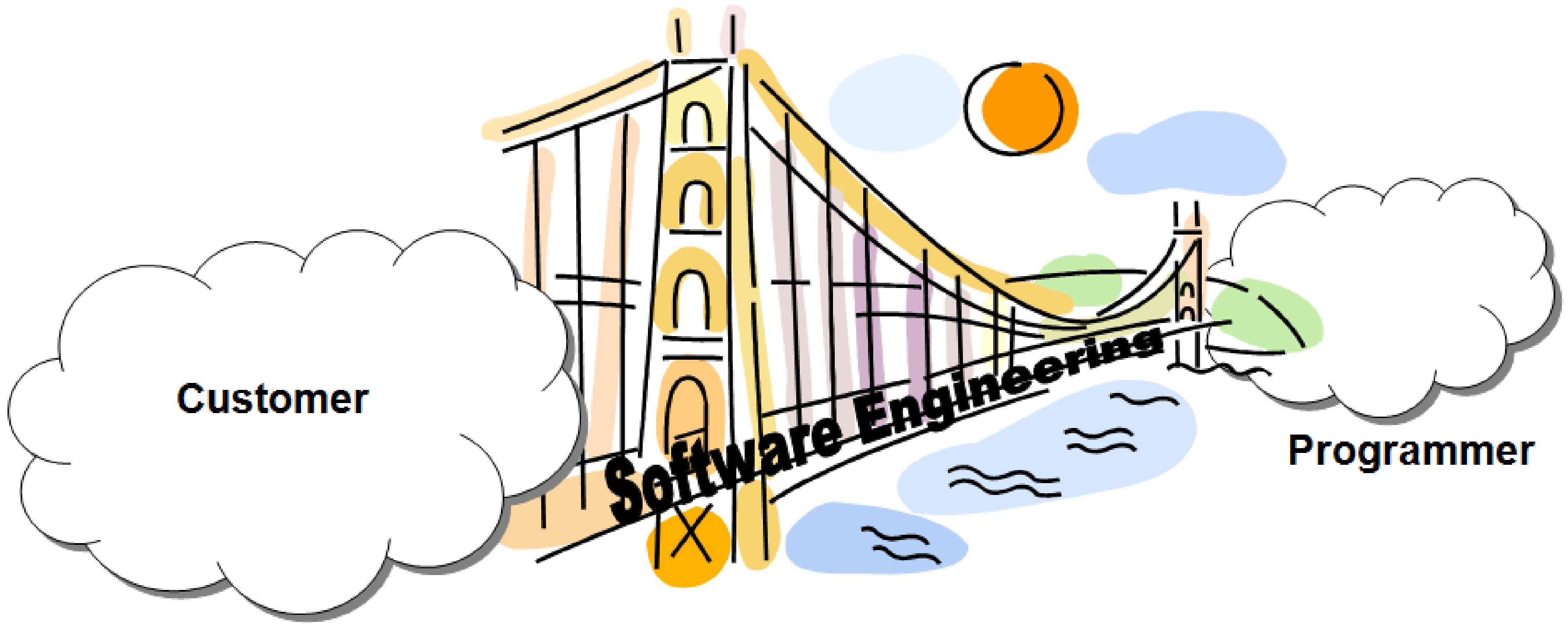
# Architecture Design

何明昕　He Mingxin, Max
Email: c.max @ yeah.net　(课程专用)
下载邮箱: c.program@ yeah.net

# The Role of Software Engg. (1)

A bridge from customer needs to programming implementation



**First law of software engineering**

**Software engineer is willing to learn the problem domain**
(problem cannot be solved without understanding it first)

## Software engineering

- Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.
- Engineering discipline
  - Using appropriate theories and methods to solve problems bearing in mind organizational and financial constraints.
- All aspects of software production
  - Not just technical process of development. Also project management and the development of tools, methods etc. to support software production.

## Software process activities

- Software specification, where customers and engineers define the software that is to be produced and the constraints on its operation.
- Software development, where the software is designed and programmed.
- Software validation, where the software is checked to ensure that it is what the customer requires.
- Software evolution, where the software is modified to reflect changing customer and market requirements.

## Software is Complex

Complex (错综复杂的) ≠ complicated (复杂难懂的)

Complex = composed of many simple parts
related to one another

Complicated = not well understood, or explained

# Software Engineering is Complex and Complicated

1) 软件规模增长带来要素数量增长，熵增长必然导致混乱（热力学第三定律）

2) 准确定义需求很难（用户通常不知道要什么，只知道不要什么），需求蔓延是软件项目失败的首要原因

3) 不同的设计可以实现同样的功能(系统结构/行为/功能的关系），判断一个设计是否是好的设计很难（Conway's law: 软件的结构倾向于与开发团队的组织架构具相似性)

$$\frac{1}{n}\sin x = ?$$

$$\frac{1}{\cancel{n}}\mathrm{si}\cancel{\mathrm{n}}\, x =$$

$$six = 6$$

After explaining to a student through various lessons and examples that:

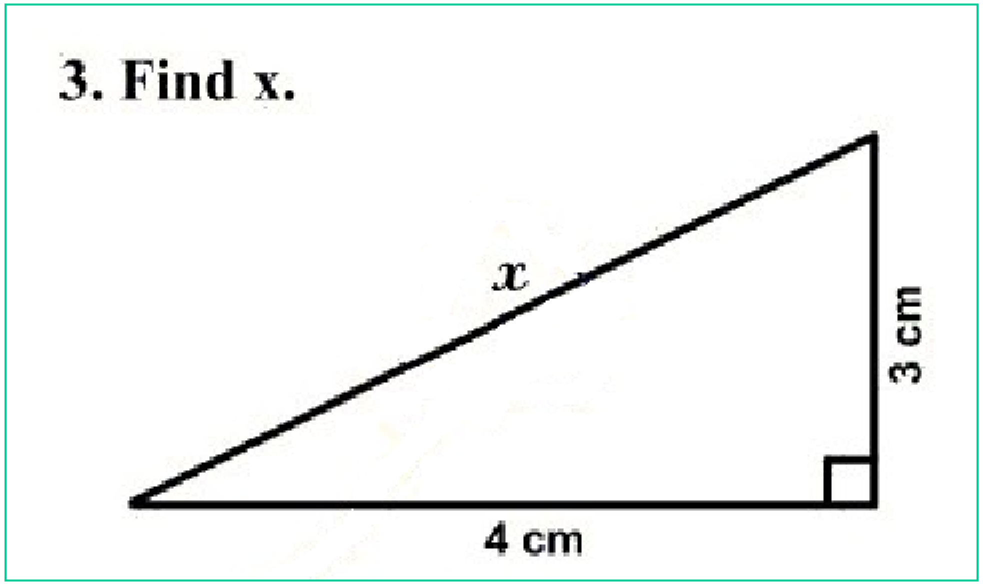
$$\lim_{x \to 8} \frac{1}{x-8} = \infty$$

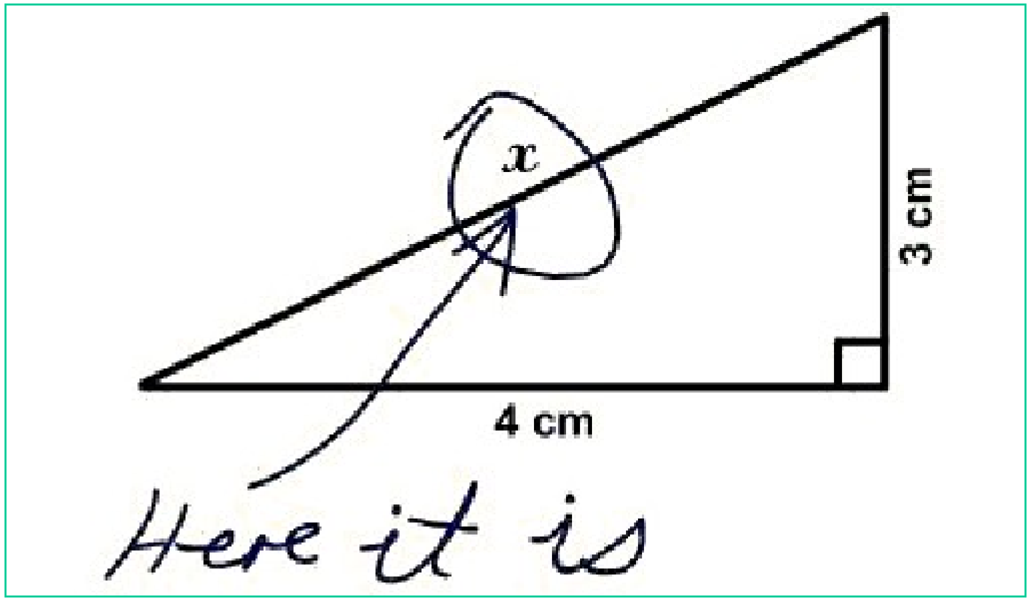
I gave a different example.

$$\lim_{x \to 5} \frac{1}{x-5} = ?$$

This was the result:

