IBM Architect Profession

Functional Aspect

Version 6.0

Architectural Thinking

Module outline

Introduction and Objectives

The functional aspect of IT architecture

Component modeling

Building a component model

- Identifying components
- Specifying components
- Implementing components

Summary and references

Learning objectives

At the end of this module, you should be able to:

- Describe what is meant by the functional aspect of IT architecture and how it can be modeled in a variety of ways
  - Component modeling
  - Data modeling
  - Service modeling
- Describe in detail one way of modeling the functional aspect through component modeling
  - Describe what makes a “good” component
  - Describe some of the key principles used in building component models

Module outline

Introduction and objectives

The functional aspect of IT architecture

Component modeling

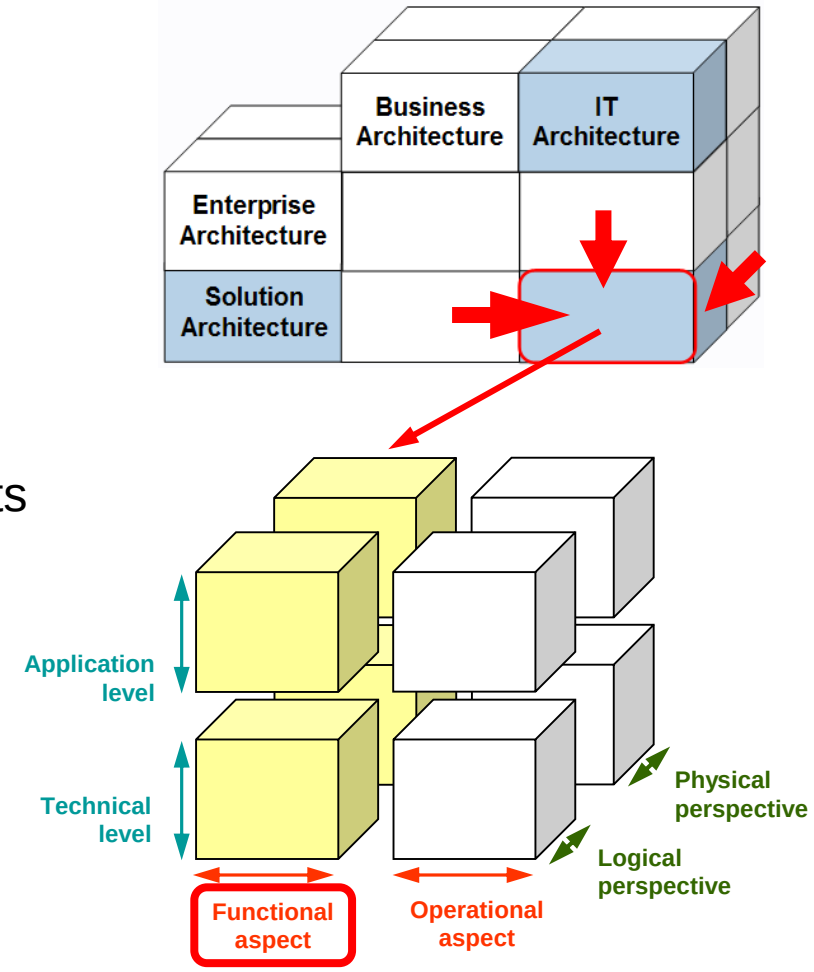
Building a component model

- Identifying components
- Specifying components
- Implementing components

Summary and references

What is the functional aspect of IT architecture?

- It is one of the basic viewpoints of an IT architecture.
- It describes and captures the system's functional behaviour.
- It can be described in a variety of ways:
  - Component modeling
  - Data modeling
  - Service modeling
- It can be created by Enterprise Architects "in outline" before a program is funded, and/or "completely" by IT Architects within programs.
- It is influenced by the business architecture, solution requirements.
- It can be influenced by the enterprise architecture, architecture guidelines and constraints.



The functional aspect of an IT system can be modeled in a variety of ways

Component modeling

- Reasoning about the structure and behavior of the system as a set of interrelated and interacting *lumps of functionality*: components
  - Components' capabilities are modeled in terms of what they can do, the activities they can perform, and the information they are responsible for.
  - Components work together through the exchange of messages sent and received via interfaces.

Data modeling

- Reasoning about the structure and relationships between the system's information entities
  - Data modeling is a key part of information architecture, which also considers the informational parts of the other three basic viewpoints such as requirements, operational, and viability aspects

Service modeling

- Reasoning about the structure and behavior of the system as a set of interrelated and interacting offers and requests for service
  - While often implied, that is, abstracted away, in the service model, services are offered and requested by components; a service may be thought of as a *component's interface with a contract.

This module focuses on modeling the functional aspect via component modeling.

- Data modeling is addressed in the "Information Architecture" module.
- Service modeling is addressed in *Service Oriented Modeling and Architecture* (SOMA).

Data modeling and component modeling are strongly interrelated

Information must be considered across all four basic viewpoints:

- Requirements, functional, operational, and validation aspects

Data modeling provides the required informational insights in the functional aspect.

- Operational modeling's data deployment units enable modeling of data distribution and management in the operational aspect.

Data modeling is separate from, but significantly related to, component modeling.

- Components must be able to:
  - Manage, that is, create, read, update, delete persistent data
  - Manipulate transient data

Component models and data models must be developed alongside each other, identifying relationships and behaviours between data, whether transient or persistent.

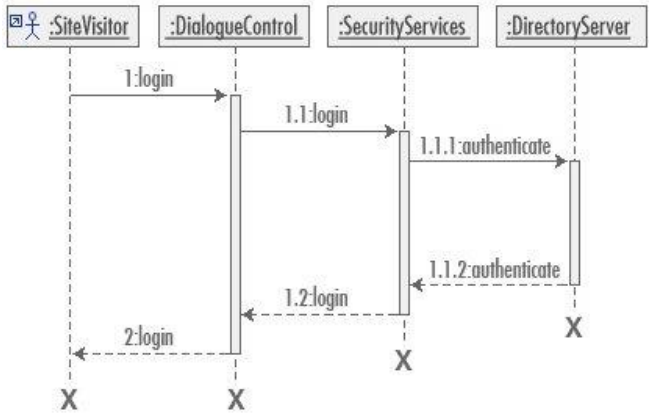
- Whereas data models document the relationships between data entities, component models show how components own, use, and request access or changes to these data entities.

Data modeling is a significant part of information architecture.

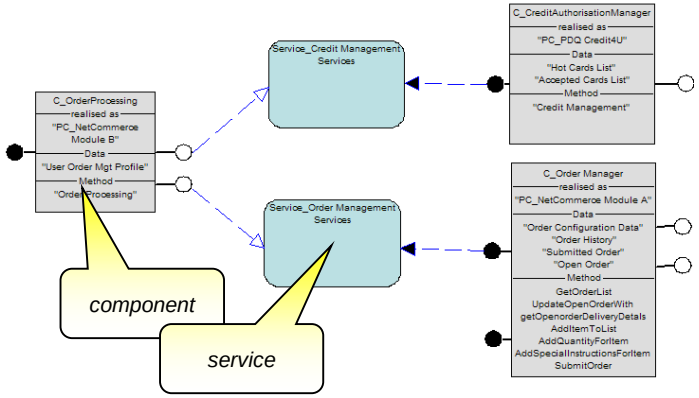
- Addressed in a separate lecture

The functional aspect can also be modeled using services

- Components *use* each other, offering and requesting information via messages that flow between interfaces.
- Rather than focusing on *active* components, it may be appropriate to focus on what the components' interfaces offer, or request.
- Reasoning about *active* interfaces elevates them to the notion of services.
  - Thinking about what is offered and requested, instead of what is doing the offering and requesting, is at the heart of *SOA.*
- But *service* is not a synonym for *component*.
  - Component-centric: Components offer and request services
  - Service-centric: Services are satisfied by components, which, if necessary, request services from other components.



Component-centric view



Service-centric view

Module outline

Introduction and objectives

The functional aspect of IT architecture

Component modeling

Building a component model

- Identifying components
- Specifying components
- Implementing components

Summary and references



What is component modeling?

- Component modeling is the technique used to identify and specify components and their interfaces.
- Component modeling is primarily concerned with the functional aspect of IT architecture.
- The technique is complementary to, and sits alongside, that of operational modeling which incorporates the non functional aspects of IT architecture.

Why build a component model?

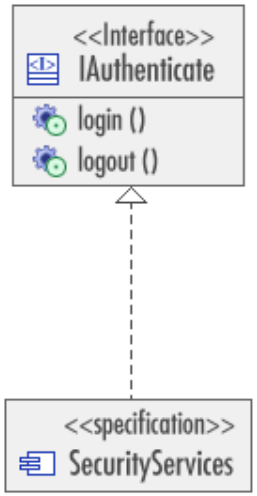
- Bridges the gap between the functional requirements like the *what* and the *how*, the solution.
- Helps us visualize and understand the system. Today's complex, software-intensive systems cannot be comprehended without models.
- Specifies the structure and behavior of the system and documents decisions made.
- Provides the necessary input for the operational model to make proper placement decisions about the components' various aspects:
  - Where should the components run?
  - Where should the components' data be?
  - Where should the components be installed?

The component is a primary concept used for describing the functional aspect of the IT architecture

- A component is a modular unit of functionality.
- A component makes its functionality and state available through one or more interfaces.

"State" – the information managed by the component

- Examples of components are:
  - At the *application level* of the component model:
    - Business processing components such as the Customer Processing component
    - Business service components such as the Account Manager component
  - At the *technical level* of the component model:
    - Technical components, middleware such as messaging or transaction software
    - System software components such as an operating system
    - Hardware components such as an encryption device
- A component is not just a programming-level concept such as an Oracle® EJB or .NET component.

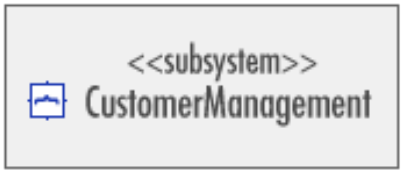


Components may be grouped into subsystems

Subsystem: An interesting group of model elements of any number or model.

- Security subsystem, printing subsystem, order management subsystem, data distribution subsystem.
- Model elements can be simultaneously in any number of subsystems.
- A subsystem has no intrinsic capabilities; it is simply an identifier.

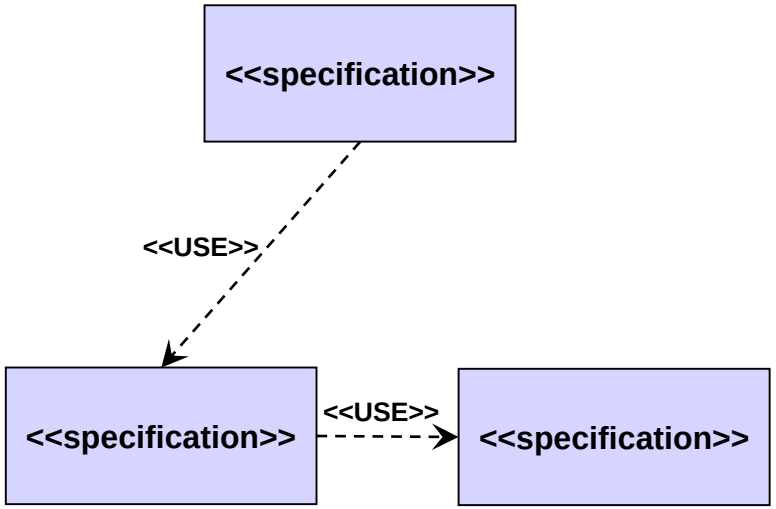
Subsystems are commonly found in component models, where they enable the modeler to:



- Label major parts of an IT system, such as the *accounting* subsystem.
- Identify a commercial software product or package, such as CICS or SAP.
- Allocate work to development groups such as platform teams.

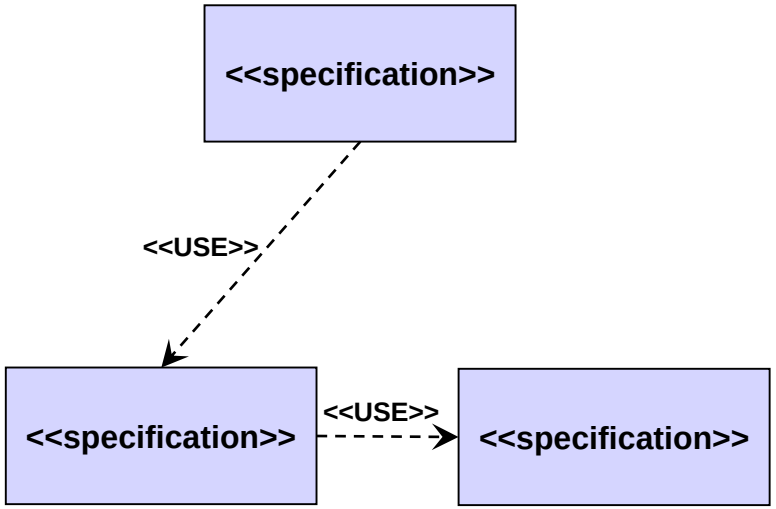
In the context of component modeling, since it is *just a label*, a subsystem does not have its own interfaces. Interfaces *to the subsystem* are actually interfaces on components in the subsystem.

Component modeling enables the functional aspect to focus (1 of 4)

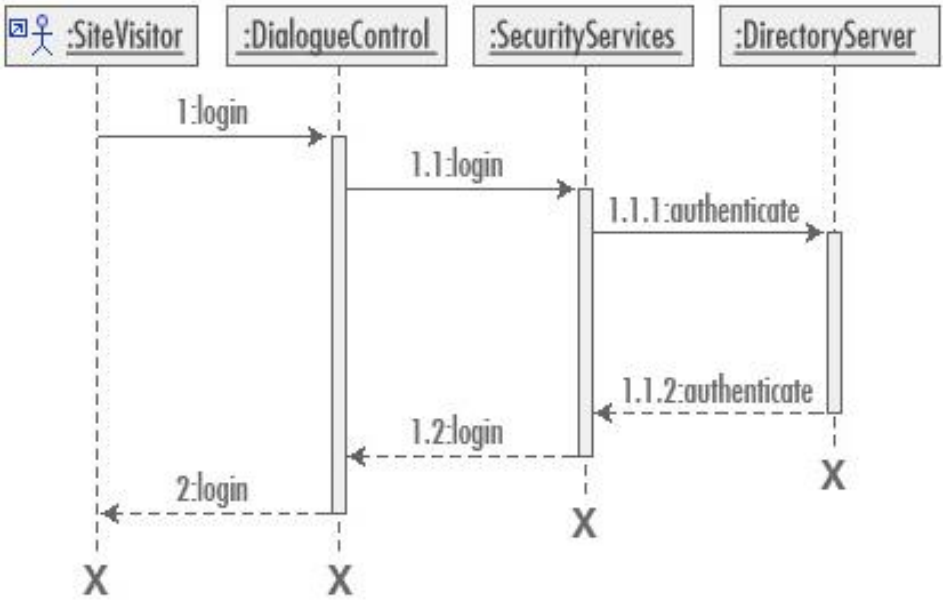


On the system's structure, as described in a component relationship diagram

Component modeling enables the functional aspect to focus (2 of 4)

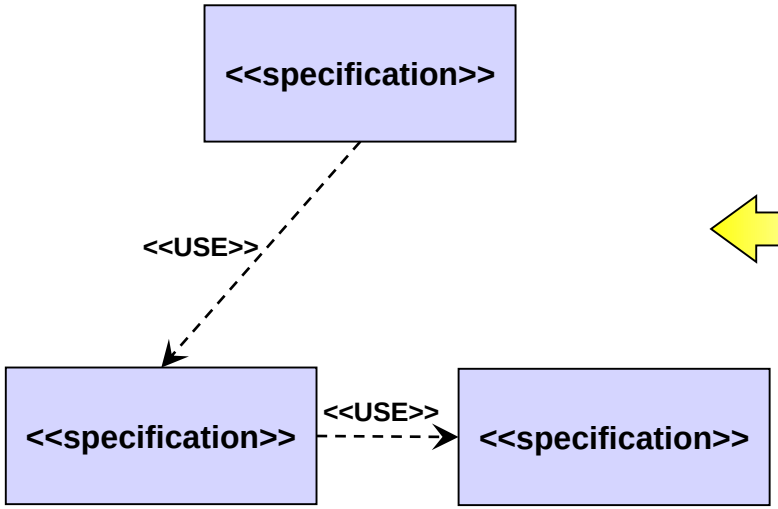


On the system's structure, as described in a component relationship diagram.

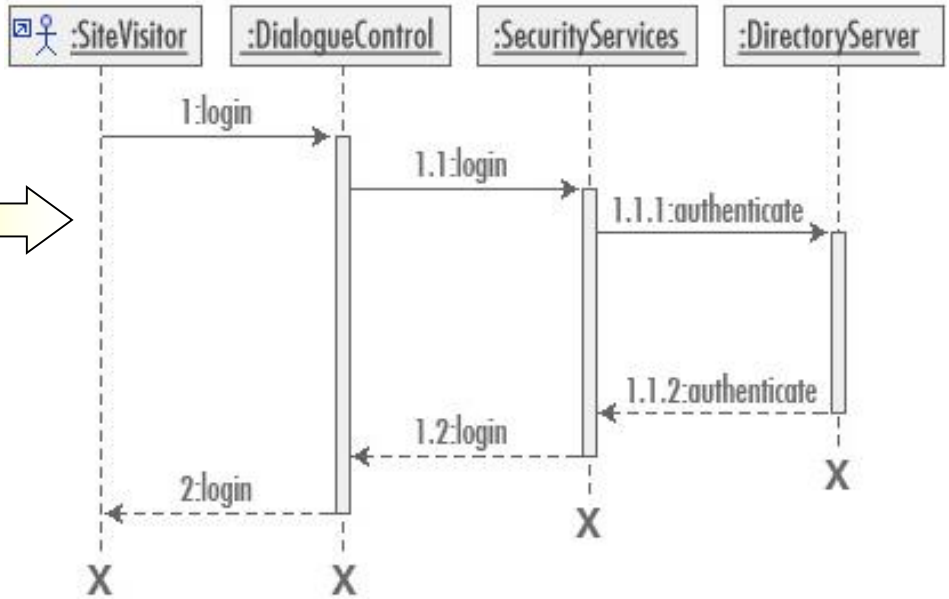


As well as, its behavior, as shown in a component interaction diagram.

