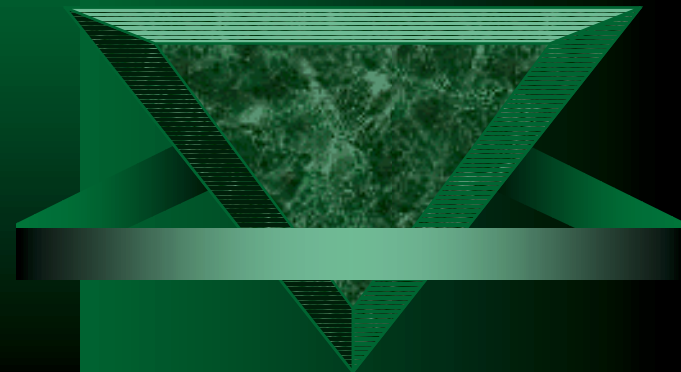


# Lesson 3: Mapping the Business Architecture to the Component-based Software Architecture

Ali Arsanjani

IBM and

Maharishi University  
of Management



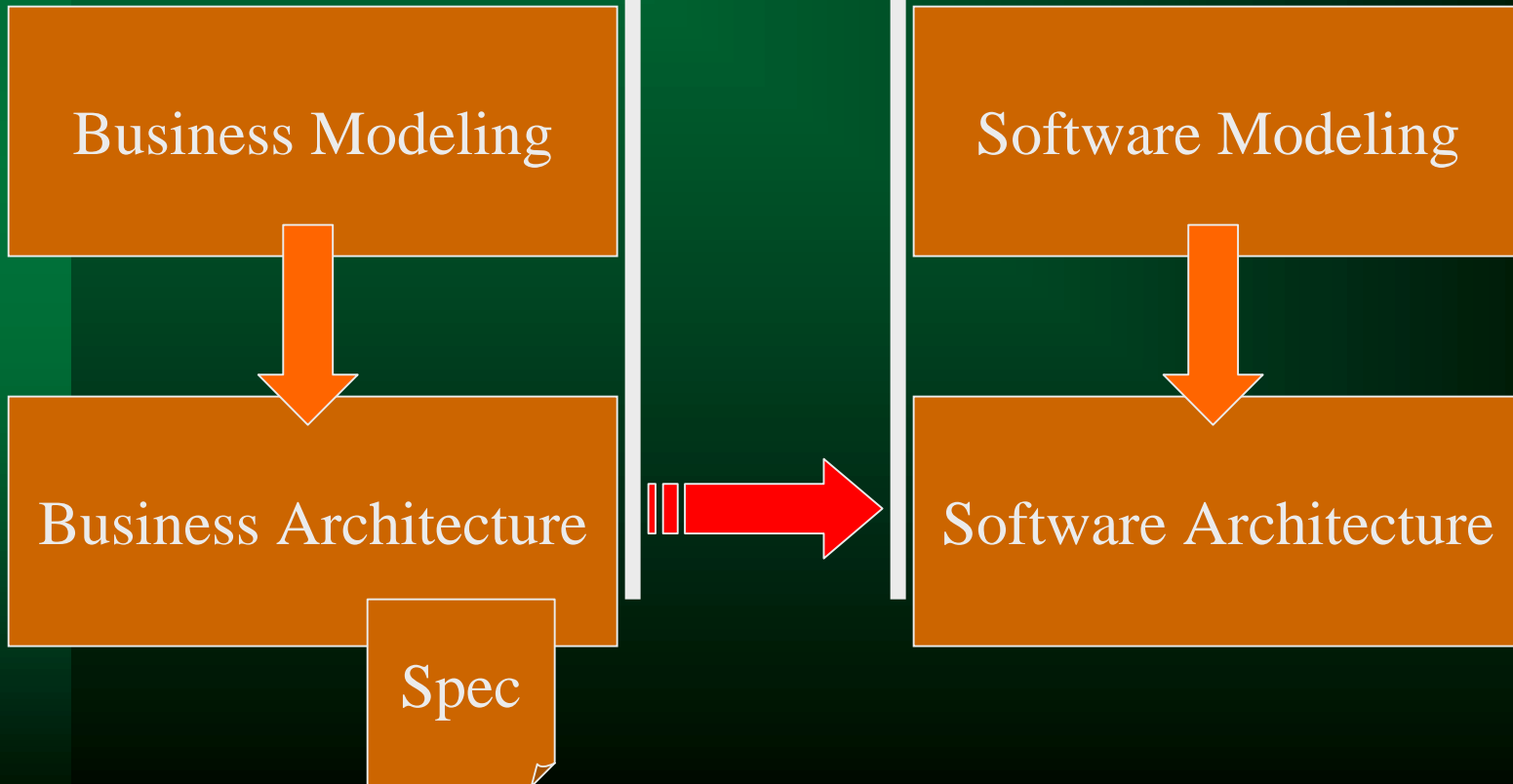
# Mapping Business Models to Software Architecture