$$\lim_{x \to 5} \frac{1}{x-5} = \rotatebox{90}{5}$$


3. Find x.
$x$
3 cm
4 cm

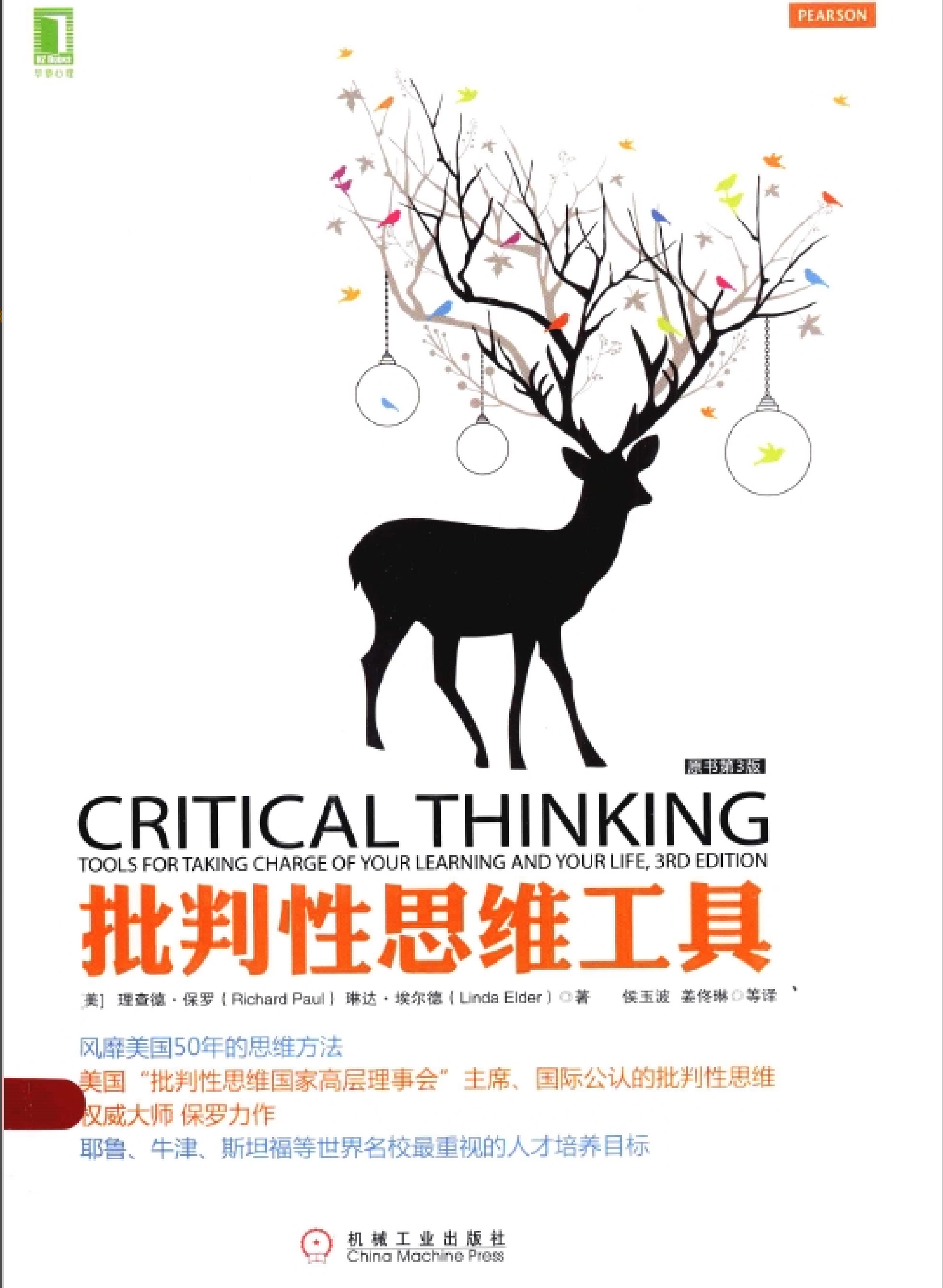

$x$
3 cm
4 cm
Here it is

# Critical Thinking

- 批判性思维
  关键性思考
  建设性思考

- Critical Observing:
  建设性观察

- 系统化认知问题 与
  构建解决方案的能力



PEARSON

原书第3版

CRITICAL THINKING

TOOLS FOR TAKING CHARGE OF YOUR LEARNING AND YOUR LIFE, 3RD EDITION

批判性思维工具

[美] 理查德·保罗（Richard Paul） 琳达·埃尔德（Linda Elder）◎著　　侯玉波 姜佟琳◎等译

风靡美国50年的思维方法

美国“批判性思维国家高层理事会”主席、国际公认的批判性思维权威大师 保罗力作

耶鲁、牛津、斯坦福等世界名校最重视的人才培养目标

机械工业出版社
China Machine Press

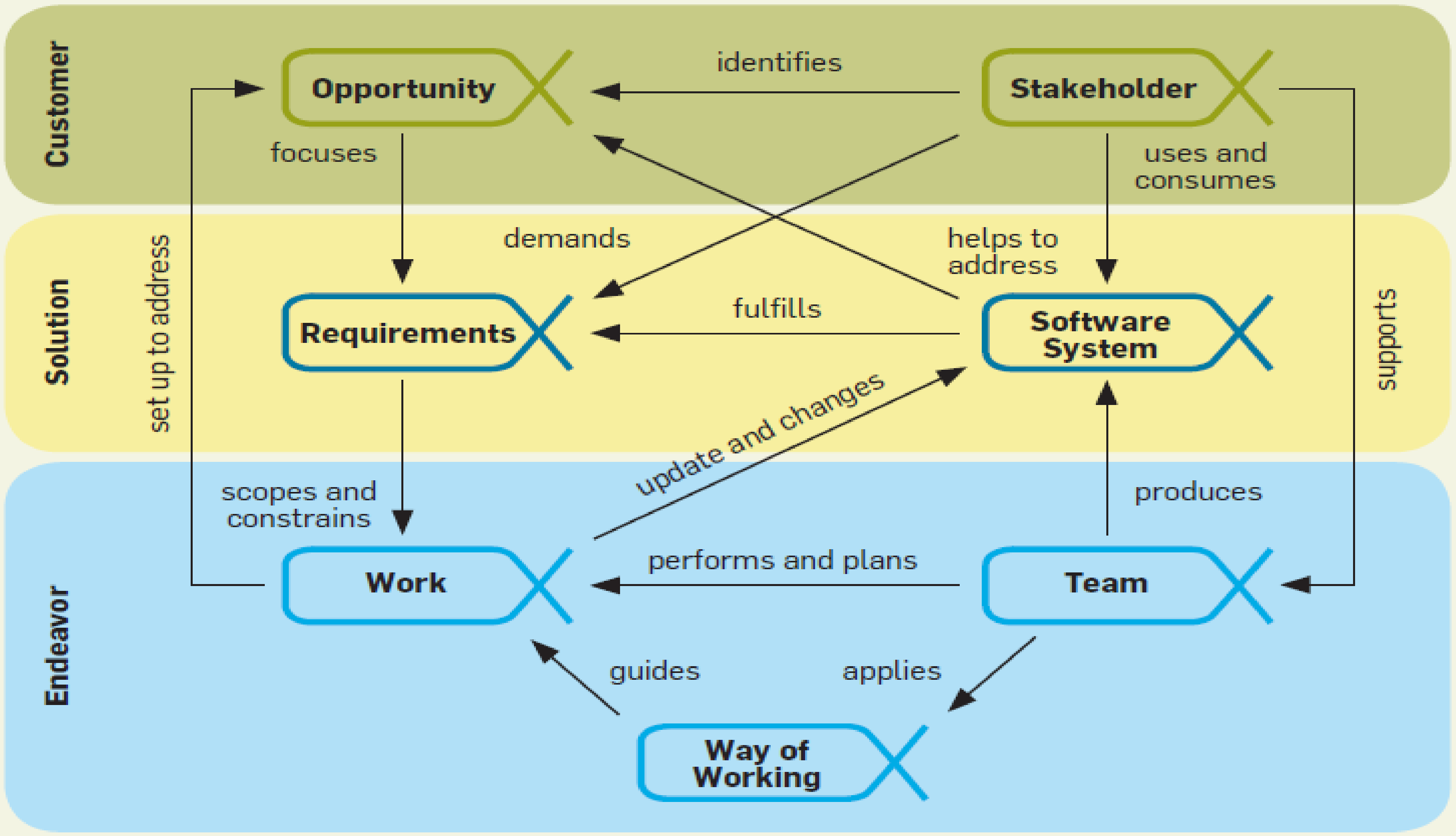
## SEMAT http://semat.org

- SEMAT (Software Engineering Method and Theory) drives a process to refound software engineering based on a solid theory, proven principles and best practices.

# The Essence of Software Engineering: The SEMAT Kernel

By Ivar Jacobson

Figure 1. Things to work with.



# The SEMAT Kernel (cont.)

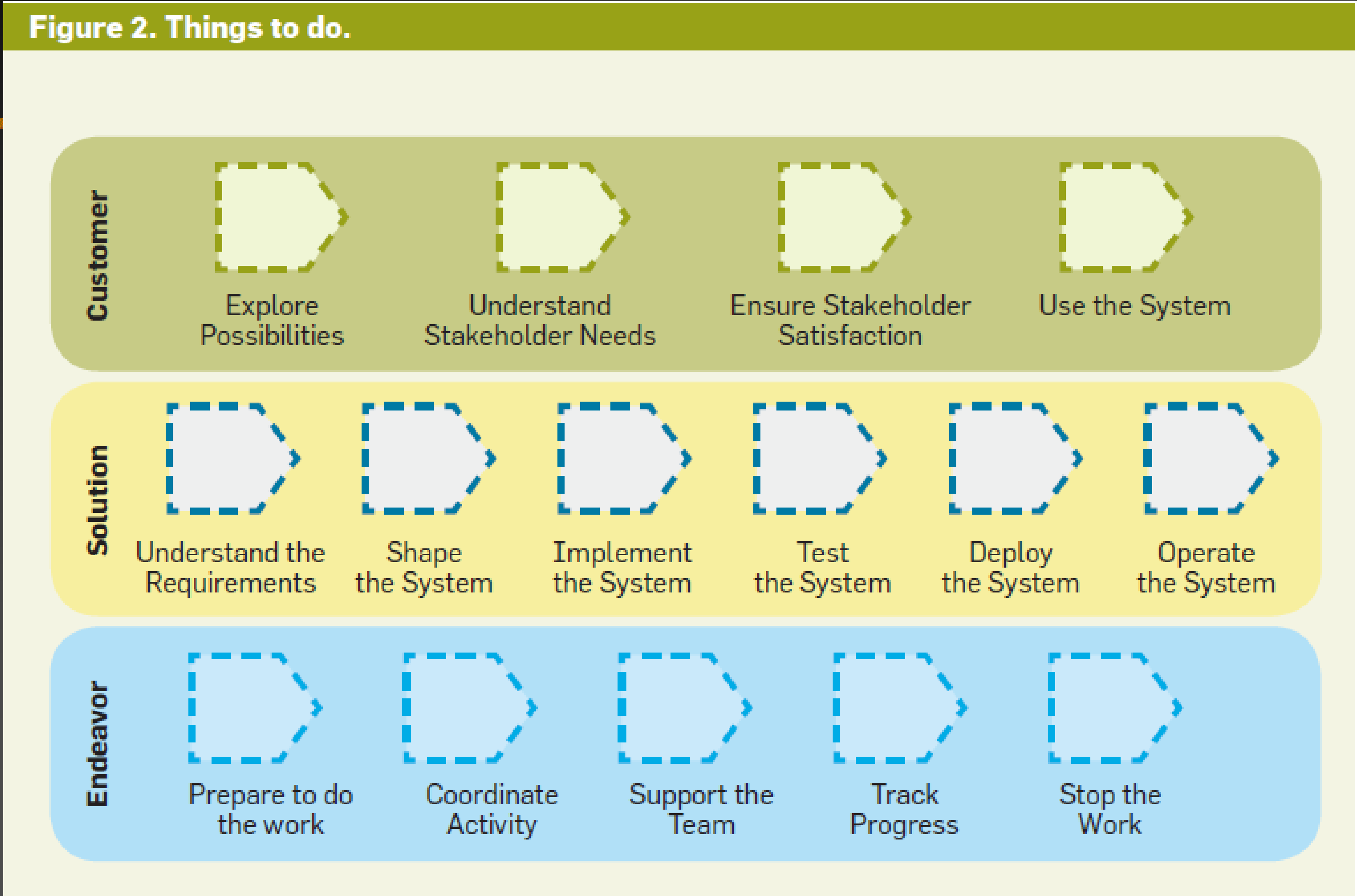


Figure 2. Things to do.

## 软件(程序)的三项职责(Robert Martin)

第一个职责是它要有效运行满足用户需要的功能。这也是该模块得以存在的原因。

第二个职责是它要应对变化。几乎所有的模块在它们的生命周期中都要变化，开发者有责任保证这种改变应该尽可能地简单。一个难以改变的模块，即使能够工作，也需要对它进行修正。

