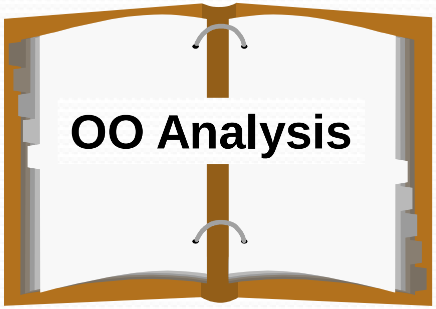
COMP 3711

OOA / OOD

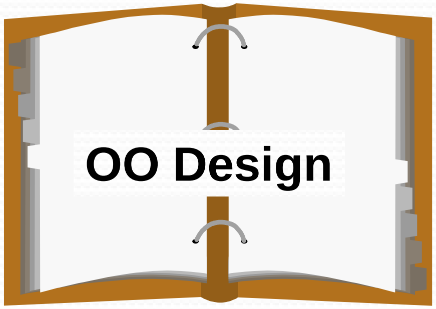
Midterm Review

Text: Larman
Inside cover front page +
1,22,2,3,4,5,6,8,30, 9, 31.2-31.17, 11, 1.32, 14, 15, 16,17

OO Approach In Development

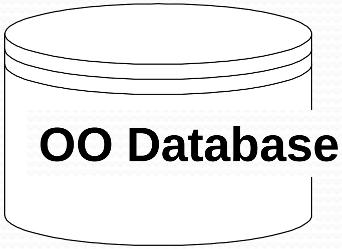


Analyse current situation and new requirement



Formulate (eleborate) conceptual solutions

Build and construct



Store data objects

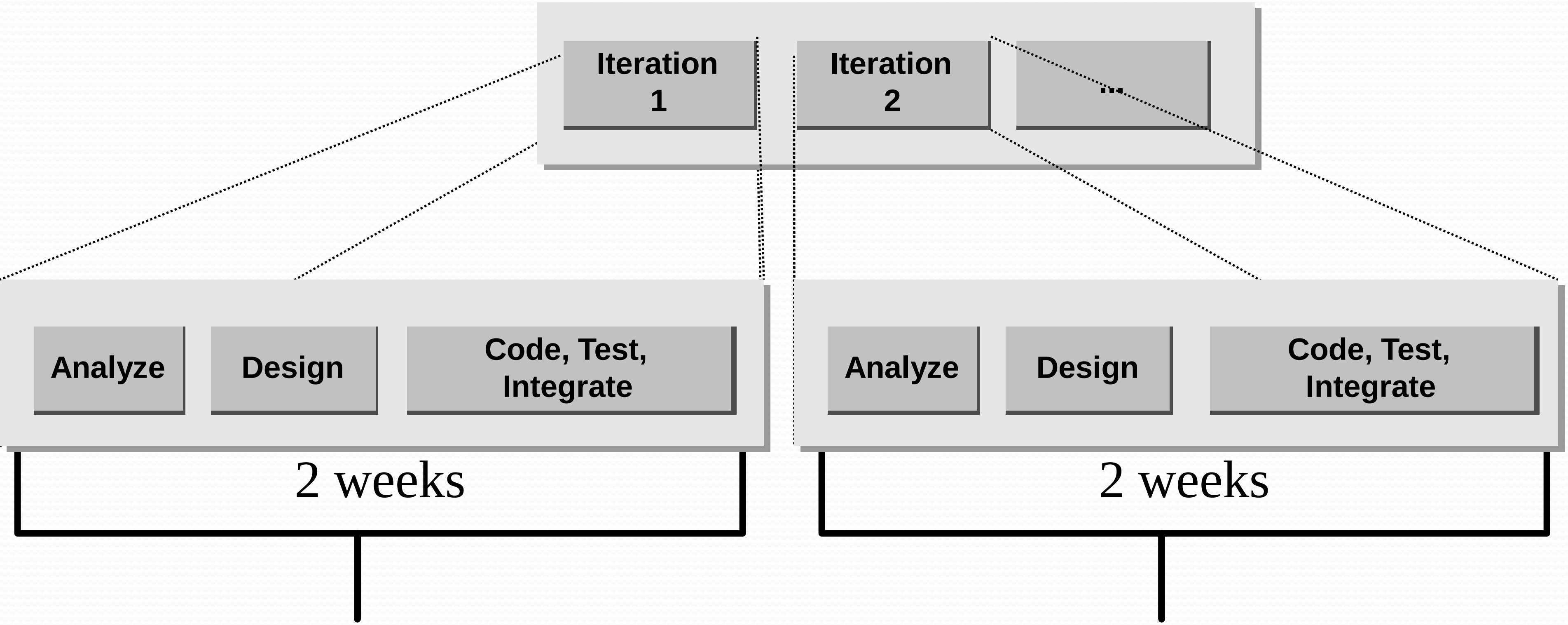
Unified Process - UP

Requirement	Design	Develop	Implement
Inception	Elaboration	Construction	Transition
Approximate vision, business case, scope	Refine vision, core architecture, refine scope	Implementation of low risk core architecture, refine scope	Beta test and deployment

