CS-3004
SOFTWARE DESIGN AND ANALYSIS

RUBAB JAFFAR

RUBAB.JAFFAR@NU.EDU.PK

Introduction

Software Processes, UP and UML

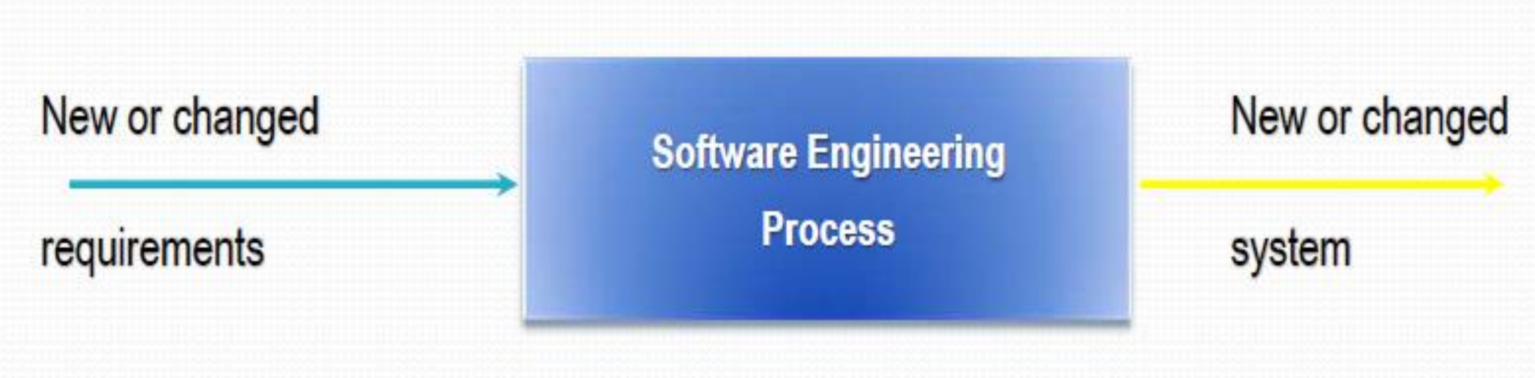
Lecture # 4, 5, 6

TODAY’S OUTLINE

- Software Process
- Software Development approaches
- Process models
- Unified Process
- UML
- Use case model

PROCESS

Defines Who is doing What, When to do it, and How to reach a certain goal.



- Workers, the ‘who’
- Activities, the ‘how’
- Artifacts, the ‘what’
- Workflows, the ‘when’

WHAT IS A PROCESS MODEL ?

It is a description of

i) what tasks need to be performed in

ii) what sequence under

iii) what conditions by

iv) whom to achieve the "desired results."

Why Process Model?

Provide "guidance" for a systematic coordination and controlling of

a) the tasks and of

b) the personnel who perform the tasks

Note the key words: coordination/control , tasks, people

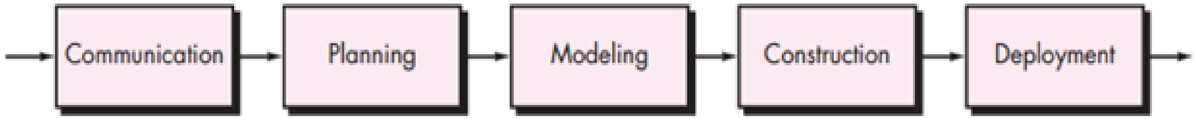
COMMON ACTIVITIES OF PROCESS MODEL

Many different software processes but all involve:

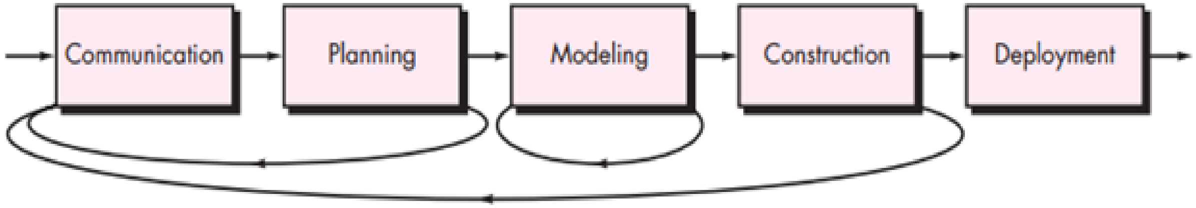
- Specification – defining what the system should do;
- Design and implementation – defining the organization of the system and implementing the system;
- Validation – checking that it does what the customer wants;
- Evolution – changing the system in response to changing customer needs.