第三个职责是要和阅读它的人进行沟通。对该模块不熟悉的开发人员应该能够比较容易地阅读并理解它。一个无法进行沟通的模块也是拙劣的，同样需要对它进行修正。

## 好的软件(程序)的特征：

第一：能正确有效地运行；

第二：能容易地扩展和修改；

第三：能容易地与人沟通(便于交流)。

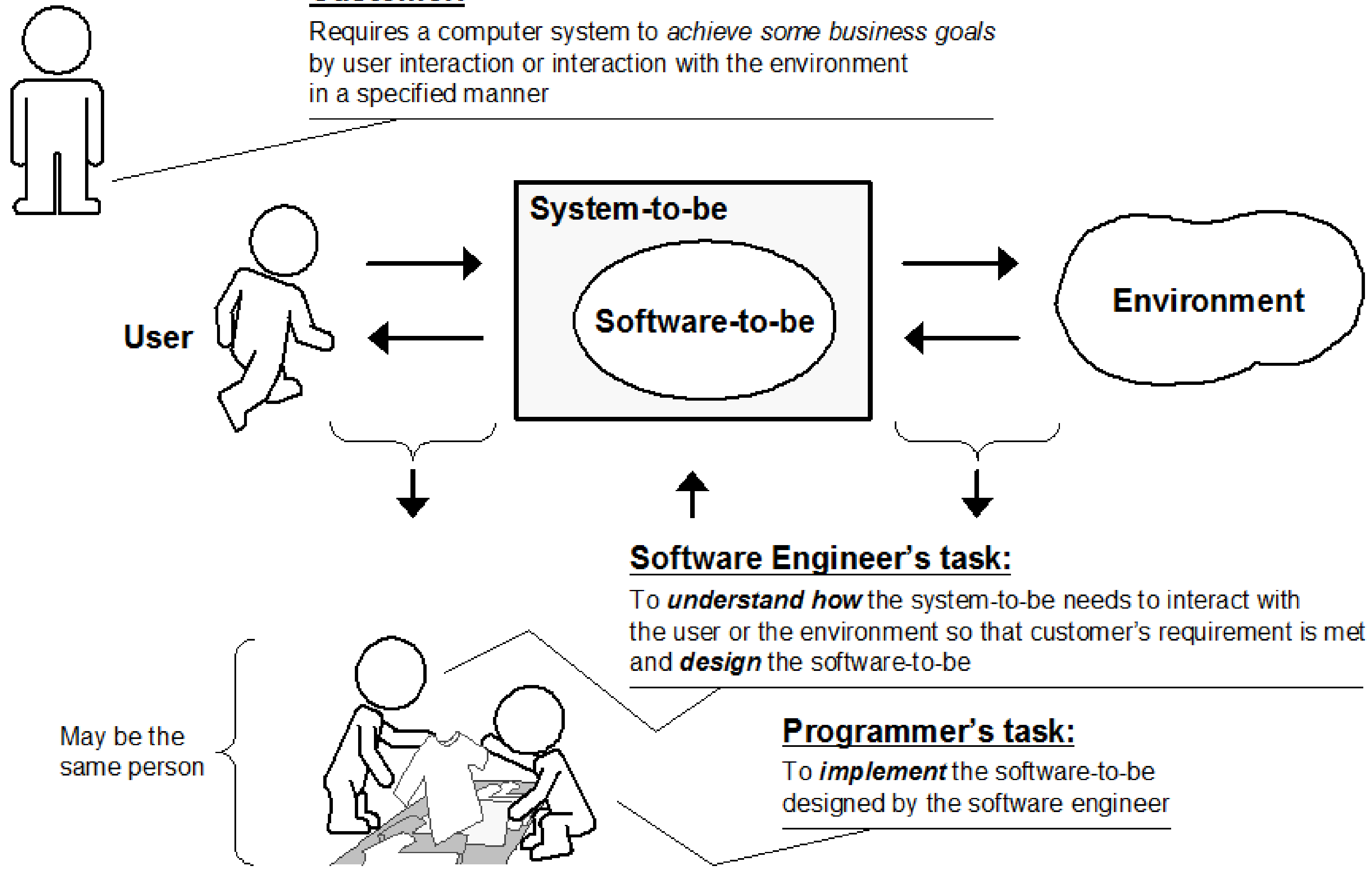
# Essential attributes of good software

| Product characteristic | Description |
|---|---|
| Maintainability | Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment. |
| Dependability and security | Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system. |
| Efficiency | Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc. |
| Acceptability | Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use. |

# The Role of Software Engg. (2)

**Customer:**

Requires a computer system to *achieve some business goals* by user interaction or interaction with the environment in a specified manner



**Software Engineer's task:**

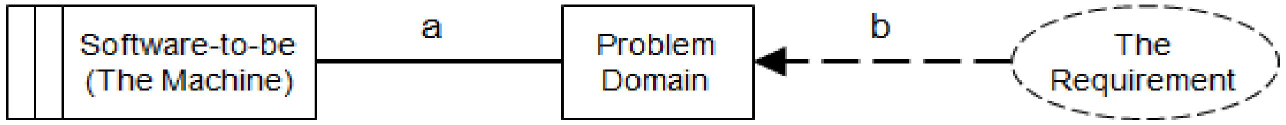
To ***understand how*** the system-to-be needs to interact with the user or the environment so that customer's requirement is met and ***design*** the software-to-be

**Programmer's task:**

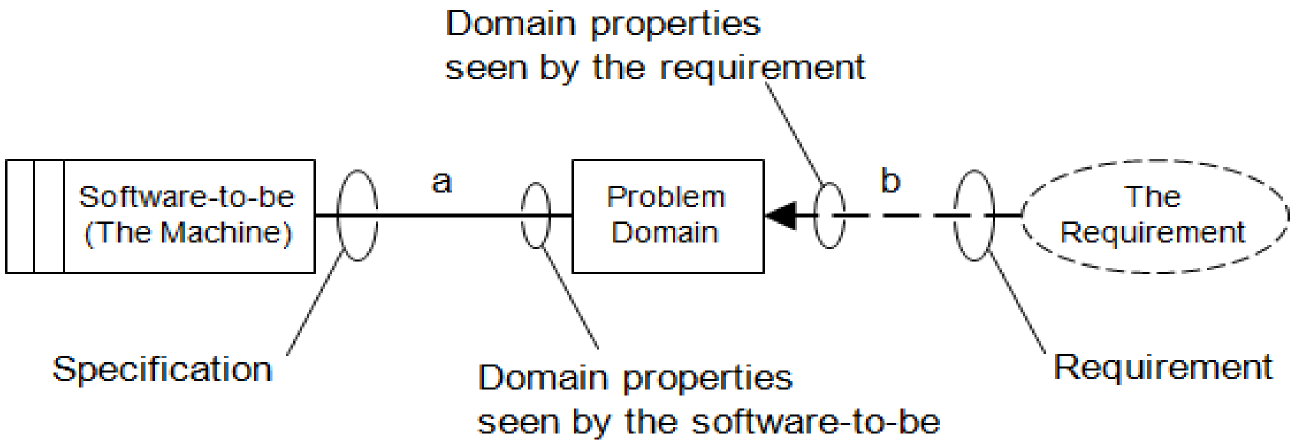
To ***implement*** the software-to-be designed by the software engineer

# Machine and Problem Domain

(a)



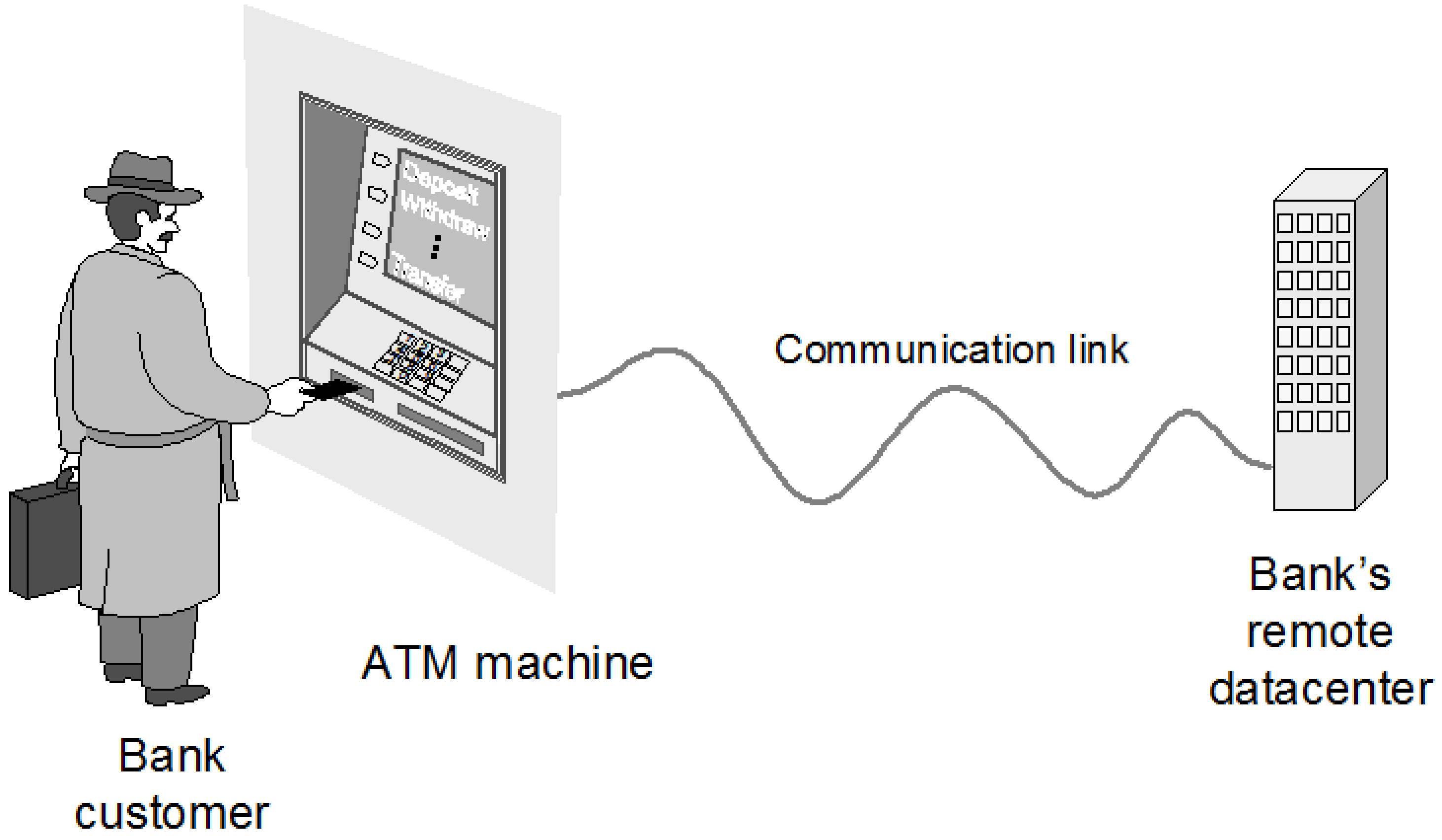
(b)