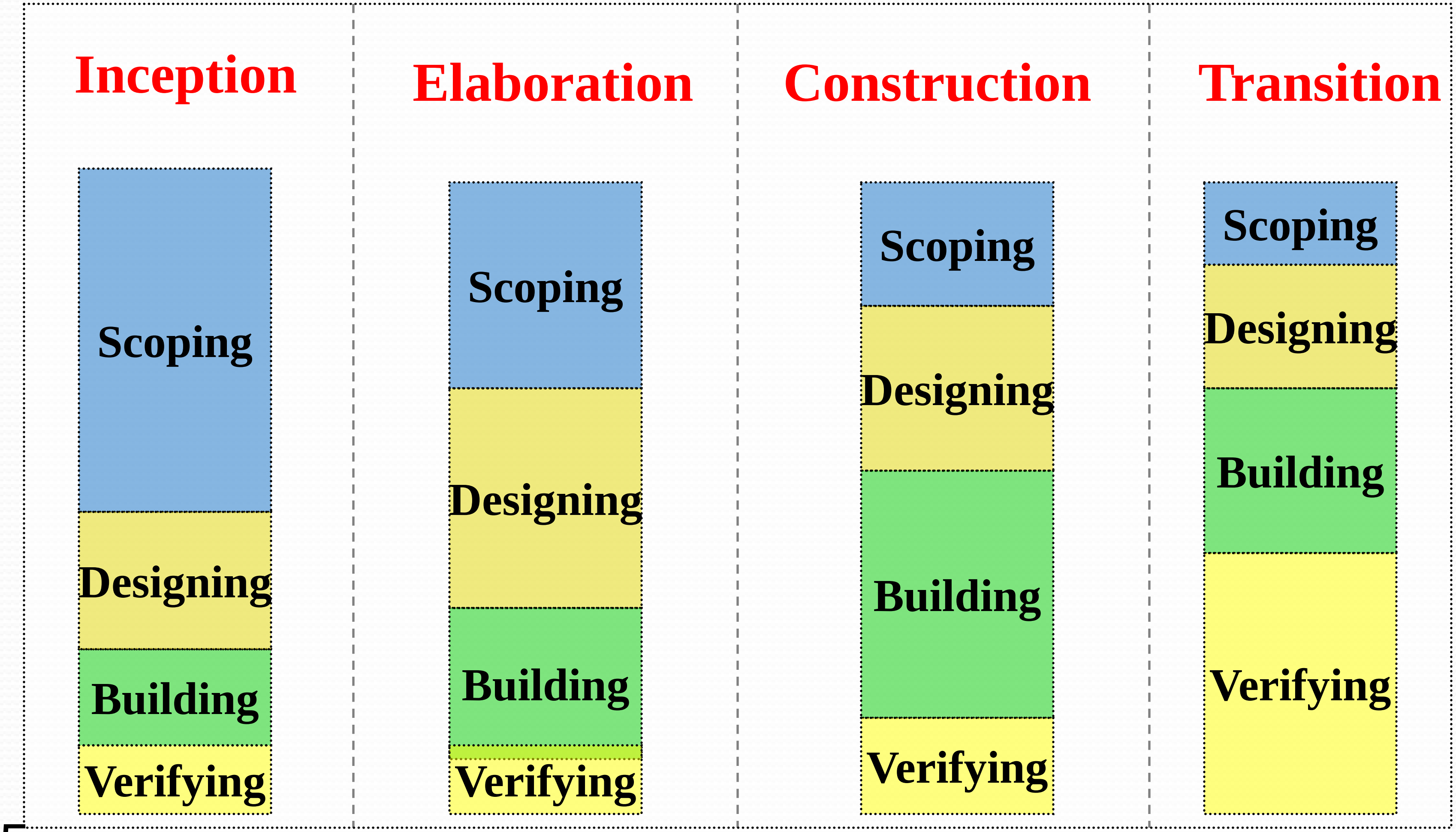
Iterative Development

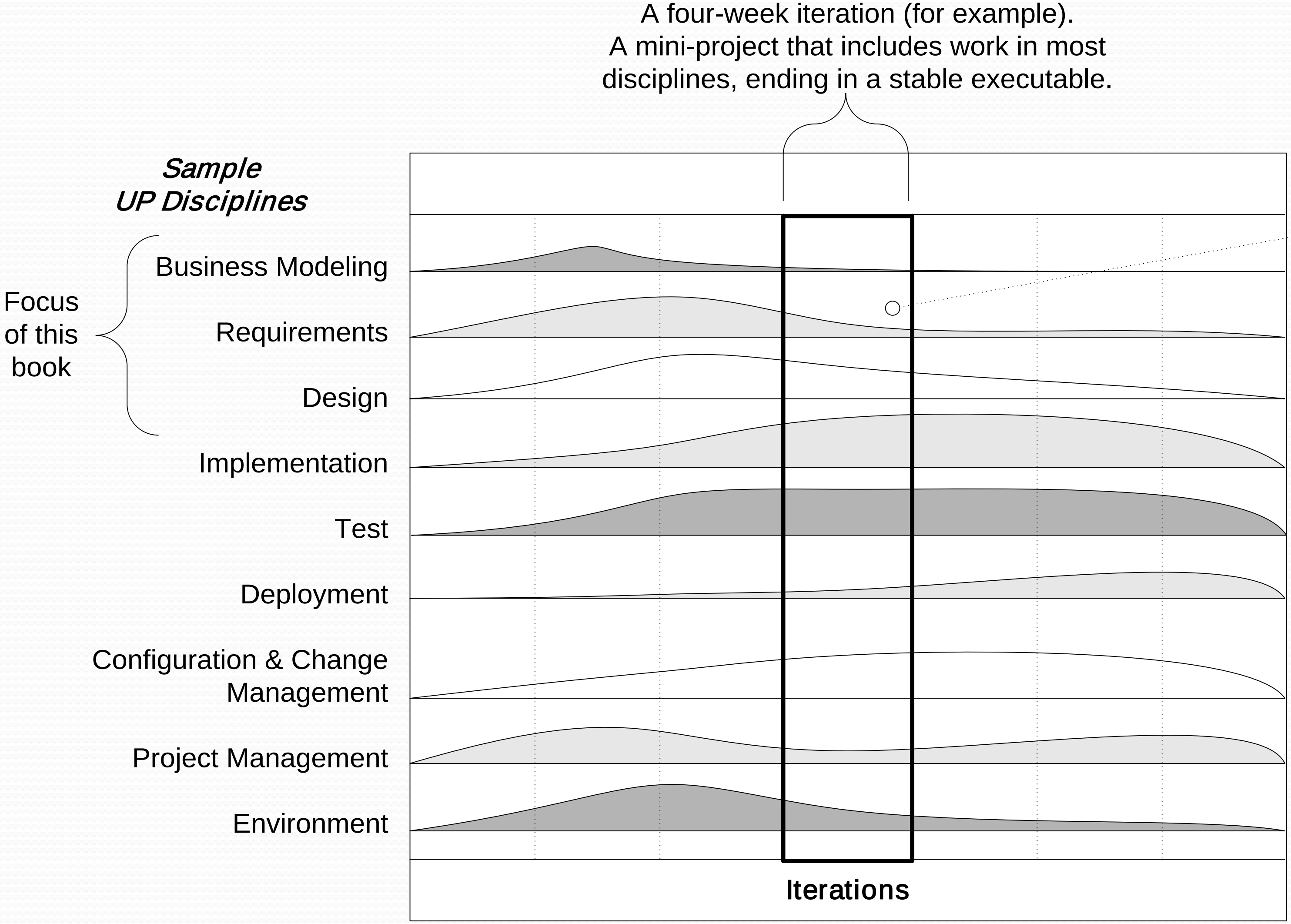
Larman 2.10

Iterative Development



Unified Process (UP) - Iterative




A four-week iteration (for example).
A mini-project that includes work in most disciplines, ending in a stable executable.
Sample
UP Disciplines
Focus of this book
Business Modeling
Requirements
Design
Implementation
Test
Deployment
Configuration & Change Management
Project Management
Environment
Iterations

Note that although an iteration includes work in most disciplines, the relative effort and emphasis change over time.

This example is suggestive, not literal.

Larman Fig 2.7

Variations of OO/UP Methodologies

- Extreme Programming (XP)
  - Ken Beck
  - Light weight
- Agile Modeling
  - Scott Ambler
  - Combine UP/XP
- Scrum
  - Takeuchi, Nonaka
  - Based on Agile
  - Adaptive Methodology

Manifesto Of Agile Alliance

•Uncover better ways

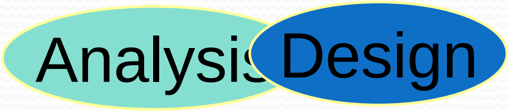
•Helping each other

Process vs Models

- Process (Methodologies)
  - Provides guidelines to follow
  - Include specific models, tools, techniques, documentation
- Model (Abstraction)
  - Representation of an important aspect of the “real world”
  - Use of drawings, diagrams, notations, symbols, conventions

Analysis Versus Design

- Analysis
  - Focus on understanding the problem domain
  - Idealized design
  - Behavior
  - Functional requirements
  - System structure



Analysis

Design

- Design
  - Focus on understanding the solution
  - Operations and Attributes
  - Performance
  - Close to real code
  - Object lifecycles
  - Non-functional requirements
  - A large model