Component modeling enables the functional aspect to focus (3 of 4)



On the system's structure, as described in a component relationship diagram



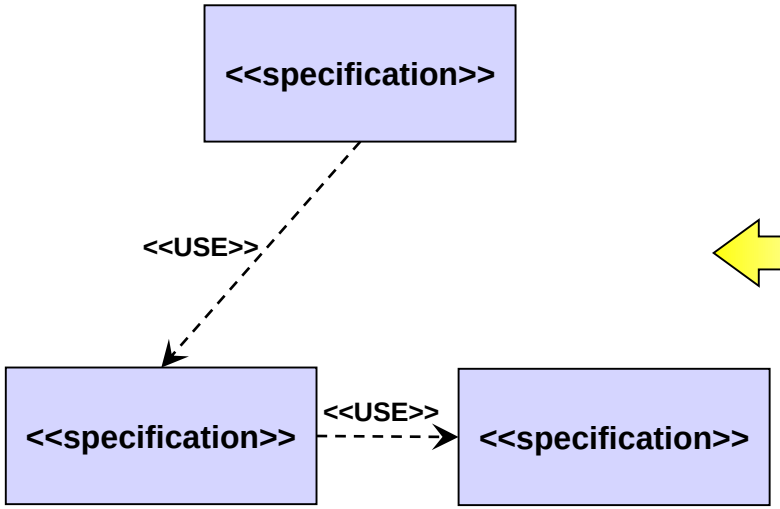
As well as, its behavior, as shown in a component interaction diagram

Other types of relationship diagrams may, for example, show how technical components support ("use") application components.

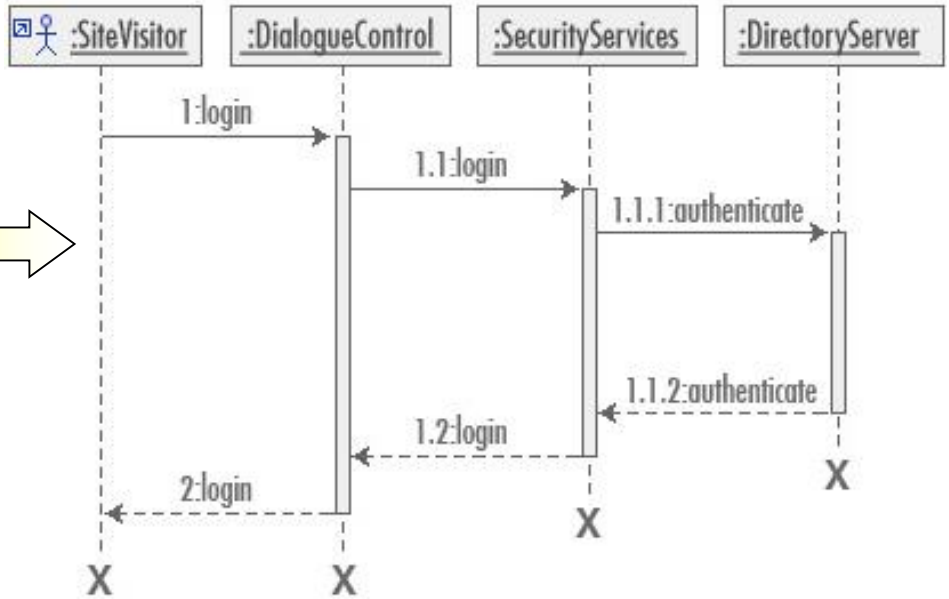
The relationship diagram may summarize the components' interactions, in which case the relationships are modeled as component connectors.

This type of relationship diagram can be called a component structure diagram.

Component modeling enables the functional aspect to focus (4 of 4)



On the system's structure, as described in a component relationship diagram



As well as, its behavior, as shown in a component interaction diagram

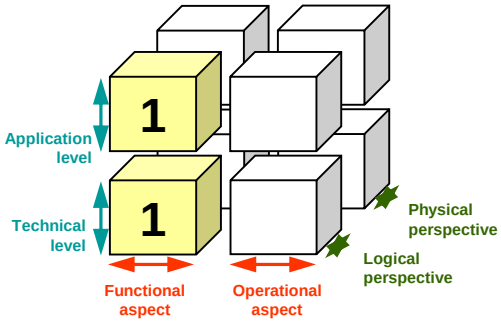
The relationship diagram may summarize the components' interactions, in which case the relationships are modeled as component connectors.

Component modeling makes extensive use of the Unified Modeling Language

Unified Modeling Language (UML) is:

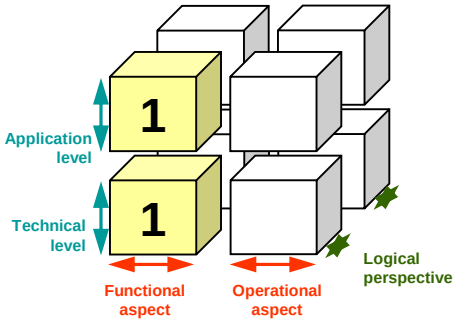
- A graphical notation, supported by a single metamodel, for describing software systems.
- An open standard controlled by the Object Management Group (OMG); Version 2 was released in July 2005.
- Based on a unification of the modeling work of Grady Booch, Jim Rumbaugh, and Ivar Jacobson.
- Supported by a number of tool providers.

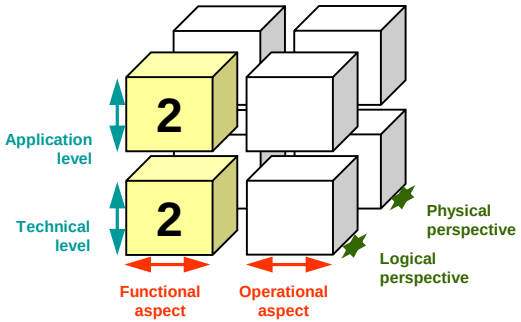
The component modeling technique consists of three steps
(1 of 3)



- **Component identification:**
  a. Partition into subsystems and components and assign responsibilities
  b. Review architectural patterns, reference architectures, and reusable assets
  c. Structure ensuring loose coupling, high cohesion, and so on

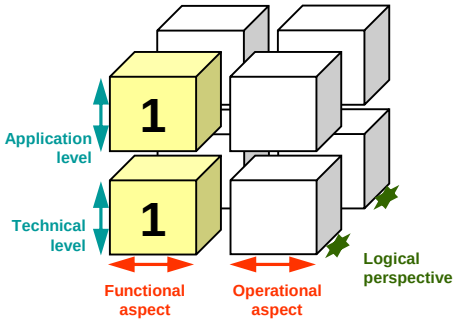
The component modeling technique consists of three steps
(2 of 3)



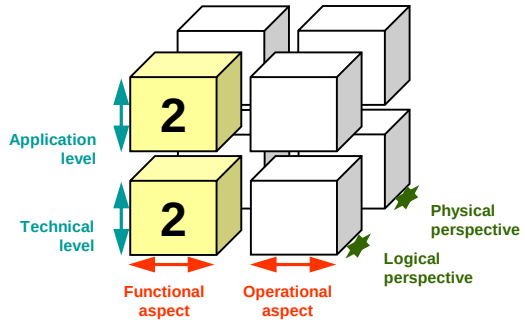


- **Component identification:**
  - a. Partition into subsystems and components and assign responsibilities
  - b. Review architectural patterns, reference architectures, and reusable assets
  - c. Structure ensuring loose coupling, high cohesion, and so on
- **Component specification:**
  - a. Specify interfaces
  - b. Specify operations and signatures
  - c. Specify pre- and post-conditions

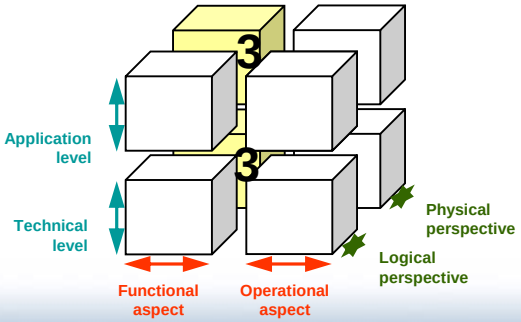
The component modeling technique consists of three steps
(3 of 3)



- **Component identification:**
  - a. Partition into subsystems and components and assign responsibilities
  - b. Review architectural patterns, reference architectures, and reusable assets
  - c. Structure ensuring loose coupling, high cohesion, and so on

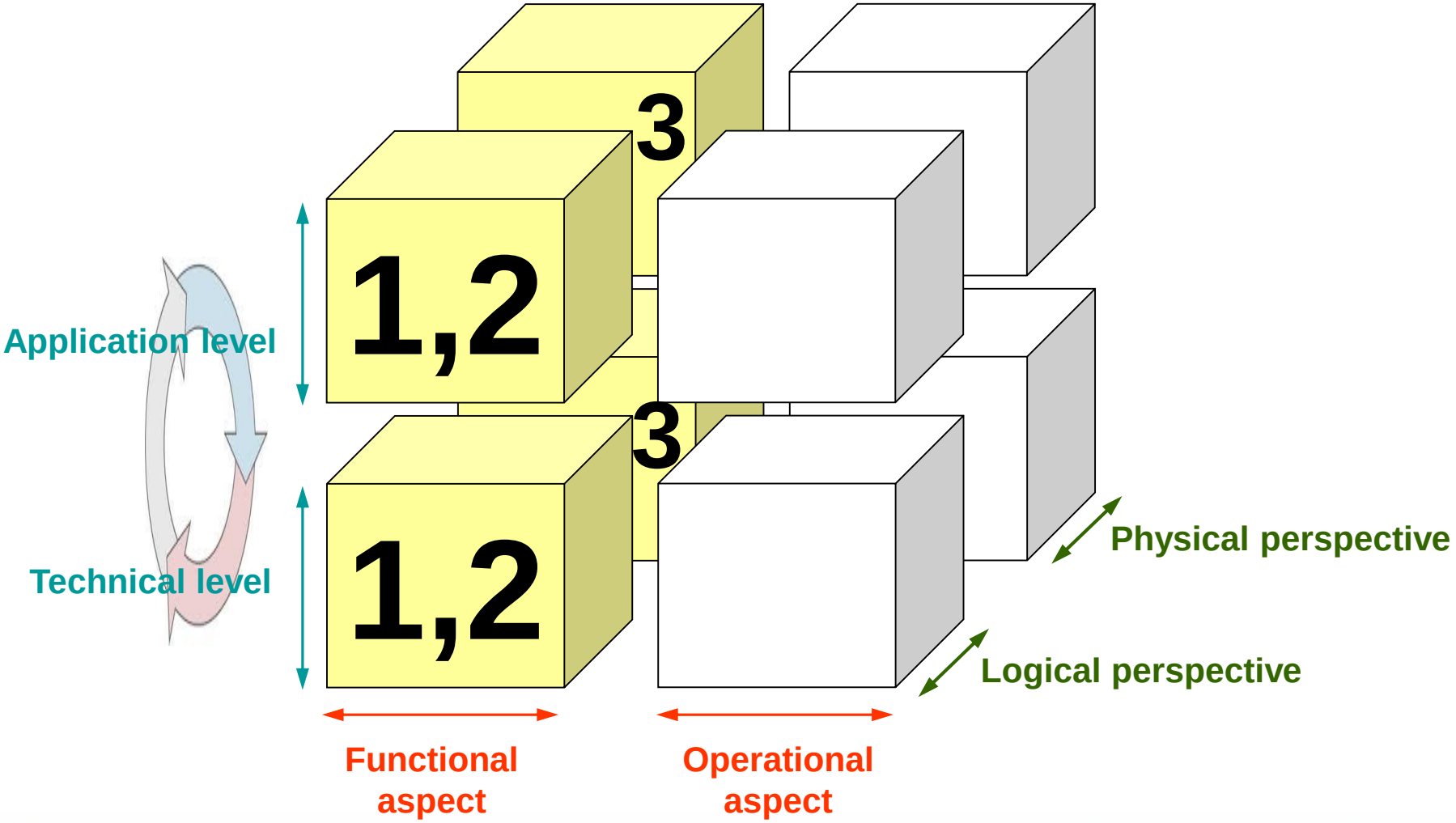


- **Component specification:**
  - a. Specify interfaces
  - b. Specify operations and signatures
  - c. Specify pre- and post-conditions

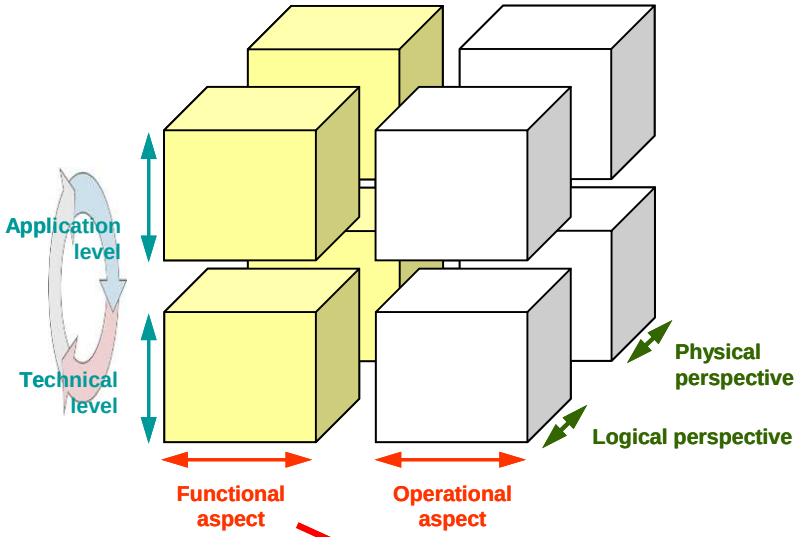


- **Component implementation:**
  - a. Identify products and packages
  - b. Define implementation approach

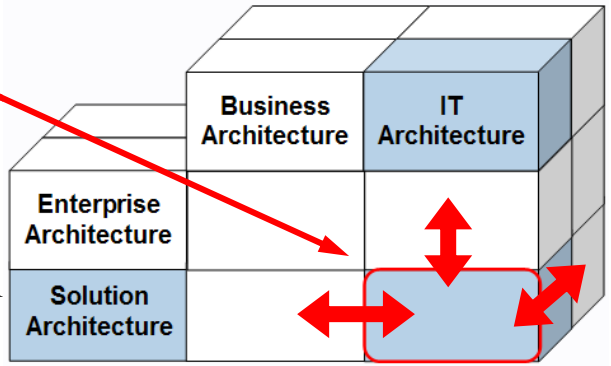
They are performed iteratively and are always driven from the business architecture, based on the enterprise architecture (1 of 2)



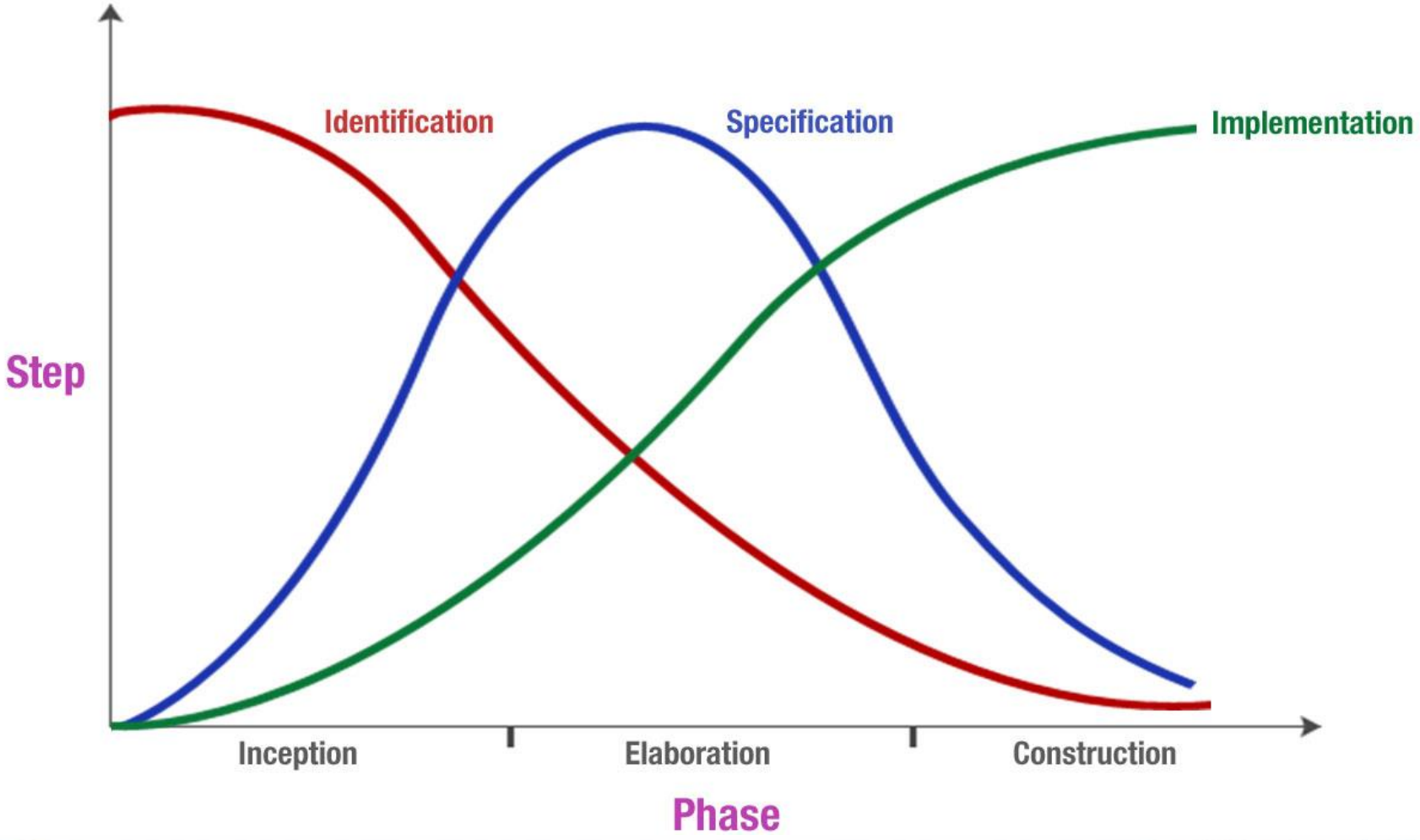
They are performed iteratively and are always driven from the business architecture, based on the enterprise architecture (2 of 2)



And don't forget that the IT architecture's component model can influence "back up" to the BA and EA.