a: specification interface phenomena
b: requirement interface phenomena

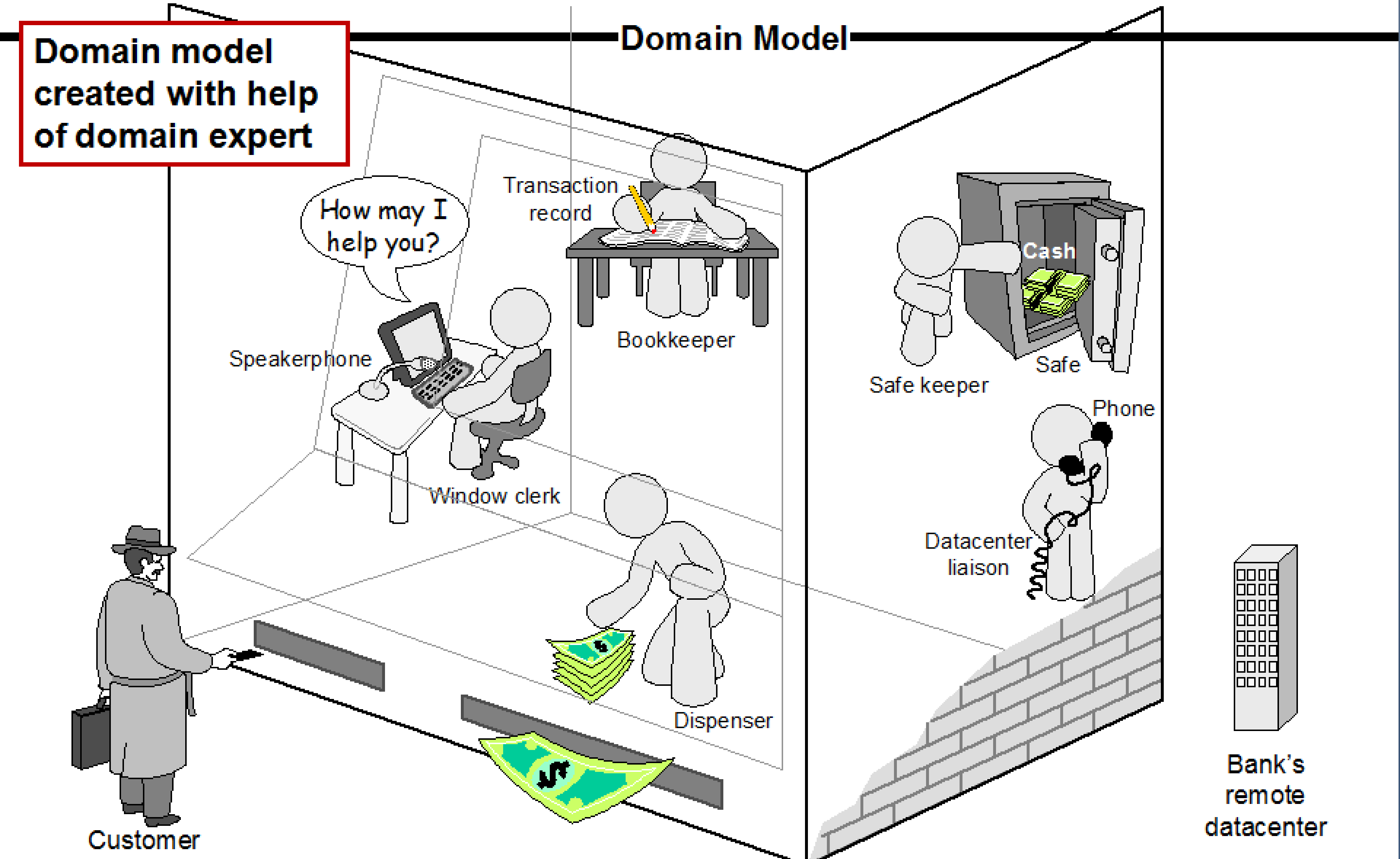
# Example: ATM Machine

**Understanding the money-machine problem:**

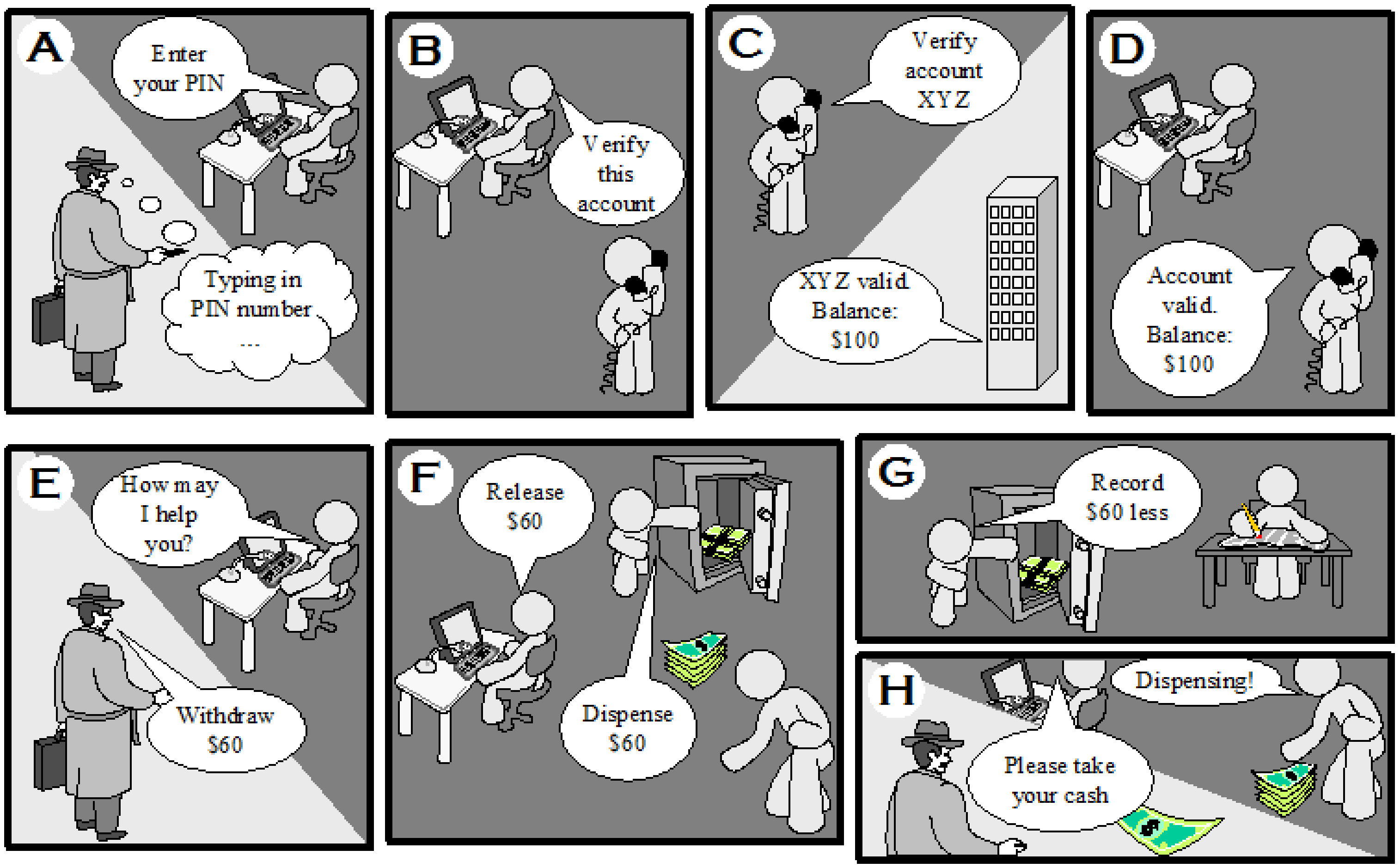


# How ATM Machine Might Work

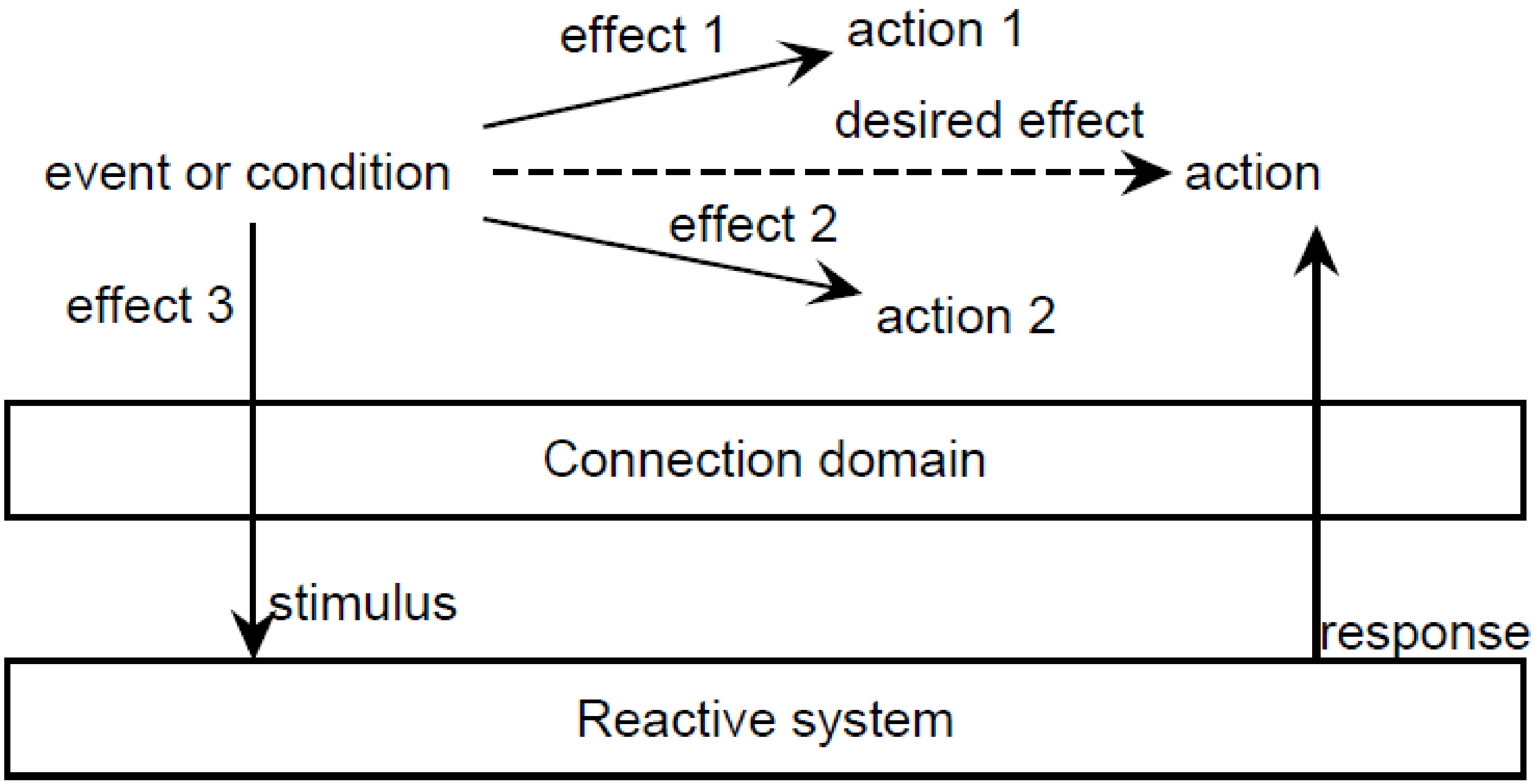
**Domain model created with help of domain expert**



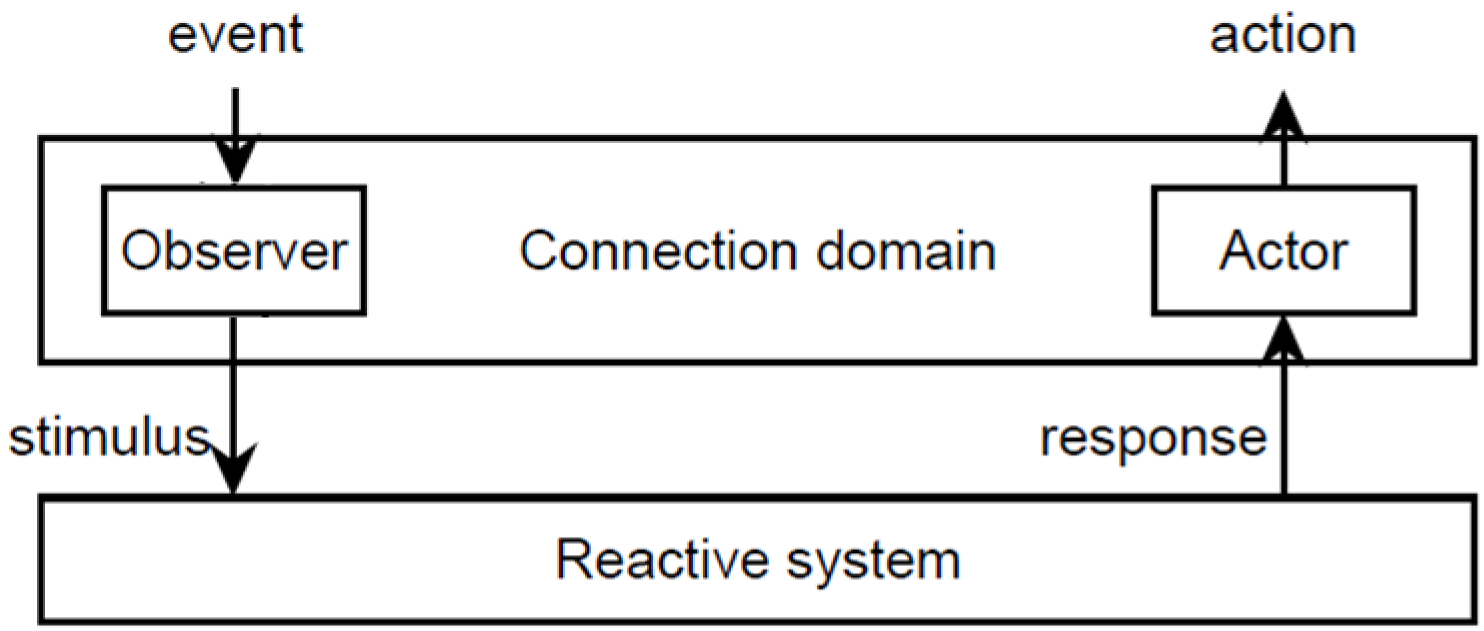
# Cartoon Strip: How ATM Machine Works



## Structure of stimulus-response behavior



## Observers and actors are in the connection domain



# What is Analysis?

- Analysis is a broad term.  In Software development, we are primarily concerned with two forms of analysis.
- Requirements Analysis is discovering the requirements that a system must meet in order to be successful.
- Object Analysis is investigating the object in a domain to discover information important to meet the requirements.

## What is Design?

- Design emphasizes a conceptual solution that fulfills the requirements.  A design is not an implementation, although a good design can be implemented when it is complete.
- There are subsets of design, including architectural design, object design, and database design.