Larman 1.4

RUP / UP

FURPS

UML

RDD

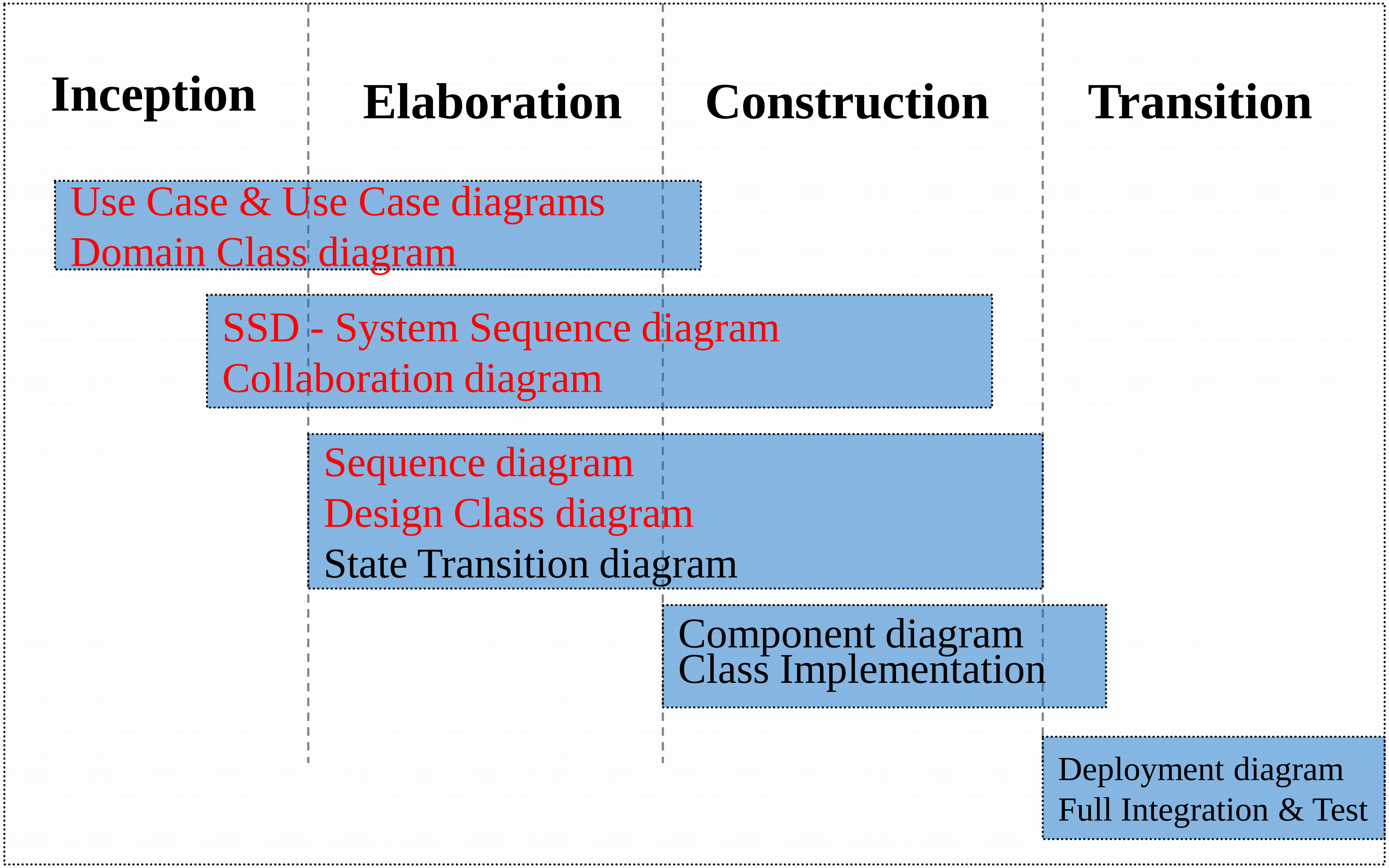
GRASP

GoF Design Patterns

Types of requirements(FURPS)

- FURPS
  - (F)unctional - features, capabilities, security
  - (U)sability - human factors, help, documentation
  - (R)eliability - frequency of failure, recoverability, predictability
  - (P)erformance - response times, throughput, accuracy, availability, resource usage
  - (S)upportability - adaptability, maintainability, internationalization, configurability