Different Teams

The Gap

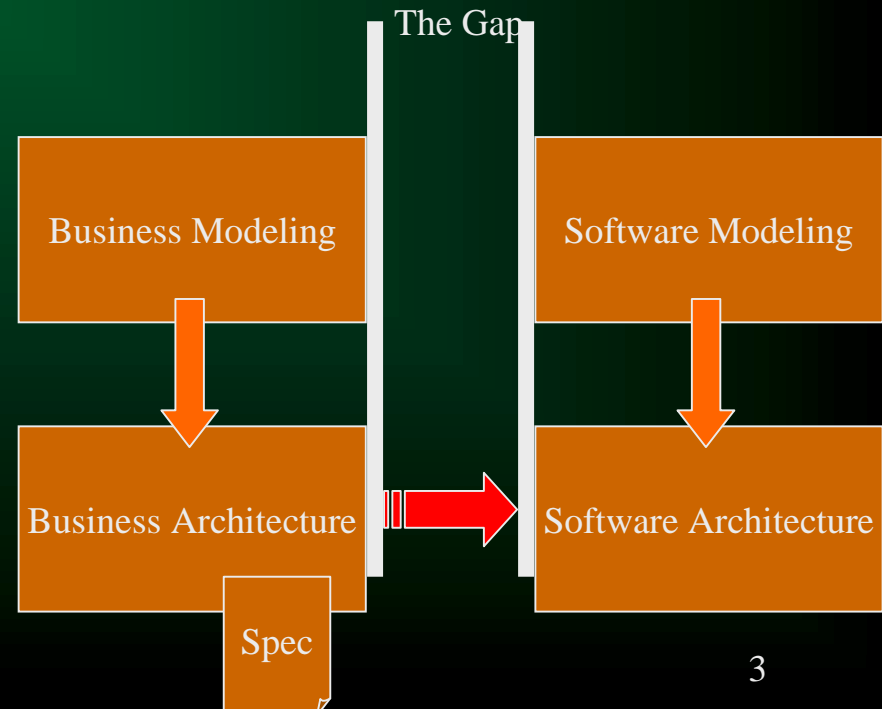
Different Notations

Conceptual Mismatch



# The Gap in Mapping Business Models to Software Architecture creates Problems

- The Gap is caused by:
  - Different Teams, Different Notations, Conceptual Mismatch
- The Gap results in:
  - Abandoned, delayed software projects
  - Software not meeting business requirements



# Where do Architectures come from?

**Input**

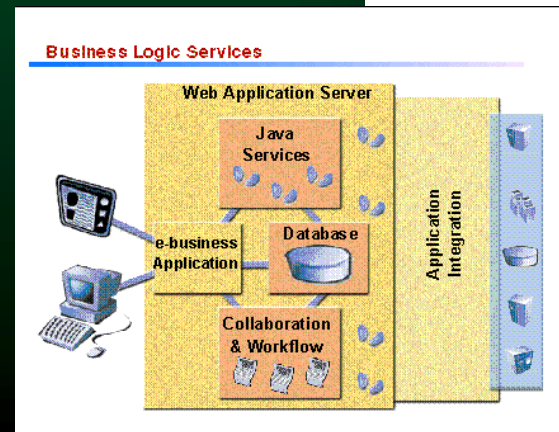
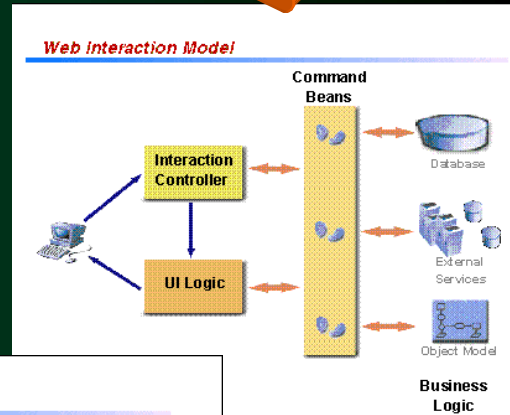
Implicit demands >> Explicit demands

Mind of Architect

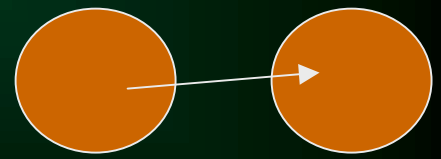
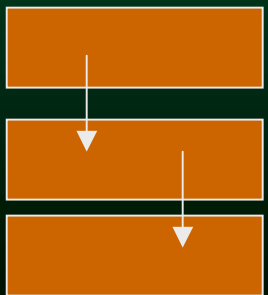
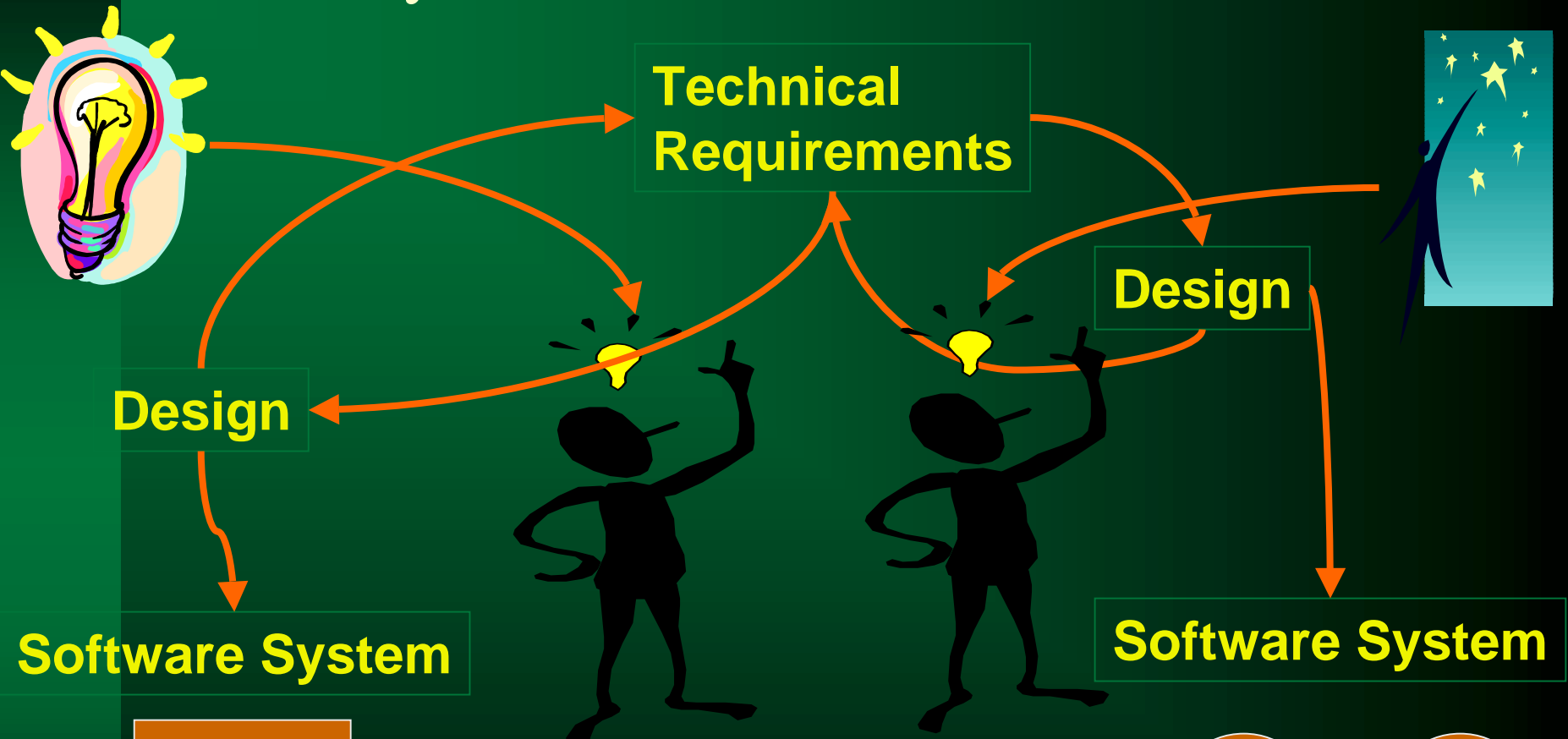
Produce  
Architecture

