using artifacts
- Designing the physical architecture
  - ✓ Example using deployment diagrams
- What's next..

## DESIGNING THE ARCHITECTURE

- As discussed before, designing architectures require the selection of particular perspectives for design that are appropriate for describing the system to be developed.
  - ✓ To this end, several model have been created that suggest popular views that are useful in the design of most systems.
  - ✓ These models propose addressing the system's architectural design from perspectives that are common to most software systems.
  - ✓ They provide a *systematic and disciplined* approach to architecture creation.
  - ✓ Two popular models are the 4+1 view model and Siemens' 4 views model.

- The 4+1 View Model
  - ✓ Concentrates on four main views:
    - Logical, process, development, and physical
  - ✓ To ensure consistency among all 4 views, the user's perspective is captured through several use cases, as part of the user view.
  - ✓ 4+1 is used as the basis for the popular rational unified process



## DESIGNING THE LOGICAL ARCHITECTURE

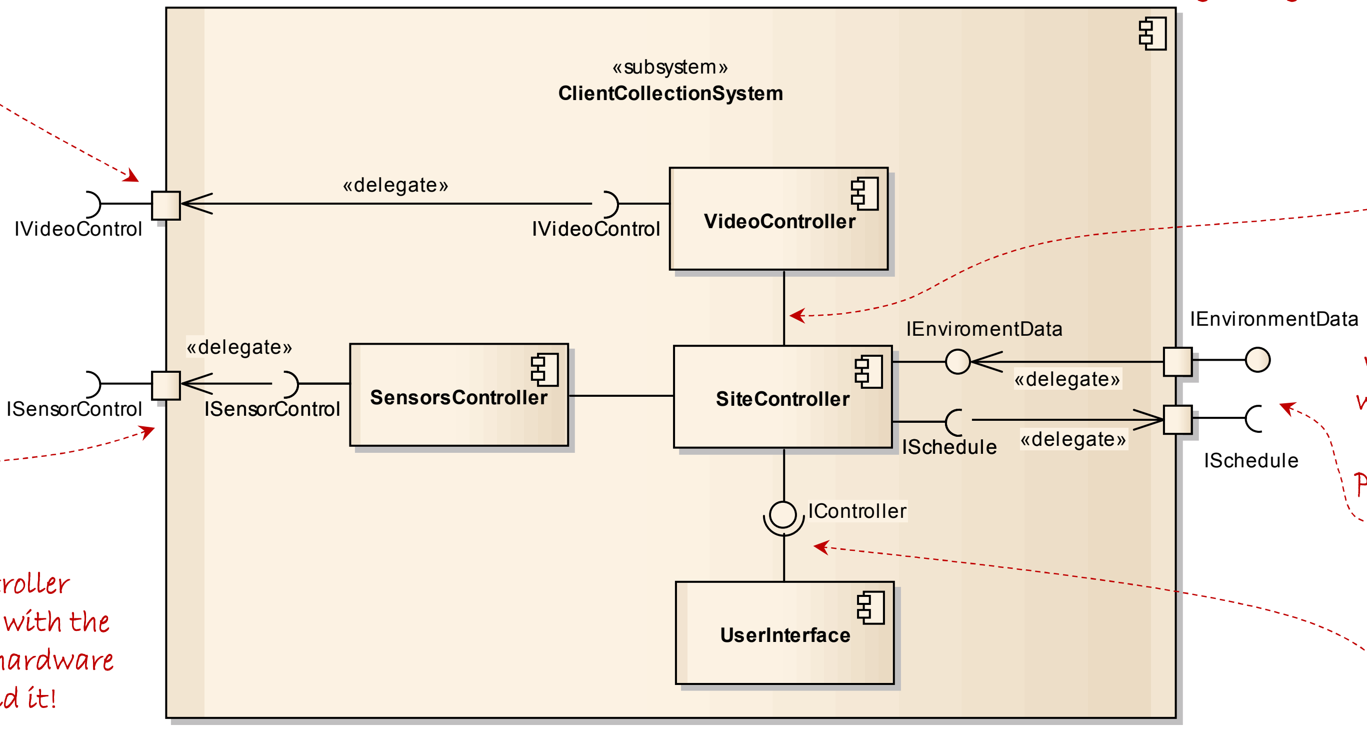
- The logical view is used to decompose systems into logical components that represent the structural integrity that supports functional and non-functional requirements. The logical view can be modeled using:
  - ✓ Component diagrams
  - ✓ Class diagrams
  - ✓ Box-and-line diagrams