## Analysis and Design

Focus on Make decisions (choices)

- Analysis
  - *do the right things*
- Design
  - *Do the things right*

## 软件设计的五个层面：

1) 产品/服务市场架构设计(定位与设想)
2) 系统(软件)架构设计
3) 软件边界(界面或接口)设计
4) 信息(数据)结构及存取设计
5) 业务逻辑(处理/计算)设计

## What is a good object design?

Applying UML and Patterns in OOA/OOD

## Applying UML

- UML is just a standard diagramming notation.  It is just a tool, not a skill that is valuable in itself.  Knowing UML helps you communicate with others in creating software, but the real work in this course is learning Object-Oriented Analysis and Design, not how to draw diagrams.

## Assigning Responsibilities

- The most important skill in Object-Oriented Analysis and Design is assigning responsibilities to objects.  That determines how objects interact and what classes should perform what operations.

## Design Patterns

- Certain tried-and-true solutions to design problems have been expressed as principles of best practice, often in the form of ***Design Patterns***.
- A Design Pattern is a named problem solution formula that apply excellent design principles.

## Requirements Analysis

- All Software Analysis and Design is preceded by the analysis of requirements.
- One of the basic principles of good design is to defer decisions as long as possible. The more you know before you make a design decision, the more likely it will be that the decision is a good one.
- TFCL: ***Think First, Code Later!***

## Use Cases

- Writing Use Cases is not a specifically Object Oriented practice. But it is a best practice for elaborating and understanding requirements. So we will study Use Cases.

## Definition of Software Architecture

- Software Architecture:There are various forms of it. But the common theme is that it has to do with large scale-the Big Ideas in the forces, organization, styles, patterns, responsibilities, collaborations, connections and motivations of a system and major subsystems.

## 软件架构

- 软件架构 = { 结构元 (Elements)，结构型 (Forms)，结构理 (Rationale) }
- 即软件架构是由若干具特定结构形按结构理组合起来的结构元的集合。
- 结构元包括处理元、数据元和连接元

## Definition variance

- In software development, architecture is thought of as both noun and a verb.
- As a noun, the architecture includes the organization and  structure of the major elements of the system.
- As a verb, architecture is part investigation and part design work.

## Definition

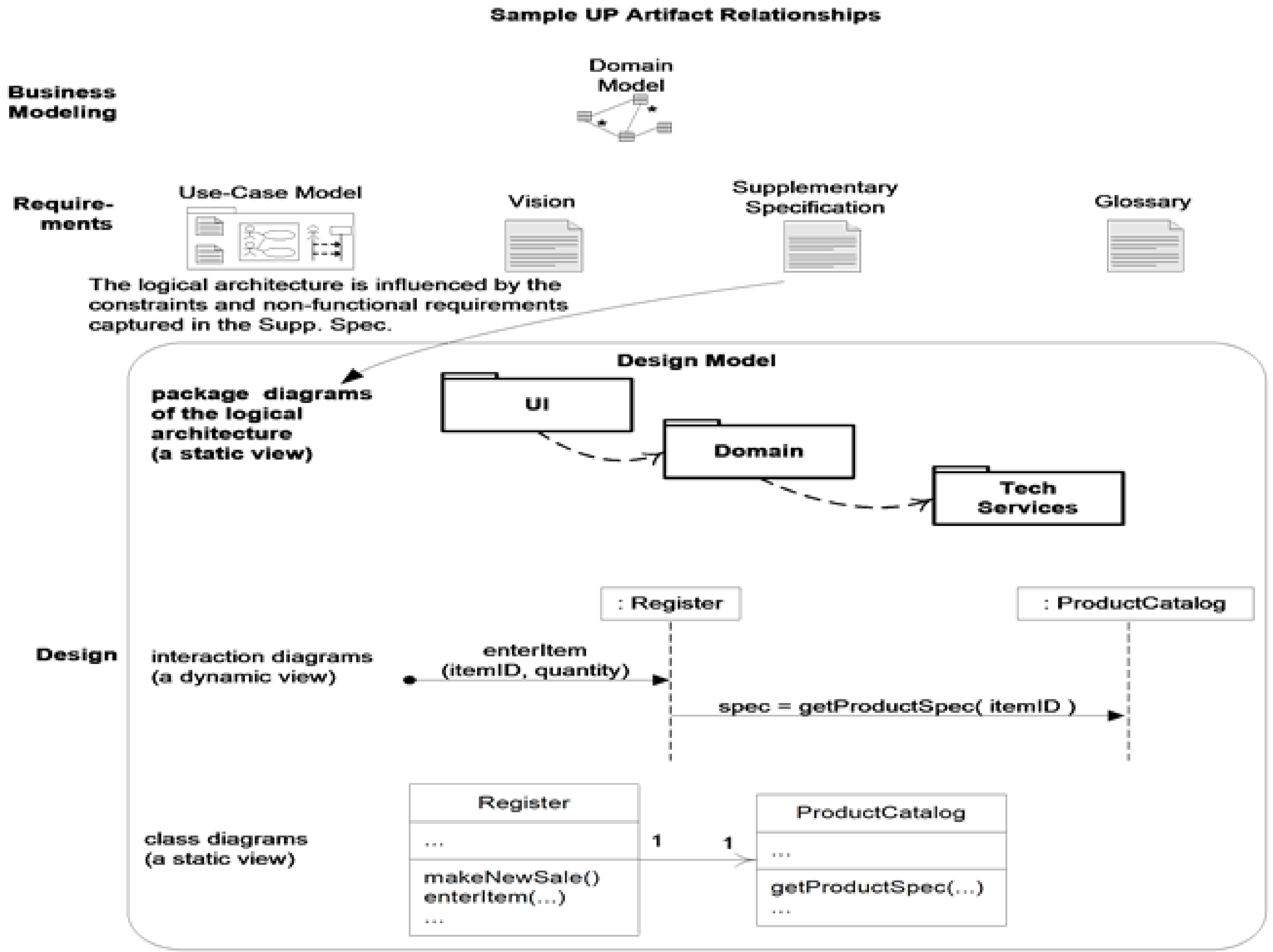
- Architectural investigation: involves functional and non-functional requirements that have impact on system design.
- Some of these are: Market trends, performance, cost and points of evolution.
- Architectural Design: is the resolution of these requirements in the design of software.

## Architectural Dimension and Views in UP

The common dimensions are:

- The logical architecture, describes the system in terms of its conceptual organization in layers, packages, classes, interfaces and subsystems.
- The deployment architecture, describes the system in terms of the allocation of process to processing unit and network configurations.

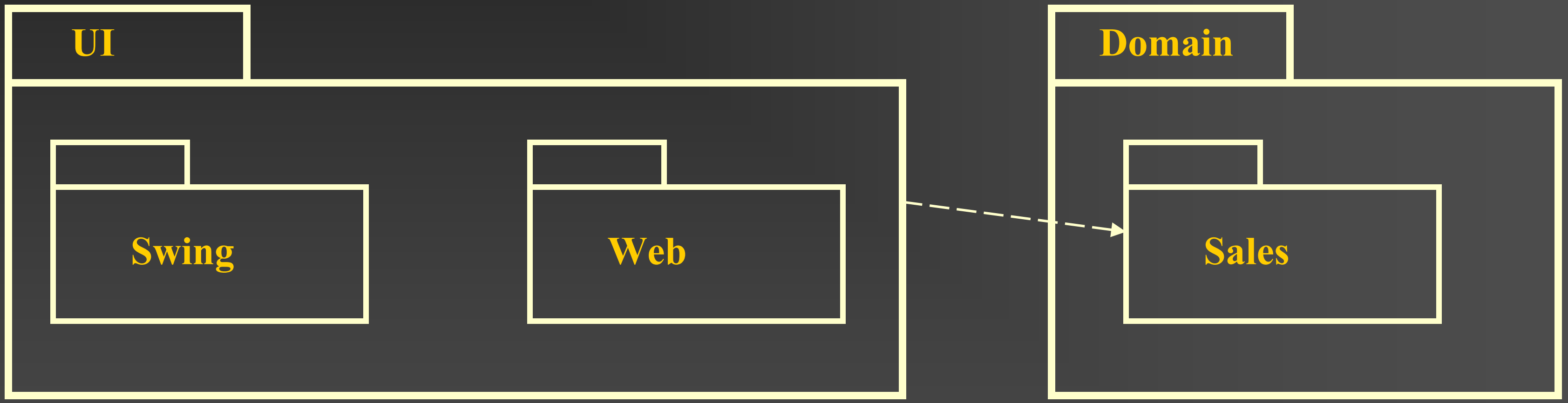
## Fig. 13.1 UP制品相互影响示例 (p146)


Sample UP Artifact Relationships
Business Modeling
Domain Model
Require-ments
Use-Case Model
Vision
Supplementary Specification
Glossary
The logical architecture is influenced by the constraints and non-functional requirements captured in the Supp. Spec.
Design Model
package diagrams of the logical architecture (a static view)
UI
Domain
Tech Services
Design
interaction diagrams (a dynamic view)
: Register
: ProductCatalog
enterItem (itemID, quantity)
spec = getProductSpec( itemID )
class diagrams (a static view)
Register
...
makeNewSale()
enterItem(...)
...
1
1
ProductCatalog
...
getProductSpec(...)
...

## Package Diagrams

- UML Package Diagrams are often used to show the contents of components, which are often packages in the Java sense.
- Each package represents a namespace.
- Packages, as components, can be nested inside other packages.