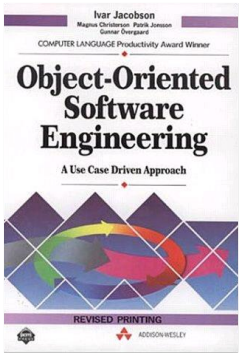


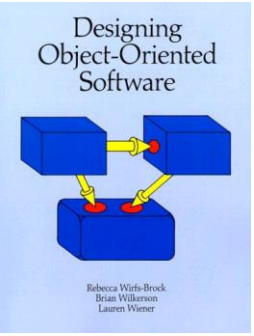
Each step is applied, to varying degrees, at different points in the delivery process

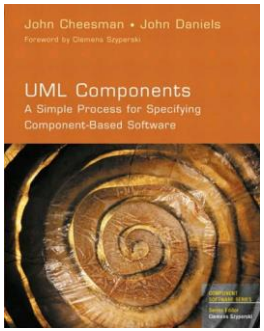


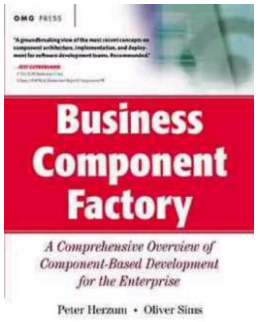
The technique is based on industry best practices

- Use-case-driven design, Jacobson
- Responsibility-driven design, Wirfs-Brock
- Design by contract, Meyer
- Cohesion and coupling, Yourdon, Page-Jones, and others
- Component architecture, Cheesman and Daniels
- Component granularity, Herzum and Sims





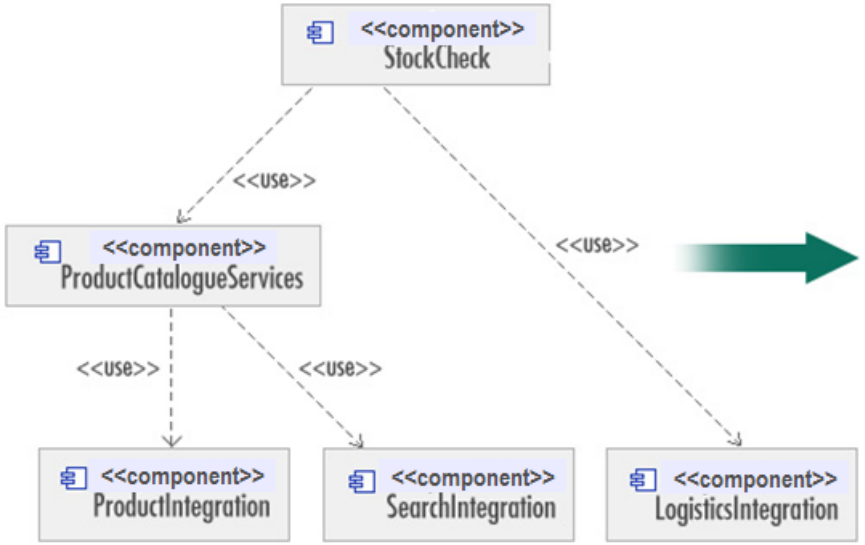




The component model is used as input into a number of activities

In application development or customization:

- Work allocation
- Version control
- Design strategy
- Reuse
- Testing
- Project management
- Product or Package selection



In the operational aspect:

- The *things* that have to be deployed

In operations:

- Systems management and so on

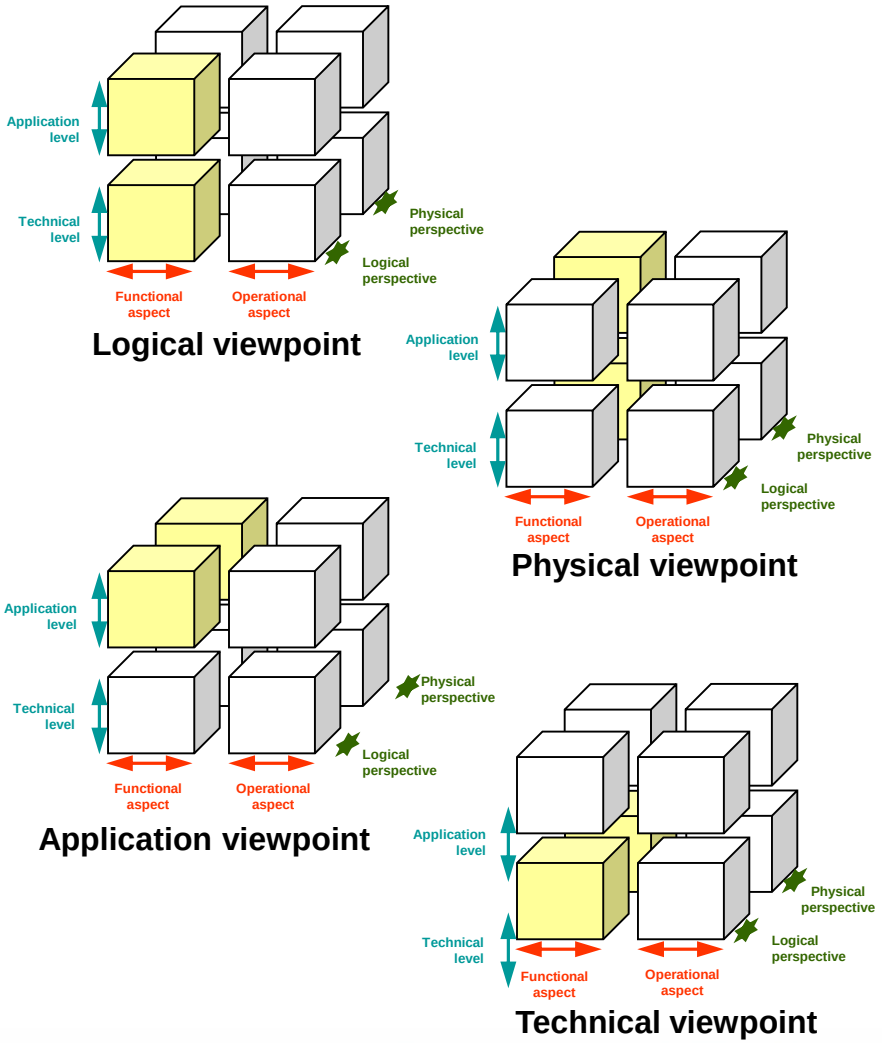
Component models have two major cross-cutting viewpoints
(1 of 3)

A logical or physical viewpoint

- Logical: Represents concepts in the domain being studied and is independent of the physical aspects of the software or hardware
- Physical: Represents actual software or hardware technology

An application or technical viewpoint, sometimes called *levels*

- Application: Components and their interrelationships that provide the application-dependent behavior of the IT system
- Technical: Components and their interrelationships that provide the application-independent behavior of the IT system



Architectural Thinking

Component models have two major cross-cutting viewpoints
(2 of 3)

What do you think?

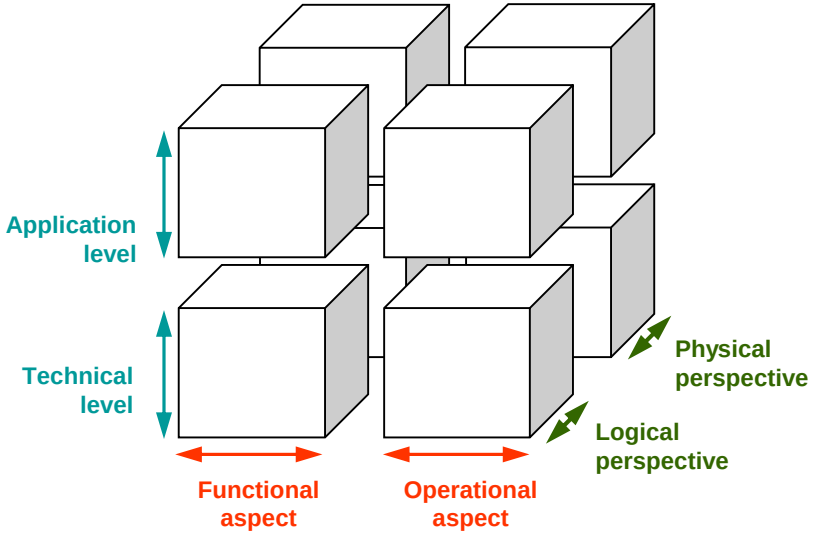
The IBM® WebSphere® Message Broker, which implements the message bus component, would be an example of what type of component?

A. Logical application

B. Physical application

C. Logical technical

D. Physical technical



Component models have two major cross-cutting viewpoints
(2 of 3)

Answer

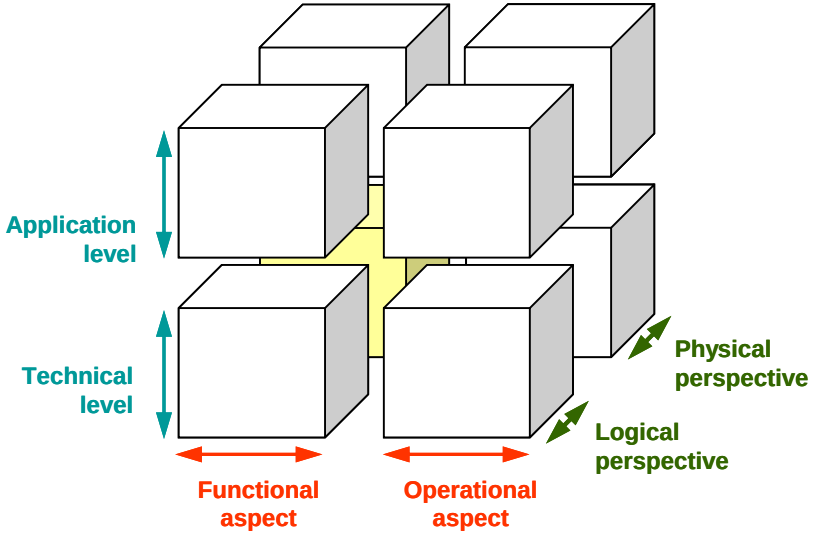
The IBM® WebSphere® Message Broker, which implements the message bus component, would be an example of what type of component?

A. Logical application

B. Physical application

C. Logical technical

D. Physical technical



Component models have two major cross-cutting viewpoints
(3 of 3)

What do you think?

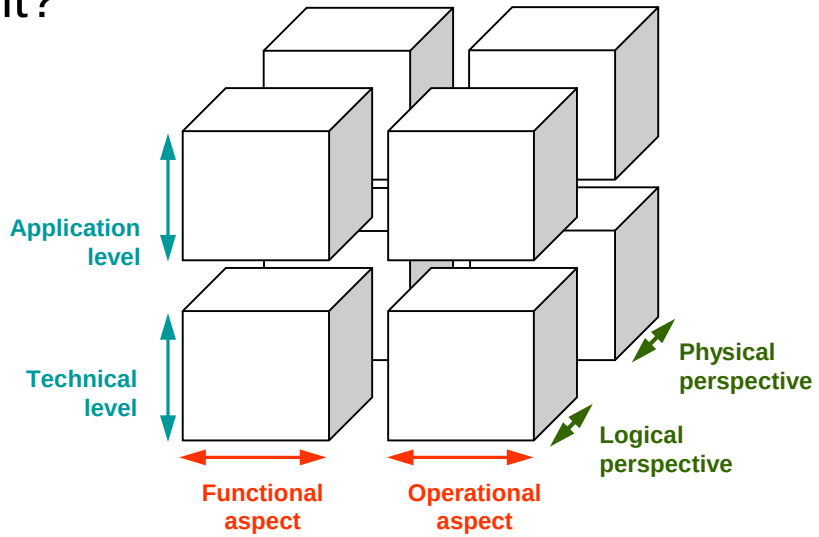
A Customer Relationship Manager having the responsibility of managing various aspects of a customer's data as well as the customer's relationships and communication channels with an organization would be an example of what type of component?

A. Logical application

B. Physical application

C. Logical technical

D. Physical technical



Component models have two major cross-cutting viewpoints
(3 of 3)

Answer

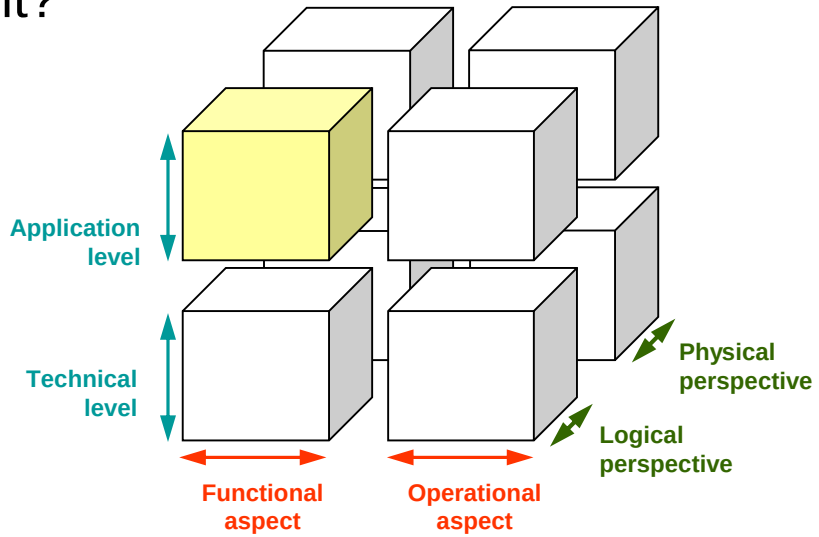
A Customer Relationship Manager having the responsibility of managing various aspects of a customer's data as well as the customer's relationships and communication channels with an organization would be an example of what type of component?

A. Logical application

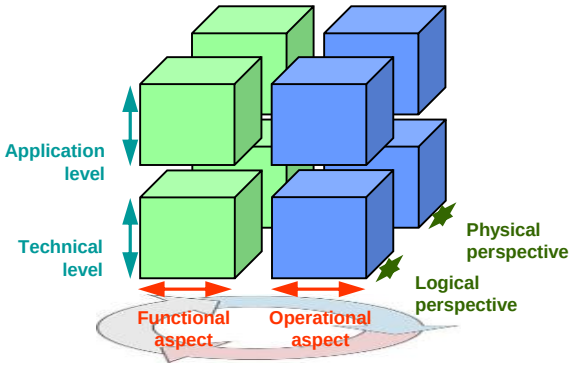
B. Physical application

C. Logical technical

D. Physical technical



The impact of the operational aspect on the component model, functional aspect, originates from multiple sources (1 of 2)



Placement considerations

- Adding components that enable existing components to collaborate between nodes; for example, asynchronous messaging
- Adding responsibilities to handle different presentation types; for example, thin versus thick client
- Adding components that handle data and software distribution

Achieving observable qualities

- Aggregating components to achieve better performance
- Segregating components with different runtime qualities

The impact of the operational aspect on the component model, functional aspect, originates from multiple sources (2 of 2)

Availability

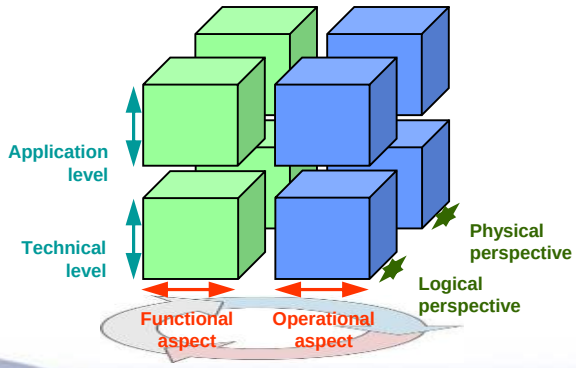
- What if individual components must be failure-resilient?
- What if the system is used by subsidiaries in other time zones?

Scalability

- What components are needed to support planned or unplanned growth?

Performance

- How can long versus short running transactions be handled?



Module outline

Introduction and objectives

The functional aspect of IT architecture

Component modeling

Building a component model

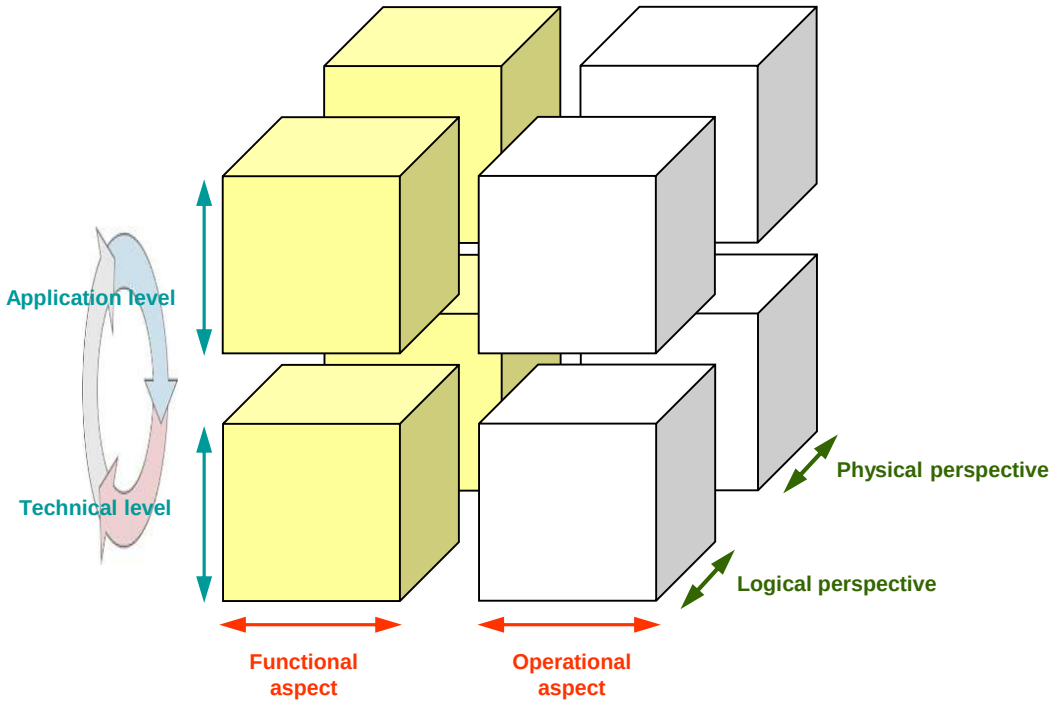
- **Identifying components**
- Specifying components
- Implementing components

Summary and references



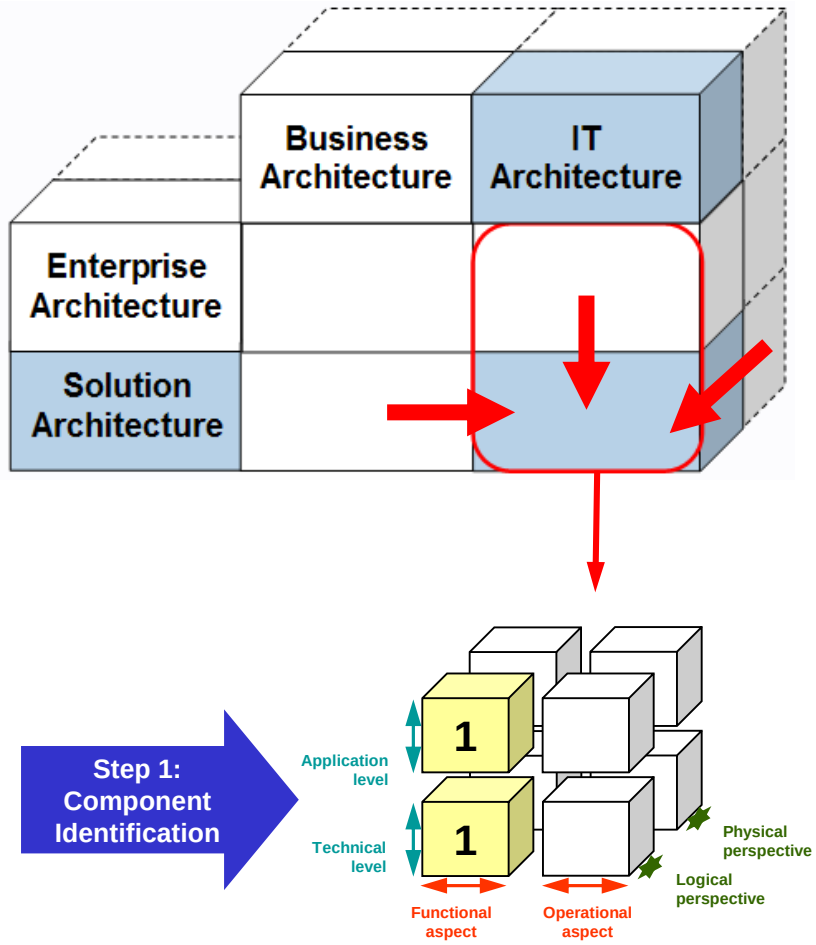
A component model evolves during the course of a project in a number of ways

- Elaboration increases the level of completeness.
- Refinement increases the level of precision.
- Implementation takes place between the logical and physical levels:
  - Logical components are implemented by physical components.
  - Physical components are specified by logical components.