**Priorities**

1. Performance
2. Scalability
3. Security
4. Flexibility
5. Reliability
6. Functionality



“Will they create the same architecture?”

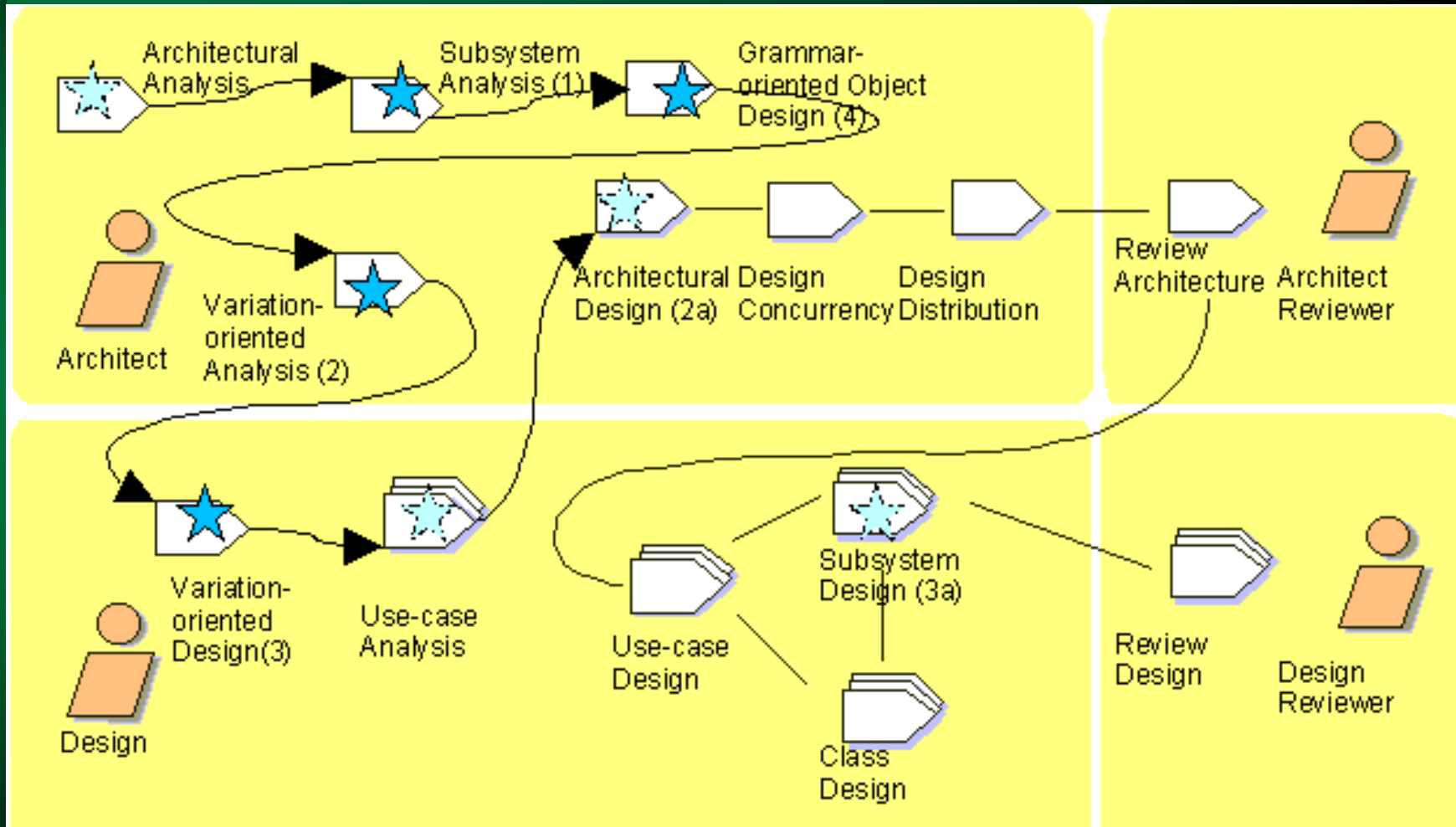


Technical requirements are only Part of the input...what are other inputs?