- Using this view the static structure of the system, including components, interfaces, and their associations are modeled. This view is appropriate for:
  - ✓ Designing for portability
  - ✓ Designing maintainability (maintaining existing features)
  - ✓ Designing for extensibility (adding new features)
  - ✓ Designing for reusability
  - ✓ Resource allocation, project structuring and planning

# DESIGNING THE LOGICAL ARCHITECTURE

Video controller communicates with the actual video collection hardware to command it!

No interface defined! Direct association suggests that these components are tightly coupled, therefore making it more difficult to modify the system later on!



SiteController communicates with the outside world. It requires a schedule and provides collected data.

Sensor controller communicates with the actual Sensor hardware to command it!

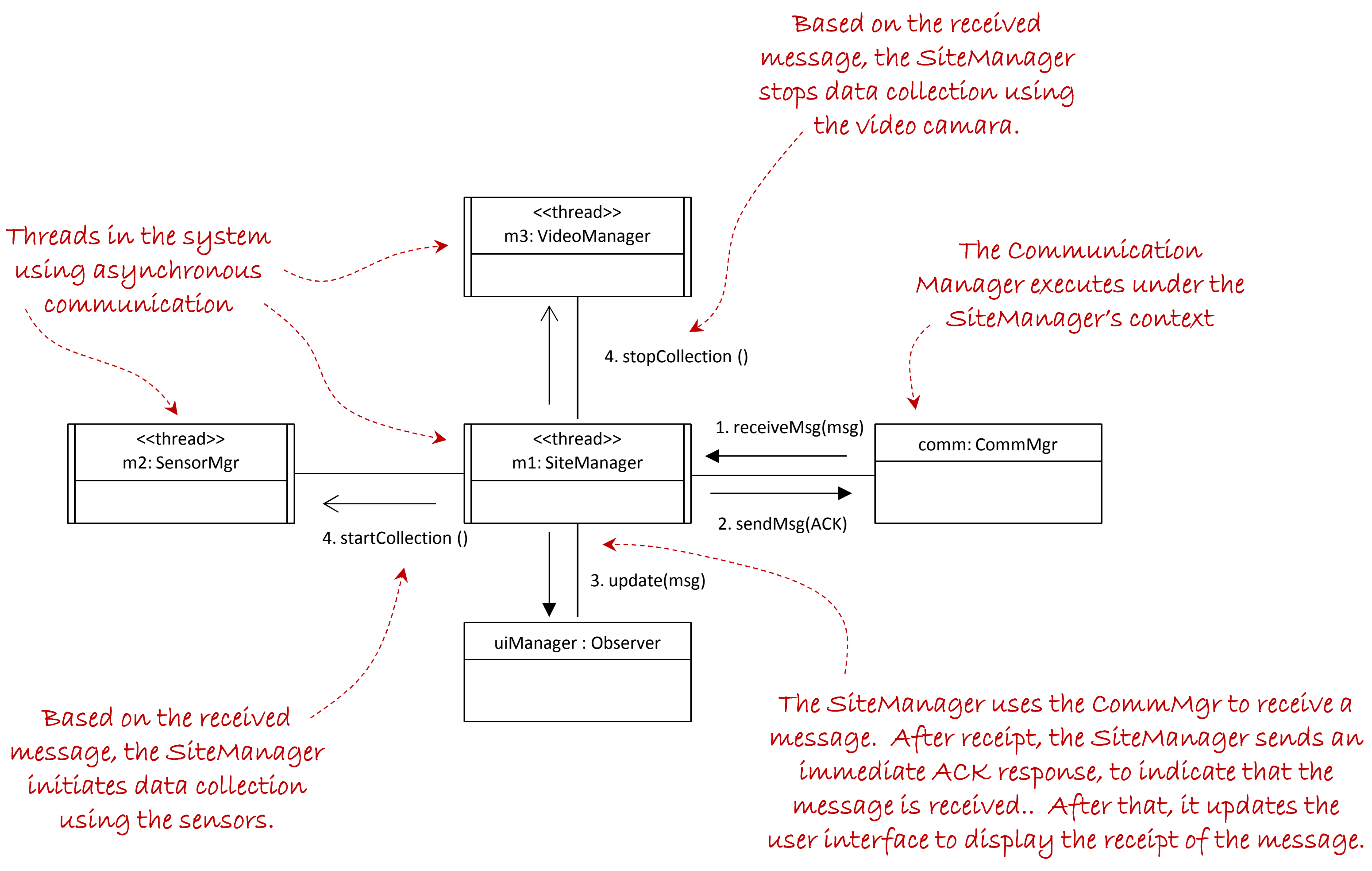
Important:
This process is recursive, i.e., each component can be further decomposed!