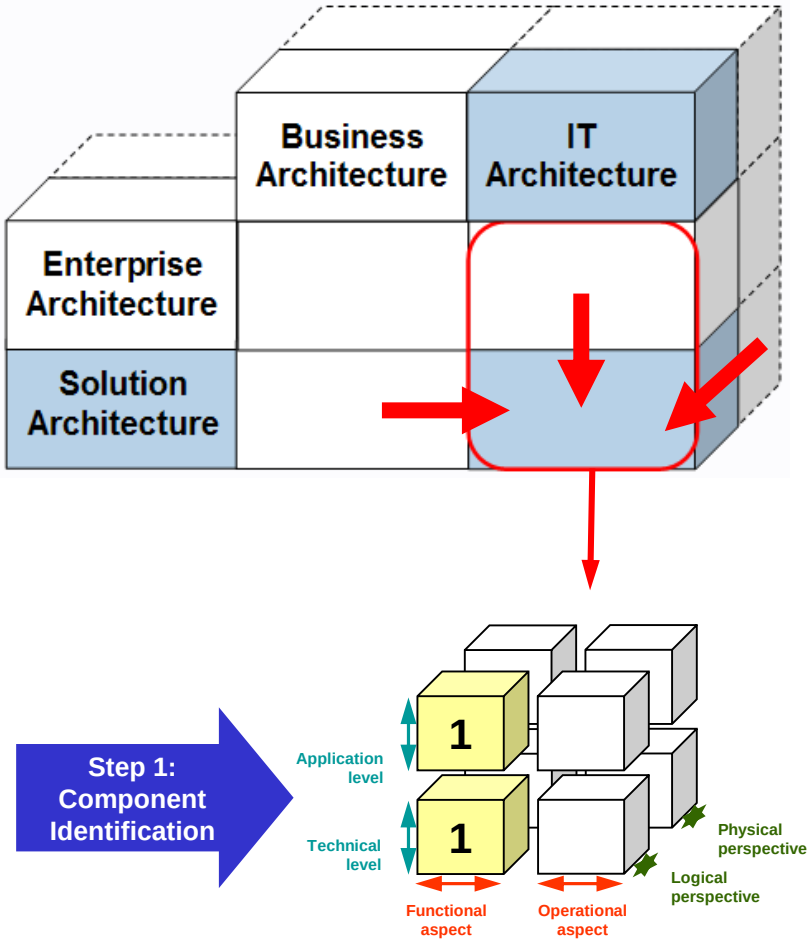
Component identification is the first step of the component modeling technique and involves the following (1 of 4)

- Driven by an understanding of the business requirements



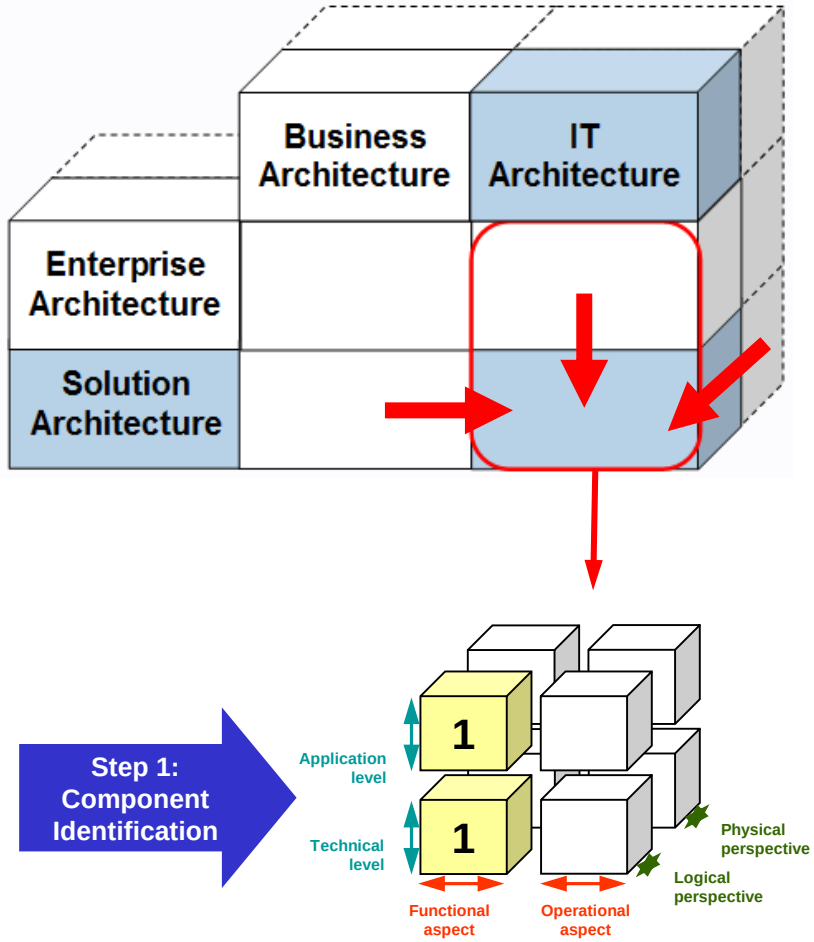
Component identification is the first step of the component modeling technique and involves the following (2 of 4)

- Driven by an understanding of the business requirements the Architect will…
- …Partition the system into subsystems and components and assign responsibilities based on an analysis of the requirements. He or she may build on work done pre proposal (by either an EA or a pre-sales Architect)



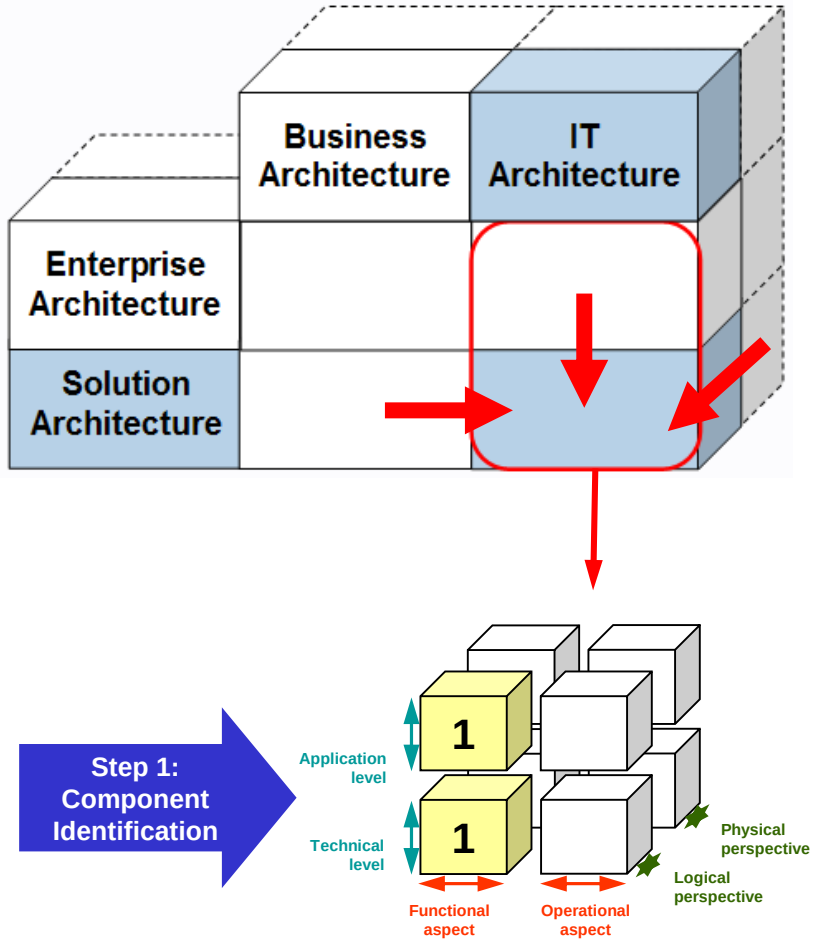
Component identification is the first step of the component modeling technique and involves the following (3 of 4)

- Driven by an understanding of the business requirements the Architect will…
- …Partition the system into subsystems and components and assign responsibilities based on an analysis of the requirements. He or she may build on work done pre-proposal (by either an EA or a pre-sales Architect)
- Use available reference architectures, architectural patterns, and other reusable assets



Component identification is the first step of the component modeling technique and involves the following (4 of 4)

- Driven by an understanding of the business requirements the Architect will…
- …Partition the system into subsystems and components and assign responsibilities based on an analysis of the requirements. He or she may build on work done pre-proposal (by either an EA or a pre-sales Architect)
- Use available reference architectures, architectural patterns, and other reusable assets
- Ensuring the components are well structured, so that they are:
  - Highly cohesive
  - Loosely coupled
  - Well isolated
  - Of the right granularity
  - Layered according to their generality



Structure and refactor, ensuring loose coupling, high cohesion, and so on

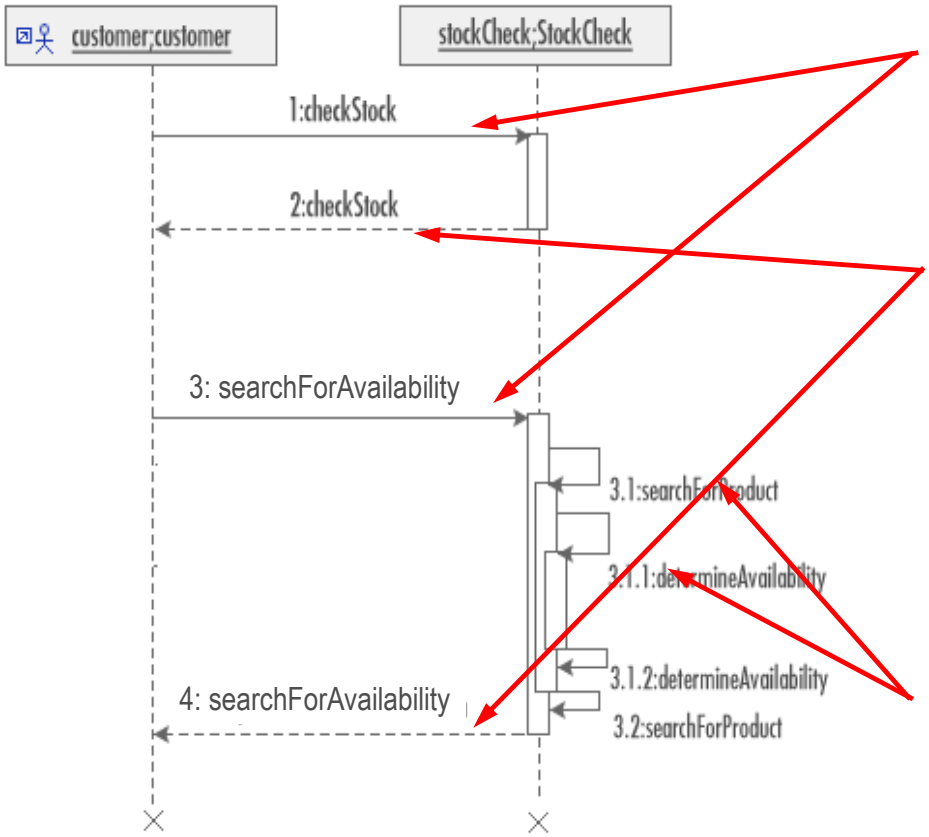
- Do the responsibilities assigned to a component make sense? Cohesiveness
- Are two or more components too dependent on each other? Coupling
- Have product or technology dependencies been confined to a few places? Isolation
- Have sufficient responsibilities been allocated to the component? Granularity
- Are components layered according to their generality? Layering

Cohesion is a measure of the strength of the dependencies *within* a component

- Responsibilities have no meaningful relationship to each other.
- Responsibilities are related in time only.
- Responsibilities are of the same general category that is selected externally.

- Responsibilities contribute to a few related business activities only.
- Responsibilities are likely to use the same input or output data.
- Responsibilities are likely to be used together.

Low or high cohesion? Good or bad



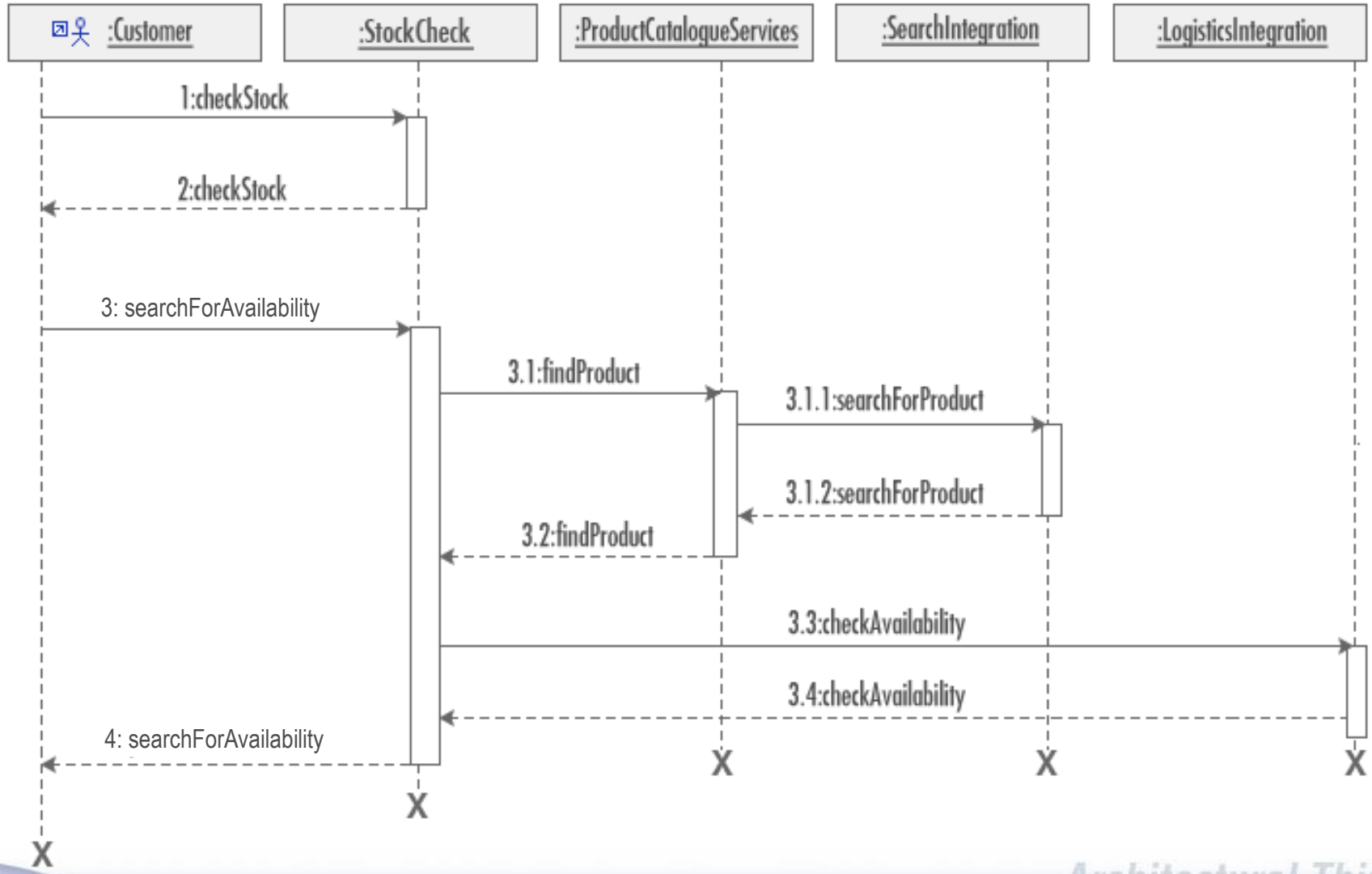
Handles user input

Handles display of user information

Handles different functions: search and stock availability

Handles overall control

An example of high cohesion: Good



Cohesion is a qualitative measure

The types of cohesion, in order of the worst to the best type, are as follows:

Coincidental cohesion (worst)

- Coincidental cohesion is when parts of a component are grouped arbitrarily, at random; the parts have no significant relationship; for example, a component of frequently used mathematical functions.