UML And UP



Inception

Elaboration

Construction

Transition

Use Case & Use Case diagrams
Domain Class diagram

SSD - System Sequence diagram
Collaboration diagram

Sequence diagram
Design Class diagram
State Transition diagram

Component diagram
Class Implementation

Deployment diagram
Full Integration & Test

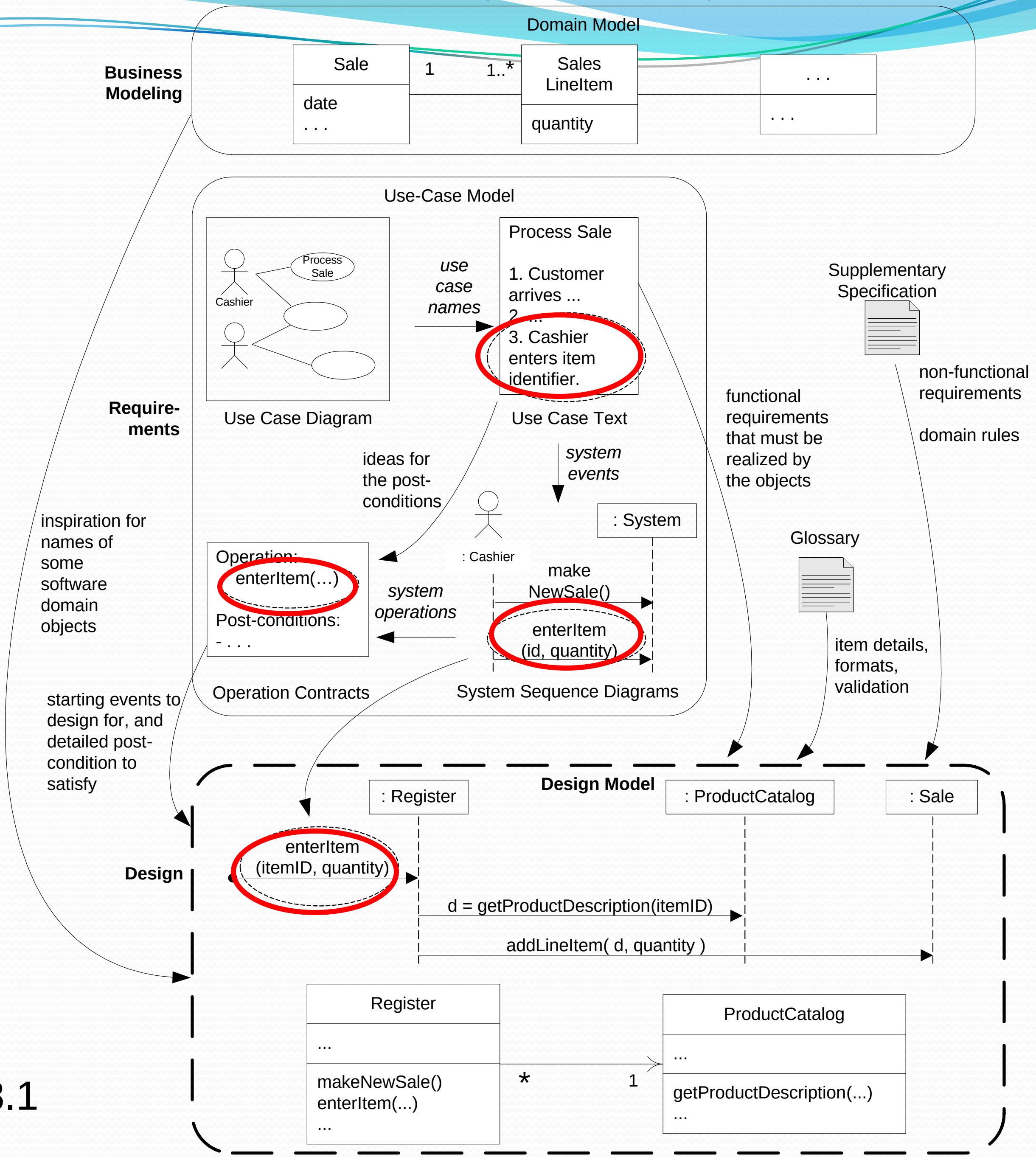
UP & Domain Models

Discipline	Artifact	Inception	Elaboration	Construction	Transition
Business Modeling	Domain Model		**start**		
Requirements	Use-Case Model	start	**refine**		
	Vision	start	**refine**		
	Supplementary Specification	start	**refine**		
	Glossary	start	**refine**		
Design	Design Model		**start**	refine	
	SW Architecture Document		**start**	refine	
	Data Model		**start**	refine	
Implementation	Implementation Model		**start**	refine	refine
Project Management	SW Development Plan	start	**refine**	refine	refine
Testing	Test Model		**start**	refine	
Environment	Development Case	start	**refine**		

Domain models normally started and completed in elaboration

OOA & OOD Design Artifacts

Sample UP Artifact Relationships



Business Modeling

Domain Model

Sale

date
. . .

1

1..*

Sales LineItem

quantity

. . .

. . .

Requirements

Use-Case Model

Process Sale

Cashier

Use Case Diagram

use case names

Process Sale

1. Customer arrives ...
2. ...
3. Cashier enters item identifier.

Use Case Text

Supplementary Specification

non-functional requirements

domain rules

functional requirements that must be realized by the objects

ideas for the post-conditions

system events

: Cashier

: System

make NewSale()

enterItem (id, quantity)

system operations

Operation: enterItem(…)

Post-conditions:
- . . .

Operation Contracts

System Sequence Diagrams

Glossary

item details, formats, validation

inspiration for names of some software domain objects

starting events to design for, and detailed post-condition to satisfy

Design

Design Model

: Register

: ProductCatalog

: Sale

enterItem (itemID, quantity)

d = getProductDescription(itemID)

addLineItem(d, quantity)

Register

...

makeNewSale()
enterItem(...)
...

*

1

ProductCatalog

...

getProductDescription(...)
...

Larman fig. 6.1 and 18.1

Use Case Definition

- Ivar Jacobson's
  - *A set of use-case instances, where each instance is a sequence of actions a system performs that yields an observable result of value to a particular actor.*

More Definitions

- Scenario (use case instance)
  - A specific sequence of actions and interactions between actors and the system
    - E.g. scenario of purchasing an item through the web

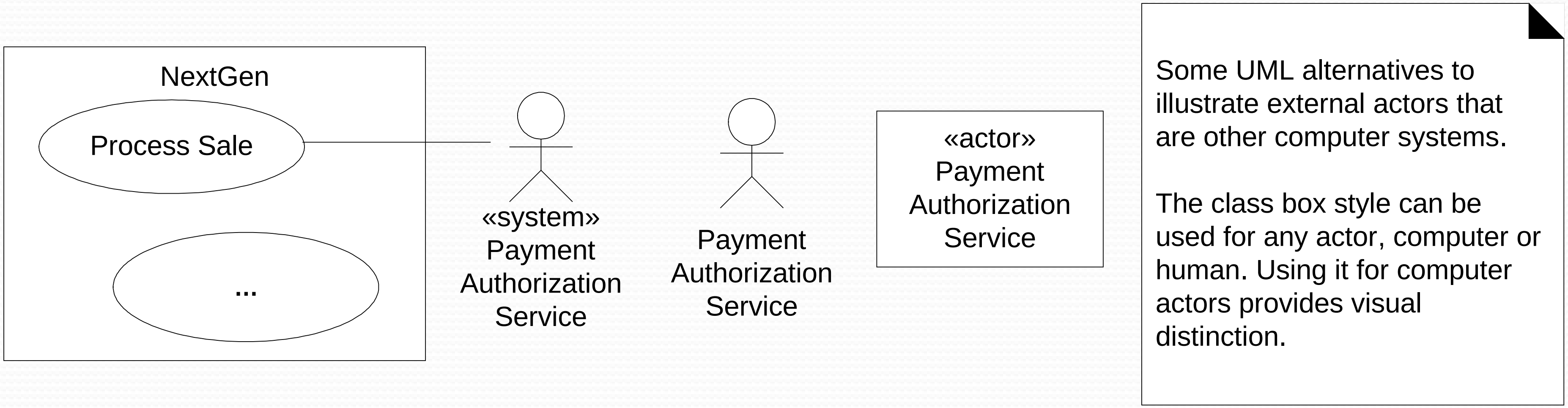
16

Use Case Documentation Template

- **Use Case Name**
- **Use Case Scenario**
- **Brief Description Of Use Case**
- **Actors**
- **Related Use Cases**
- **Stakeholders**
- **Preconditions**
- **Postconditions**
- ***Activities Flow (Actor / Action)***
- ***Systems Response***
- **Exception Conditions**
-

Which is the best format?

UML Use Case Diagramming



NextGen

Process Sale

...

«system»
Payment
Authorization
Service

Payment
Authorization
Service

«actor»
Payment
Authorization
Service

Some UML alternatives to illustrate external actors that are other computer systems.

The class box style can be used for any actor, computer or human. Using it for computer actors provides visual distinction.