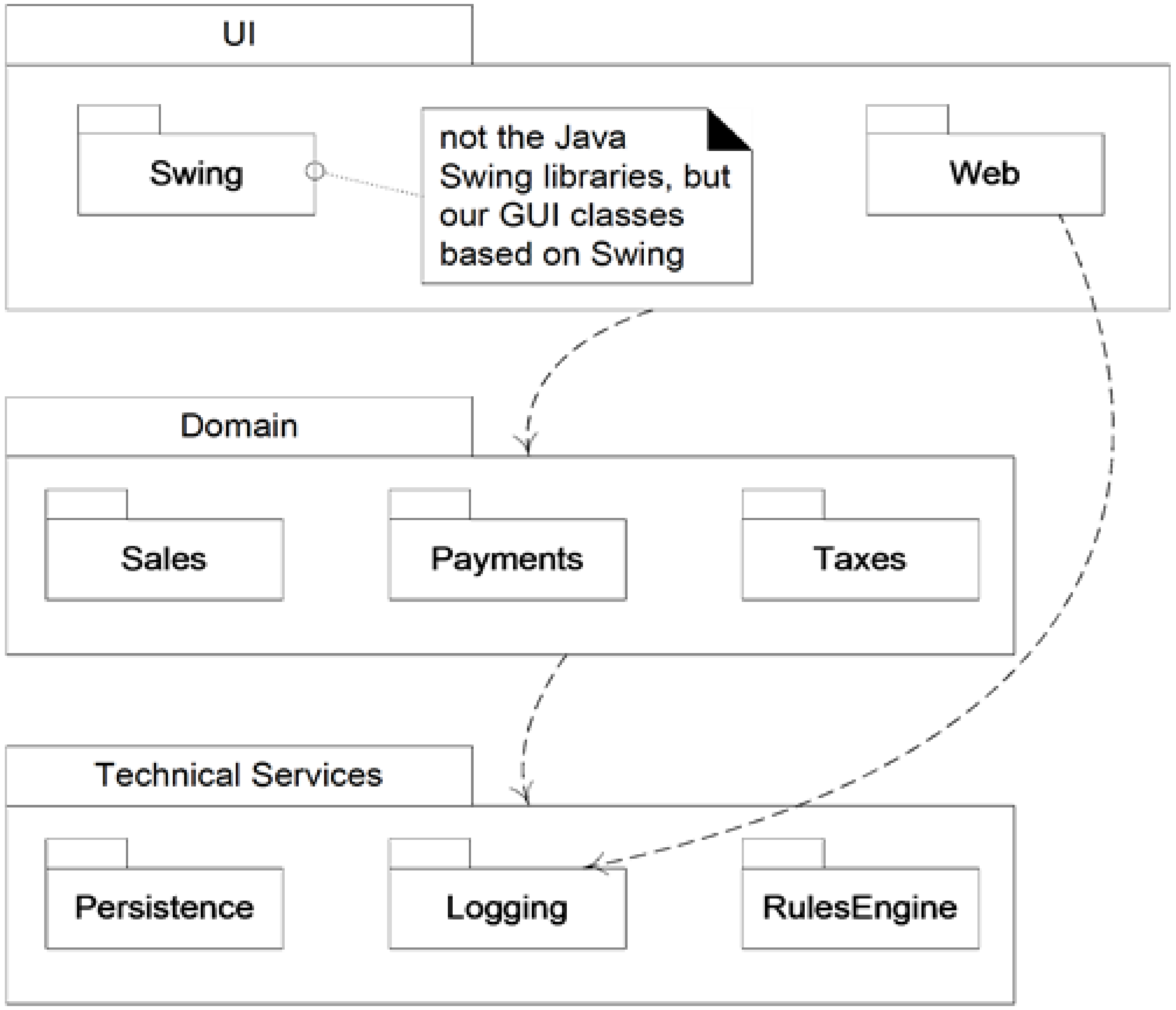
# Package Diagram



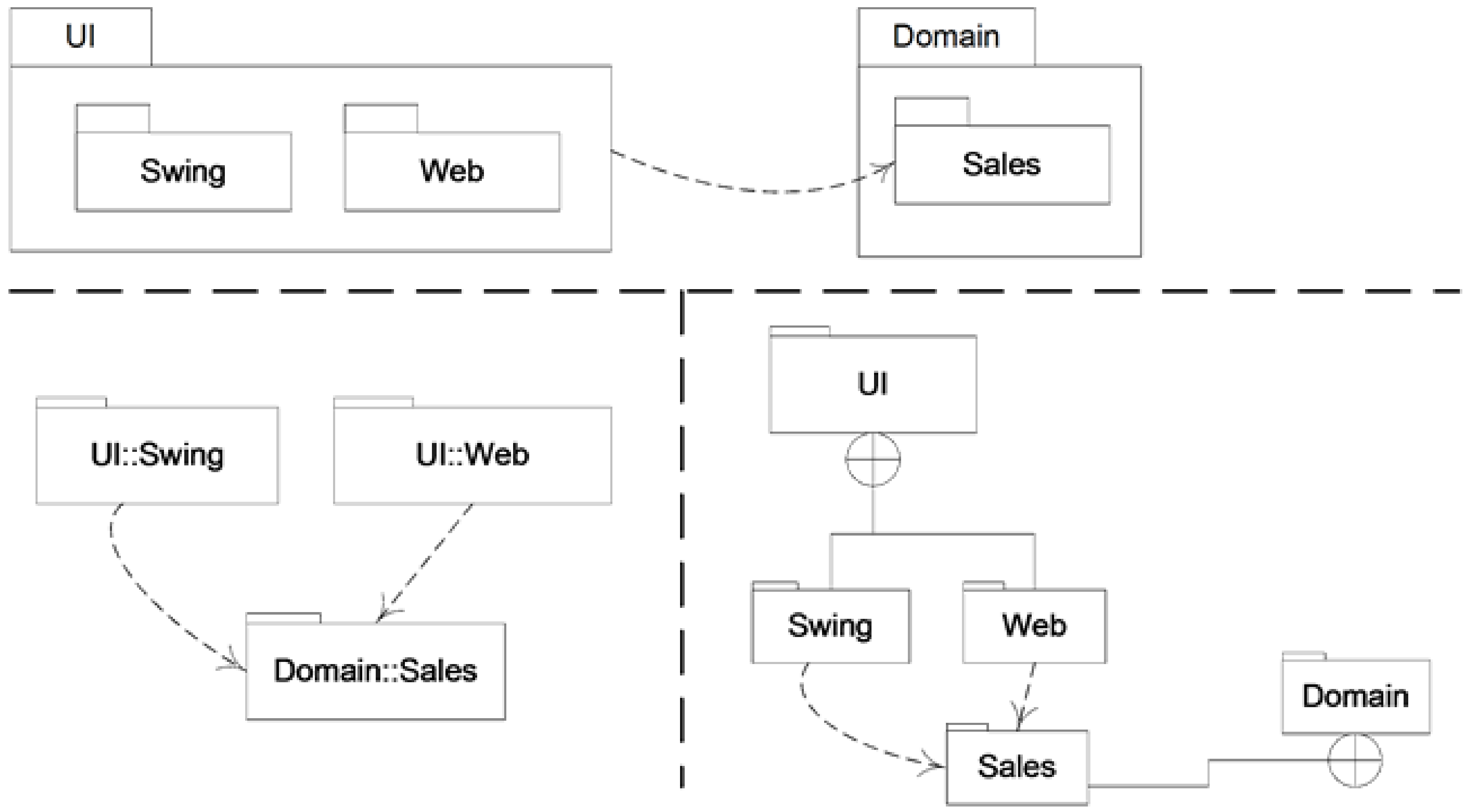
## What is a layer?

- “A layer is a coarse grained grouping of classes packages or subsystems that has cohesive responsibility for a major aspect of the system.”
- Higher layers call upon the services of lower layers.

# Fig. 13.2 UML包图所表示的层（p147)



# Fig. 13.3 嵌套包的另一UML表示（p148)



## Architectural Patterns and Pattern Categories

- Architectural patterns: Relates to large-scale design and typically applied during the early iterations(in elaboration phase).
- Design patterns: Relates to small and medium-scale design of objects and frameworks.
- Idioms: Relates to language or implementation-oriented low-level design solutions.

## Architectural Pattern: Layers

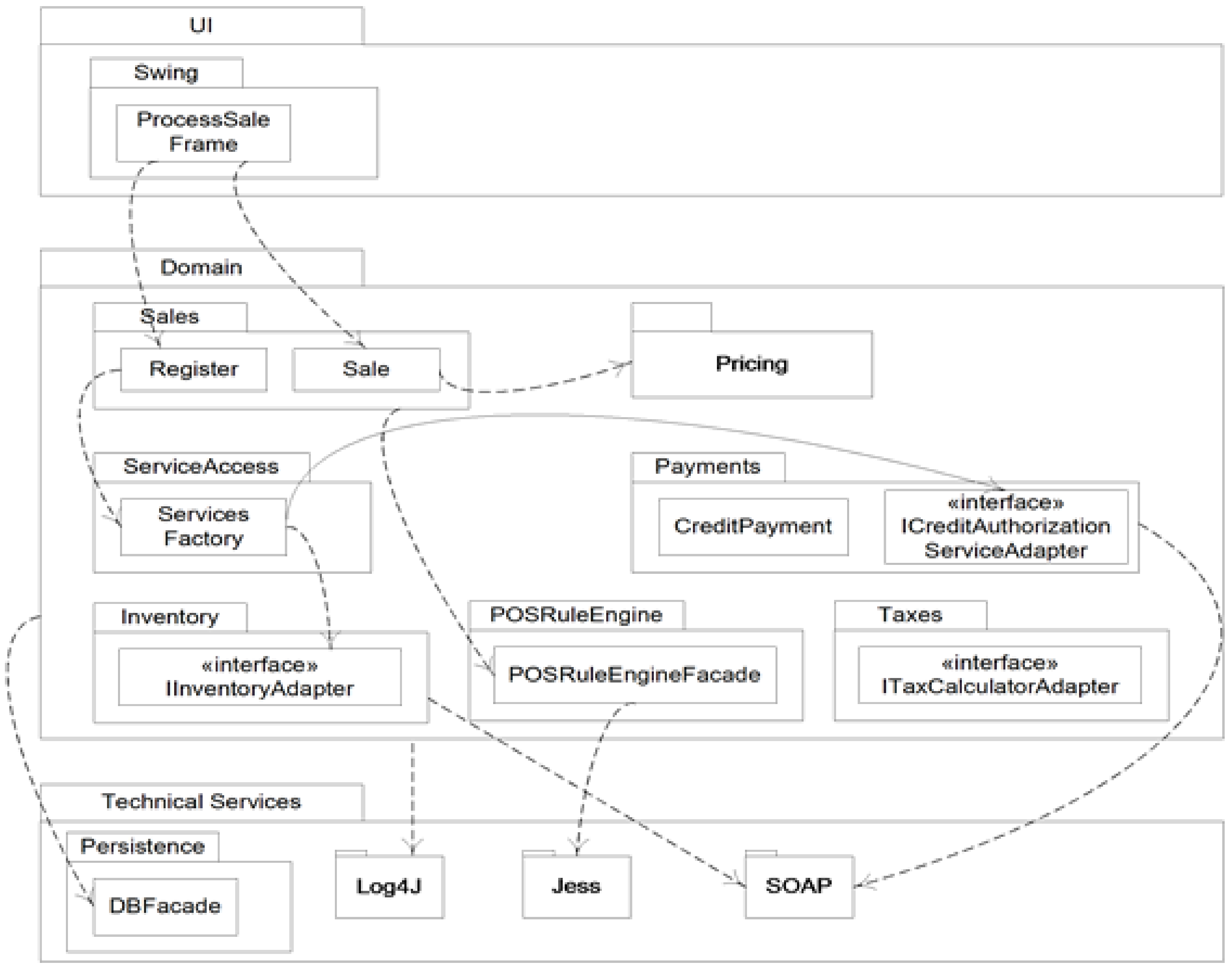
Idea behind Layer patterns:

- Organize the large-scale logical structure of a system into discrete layers of distinct, related responsibilities with a clean, cohesive separation of concerns such that the “lower” layers are low-level and general services, and the higher layers are more application specific.
- Collaboration and coupling is from higher to lower layers.

## Inter-Layer and Inter-Package Coupling

- It is informative to include a diagram in the logical view that shows the coupling between the layers and packages.
- Following figure shows the coupling.

# Fig. 34.2 Partial coupling between Packages (p405)



## Inter-Layer and Inter-Package Interaction

- Emphasizes the dynamics of how objects across the layers connect and communicate.
- The interaction diagram focuses on the logical view and on the collaborations between the layers and package boundaries.

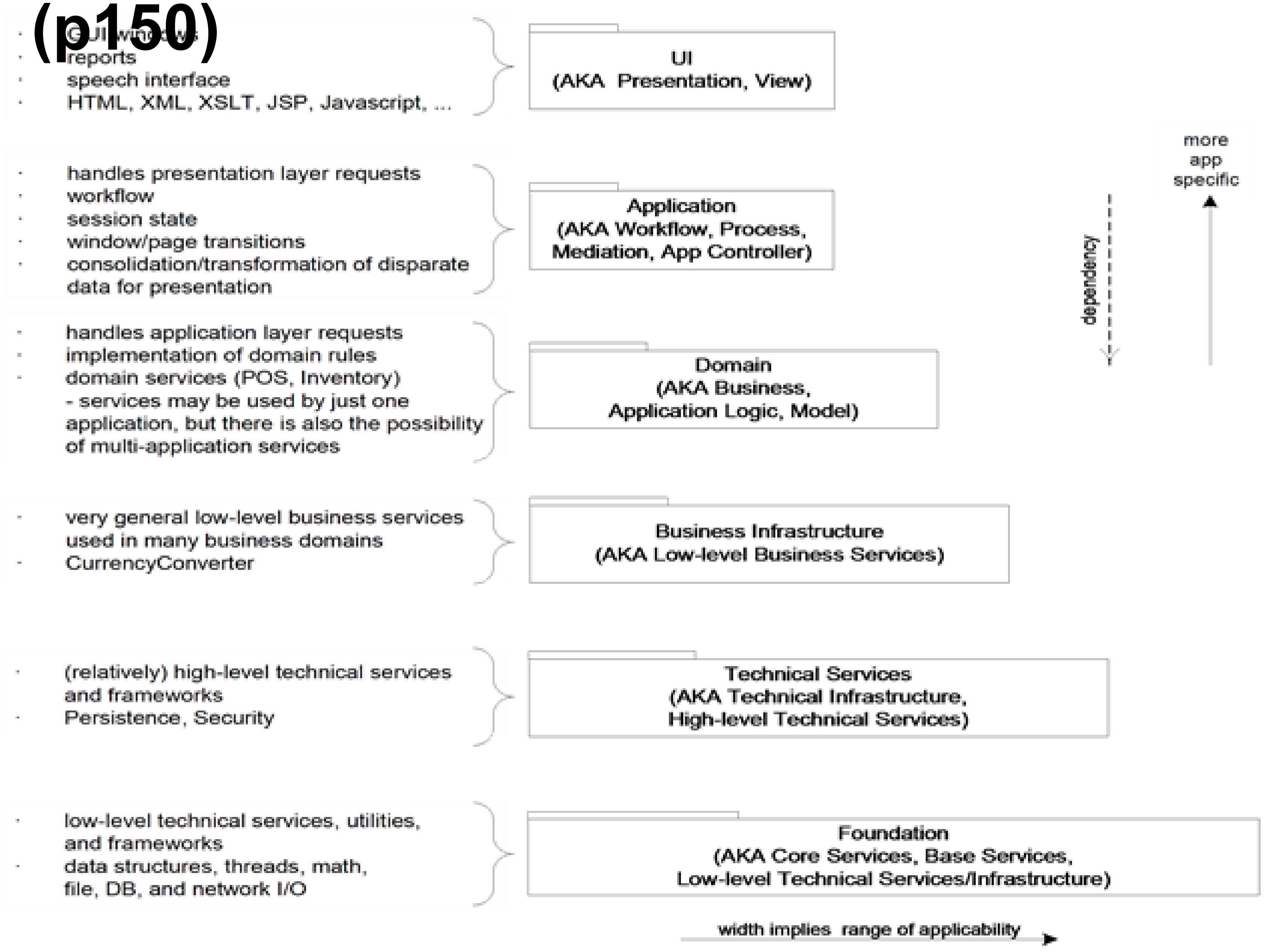
## Logical vs. Process and Deployment of Architecture

- Architectural Layers are a logical view of the architecture
- They are not a deployment view of elements to process.
- Depending on platform, all layers could be deployed within the same process on same node.
- Or across many computers.