GENERIC PROCESS FLOWS

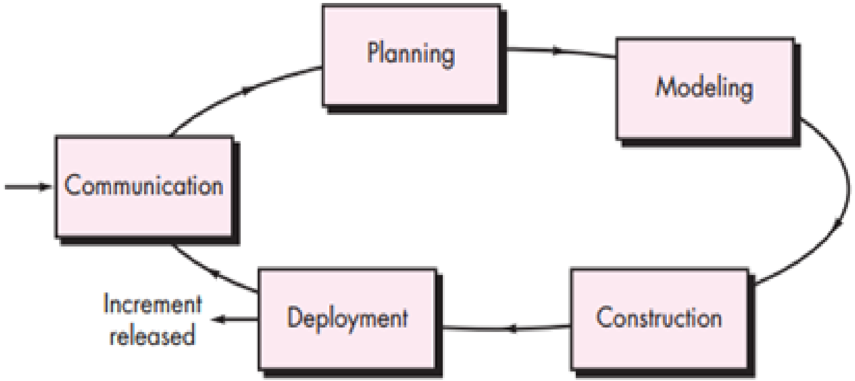
- Linear process flow
- Iterative process flow
- Incremental process flow



(a) Linear process flow

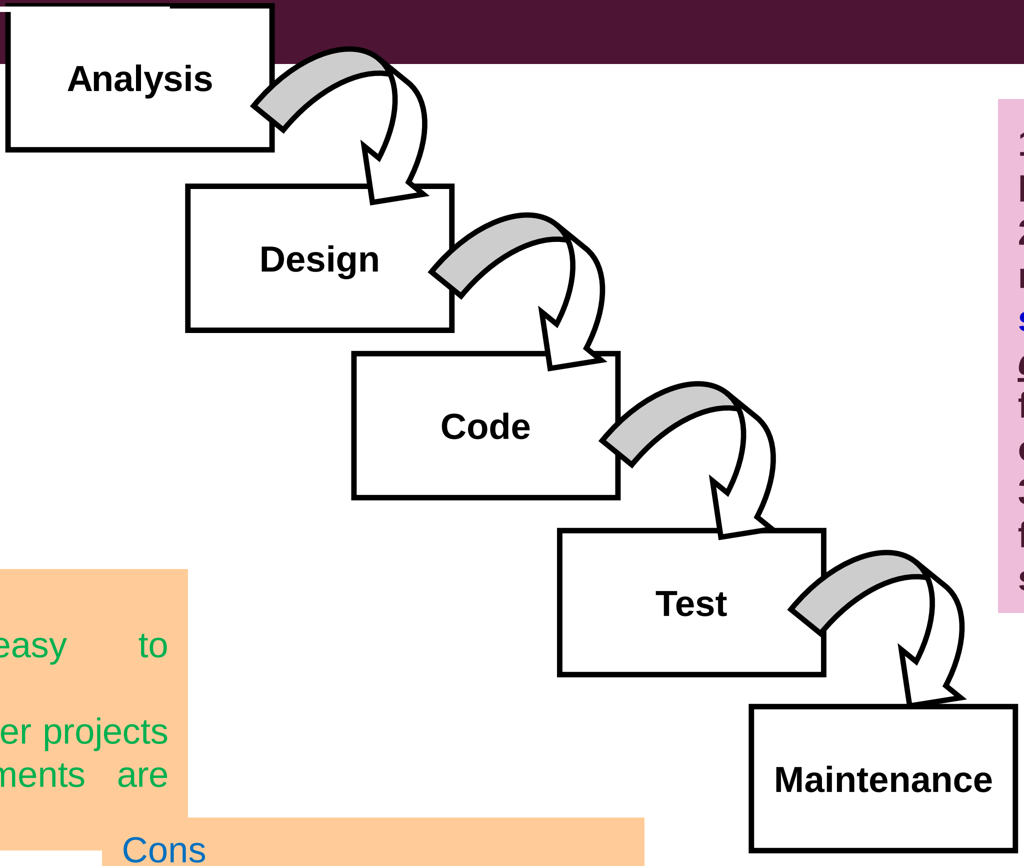


(b) Iterative process flow



(c) Evolutionary process flow

WATERFALL MODEL



1. Requirements must be specified.
2. Four main tasks must be completed in sequence: *Analysis*, *design*, *code*, and *test,* followed by integration of the system.
3. Output of one stage feeds into the next stage in sequence