# Software Architecture

- ✔ ... Is the result of technical, business and social influences
- ✔ ... Balances forces between the functionality of software with the environment in which the system will be constructed and will operate
  - (Functional vs. Non-functional) vs. business drivers
  - (Features vs. “-abilities” ) vs. stakeholder vision
    - Reusability, scalability, security, performance, reliability, flexibility, quality
    - Vs
    - Time-to-market, image, branding, usability, business process, events, mergers, acquisitions,

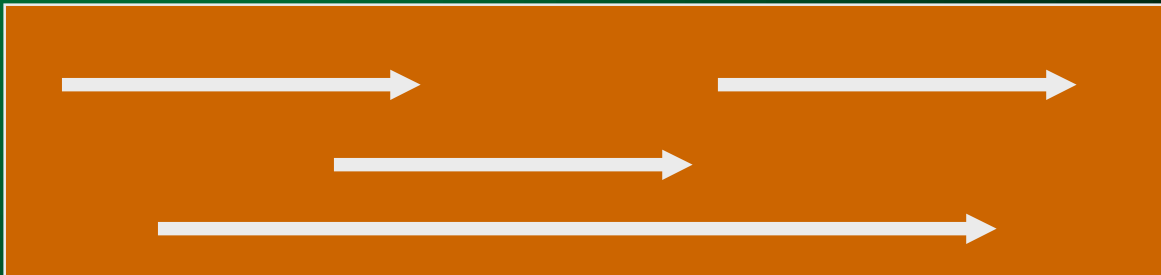
# Extensions to the Unified Process



# Extensions to UP for Business Modeling

## ✓ Domain Analysis

- What are the business processes in the domain?



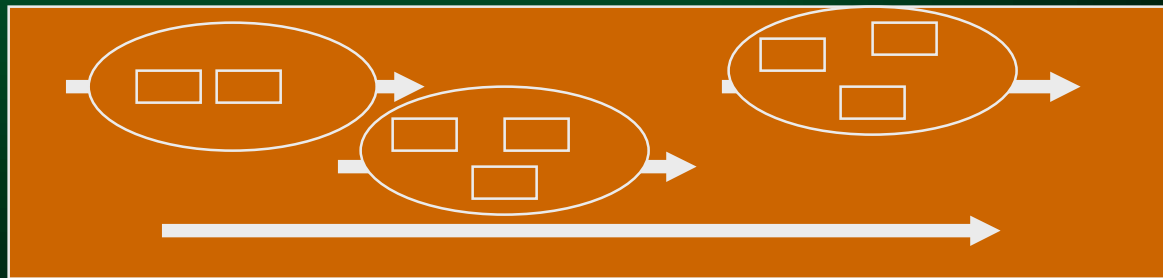
## ✓ Subsystem Analysis

- Analyze the business domain into its constituent subsystems
- Describe the rules governing the behavior of the subsystems among themselves
- Describe the rules governing the behavior of the internals of each subsystem

# Extensions to UP for Business Modeling

## Subsystem Analysis

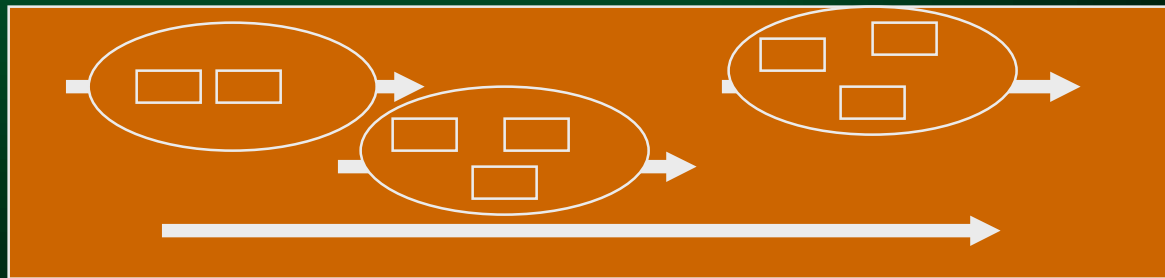
- Analyze the business domain into its constituent subsystems
- Describe the rules governing the behavior of the subsystems among themselves
- Describe the rules governing the behavior of the internals of each subsystem



# Extensions to UP for Business Modeling

Describe how the domain and the subsystems should behave

Let's examine a case study: the e-bazaar, an order management system over the web.



# Order-processing Subsystem Analysis

1. Customer Relationship Management = {Account Management, Contact Management (Addresses), Security, Customer Profile and Preferences}
2. Order Management={Order Entry {Shopping Cart Management}, Billing, Fulfillment}
3. Product Management = {Products, Catalogs, Pricing}
4. Inventory Management = {Fulfillment {Picking and Shipping}, Vendor Management}
5. Financial Management = {Billing, Accounting and Bookkeeping, Accounts Payable, Accounts Receivable}



# Make Online Purchase

## 1.1 Use-case Grammar

**Online Purchase =  
{[Identification], Presentation, Selection,  
Purchase [Identification], Confirmation, Order  
Fulfillment}**

# Make Online Purchase

## 1.1 Use-case Grammar

Online Purchase = {**Identification**, **Catalog Selection**,  
Selection, Purchase, Confirmation, Order Fulfillment}

**Identification** = {Challenge User with Login, Verify UserID and  
Password}

**Catalog Selection** = {Display Menu }

# Make Online Purchase

## 1.1 Use-case Grammar

**Online Purchase** = {Identification, Catalog Selection, **Selection**, Purchase, Confirmation, Order Fulfillment}

**Selection** = {Browse Product Catalog, Select a Product Item, **Shopping Cart Operation**, {Selection | Quit}}

**Shopping Cart Operation** = { {Add Item to Shopping Cart | Delete Item From Shopping Cart | Save Shopping Cart | [Shopping Cart Operation] }, **Checkout** }

**Checkout** = {Complete Order Info}

# Make Online Purchase

## 1.1 Use-case Grammar

Online Purchase = {Identification, Presentation, Selection, **Purchase**, Confirmation, Order Fulfillment}

**Purchase** = {Review Order, Review Terms of Agreement {Acknowledge Terms of Agreement, Submit Order | Cancel Order | Change Order to Quote}}.