Use UML keywords and stereotypes including guillemet symbols

Larman 6.5

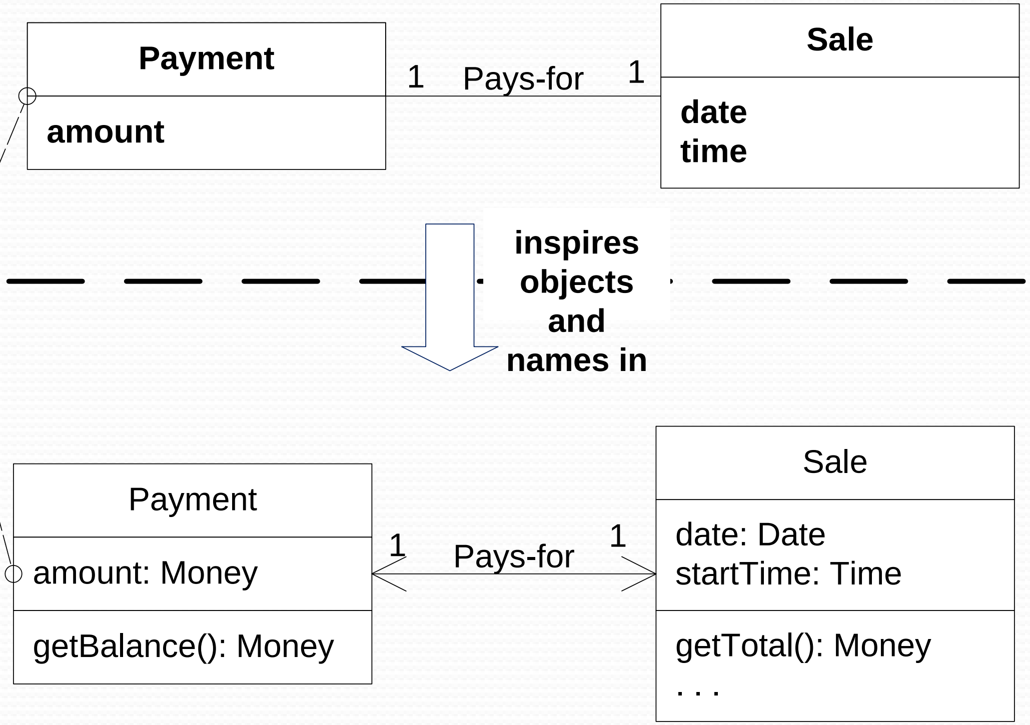
Domain Model drives Design Model

UP Domain Model
Stakeholder's view of the noteworthy concepts in the domain.

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.



Conceptual Classes

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.

Larman Fig 9.6

Conceptual classes from nouns

Simple cash-only Process Sale scenario:

1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier and quantity, if greater than one.
4. System records sale line item and presents item description, price, and running total.
5. Cashier repeats steps 2-3 until indicates done.
6. System presents total with taxes calculated.
7. Cashier tells Customer the total, and asks for payment.
8. Customer pays with cash.
9. Cashier enters cash tendered.
10. System records payment and presents change due.
11. System logs the completed sale, but does not interact with external systems.
12. System presents receipt.
13. Customer leaves with receipt and goods.

Domain Modeling Guidelines

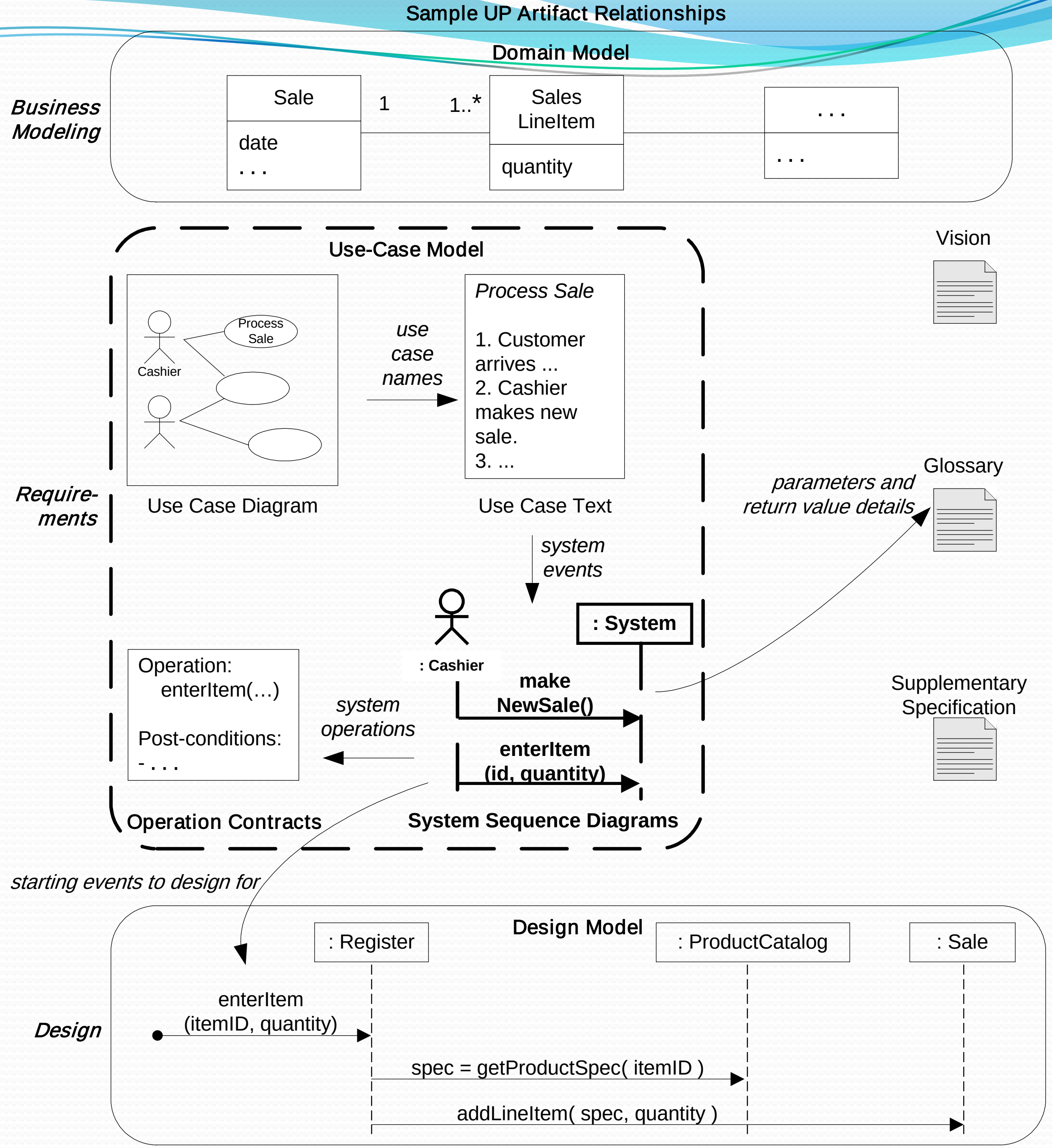
- List the candidate conceptual classes using following techniques in a domain class model
  - Conceptual Class Category List
  - and/or Noun Phrase Identification
- Draw them in the Domain Model.
- Add associations necessary to record relationships.
- Add the attributes necessary to fulfill information requirements.

Conceptual Class Relationships

- Four types of relationships:
  - Association
  - Aggregation (Composition)
  - Dependency
  - Generalization (Specialization)