Logical cohesion

- Logical cohesion is when parts of a component are grouped because they logically are categorized to do the same thing, even if they are different by nature; for example, grouping all I/O handling routines.

Temporal cohesion

- Temporal cohesion is when parts of a component are grouped by when they are processed; the parts are processed at a particular time in program execution; for example, a function that is called after catching an exception that closes open files, creates an error log, and notifies the user.

Procedural cohesion

- Procedural cohesion is when parts of a component are grouped because they always follow a certain sequence of execution; for example, a function that checks file permissions and then opens the file.

Communicational cohesion

- Communicational cohesion is when parts of a component are grouped because they operate on the same data; for example, a component that operates on the same record of information.

Sequential cohesion

- Sequential cohesion is when parts of a component are grouped because the output from one part is the input to another part, as on an assembly line; for example, a function that reads data from a file and processes the data.

Functional cohesion (best)

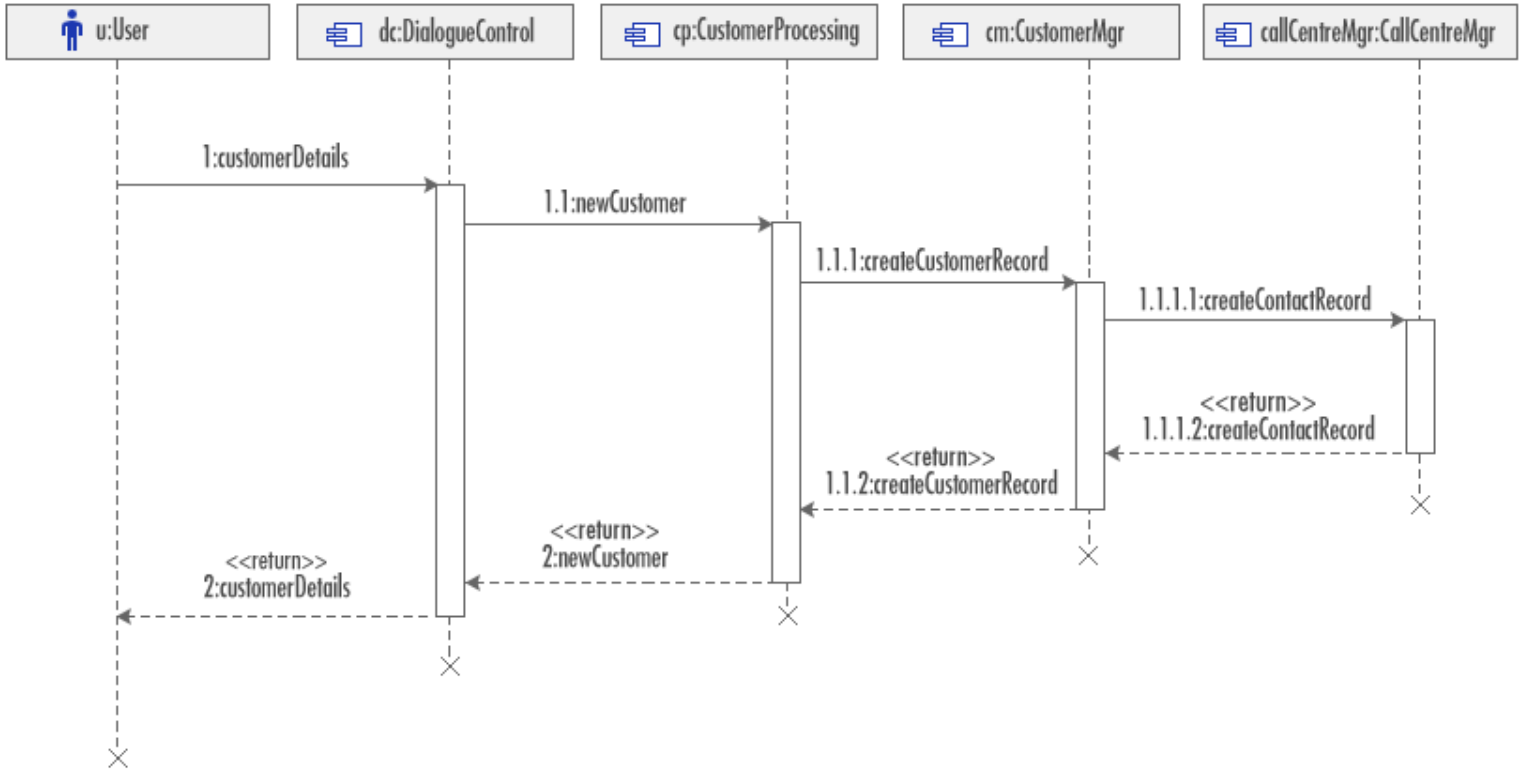
- Functional cohesion is when parts of a component are grouped because they all contribute to a single well-defined task of the module.

Coupling is a measure of the strength of the dependencies *between* components

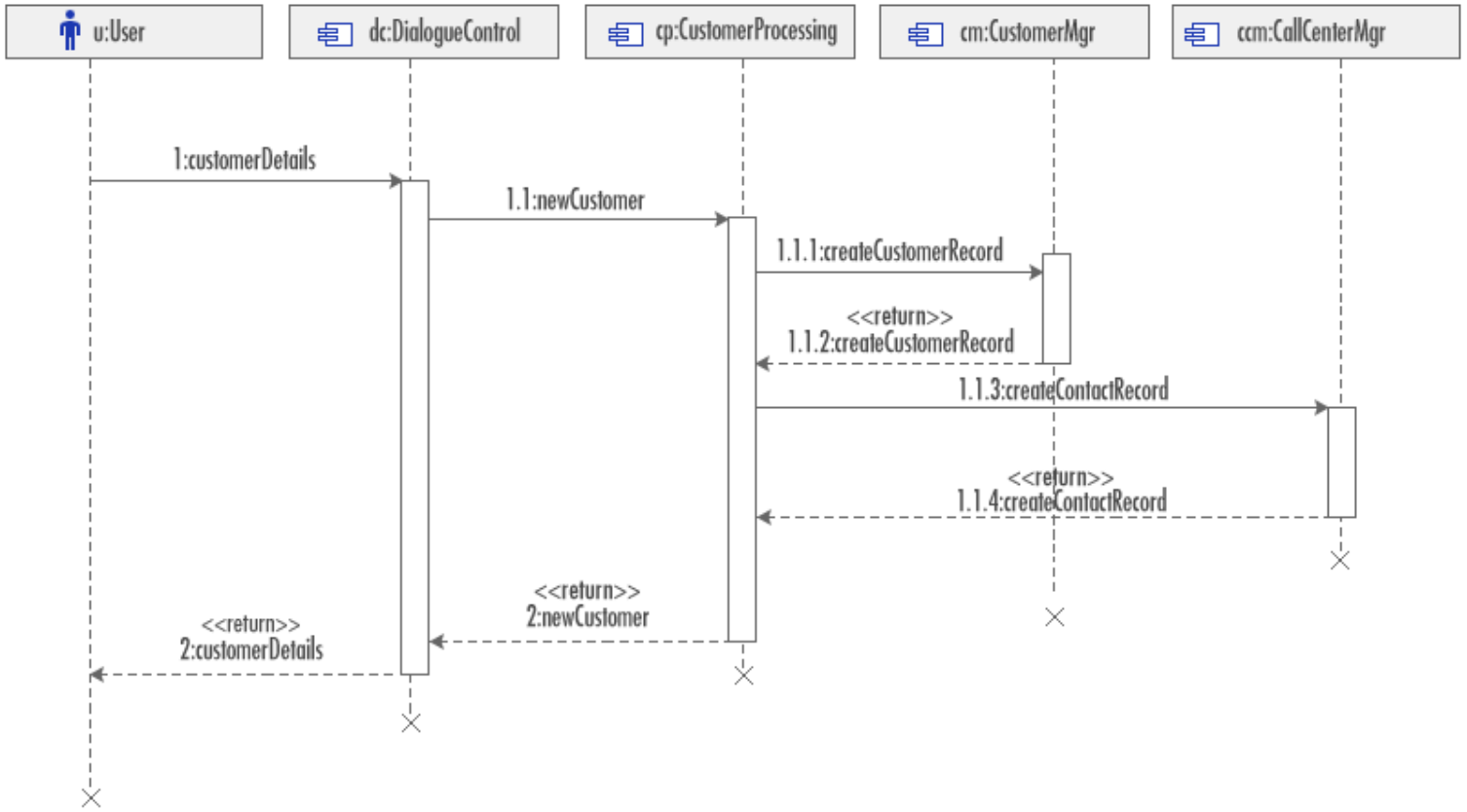
- Data that is intended to control the internal component logic is passed .
- Components refer to the same global data area.
- One component refers to the inside of another.
- Different meanings are assigned to different parts of a range of data.

- There is no unnecessary inter-component communication: Don't Talk to Strangers.
- Communication is via fundamental data types.
- If communication is via composite data, then it is self-describing data, such as XML.

An example of strong coupling: Bad



An example of loose coupling: Good



Coupling is also a qualitative measure

Content coupling, high

- Content coupling is when one component modifies or relies on the internal workings of another component; for example, accessing local data of another component.

Common coupling

- Common coupling is when two components share the same global data; for example, a global variable. Changing the shared resource implies changing all the components using it.

External coupling

- External coupling occurs when two components share an externally imposed data format, communication protocol, or device interface.

Control coupling

- Control coupling is one component controlling the logic of another, by passing its information on what to do; for example, passing a what-to-do flag.

Stamp coupling, data-structured coupling

- Stamp coupling is when components share a composite data structure and use only a part of it, possibly a different part; for example, passing a whole record to a function which only needs one field of it.

Data coupling

- Data coupling is when components share data through, for example, parameters. Each datum is an elementary piece, and these are the only data which are shared; for example, passing an integer to a function that computes a square root.

Message coupling, low

- Components are not dependent on each other; instead, they use a public interface to exchange parameter-less messages or events.

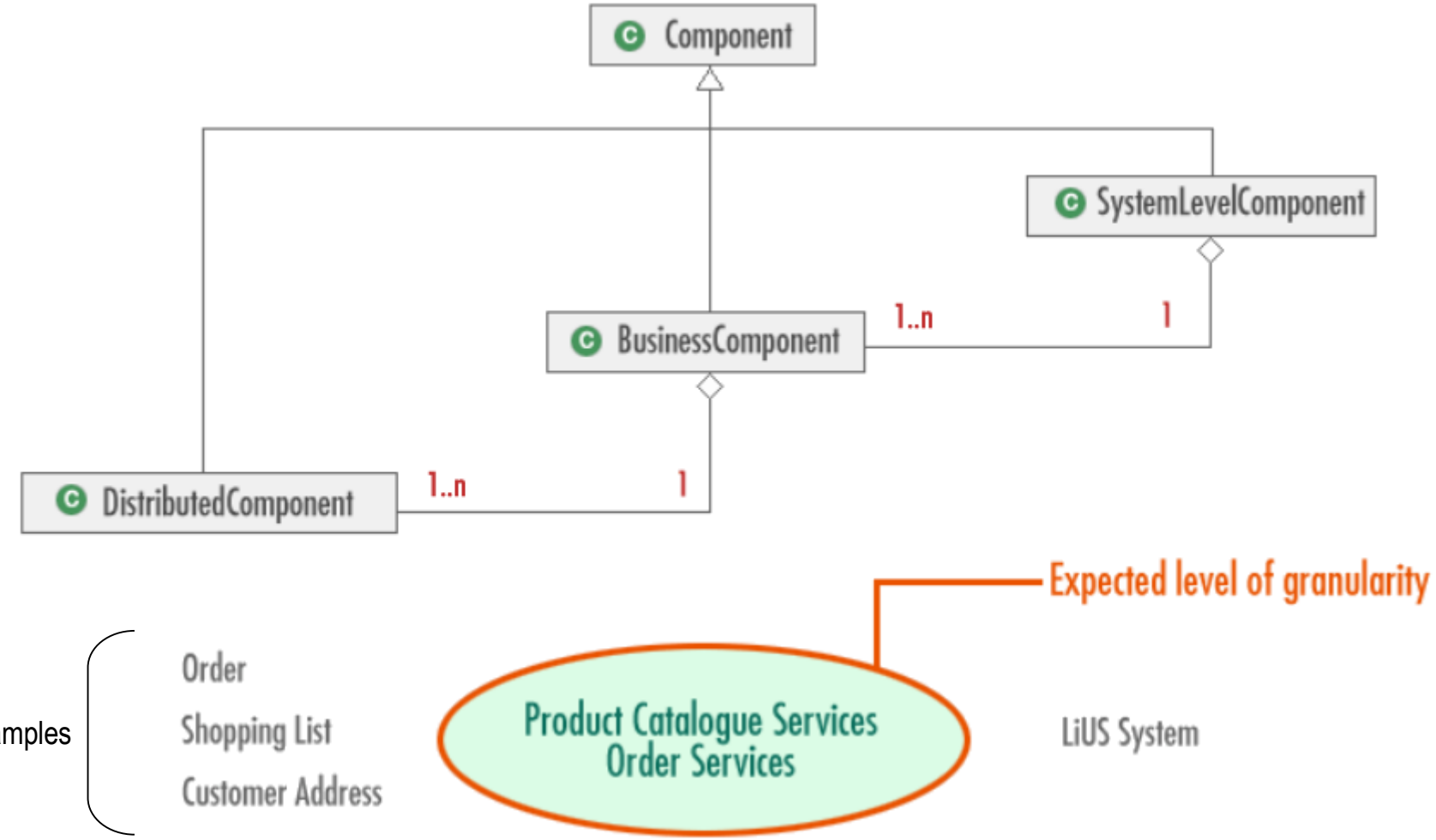
Isolation is a measure of the degree to which product and technology dependencies are isolated

- Every component has a dependency on the product-specific or technology-specific aspects of other components.
- There is no use of patterns.

- Product and technology dependencies are decoupled.
- Patterns, such as the Gang of Four* “Proxy,” “Bridge,” and “Mediator” or GRASP** “Indirection” are used.

*See [GAMMA95]
**General Responsibility Assignment Software Patterns

Some examples of components at the three levels of granularity



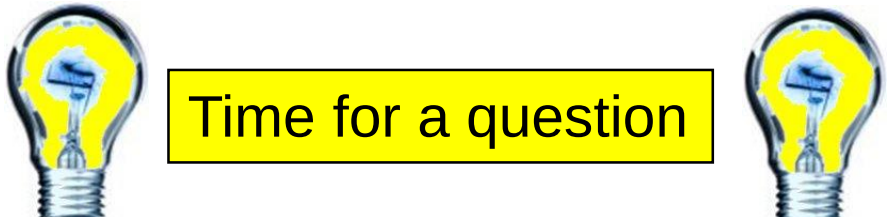
Granularity is a measure of the size or amount of functionality assigned to a component (1 of 4)

- A software element called at run time with a clear interface and a separation between interface and implementation
- Also known as a *distributed component*

- An autonomous business concept or business process
- Also known as a *business component*

- A set of cooperating business components assembled to deliver a solution to a business problem
- Also known as a *system-level component*

Granularity is a measure of the size or amount of functionality assigned to a component (2 of 4)



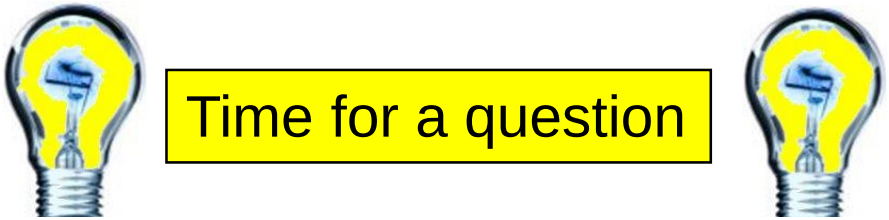
Account manager and account processing are examples of a:

A. Distributed component

B. Business component

C. System-level component

Granularity is a measure of the size or amount of functionality assigned to a component (3 of 4)



Time for a question

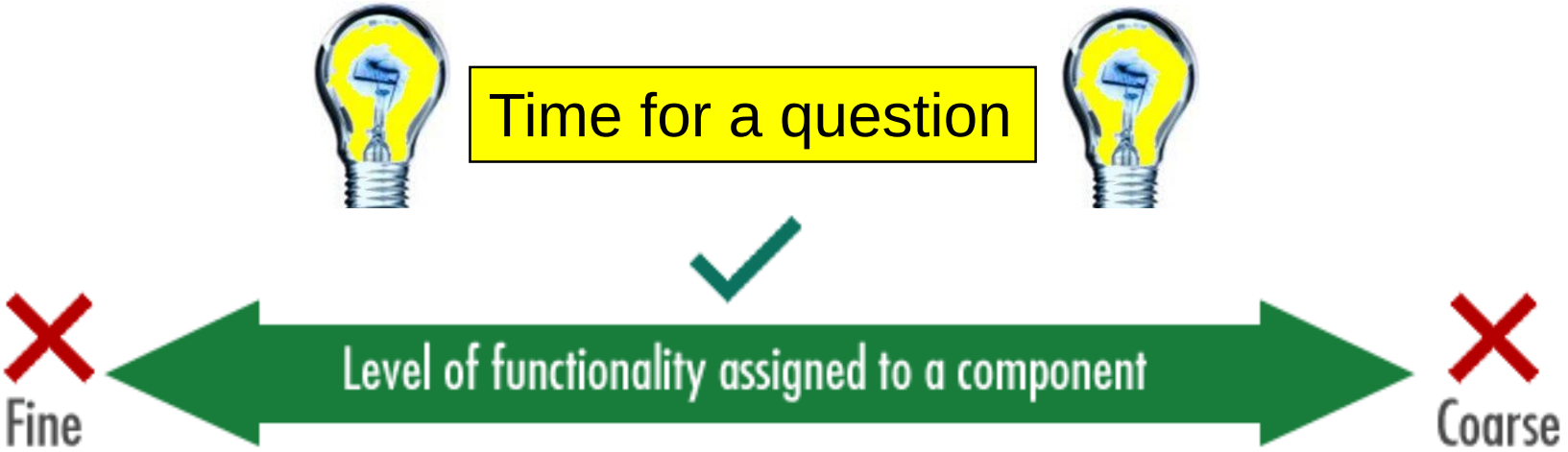
Interest calculator, account, and account rules are examples of a:

A. Distributed component

B. Business component

C. System-level component

Granularity is a measure of the size or amount of functionality assigned to a component (4 of 4)



An Internet bank system is an example of a:

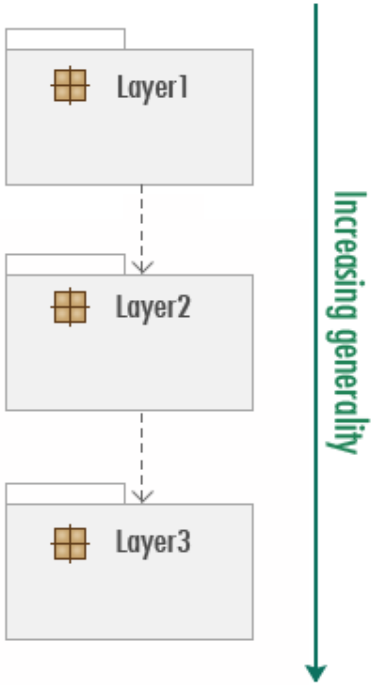
A. Distributed component

B. Business component

C. System-level component

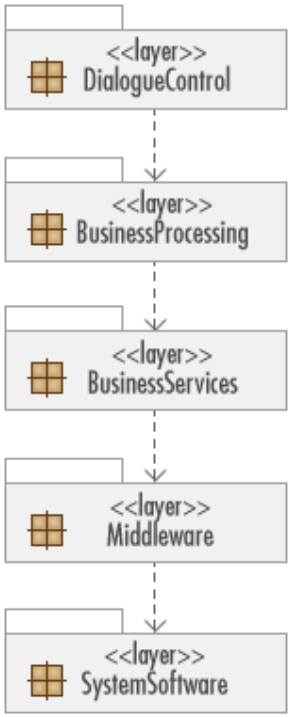
Layering involves separating out components according to their generality

- Layering provides a logical partitioning of components into a number of sets or layers.
- Rules define relationships between layers:
  - Strict: Components only depend on components in the same layer or the one below.
  - Non-Strict: Components may depend on components in any lower layer.
- Layering provides a way to restrict inter-component dependencies.
- Well-layered systems are more loosely coupled and therefore more easily maintained.