# Make Online Purchase

## 1.1 Use-case Grammar

**Online Purchase = {Identification, Presentation, Selection, Purchase, Confirmation, Order Fulfillment}**

**Confirmation = {Send confirmation number to user}**

**Order Fulfillment = {Pick and Ship Order}**

# Make Online Purchase Use-case Grammar

**Online Purchase = {Identification, Presentation, Selection, Purchase, Confirmation, Order Fulfillment}**

**Identification = {Challenge User with Login, Verify UserID and Password}**

**Presentation = {Display Menu}**

**Selection = {Browse Product Catalog, Select a Product Item, Shopping Cart Operation, Selection}**

**Shopping Cart Operation = { {Add Item to Shopping Cart | Delete Item From Shopping Cart | Save Shopping Cart | [Shopping Cart Operation] }, Checkout }**

**Checkout = {Complete Order Info}**

**Complete Order Info = {{Verify Billing and Shipping Address| Select Billing and Shipping Addresses}, {Verify Shipping Method | Select Shipping Method}}.**

**Purchase = {Review Order, Review Terms of Agreement {Acknowledge Terms of Agreement, Submit Order | Cancel Order | Change Order to Quote}}.**

**Confirmation = {Send confirmation number to user}**

**Order Fulfillment = {Pick and Ship Order}**

# Make Online Purchase Use-case Grammar

**Online Purchase = {Identification, Presentation, Selection, Purchase, Confirmation, Order Fulfillment}**

**Identification = {Challenge User with Login, Verify UserID and Password}**

**Presentation = {Display Menu}**

**Selection = {Browse Product Catalog, Select a Product Item, Shopping Cart Operation, Selection}**

**Shopping Cart Operation = { {Add Item to Shopping Cart | Delete Item From Shopping Cart | Save Shopping Cart | [Shopping Cart Operation] }, Checkout }**

**Checkout = {Complete Order Info}**

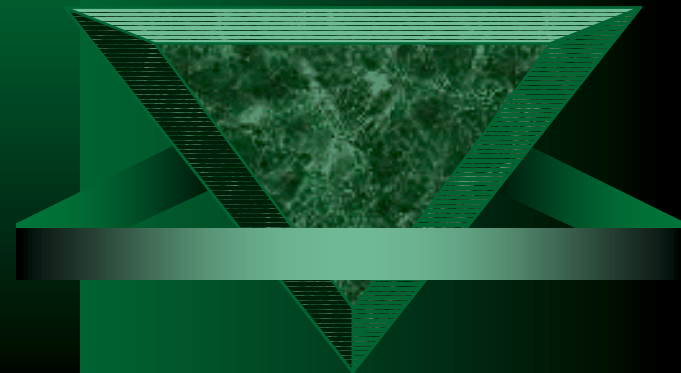
**Complete Order Info = {{Verify Billing and Shipping Address| Select Billing and Shipping Addresses}, {Verify Shipping Method | Select Shipping Method}}.**

**Purchase = {Review Order, Review Terms of Agreement {Acknowledge Terms of Agreement, Submit Order | Cancel Order | Change Order to Quote}}.**