- Other stuff:
  - Association name, Role name
  - Multiplicity, Visibility

SSD
Part of requirements gathering

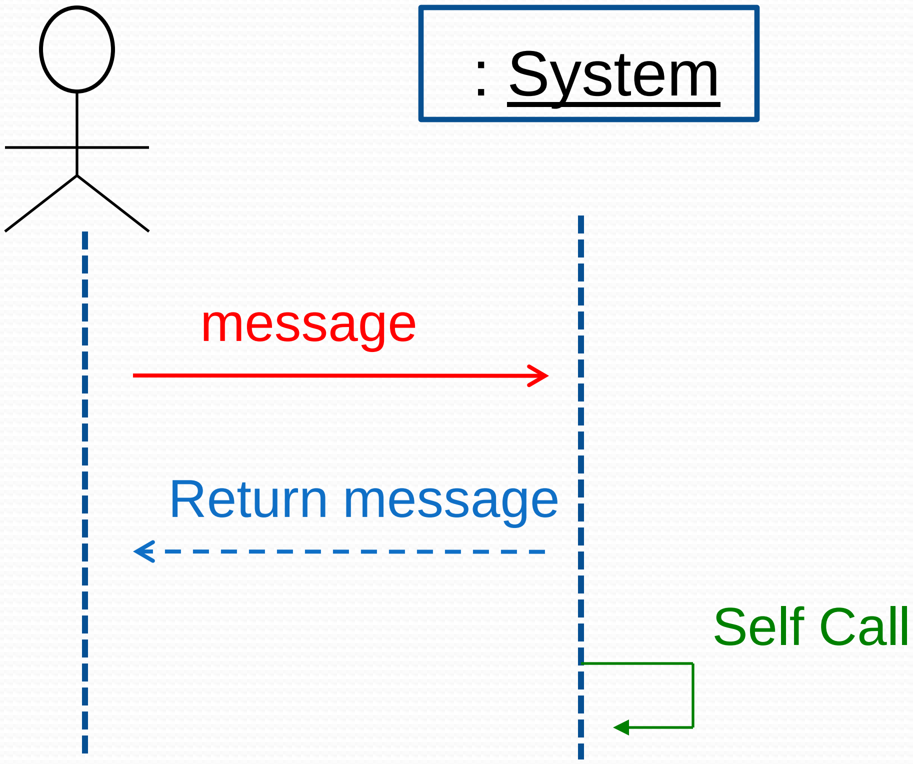


UML Requirement Model

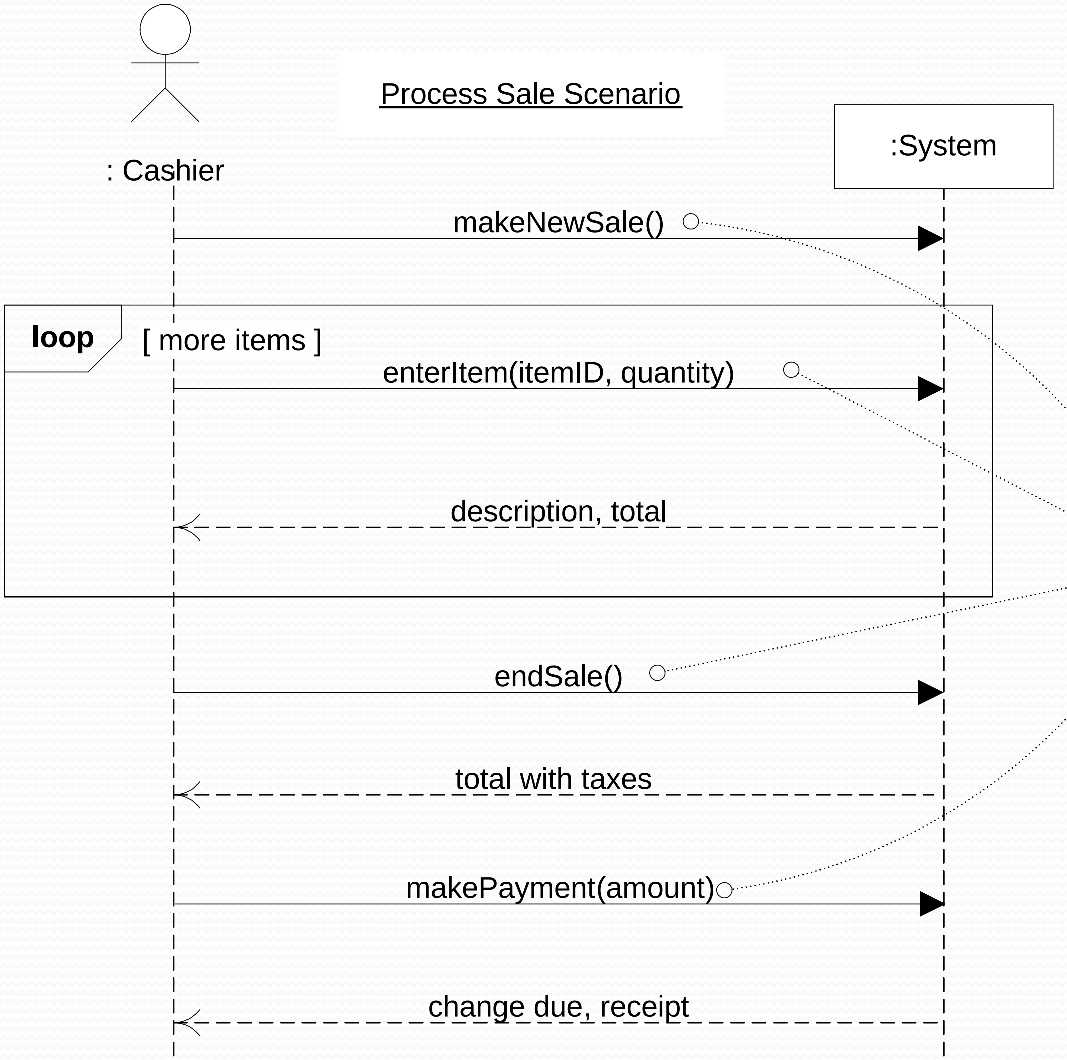
- Use Case
  - The focus in the Use Case is the Actors interacting with the System.
  - The actor generates events to the system
  - Events initiate operations upon the system
- Domain Model
  - The focus in the Domain model is the relationships between the conceptual classes
- System Sequence Diagrams (SSDs)
  - Derived from Use Cases
  - A SSD shows *one* Use Case Scenario.

SSD (System Sequence Diagram) Notation

- Messages are labelled on arrows to show messages sent to or received by actor or system