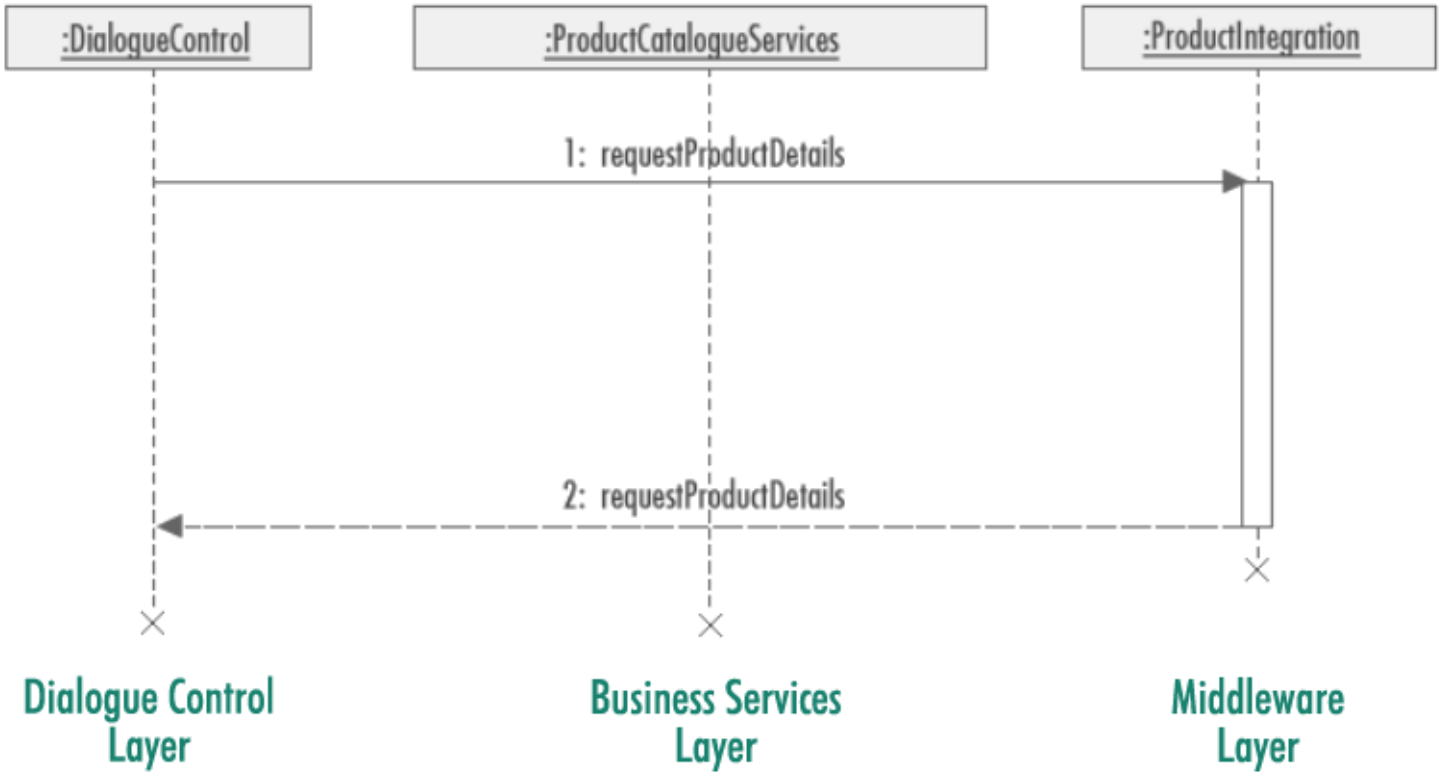


An example of layered architecture

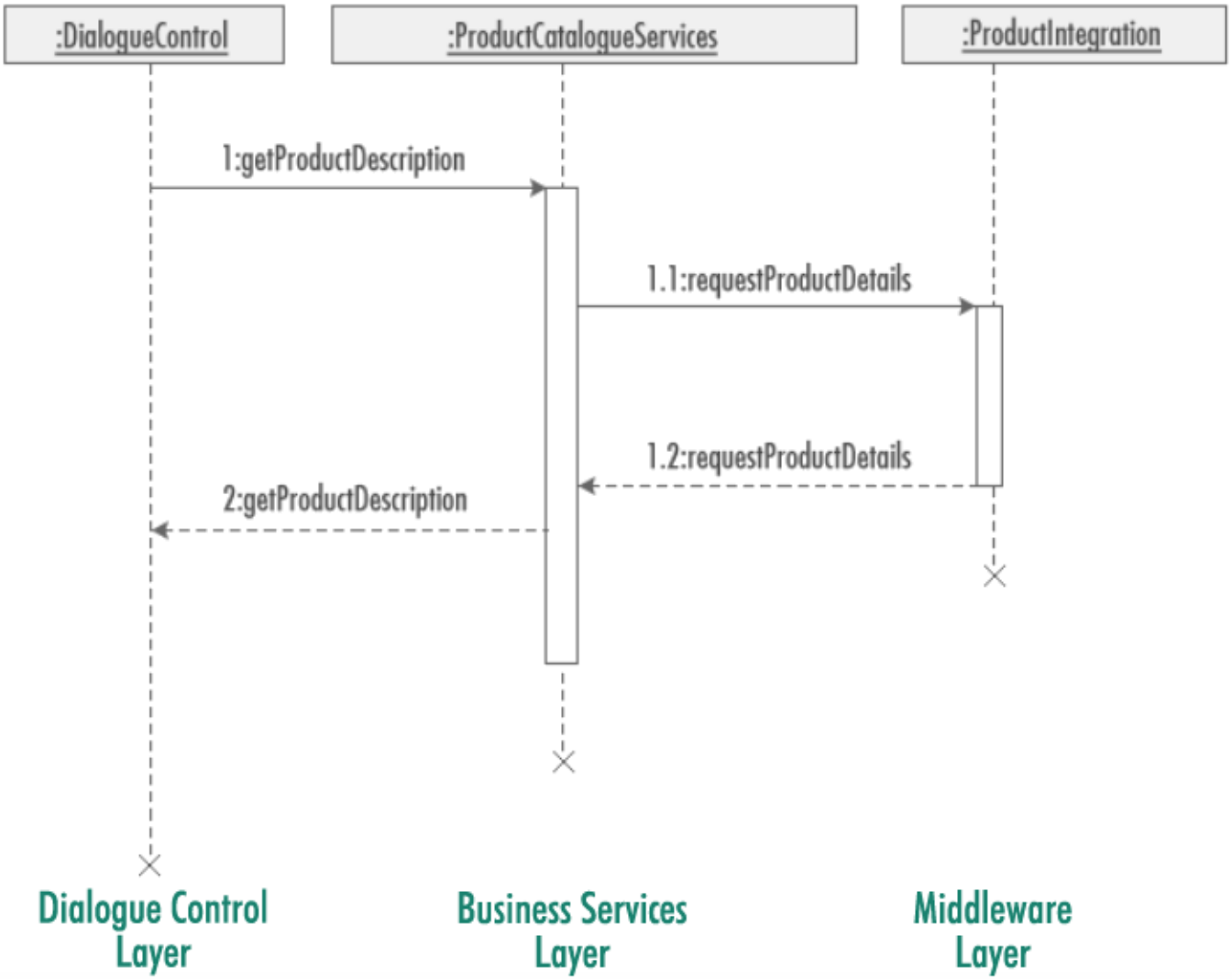
- The dialog control layer handles user-system interactions and uses case logic.
- The business processing layer contains application-specific services that handle use case step logic and choreography.
- The business services layer contains more general business components that may be used in several applications.
- The middleware layer contains components such as interfaces to databases and platform-independent operating system services.
- The system software layer contains components such as operating systems and databases.



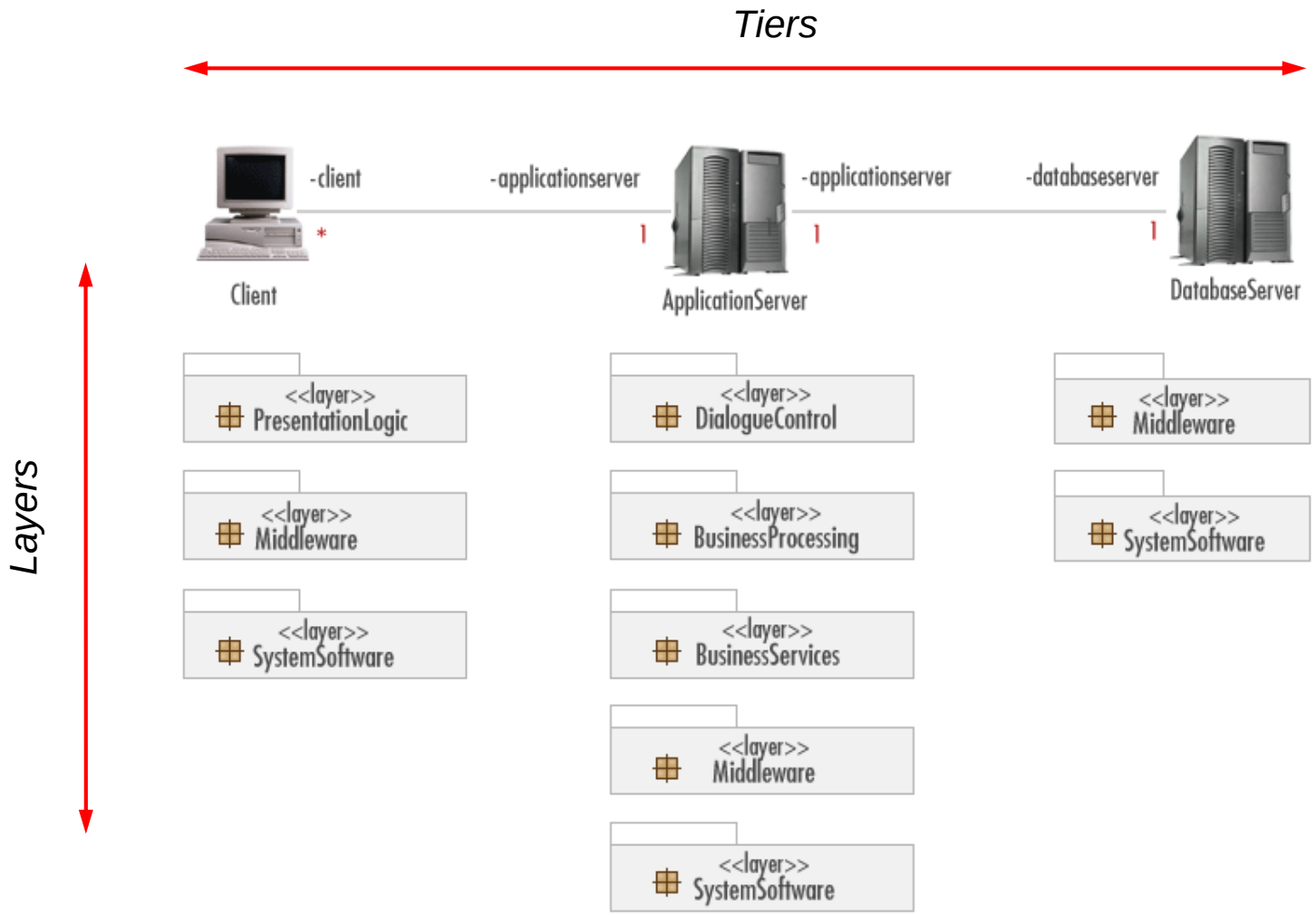
Strict or non-strict layering? Good or bad



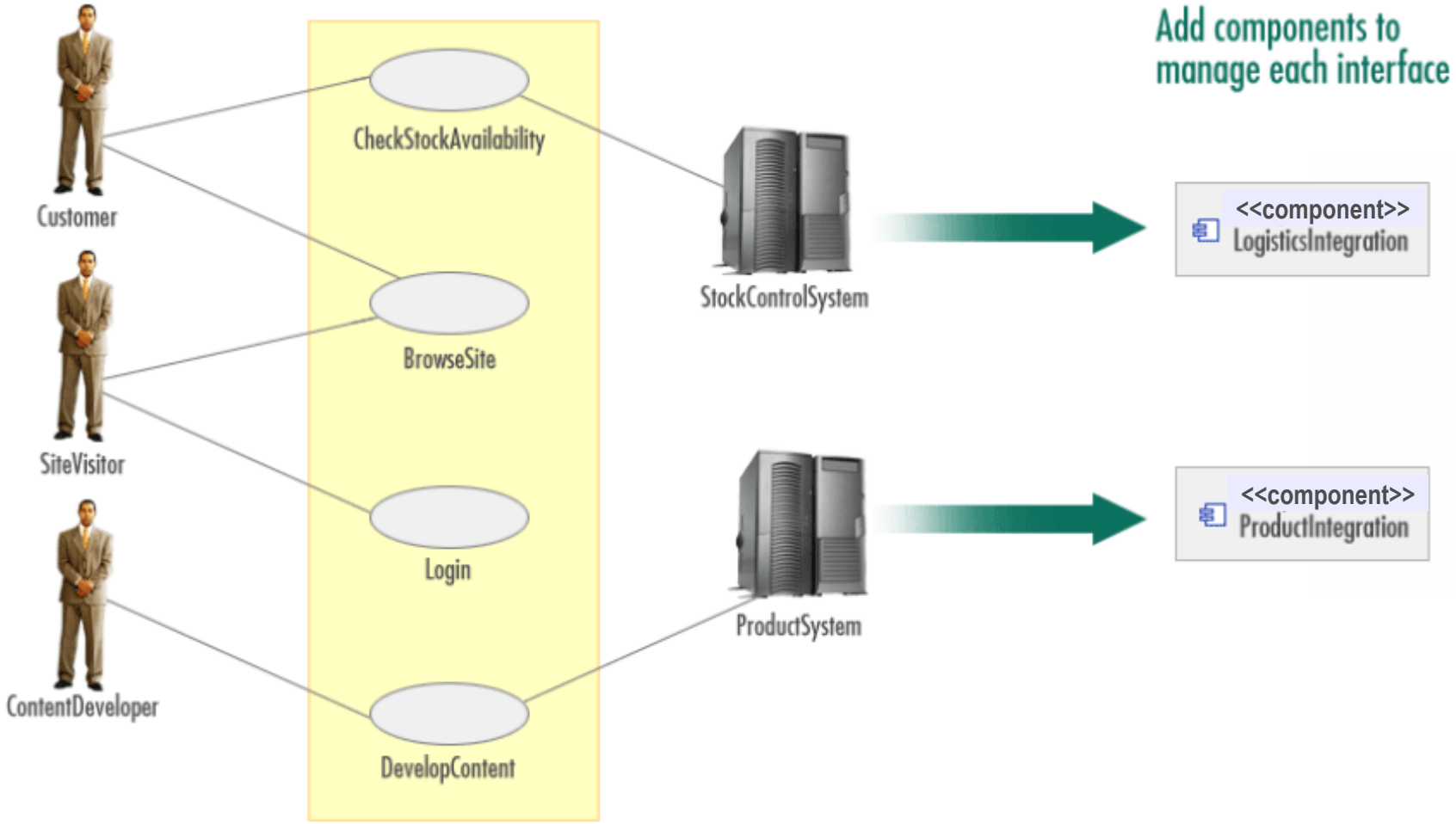
An example of strict layering: Good



Layers and tiers



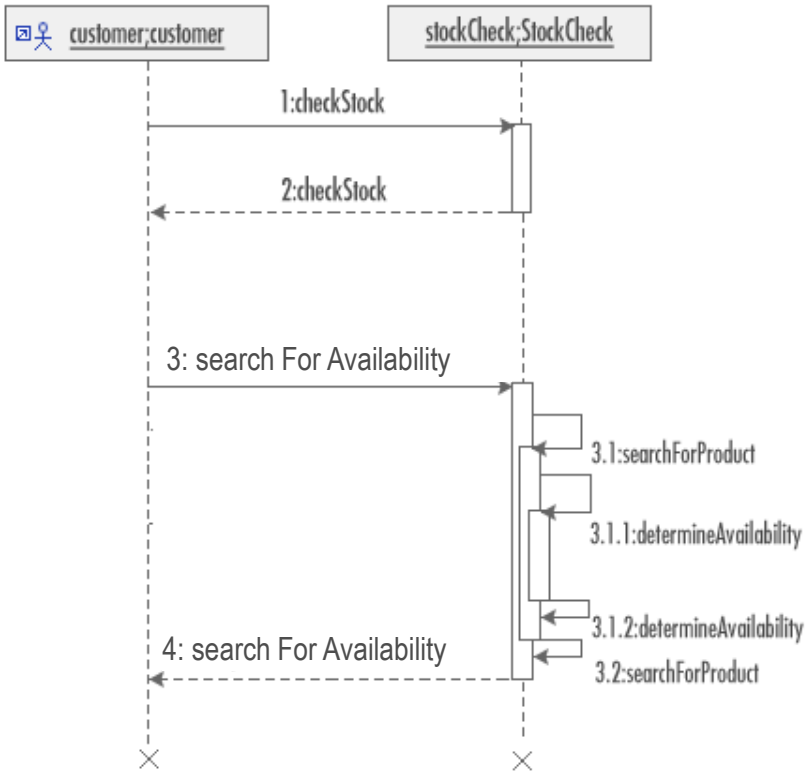
Partition into subsystems and components and assign responsibilities; identify interfaces to external systems