**Confirmation = {Send confirmation number to user}**

**Order Fulfillment = {Pick and Ship Order}**

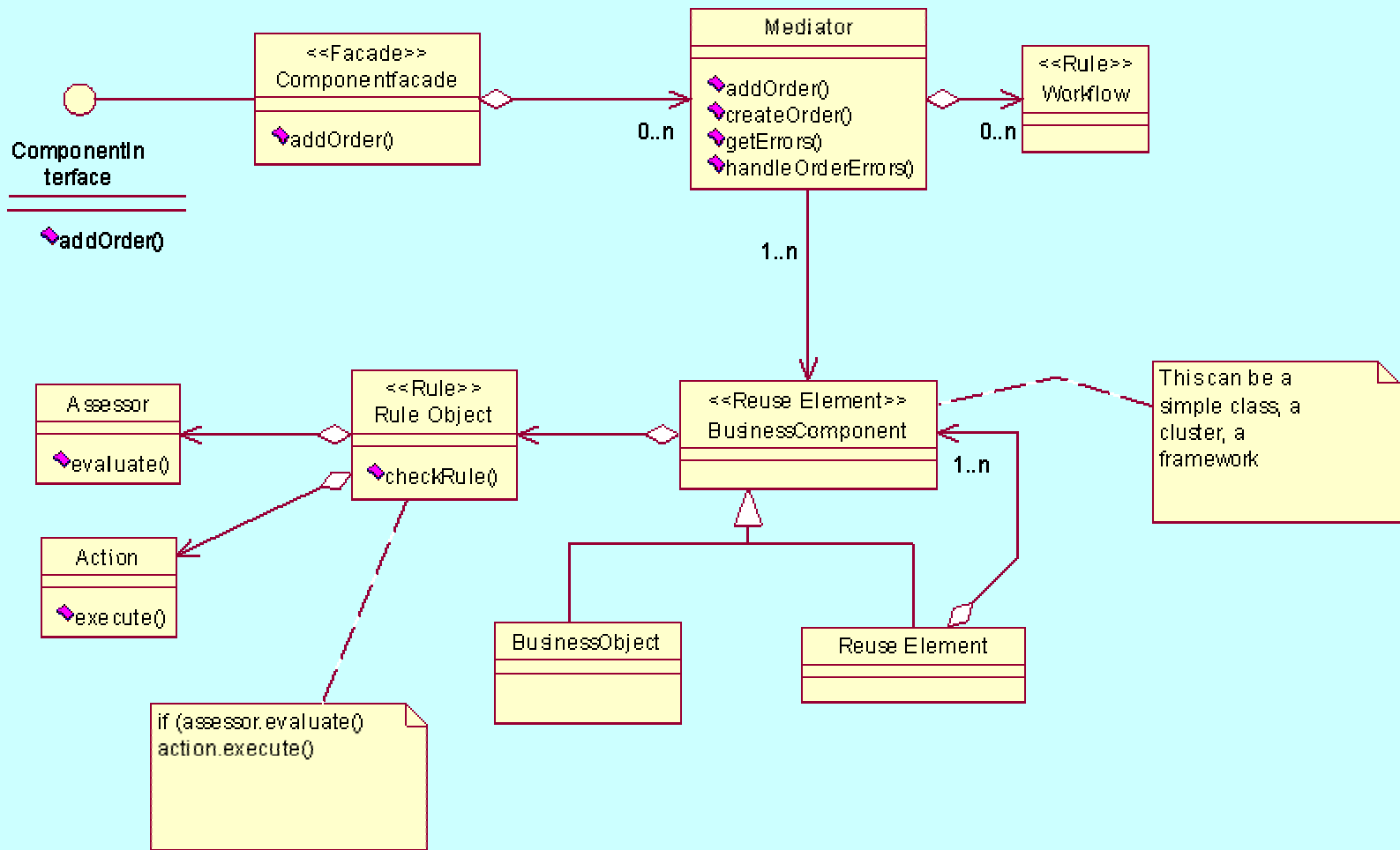
# Problems in Component Engineering



# What should become a component?

- ✓ Proposition: Subsystems should become components
  - But how do I find subsystems in a business domain?
  - Component / Subsystem Discovery
  - → we need activities in OO methods to handle this
    - **Subsystem Analysis**
- ✓ Currently: Inadequate domain partitioning into subsystems
  - What are the large-grained subsystems that collaborate to achieve business goals?
  - Subsystems are then implemented as components

# Enterprise Component



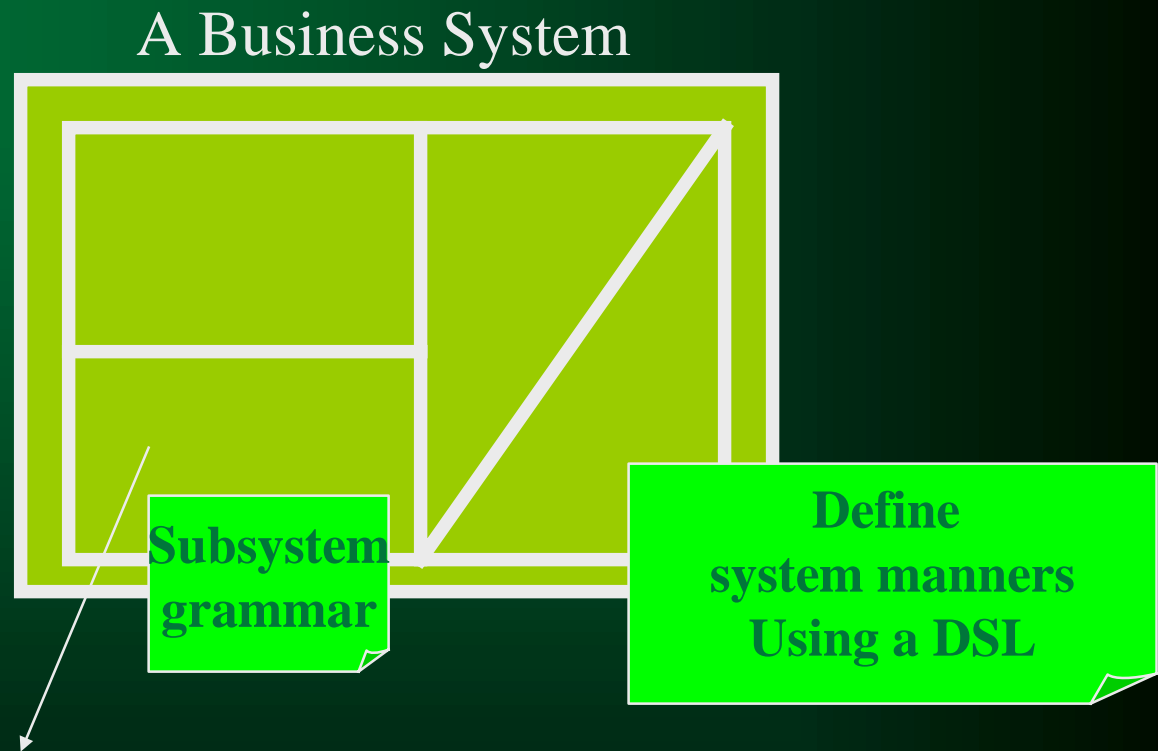
# Component Interfaces are Inadequate

- ✔ Now we have a semantic notion of a business subsystem: how do we precisely describe it and use it?
- ✔ *Manners*: Need a precise description of how the system behaves in a given context
  - Business Rules
  - Context
  - Meta-data
  - Events
- ✔ Need to know how a subsystem behaves in a given domain context
  - There are many ways of using a subsystem; what is a well-mannered subsystem
  - Subsystems and components have “manners”
  - Manners are rules governing system behavior