Pros

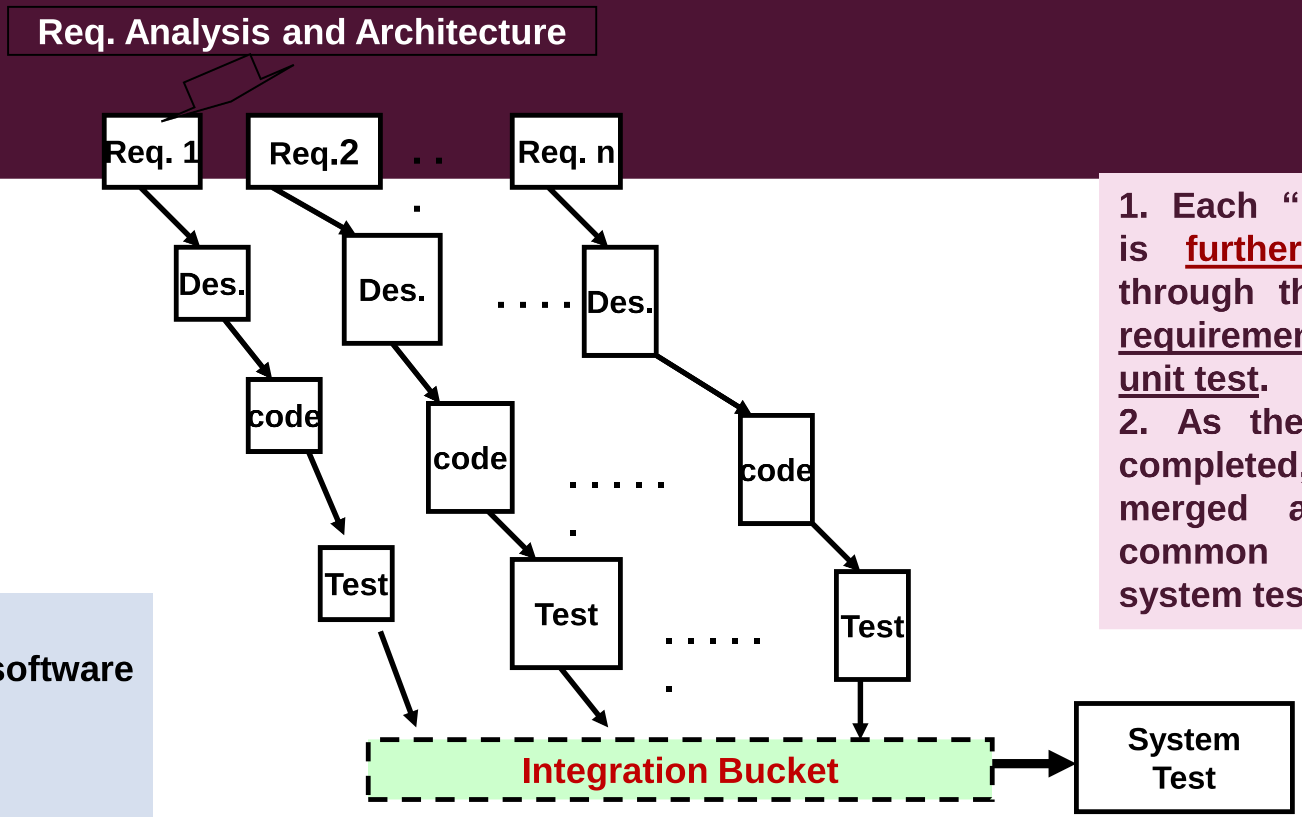
•Simple and easy to understand.

•Works well for smaller projects where the requirements are well-understood.

Cons

- No working product
- Inflexible

INCREMENTAL MODEL



1. Each “major requirement/item” is <u>further developed separately</u> through the same sequence of : <u>requirement</u>, <u>design</u>, <u>code</u>, and <u>unit test</u>.