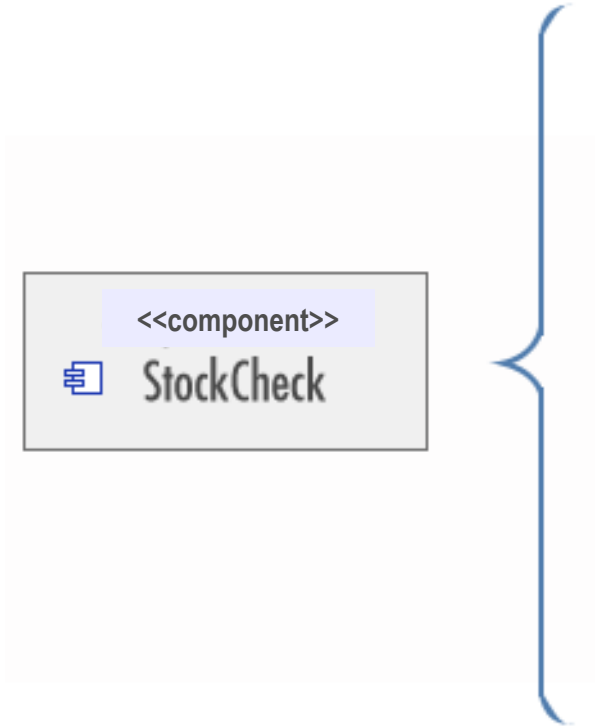
Partition into subsystems and components and assign responsibilities; identifying components for architecturally significant use cases

- The actor requests a stock query.
- The system prompts the actor to select a store name and enter an article or product name or number.
- The actor selects a store name and enters an article or product name. If a product name is entered, the user may also be required to specify further selection criteria, such as selecting color and so on, for that product.
- The system searches for the entered article or product in the selected store and checks its availability.
- The system determines that the article or product is available.
- The system returns the availability information for the article or product.
- The use case ends successfully.

Use Case: Check stock availability

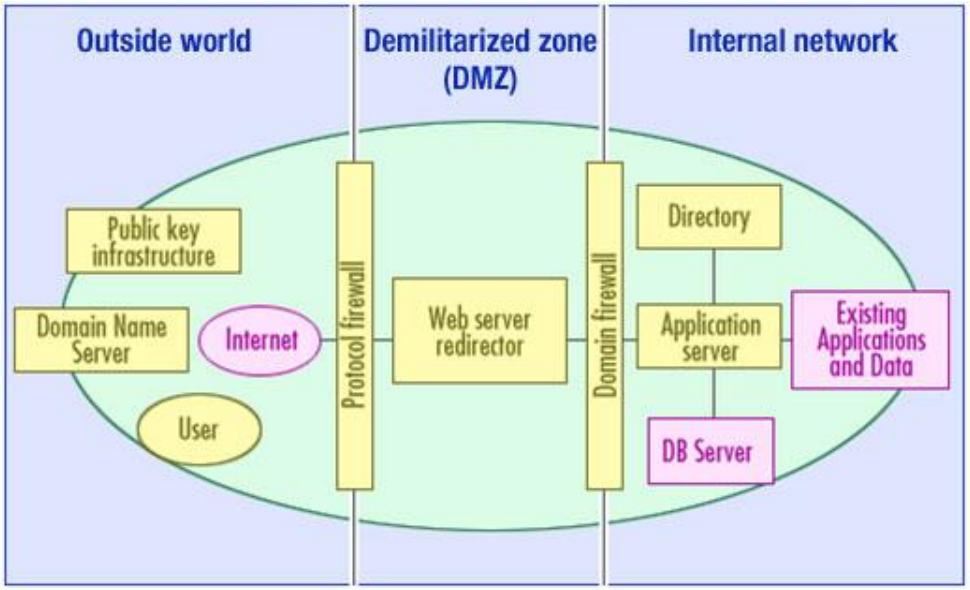


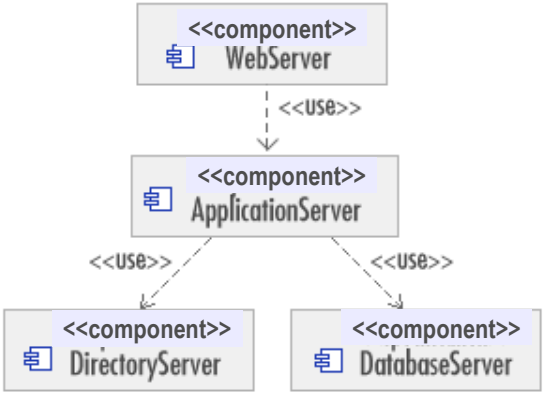
Partition into subsystems and components and assign responsibilities to component



- Handle stock requests
- Request product information
- Perform search
- Determine availability
- Provide availability information

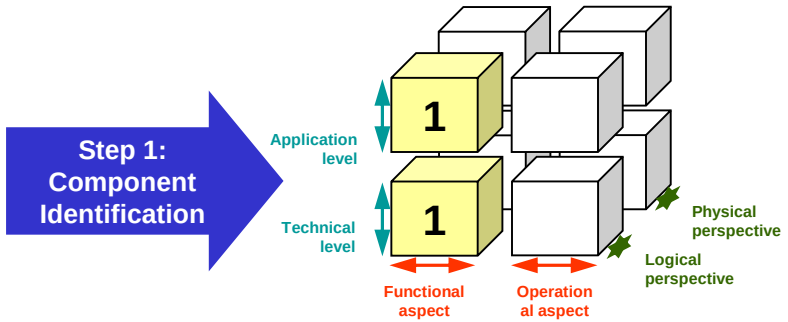
Review architectural patterns, reference architectures, and reusable assets to derive an initial set of components





At the completion of the component identification step, there should be

- An initial set of components with associated responsibilities
- One or more component relationship diagrams showing dependencies between components
- One or more component interaction diagrams showing collaborations between components
- Subsequent steps will:
  - Specify components: interfaces, operations, and signatures
  - Show how logical components are implemented into physical components



Module outline

Introduction and objectives

The functional aspect of IT architecture

Component modeling

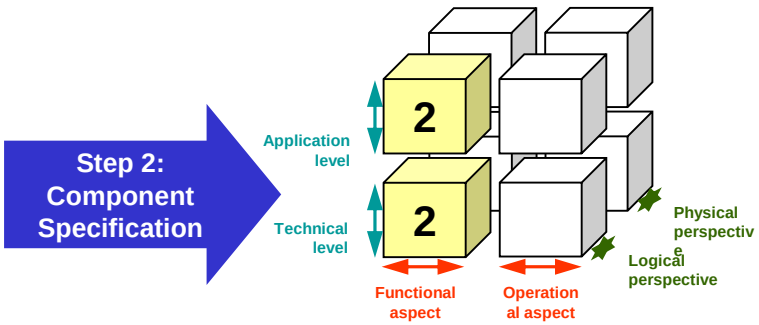
Building a component model

- Identifying components
- **Specifying components**
- Implementing components

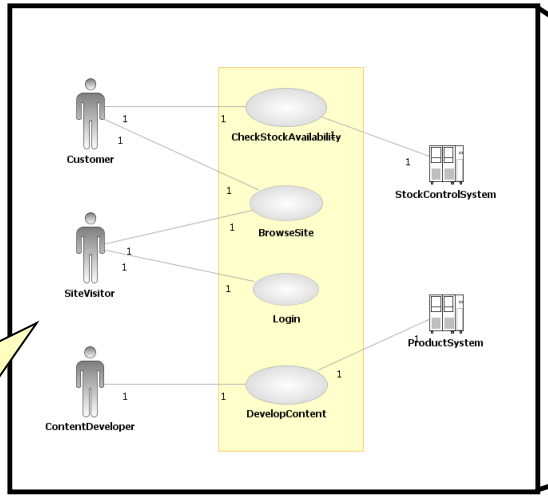
Summary and references

Component specification is the second step of the component modeling technique and involves

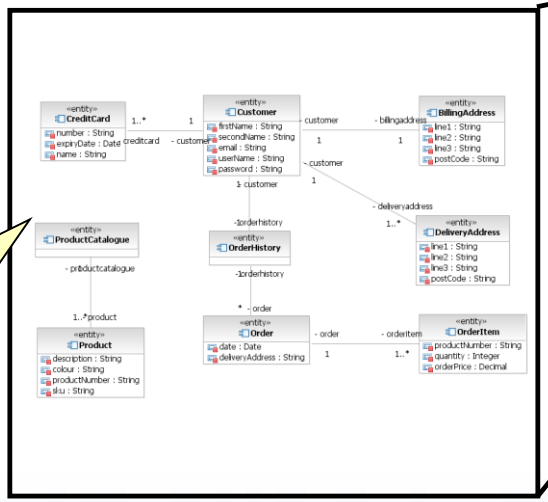
- Defining component interfaces by separating out the responsibilities placed on the components
- For each interface, specifying the operations and their signatures, that is, parameters passed and return values
- For each operation, specifying its contract of behavior or pre- and post-conditions



There are various approaches to identifying interfaces on IT components, depending on which layer they are in



Use Case Model



Data Model

«layer» DialogueControl

«layer» BusinessProcessing

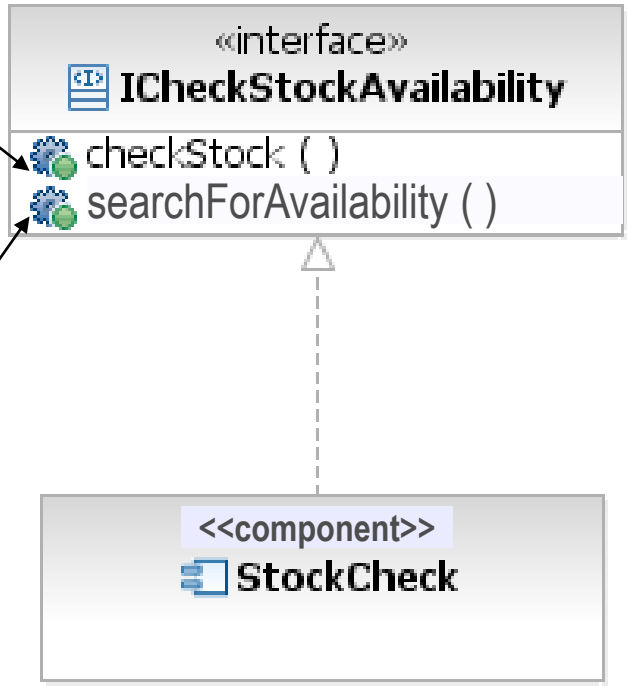
«layer» BusinessServices

«layer» Middleware

«layer» SystemSoftware

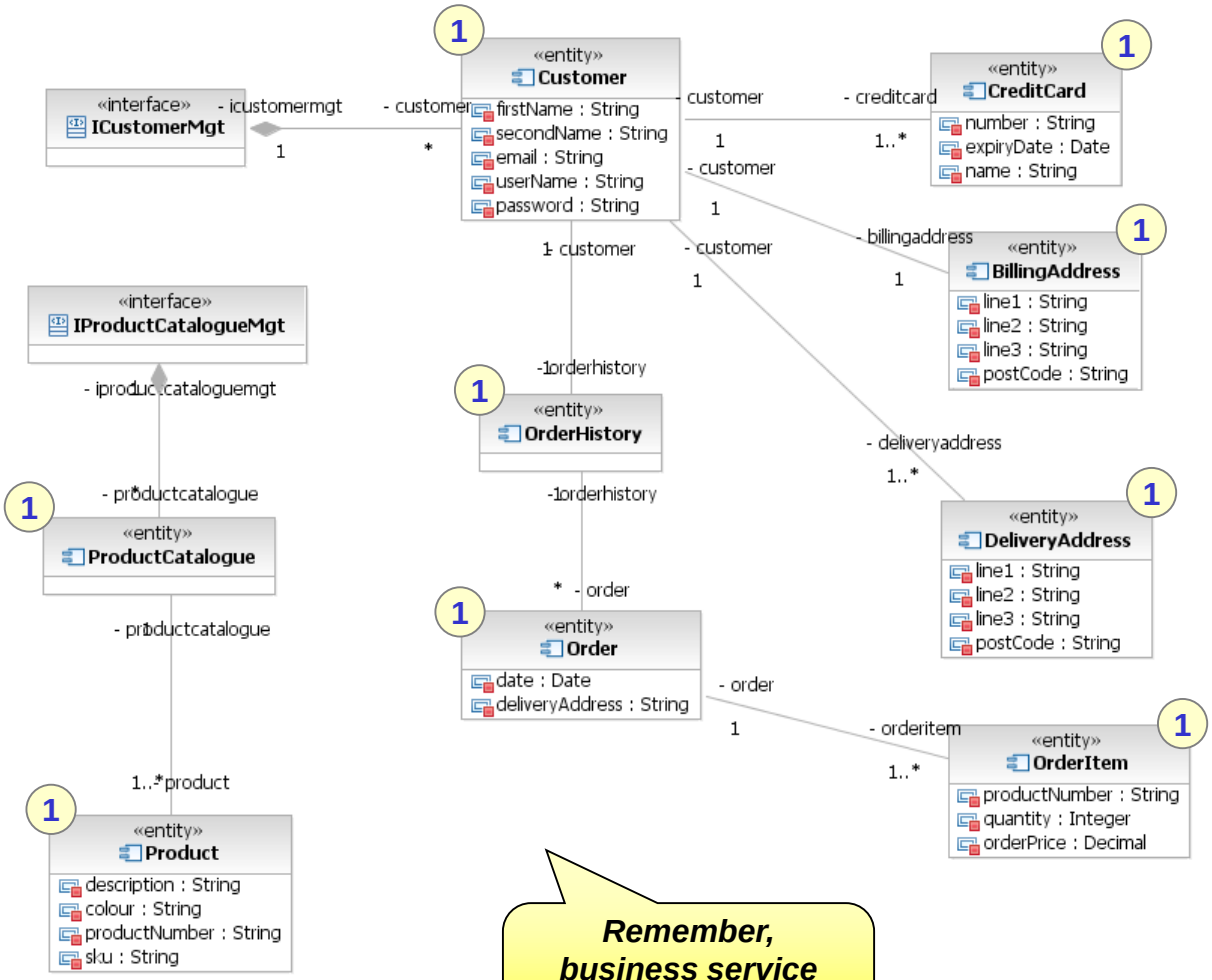
Identifying business processing component interfaces from use cases

1. The actor requests a stock query.
2. The system prompts for the actor to select a store name and enter an article or product name or number.
3. The actor selects a store name and enters an article or product name. If a product name is entered, the user may also be required to specify further selection criteria (for example, selecting color) for that product.
4. The system searches for the selected article or product in the named store and checks its availability.
5. The system determines the article or product is available.
6. The system returns the availability information for the article or product.
7. The use case ends successfully.



Identifying business service component interfaces from business entities (1 of 3)

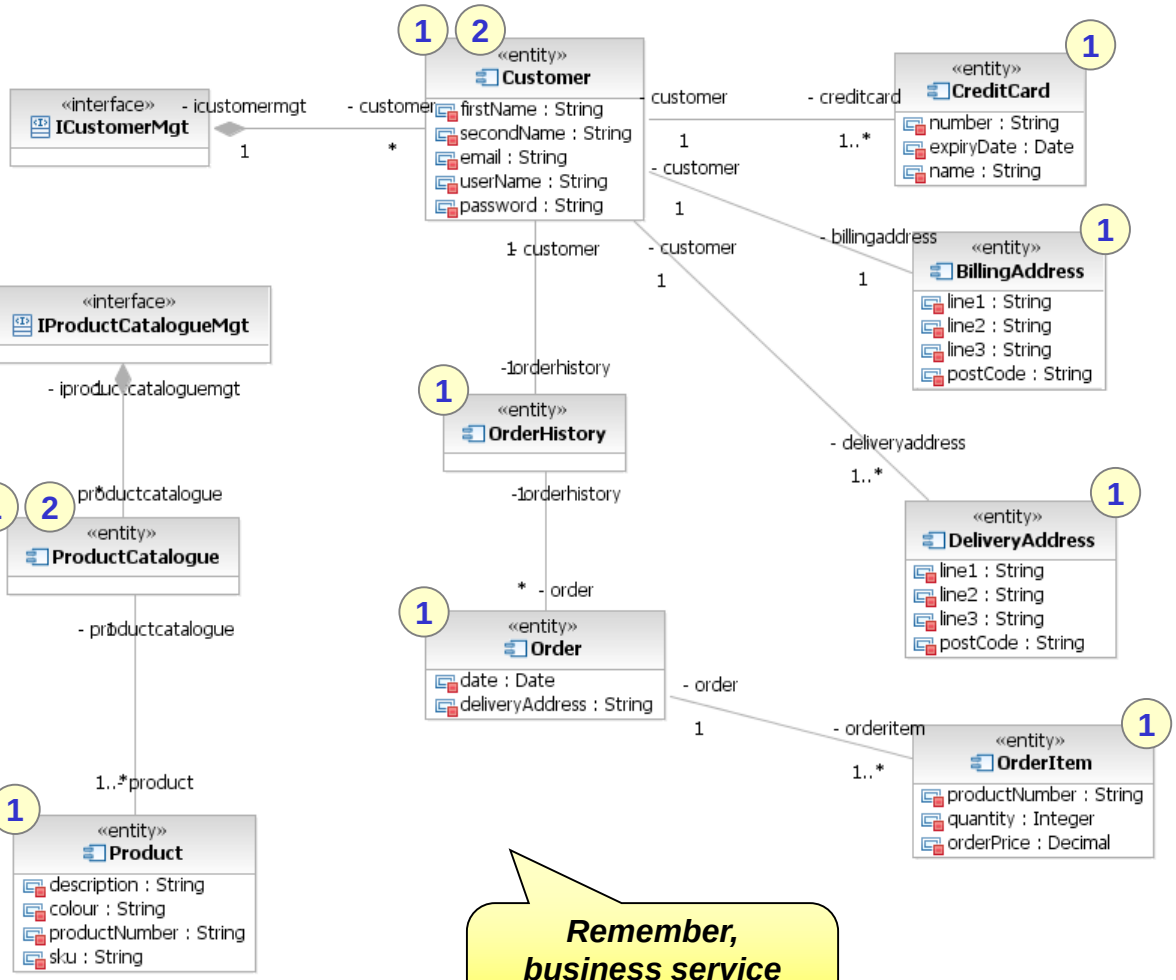
1. Produce a logical data model showing business entities.