Example: POS Input Events



SSD shows system events

these input system events invoke system operations

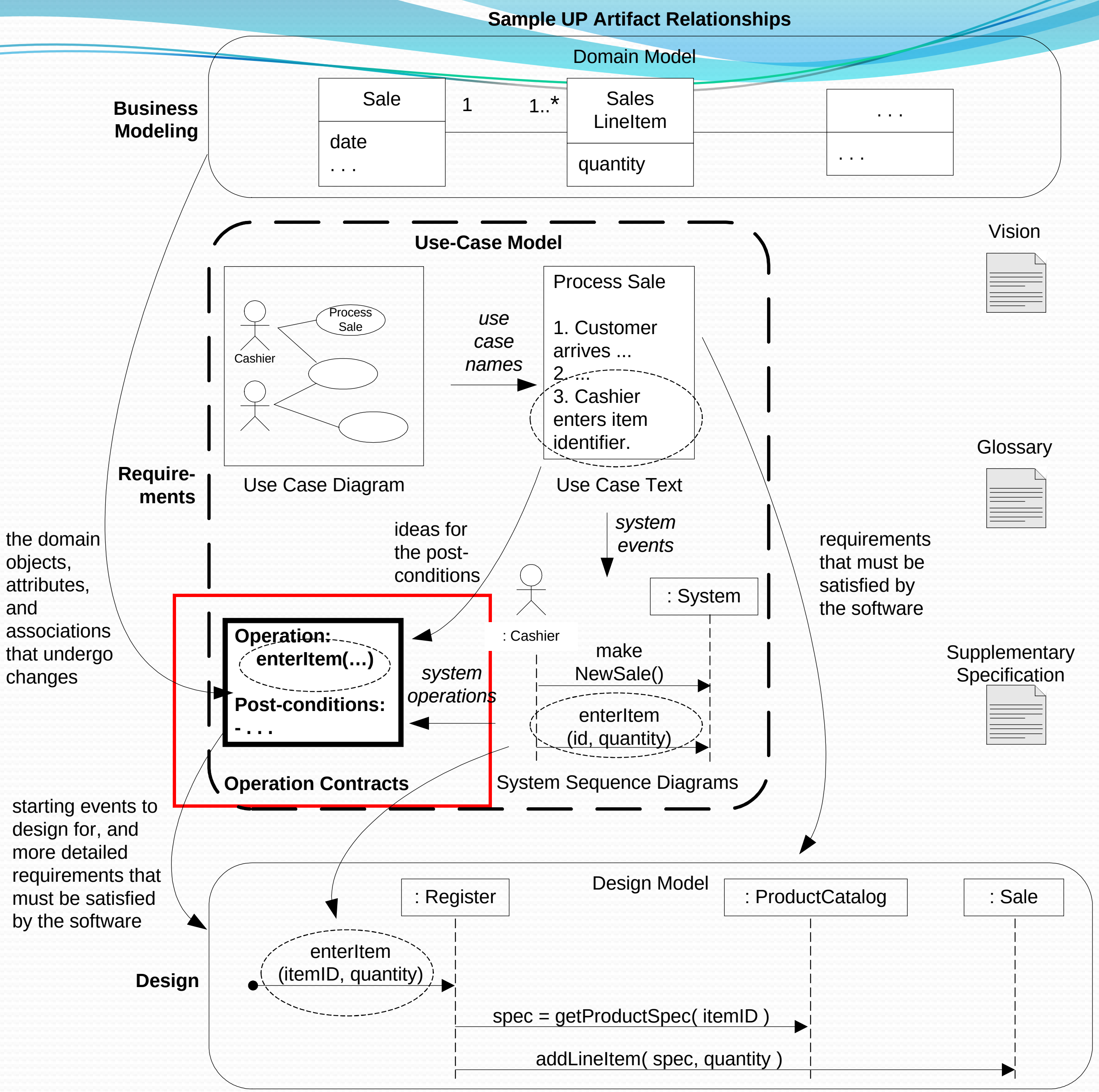
the system event *enterItem* invokes a system operation called enterItem and so forth

this is the same as in object-oriented programming when we say the message foo invokes the method (handling operation) foo

Larman Fig 11.2

Operation Contracts

Part of requirements gathering with the SSD being the prime-input



Operation Contracts - Why

- Operation Contracts become necessary when Use Cases are insufficient for describing system behavior.

- In most instances, Operation Contracts may not be necessary.

Operation Contract Sections

Operation:	Name of operation and parameters
Cross References:	(optional) Use cases this operation can occur within
Preconditions:	Noteworthy (non-trivial) *assumptions* about the state of the system or objects in the Domain Model before execution of the operation.
Postconditions:	The state of objects in the Domain Model after completion of the operation.

Example Of *enterItem* Operation Contract

Operation:	enterItem(itemID : ItemID, quantity : integer)
Cross References:	Use Cases: Process Sale
Preconditions:	There is a sale underway.
Postconditions:	✹ A *SalesLineItem* instance *sli* was created (instance creation) ✹ *sli* was associated with the current *Sale* (association formed) ✹ *sli.quantity* became *actual quantity* (attribute modification) ✹ *sli* was associated with a *ProductSpecification*, based on *itemID* match (association formed)

Object Design

“After identifying your requirements, documenting in Use Cases, creating a Domain Model, SSD and Sequence Diagram .....

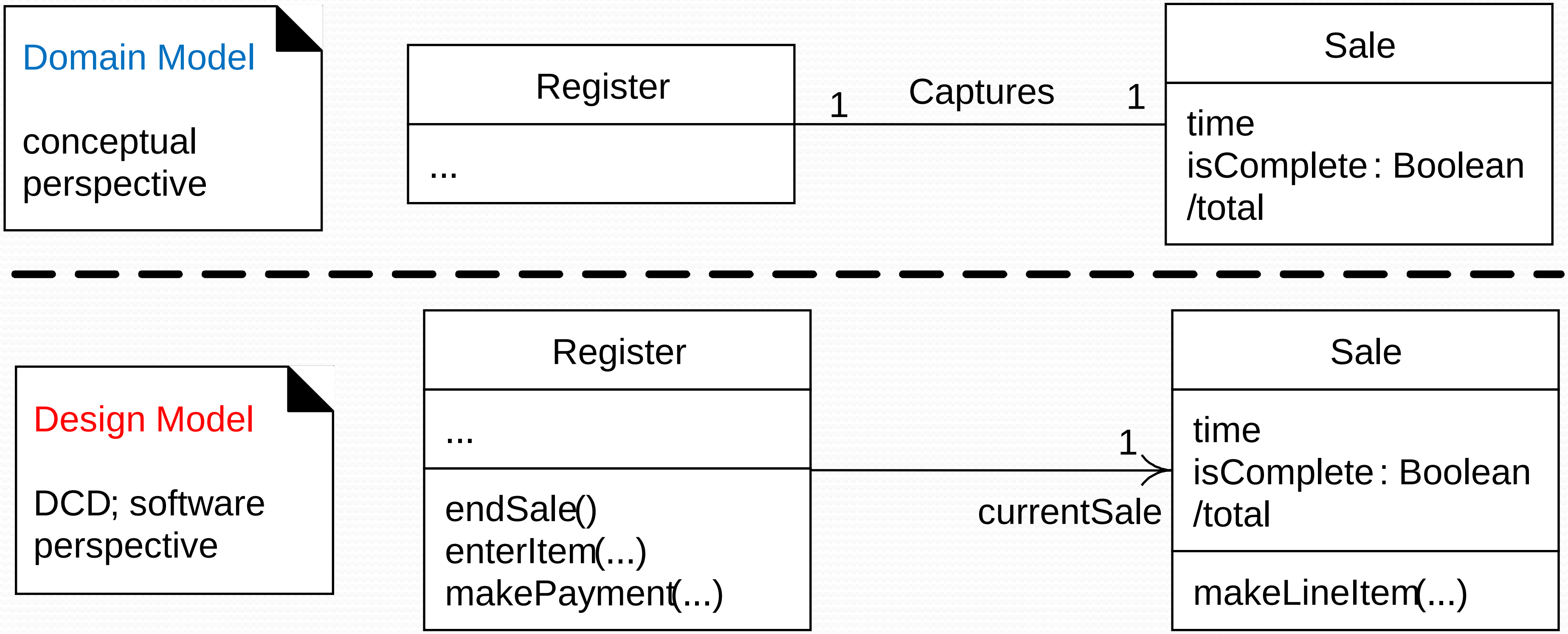
What is next?