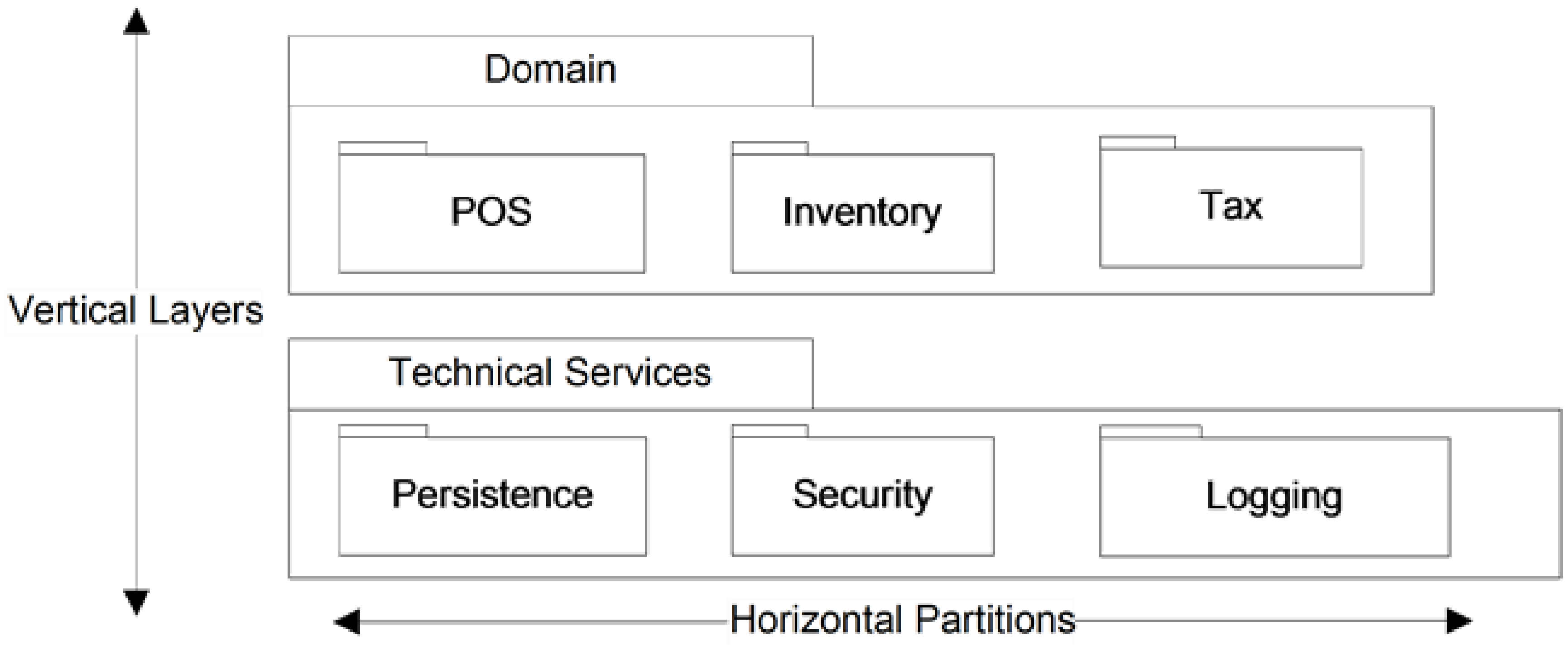
## Terminology:Tier, Layers, and Partitions

- Tier relates to physical processing node or clusters of node, such as “client tier”.
- Layers of an architecture represent the vertical slices
- Partitions represents a horizontal division of relatively parallel subsystems of a layer.

# Fig. 13.4 信息系统逻辑架构常见层 (p150)


GUI windows
reports
speech interface
HTML, XML, XSLT, JSP, Javascript, ...
UI
(AKA Presentation, View)
handles presentation layer requests
workflow
session state
window/page transitions
consolidation/transformation of disparate data for presentation
Application
(AKA Workflow, Process, Mediation, App Controller)
handles application layer requests
implementation of domain rules
domain services (POS, Inventory)
- services may be used by just one application, but there is also the possibility of multi-application services
Domain
(AKA Business, Application Logic, Model)
very general low-level business services used in many business domains
CurrencyConverter
Business Infrastructure
(AKA Low-level Business Services)
(relatively) high-level technical services and frameworks
Persistence, Security
Technical Services
(AKA Technical Infrastructure, High-level Technical Services)
low-level technical services, utilities, and frameworks
data structures, threads, math, file, DB, and network I/O
Foundation
(AKA Core Services, Base Services, Low-level Technical Services/Infrastructure)
more app specific
dependency
width implies range of applicability

## Fig. 13.6 层和分区 (p153)



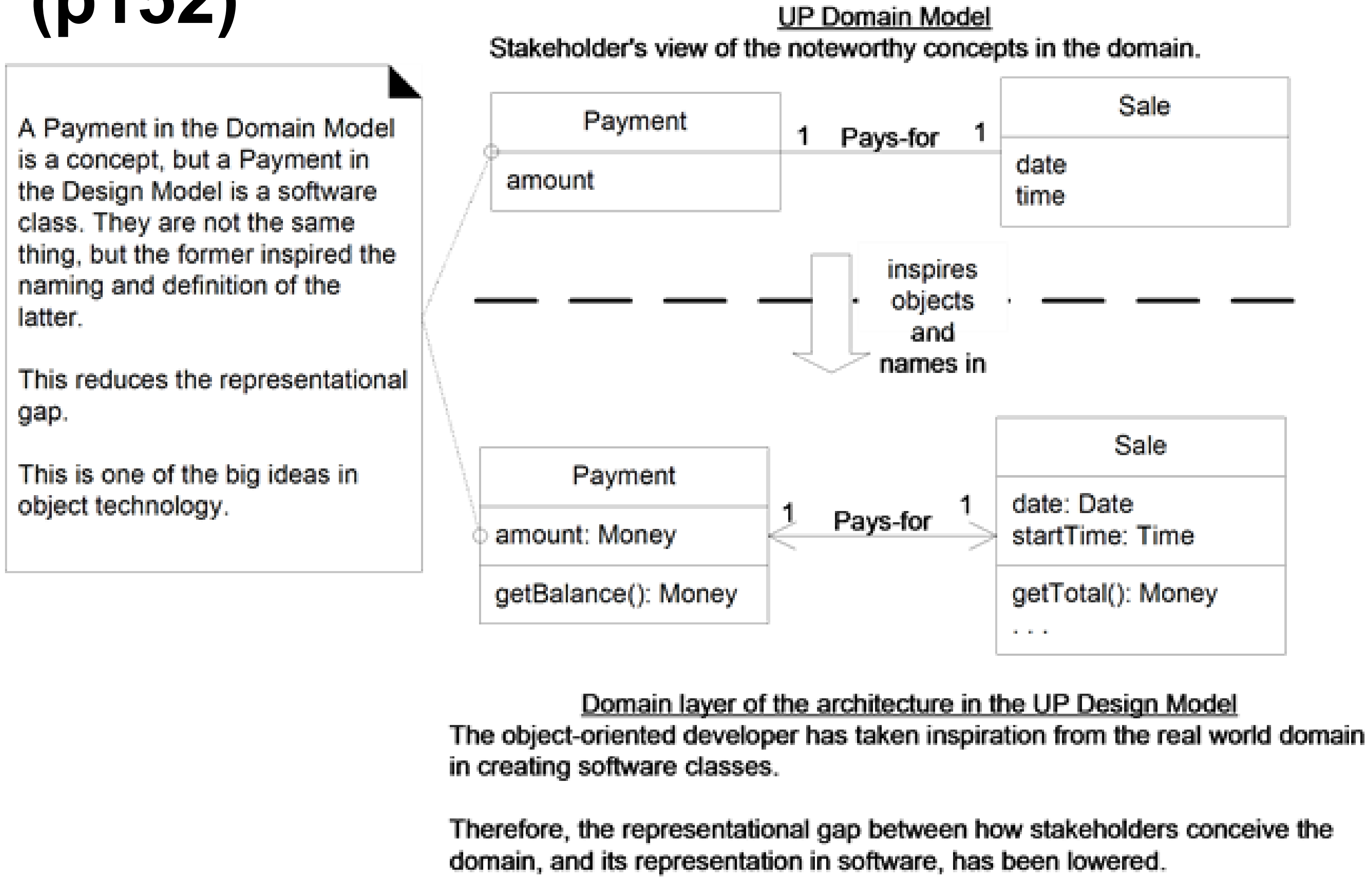
## How do we design application logic with objects?

- We could create one class and put all logic in it, but that violates the whole spirit of object orientation.
- We create software objects with names drawn from the real world, and assign application logic responsibilities to them.
- It takes a lot of skill and experience to do a good job of choosing objects and assigning responsibilities.

## Domain Layer and Domain Model

- These are not the same thing. Domain model shows the real world, while the Domain layer shows the software architecture.
- But the Domain model inspires the Domain layer, and is the source of many of the concept, especially class names.
- Do not confuse the problem with the solution.

# Fig. 13.5 领域层和领域模型的关系 (p152)



A Payment in the Domain Model is a concept, but a Payment in the Design Model is a software class. They are not the same thing, but the former inspired the naming and definition of the latter.

This reduces the representational gap.

This is one of the big ideas in object technology.

UP Domain Model

Stakeholder's view of the noteworthy concepts in the domain.

Domain layer of the architecture in the UP Design Model

The object-oriented developer has taken inspiration from the real world domain in creating software classes.

Therefore, the representational gap between how stakeholders conceive the domain, and its representation in software, has been lowered.