Remember, business service components are at the IS level of IT

Identifying business service component interfaces from business entities (2 of 3)

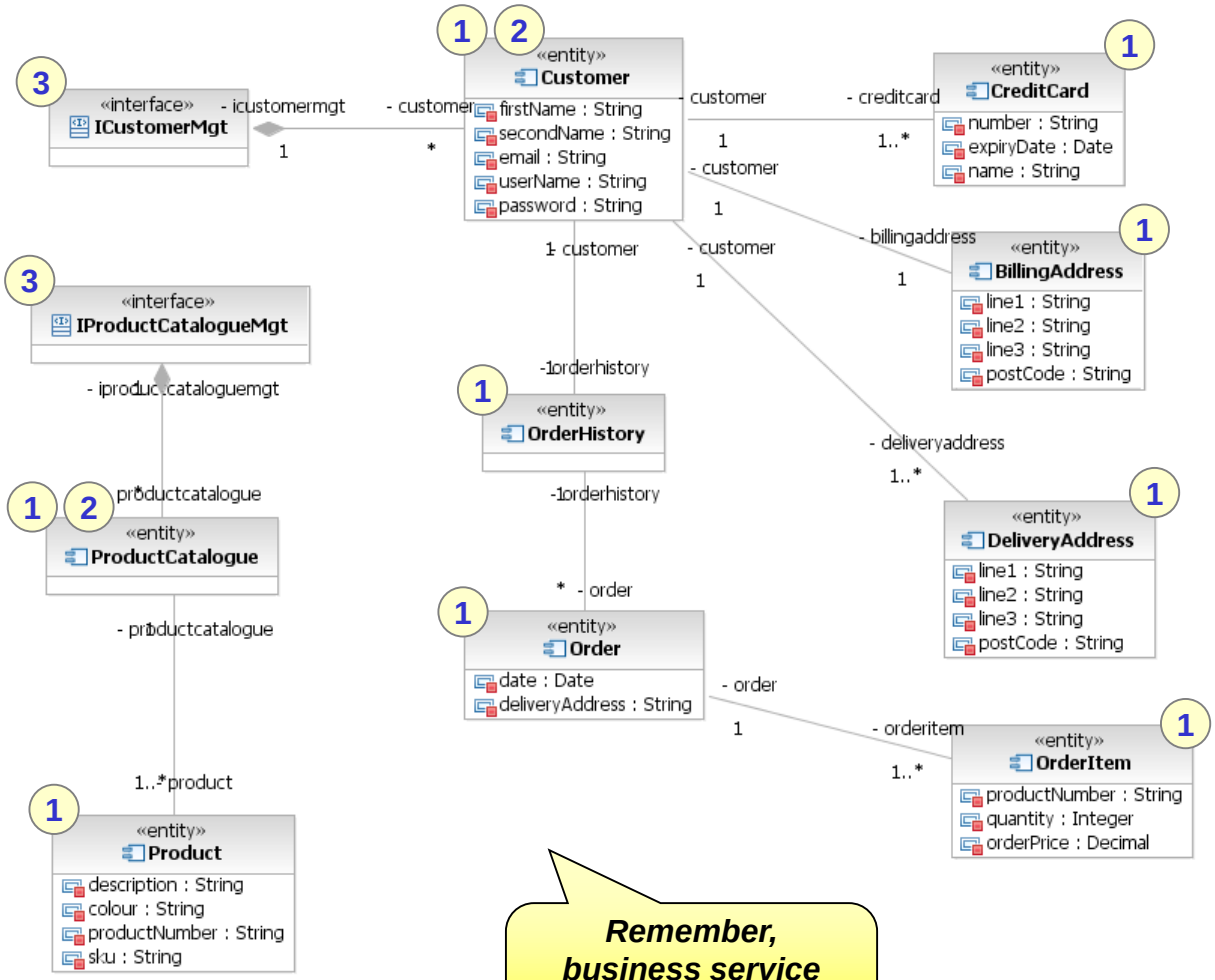
1. Produce a logical data model showing business entities.
2. Identify core business entities.



Remember, business service components are at the IS level of IT

Identifying business service component interfaces from business entities (3 of 3)

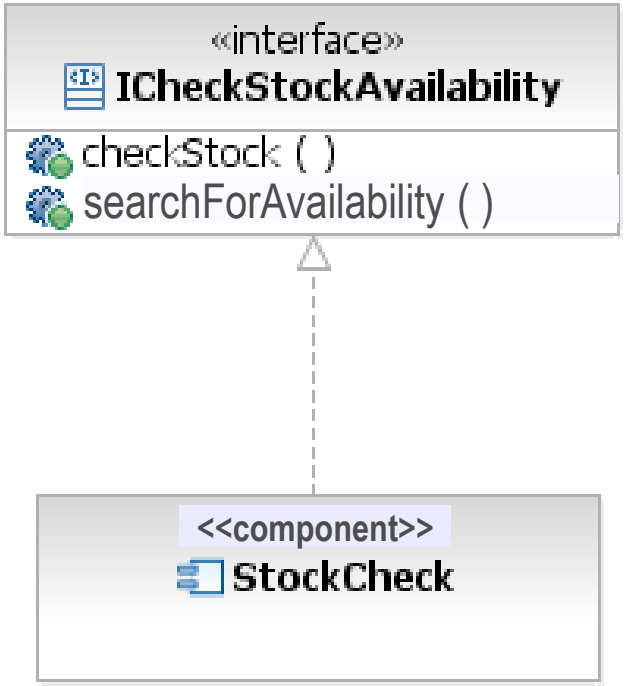
1. Produce a logical data model showing business entities.
2. Identify core business entities.
3. Create interfaces and operations to manage core business entities.



Remember, business service components are at the IS level of IT!

Assigning operation signatures to operations

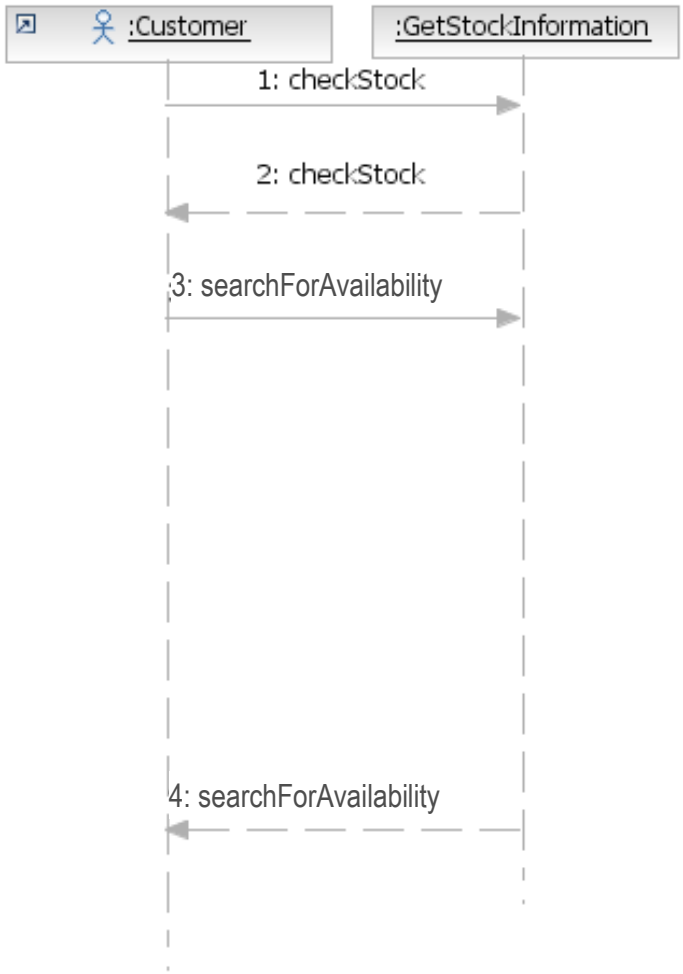
- For each operation identified, decide what data flows in and out.
- Look at the collaborations between components in component interaction diagrams.
- Make decisions about what data is required to provide the required functionality.
- Create structured data types as appropriate



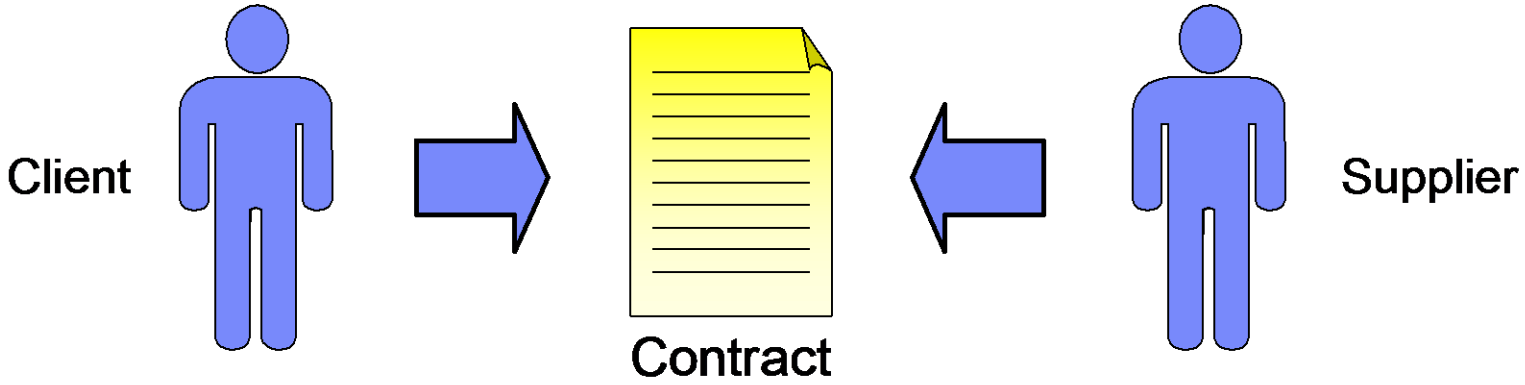
Let's look at the interaction searchForAvailability

- A store name and a product name, both strings, are required as input.
- The quantity available in the store will be returned.
- Alternately, a simple true for "available" or false for "not available" might be acceptable as a return, depending on the stakeholders for the use case.
- The signature then becomes:

searchForAvailability(storeName:String, product:String):Integer



Pre- and post-conditions are contracts on how an operation will work between a client and a supplier



Pre-condition

- What must be true for the operation to succeed
- An obligation for the client and a benefit for the supplier
- A pre-condition violation means a “bug” in the client

Post-condition

- What the operation will guarantee to do if the pre-condition is true
- A benefit for the client and an obligation for the supplier
- A post-condition violation means a “bug” in the supplier

Considering pre- and post-conditions for the operation searchForAvailability

searchForAvailability(storeName:String, product:String):Integer

Pre-condition: what must be true for the operation to succeed

- `storeName` must be a valid store name
- `product` must be a valid product identifier

Post-condition: what the operation will guarantee to do if the pre-condition is true

- System will provide product availability; for example, quantity, at the specified store

At the completion of the component specification step,

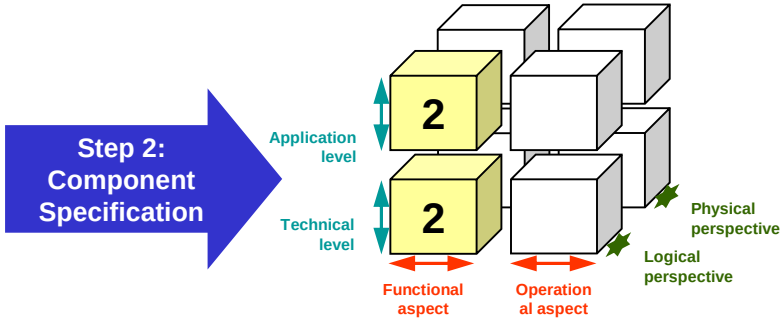
There should be a refined component model with:

Defined component interfaces by separating out the responsibilities placed on the components:

- For each interface, specifying the operations and their signatures, that is, parameters passed and return values
- For each operation, specifying its contract of behavior or pre- and post-conditions

Subsequent steps will:

- Map the specified components to the appropriate products, packages, and custom-built applications; an architecture decision needs to be taken
- Identify implementation approach of physical components



Module outline

Introduction and objectives

The functional aspect of IT architecture

Component modeling

Building a component model

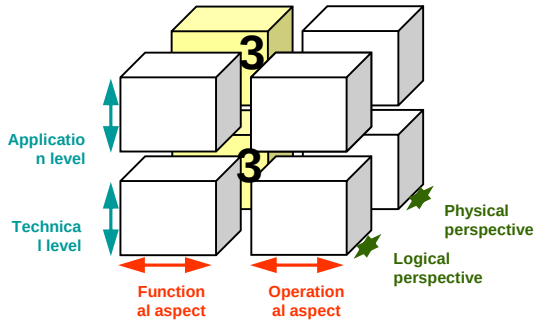
- Identifying components
- Specifying components
- **Implementing components**

Summary and references



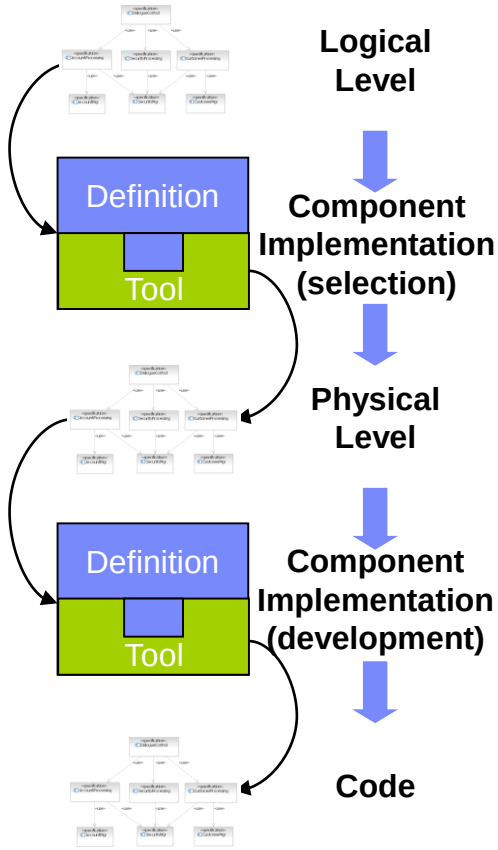
Component Implementation is the third step of the component modeling technique and involves

- For those components that are to be implemented using commercial-off-the-shelf (COTS) products, map the specified-level components to the appropriate products or packages.
- For those components that are to be built, identify the approach to implementation by defining the physical-level components. Use patterns or other assets to help in this definition.
- And don't forget that many components will be realized as standard technologies defined with selection criteria in the EA.
  - Whether at the *IS* level of IT business functionality
  - Or the *technology* level of IT middleware and so on



During component implementation, the logical-level model is transformed into a physical-level model

- Decisions must be made as to what technology best realizes the specified-level components.
- Implementing a specified-level component model into a physical-level component model involves:
  - Selecting products and packages
  - Identifying frameworks and patterns
  - Deciding what components have to be developed
  - Making choices about what technology should be used to realize the components
- There can be more than one physical component model for each specified-level model.
- Strictly speaking, this notion of component implementation is implementation selection, to be followed by application development or package customization.



Module outline

Introduction and objectives

The functional aspect of IT architecture

Component modeling

Building a component model

- Identifying components
- Specifying components
- Implementing components

Summary and references

Summary

The functional aspect of a system's IT architecture can be described in various ways:

- Components
- Data
- Services

Component modeling describes the functional structure and behavior of the system's IT architecture via components.

- Components' capabilities are modeled in terms of what they can do, the activities they can perform and the information they are responsible for.
- Components work together through the exchange of messages sent and received via interfaces.

Component models can include:

- Static, structural views and dynamic, behavioral views
- Application and technical cross-cutting viewpoints
- Logical and physical cross-cutting viewpoints

Component modeling consists of:

- Identifying components
- Specifying components
- Transforming components

References

[BACHMANN00] Bachmann, et al. *Software Architecture Documentation in Practice: Documenting Architectural Layers*, SEI, March 2000

[BOOCH93] Booch, Grady. *Object Oriented Analysis and Design with Applications*, Benjamin Cummings, 1993, ISBN: 0805353402

[CHEESMAN00] Cheesman, John and Daniels, John. *UML Components*, Addison-Wesley, 2000, ISBN: 0201708510

[GAMMA95] Gamma, Erich, et al. *Design Patterns: Elements of Reusable Object Oriented Software*, Addison-Wesley, 1995, ISBN: 0201633612

[HERZUM99] Herzum, Peter and Sims, Oliver. *Business Component Factory - A Comprehensive Overview of Component-based Development for the Enterprise*, John Wiley, 1999, ISBN: 0471327603

[JACONSON94] Jacobsen, Ivar, et al. *Object Oriented Software Engineering: A Use Case Driven Approach*, Addison Wesley, 1994, ISBN: 0201544350

[MEYER97] Meyer, Bertrend. *Object-Oriented Software Construction (2nd Edition)*, Prentice-Hall, 1997, ISBN: 0136291554

[PAGE-JONES88] Page-Jones, Meiler. *The Practical Guide to Structured Systems Design*, Prentice Hall, 1988, ISBN: 0136907776

[WIRFS-BROCK90] Wirfs-Brock, Rebecca, et al. *Designing Object Oriented Software*, Prentice Hall, 1990, ISBN: 0136298257

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Other company, product, or service names may be trademarks or service marks of others.