Well-defined interface! This allows different user interfaces to be exchangeable easier, e.g., from console-to graphical user interface-based. on!

## DESIGNING THE PROCESS ARCHITECTURE

- The process view is used to represent the dynamic or behavioral aspects of software systems, where the main units of analysis are processes and threads.
  - ✓ With this view, the system is decomposed into processes and threads to address design issues that deal with the dynamic flow of control between architectural elements.
- The process view can be modeled with UML using:
  - ✓ Sequence diagrams
  - ✓ Communication diagrams
- This view is appropriate for evaluating important characteristics of the system, such as concurrency, fault tolerance, and the system's integrity. Specific quality attributes that can be evaluated with this view include:
  - ✓ Performance
  - ✓ Availability

## DESIGNING THE PROCESS ARCHITECTURE



Based on the received message, the SiteManager stops data collection using the video camara.
Threads in the system using asynchronous communication
<<thread>> m3: VideoManager
4. stopCollection ()
The Communication Manager executes under the SiteManager's context
<<thread>> m2: SensorMgr
<<thread>> m1: SiteManager
1. receiveMsg(msg)
comm: CommMgr
2. sendMsg(ACK)
4. startCollection ()
3. update(msg)
uiManager : Observer
Based on the received message, the SiteManager initiates data collection using the sensors.
The SiteManager uses the CommMgr to receive a message. After receipt, the SiteManager sends an immediate ACK response, to indicate that the message is received.. After that, it updates the user interface to display the receipt of the message.

## DESIGNING THE DEVELOPMENT ARCHITECTURE

- The development view of software systems represents the software development configuration aspects of the software system.
  - ✓ The main units of decomposition are actual physical files and directories.