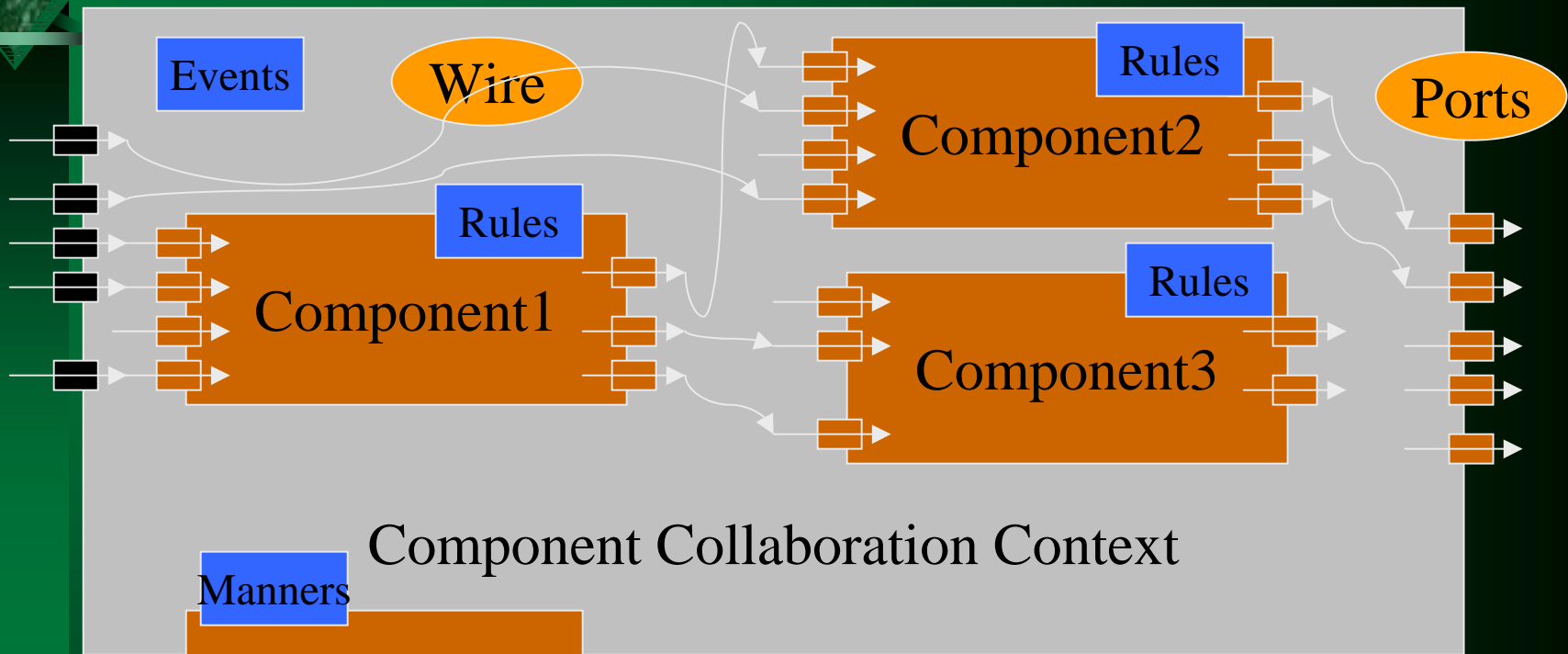
# Component Granularity

- ✓ Components are currently too fine grained
- ✓ We need medium- to large-grained components
- ✓ From a methodology point of view, what level of granularity should we commence modeling and designing components?
  - Objects?
    - Too fine-grained
  - There are multiple reuse levels
    - One starting point is the subsystem level

# A Partitioned Business System with Manners



A Business System Partitioned into its Subsystems.  
Subsystems collaborate using interfaces and contracts  
Based on their *manners*.



## Component Collaboration Context

### Manners

Business Domain  
Language  
 $S \rightarrow P, Q, T.$   
 $P \rightarrow Q, R, P \mid \text{null}.$   
 $Q \rightarrow \text{abc} \mid \text{bdg}.$   
 $T \rightarrow \text{ab} Q \mid \text{bc} P.$

A Business Subsystem becomes a component.

Subsystems collaborate using interfaces and contracts

Based on their *manners*.

Manners govern events, rules, context

# Grammar-oriented Object Design (GOOD)

- ✔ Grammar-oriented Object Design is the process of Designing Component-based software architectures for adaptive business systems by capturing the manners of such systems and their subsystems in domain-specific languages (DSLs)
- ✔ GOOD bridges the gap between business modeling and software architecture by providing a semantics and syntax that can be used in business modeling and trickled down to build software architectures

# Grammar-oriented Object Design (GOOD)

- ✓ GOOD helps create adaptive component-based enterprise-scale business applications that
  - Can be dynamically [re-]configured
  - Can change a collaboration (workflow) dynamically
  - Can provide contextual and meta-information about its business rules (self-description)
- ✓ This allows medium- to large-grained components to be customized and re-assembled into new applications
- ✓ To negotiate with other applications with minimal human intervention
  - The protocols of interaction and business rules are declaratively expressed in a domain-specific language captured by each subsystem's “manners”