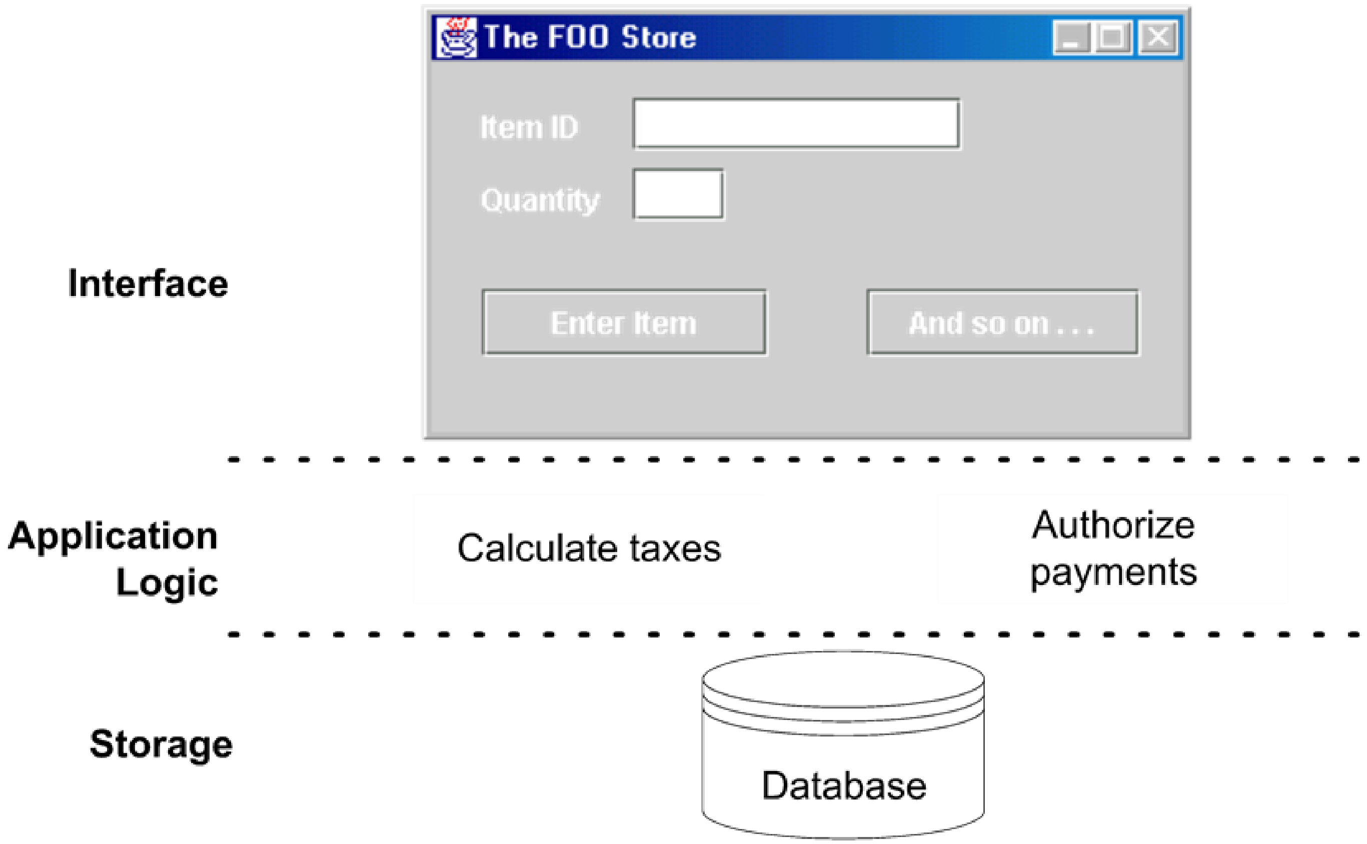
## Information Systems

- In IS layered architecture was known as three-tier architecture.
- A three-tier architecture has interface, Application logic and a storage.
- The singular quality of 3-tier architecture is:
- Separation of the application logic into distinct logical middle tier of software.
- The interface tier is relatively free of application processing.

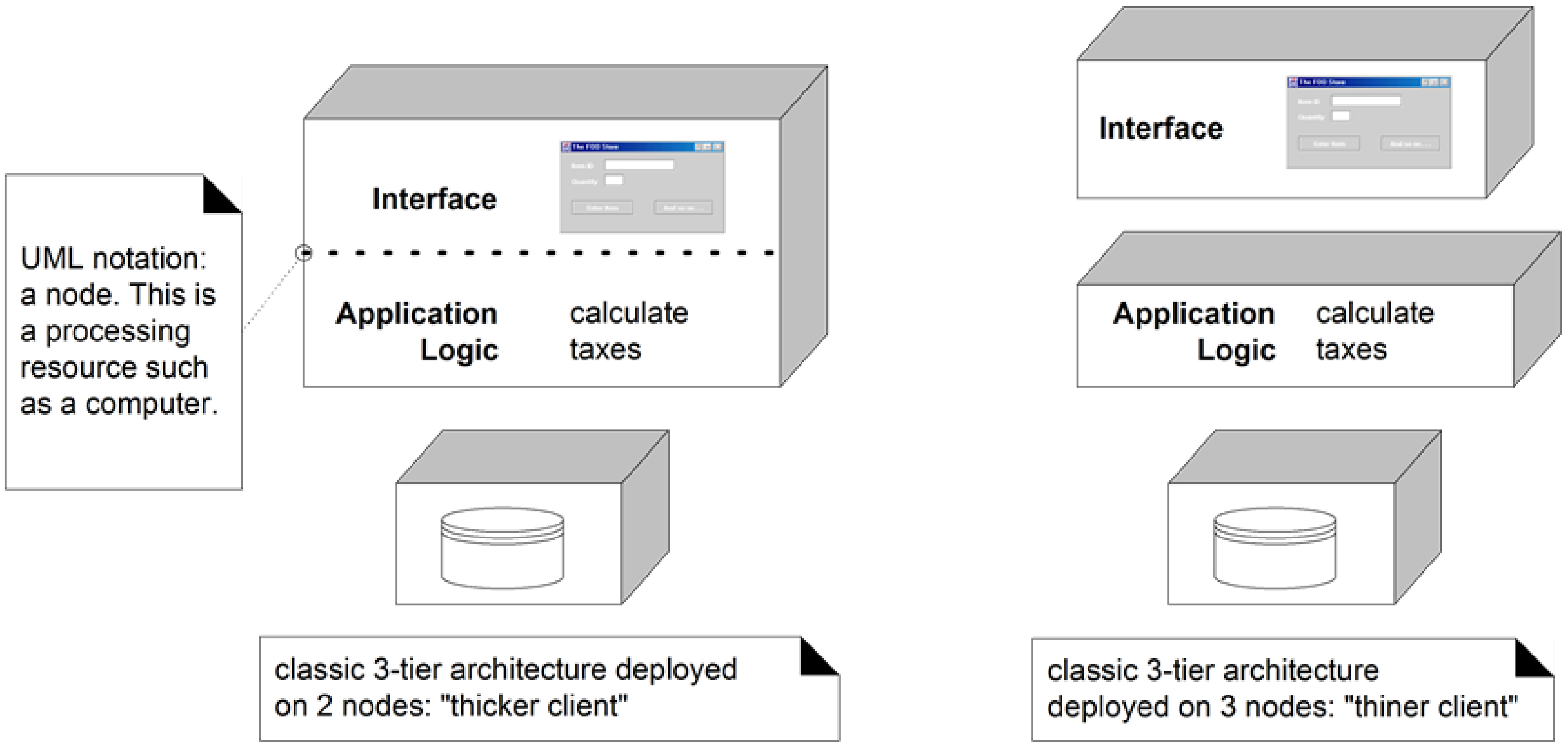
## Information Systems(cont..)

- The middle tier communicates with the back-end storage layer.
- The following is an example of 3-tier architecture.

# Fig. 34.11 三层架构经典视图（p413)



## Fig. 34.12 三层结构布署在两层/三层物理架构上（p414)


UML notation: a node. This is a processing resource such as a computer.
Interface
Application Logic
calculate taxes
classic 3-tier architecture deployed on 2 nodes: "thicker client"
Interface
Application Logic
calculate taxes
classic 3-tier architecture deployed on 3 nodes: "thiner client"

## Two-tier Design

- In this design, the application logic is placed within window definitions, which read and writes directly to database.
- There is no middle tier that separates out the application logic.

## The Model-View Separation Principle

- The principle states that model(domain) objects should not have direct knowledge of view(presentation) objects.
- Furthermore, the domain classes should encapsulate the information and behavior related to application logic.

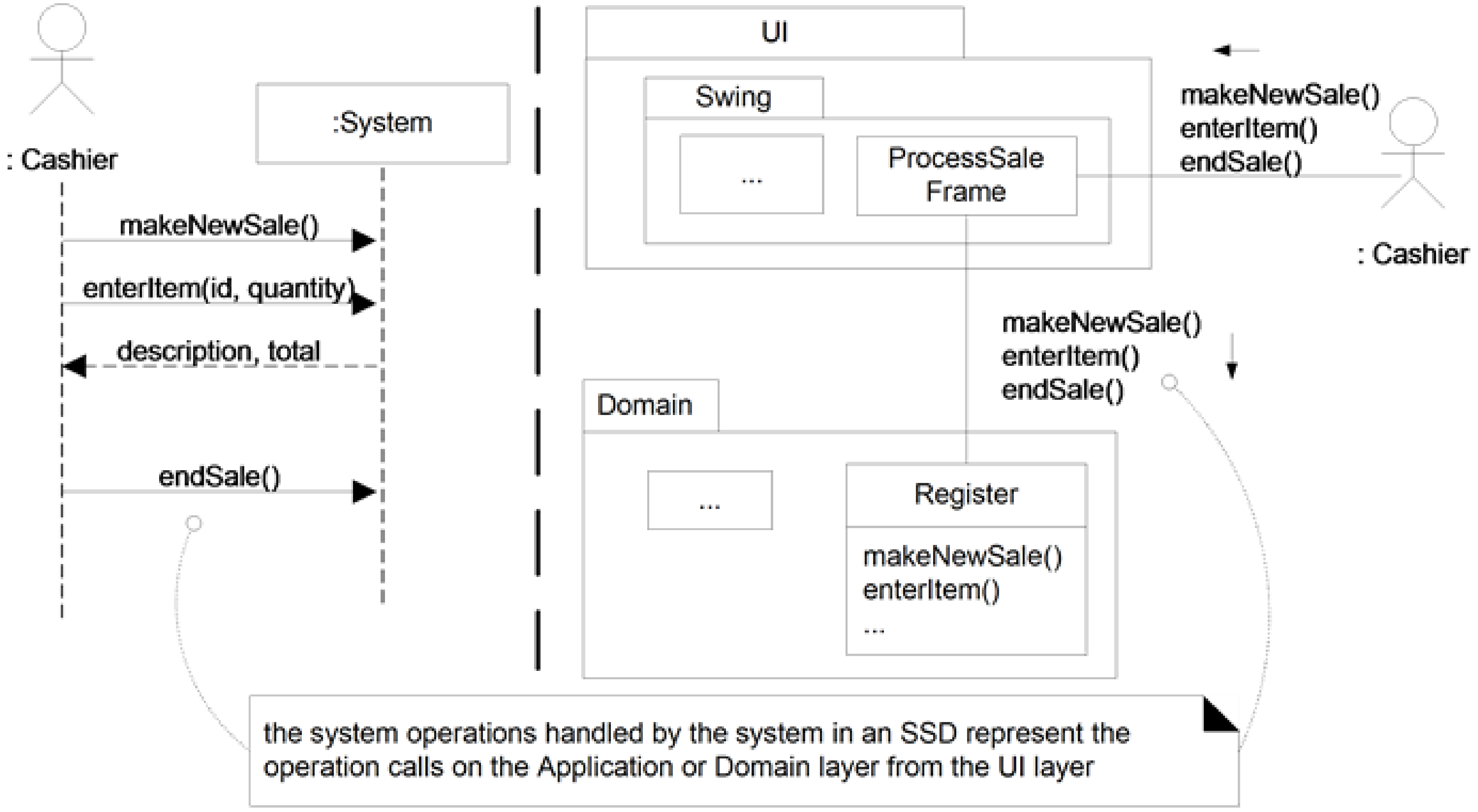
## Need for Model-View separation

- To support cohesive model definitions that focus on the domain process, rather than on interfaces.
- To allow separate development of the model and user interface layers.
- To minimize the impact of requirements changes in the interface upon the domain layer.
- To allow new views to be easily connected to an existing domain layer, without affecting the domain layer.

## Continue..

- To allow multiple simultaneous views on the same model object.
- To allow execution of the model layer independent of the user interface layer
- To allow easy porting of the model layer to another user interface framework.

# Fig. 13.8 SSD和层间系统操作（p155)


: Cashier
:System
makeNewSale()
enterItem(id, quantity)
description, total
endSale()
UI
Swing
...
ProcessSale Frame
makeNewSale()
enterItem()
endSale()
: Cashier
makeNewSale()
enterItem()
endSale()
Domain
...
Register
makeNewSale()
enterItem()
...
the system operations handled by the system in an SSD represent the operation calls on the Application or Domain layer from the UI layer