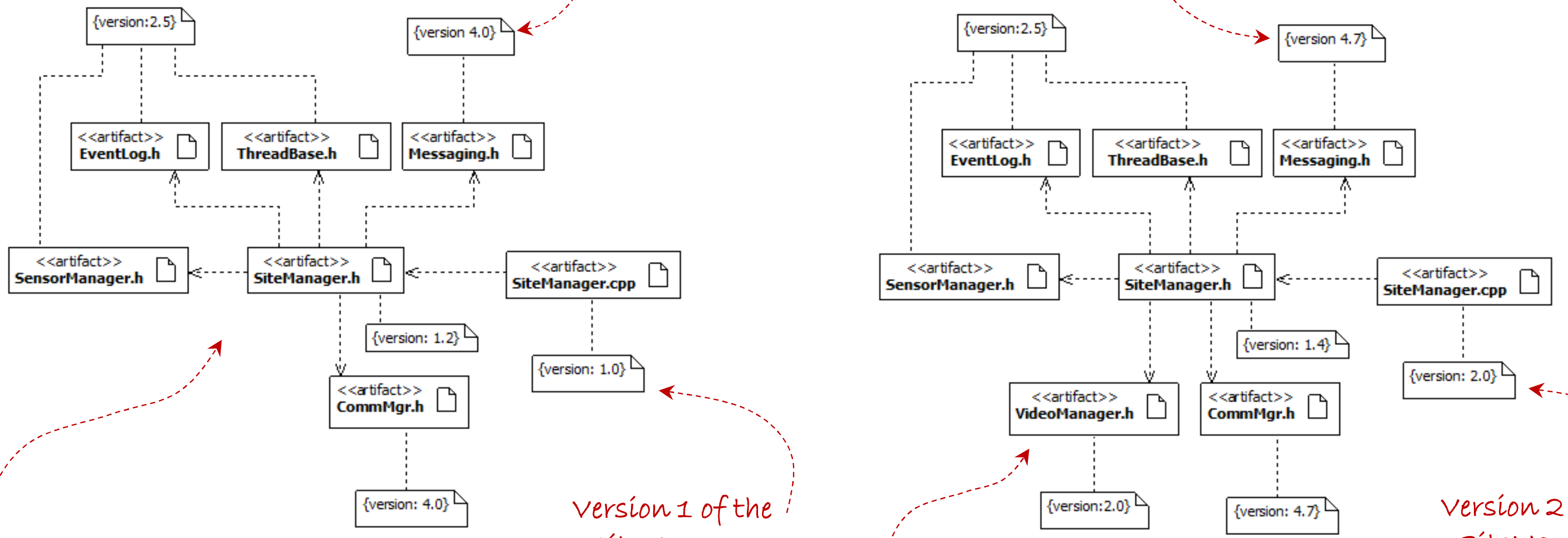
- This view can be used to analyze the system from the perspective of how logical components map to physical files and directories.
  - ✓ These analyses can be employed to address concerns that deal with:
    - Ease of development
    - Reusability
    - Compile-time dependencies of the system
    - Configuration management
    - Allocation of work to teams
    - Cost evaluation and planning
    - Project monitoring
    - Portability

# DESIGNING THE DEVELOPMENT ARCHITECTURE

Version 1.0

Notice how versions change with the addition of new functionality!

Version 2.0



Version 1 of the SiteManager

Version 2 of the SiteManager

Version 1.0 is used by a specific customer that does not require video capture capabilities.

Version 2.0 is used by a specific customer that requires video capture capabilities.