# Grammar-oriented Object Design (GOOD)

- ✓ GOOD helps span the gap between business and software architecture

# Two layers of implementing GOOD

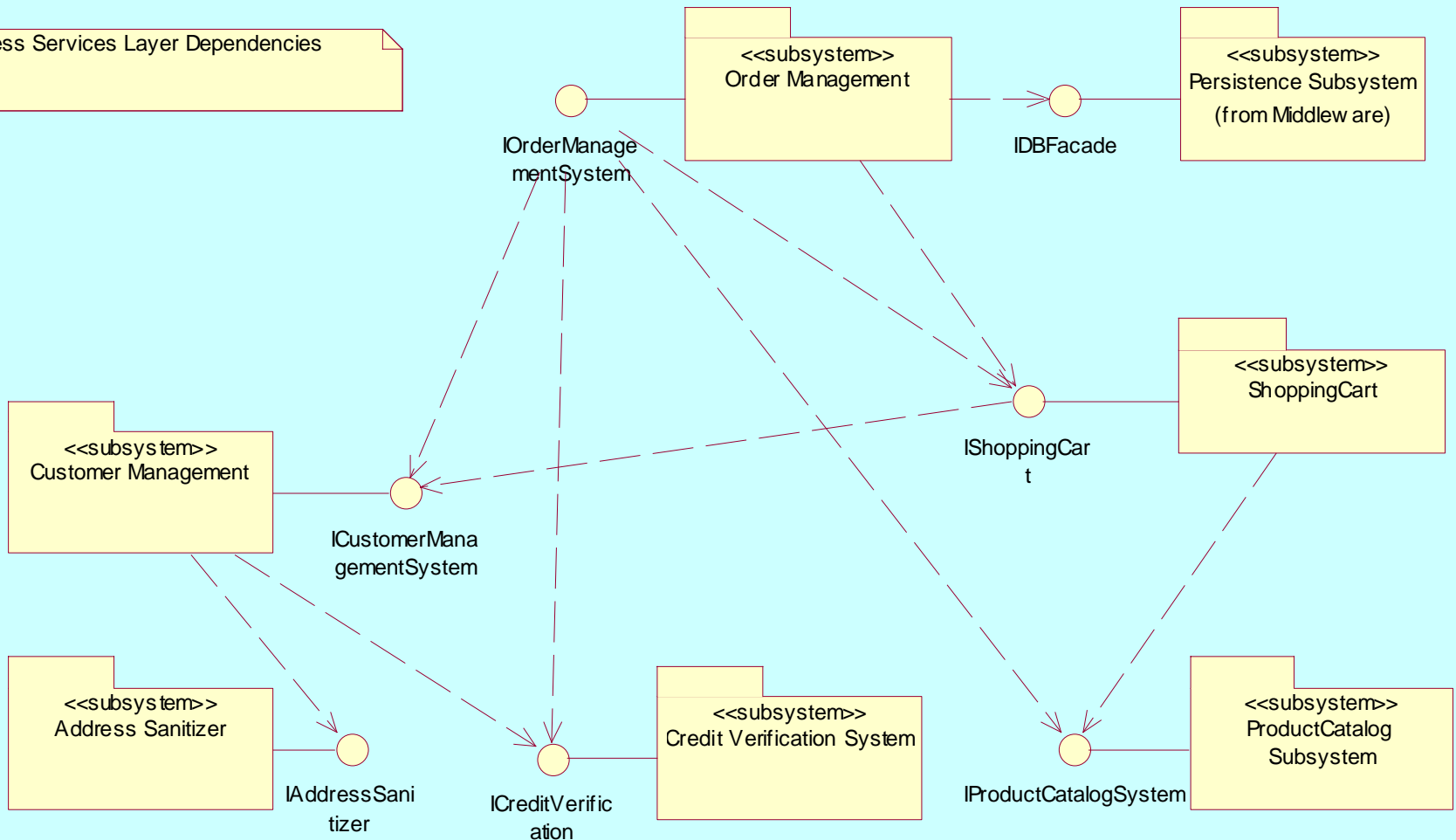
- ✓ Methodological
- ✓ Architectural

# OO Method Extensions to Support GOOD

- ✔ A blueprint of a business domain (e.g., banking) is analyzed using *subsystem analysis*
  - Business domain is partitioned into dynamically configurable, collaborating enterprise components exposing services
  - Components also expose their “manners”
- ✔ Domain variation points are identified using *variation-oriented design*
  - For each we have a dynamic collaboration, which defies how it behaves “this time”
- ✔ Its common points are expressed as a domain-specific language for which a grammar can be written to be executed within a subsystem’s workflow manager component

# Order-Processing Subsystems

Business Services Layer Dependencies



# Manners for Subsystems: Use-Case Grammar

- ✔ Online Purchase = {Identification, Presentation, Selection, Purchase, Confirmation, Order Fulfillment}
- ✔ Identification = {Challenge User with Login, Verify UserID and Password}
- ✔ Presentation = {Display Menu}
- ✔ Selection = {Browse Product Catalog, Select a Product Item, Shopping Cart Operation, Selection}
- ✔ Shopping Cart Operation = { {Add Item to Shopping Cart | Delete Item From Shopping Cart | Save Shopping Cart | [Shopping Cart Operation] }, Checkout }
- ✔ Checkout = {Complete Order Info}
- ✔ Complete Order Info = {{Verify Billing and Shipping Address| Select Billing and Shipping Addresses}, {Verify Shipping Method | Select Shipping Method}}.
- ✔ Purchase = {Review Order, Review Terms of Agreement {Acknowledge Terms of Agreement, Submit Order | Cancel Order | Change Order to Quote}}.
- ✔ Confirmation = {Send confirmation number to user}
- ✔ Order Fulfillment = {Pick and Ship Order}

# Conclusion

- ✔ Helps span the conceptual gap between the business and software architecture
- ✔ Based on extensions to current methods such as the Unified Process and to modeling languages such as the UML
- ✔ Based on patterns and best-practices for building robust software architectures.

# Component with Manners: Dynamic Rules