“The next task is to add methods to the software classes, and define the messaging between the objects to fulfill the requirements”

UML Object Modeling

- Domain Model
  - Use Case / Use Case Diagrams
  - Conceptual Classes Diagrams
  - SSD
- Design Model
  - Design Classes Diagrams (DCD)
  - Interaction Diagrams
  - Package Diagrams

Conceptual vs Design Class Diagrams



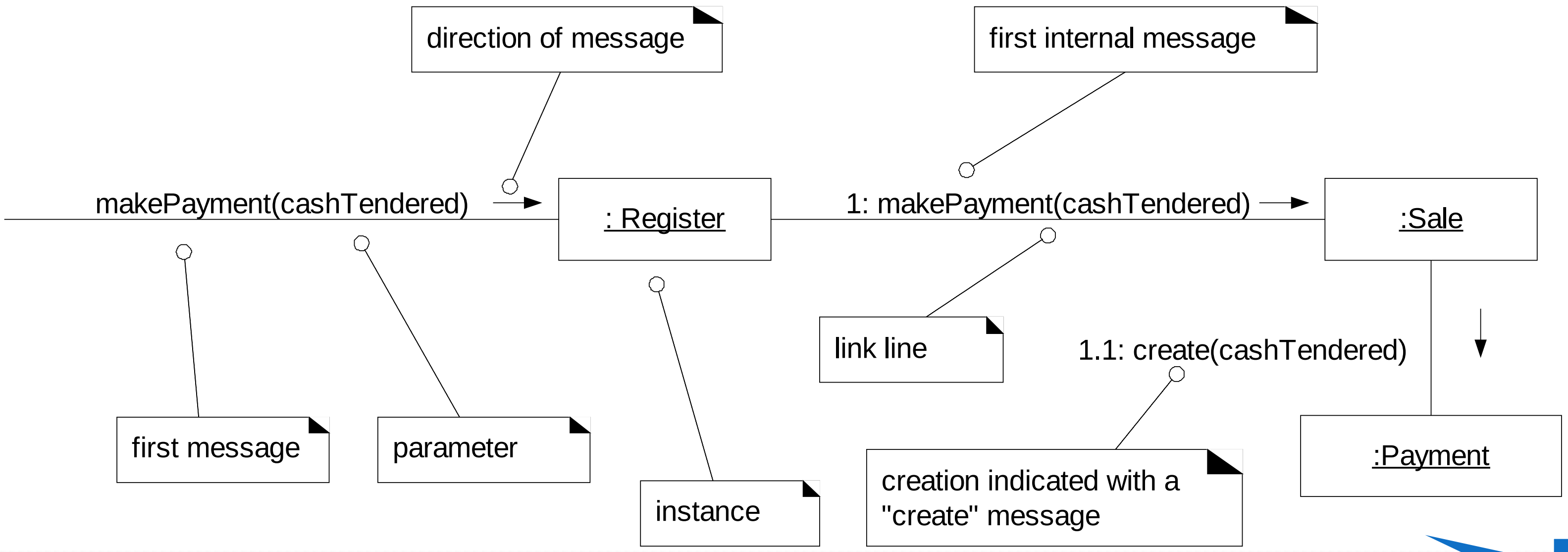
Main Steps in Developing a DCD

- Identify the classes:
  - nouns in the Use Cases Lab 2
  - scan the Conceptual Class Diagram (Domain Model) Lab 3
  - scan the Interaction Diagrams Lab 4
  - list out classes mentioned & those that appear needed:
    - controllers
    - database classes
    - parent classes for classes with a common heritage
    - etc…
- Draw the class diagram Lab 5

Interaction Diagrams
Sequence vs. Collaboration

Type	Strengths	Weaknesses
Sequence	✹ clearly shows sequence or time ordering of messages ✹ simple notation	✹ forced to extend to the right when adding new objects – consumes horizontal space
Collaboration (Communication)	✹ space economical – flexibility to add new objects in two dimensions ✹ better to illustrate complex branching, iteration, and concurrent	✹ difficult to see sequence of messages ✹ more complex notation

Example Interaction Diagrams: makePayment



direction of message

first internal message

makePayment(cashTendered)

: Register

1: makePayment(cashTendered)

:Sale

link line

1.1: create(cashTendered)

first message

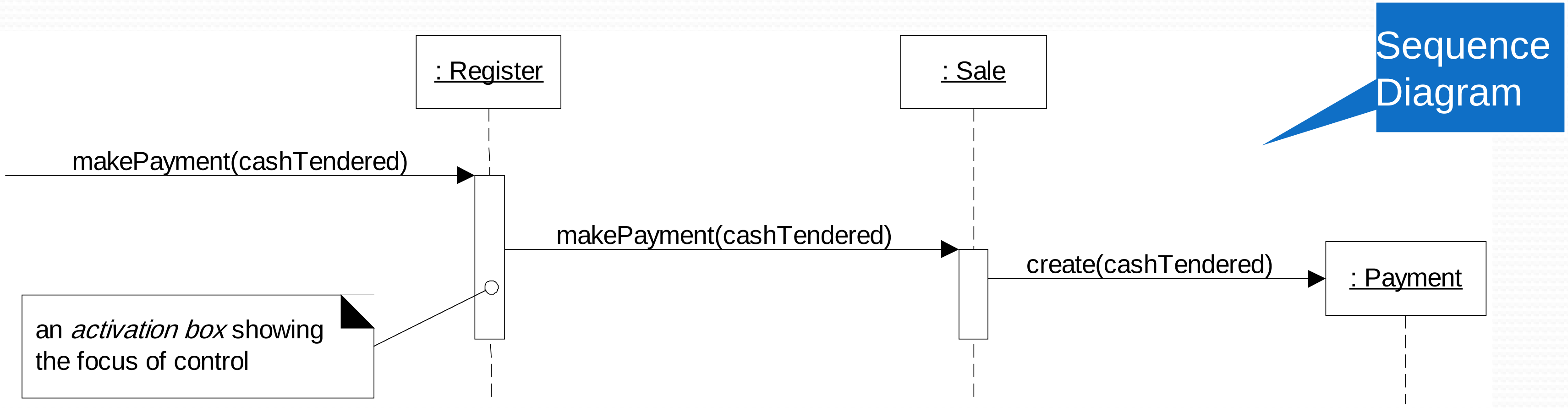
parameter

instance

creation indicated with a "create" message

:Payment

Collaboration Diagram



Sequence Diagram

: Register

: Sale

makePayment(cashTendered)

makePayment(cashTendered)

create(cashTendered)

: Payment

an *activation box* showing the focus of control

Dynamic Object Modeling

- *Guideline*
  - *Spend significant time doing interaction diagrams (sequence or communication diagrams), not just static object modeling with class diagrams.*
  - *Ignoring this guideline is a very common worst-practice with UML*

Quote from Larman, p.217

RDD - Responsibility Driven Design

- Think of software objects as having responsibilities ⟶ what they do
- Responsibilities are related to the obligations or behaviour of an object in terms of its role (its is abstraction)
- Methods fulfill responsibilities
- RDD – a general *Metaphore* of a community of collaborating responsible objects

GRASP

- G)eneral (R)esponsibility (A)ssignment (S)oftware (P)atterns (or Principles)
- Learning aid for OO Design with Responsibilities
- Key: Understand how to apply GRASP for OOD
- GRASP defines ***nine*** basic OOD principles

Nine **GRASP** Principles

- Information Expert
- Creator
- Controller
- Low Coupling
- High Cohesion
- Polymorphism
- Pure Fabrication
- Indirection
- Protected Variations

Important to grasp the first 5 principles

See inside front textbook cover