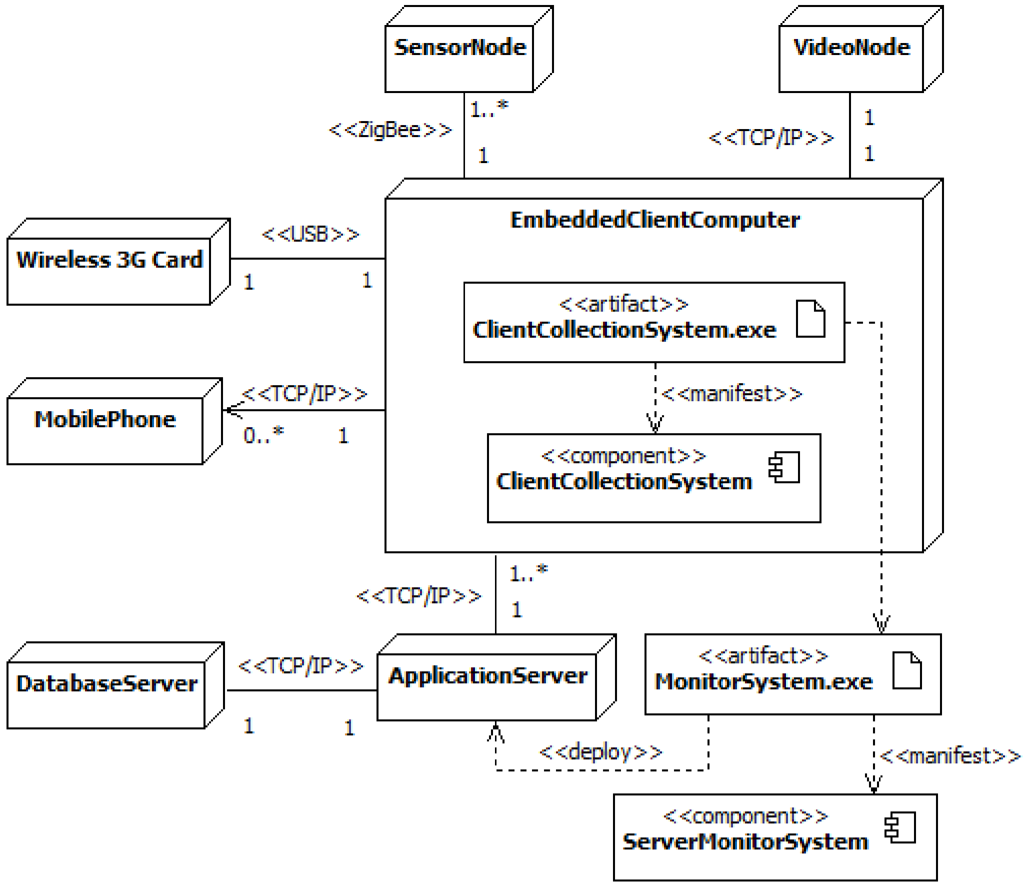
## DESIGNING THE PHYSICAL ARCHITECTURE

- The physical view represents the deployment aspects of software systems.
  - ✓ The main elements of analysis are nodes, connections between nodes, and the mapping of artifacts to these nodes.
  - ✓ This view can be used to model the run-time dependencies of the system.

- The physical view can be modeled in UML using:
  - ✓ Deployment diagrams

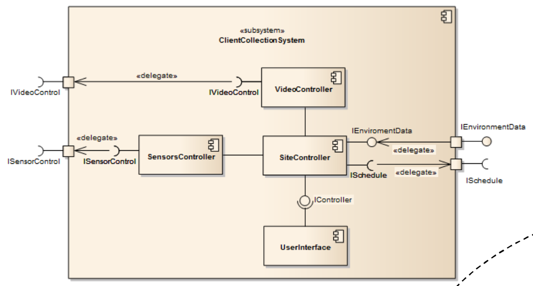
- The physical view is appropriate to evaluate the system’s:
  - ✓ Availability
    - For example, specifying the need for redundant nodes to increase the system’s availability
  - ✓ Performance
    - For example, required processor speed, bandwidth, etc. to meet performance requirements.
  - ✓ Scalability

## Designing the Physical Architecture

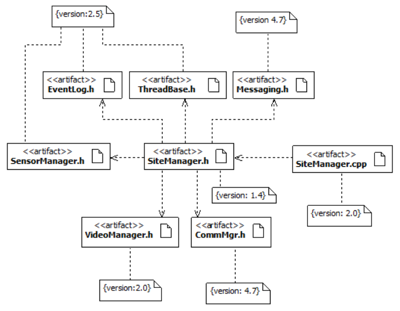


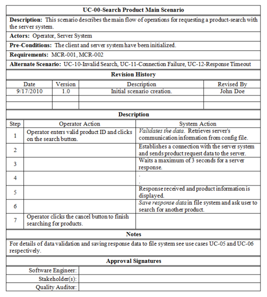
# EXAMPLE OF PARTIAL ARCHITECTURE OF CLIENT COLLECTION SYSTEM

Logical View



Development View





| UC-00-Search Product Main Scenario | | | |
|---|---|---|---|
| **Description:** This scenario describes the main flow of operations for requesting a product-search with the server system. | | | |
| **Actors:** Operator, Server System | | | |
| **Pre-Conditions:** The client and server system have been initialized. | | | |
| **Requirements:** MCR-001, MCR-002 | | | |
| **Alternate Scenario:** UC-10-Invalid Search, UC-11-Connection Failure, UC-12-Response Timeout | | | |

**Revision History**

| Date | Version | Description | Revised By |
|---|---|---|---|
| 9/17/2010 | 1.0 | Initial scenario creation. | John Doe |
| | | | |
| | | | |

**Description**

| Step | Operator Action | System Action |
|---|---|---|
| 1 | Operator enters valid product ID and clicks on the search button. | *Validates the data.* Retrieves server's communication information from config file. |
| 2 | | Establishes a connection with the server system and sends product request data to the server. |
| 3 | | Waits a maximum of 3 seconds for a server response. |
| 4 | | . . |
| 5 | | Response received and product information is displayed. |
| 6 | | *Save response data* in file system and ask user to search for another product. |
| 7 | Operator clicks the cancel button to finish searching for products. | |

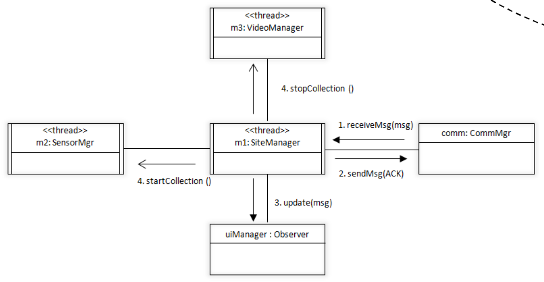
**Notes**

For details of data validation and saving response data to file system see use cases UC-05 and UC-06 respectively.

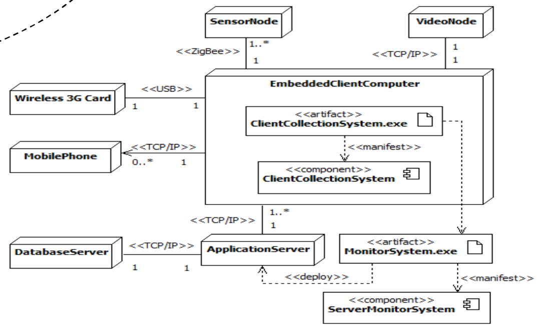
**Approval Signatures**

| | |
|---|---|
| Software Engineer: | |
| Stakeholder(s): | |
| Quality Auditor: | |

Process View



Physical View



## ONE LAST NOTE…

- Do NOT make the assumption that a particular UML classifier belongs exclusively to a particular view of the 4+1 model!!
  - ✓ Although some classifiers are certainly "*the main characters*" in particular views, they can serve as "supporting characters in other views."
    - For example, components are used mostly in the logical view, BUT,
    - They can also be used in the physical view to model how they are manifested by artifacts!

- UML Artifacts are tricky, because they can be used as the dominant design unit (i.e., "main character") in both development and physical view:
  - ✓ In the physical view, they model ***deployment artifacts***.
    - These are artifacts necessary to form an executable system (e.g., .dll, .jar, .class, .ini file, etc.)
    - These are essential to model the run-time dependencies of the system (e.g., .NET 4.1)
  - ✓ In the development view, they model ***work product artifacts***
    - These are artifacts used to create deployment artifacts, i.e., they are necessary to build the software (e.g., source code files [.h, .cpp, .java], data files])
    - These artifacts do not directly exist in the physically deployed system; they are the work products of development used to build the system.
    - These are essential to model the compile-time dependencies to build the system as well as versioning and configuration management of the system.

  - ✓ Keep in mind that when using *UML artifacts*, modeling occurs in the physical dimension, which can be addressed in both *development* and *physical* views of the 4+1 model. Artifacts do not exist in the logical dimension, as opposed to other UML classifiers, such as components.

## WHAT'S NEXT…

- In this session, we presented the 4+1 view model for designing software architectures, which includes
  - Logical view
  - Development View
  - Process View
  - Physical View
  - User View

- In the next module, we'll delve deep into popular patterns and styles used to model logical architectures; these are presented for the following types of systems:
  - ✓ Data-centered
  - ✓ Data flow
  - ✓ Distributed
  - ✓ Interactive
  - ✓ Hierarchical