2. As the developed pieces are completed, they are continuously merged and integrated into a common bucket for integrated system test

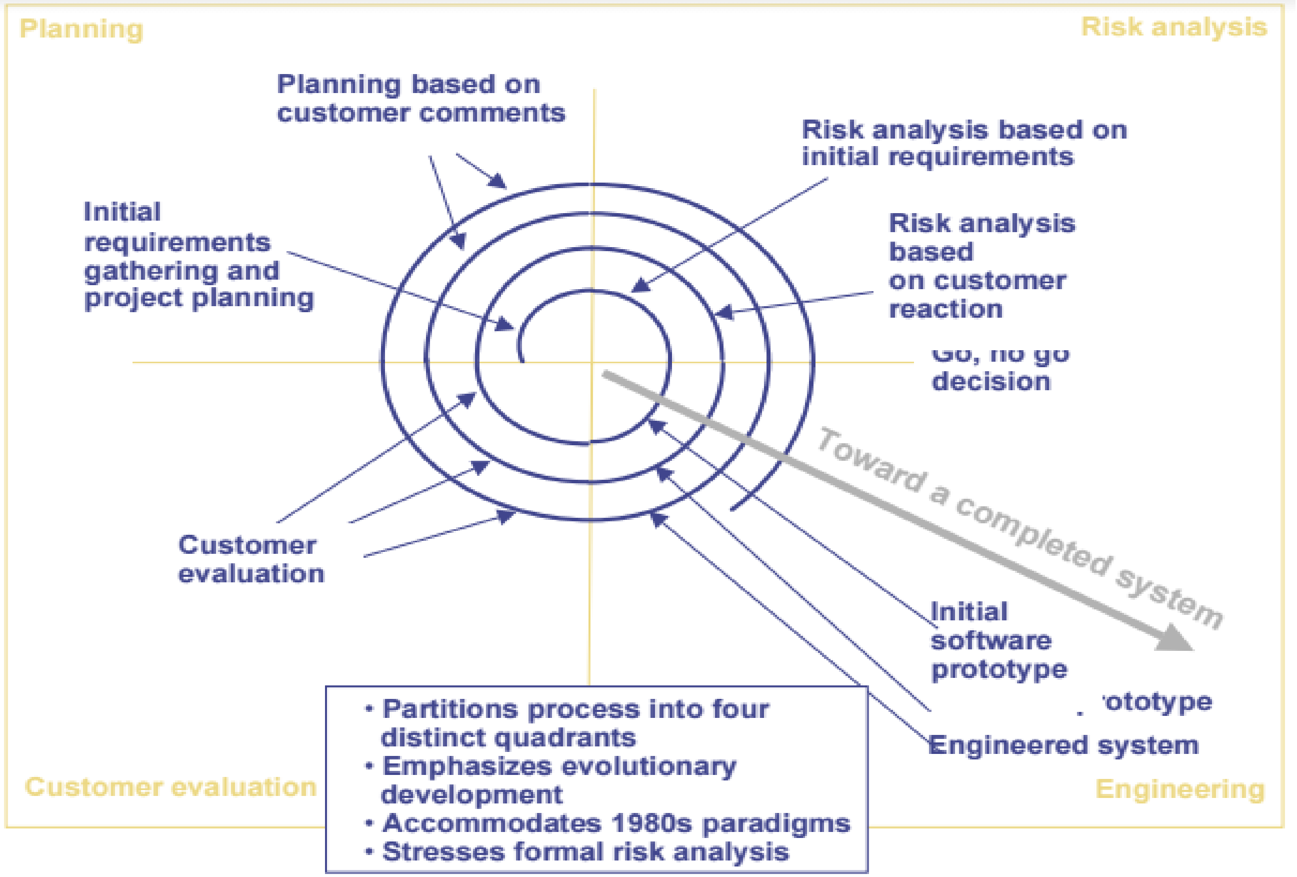
Pros

- Generatesworking software quickly.
- More flexible
- Less costly
- easier to test and debug.
- customer can respond to each built.

Cons

- Needs good planning and design.
- Needs a clear and complete definition of the whole system

ITERATION MODEL


Planning
Risk analysis
Planning based on customer comments
Risk analysis based on initial requirements
Initial requirements gathering and project planning
Risk analysis based on customer reaction
Go, no go decision
Toward a completed system
Customer evaluation
Initial software prototype
ototype
Engineered system
Customer evaluation
Engineering

- Partitions process into four distinct quadrants
- Emphasizes evolutionary development
- Accommodates 1980s paradigms
- Stresses formal risk analysis

RAPID SOFTWARE DEVELOPMENT

- Rapid development and delivery is now often the most important requirement for software systems
  - Businesses operate in a fast –changing requirement and it is practically impossible to produce a set of stable software requirements
  - Software has to evolve quickly to reflect changing business needs.
- Plan-driven development is essential for some types of system but does not meet these business needs.
- Agile development methods emerged in the late 1990s whose aim was to radically reduce the delivery time for working software systems

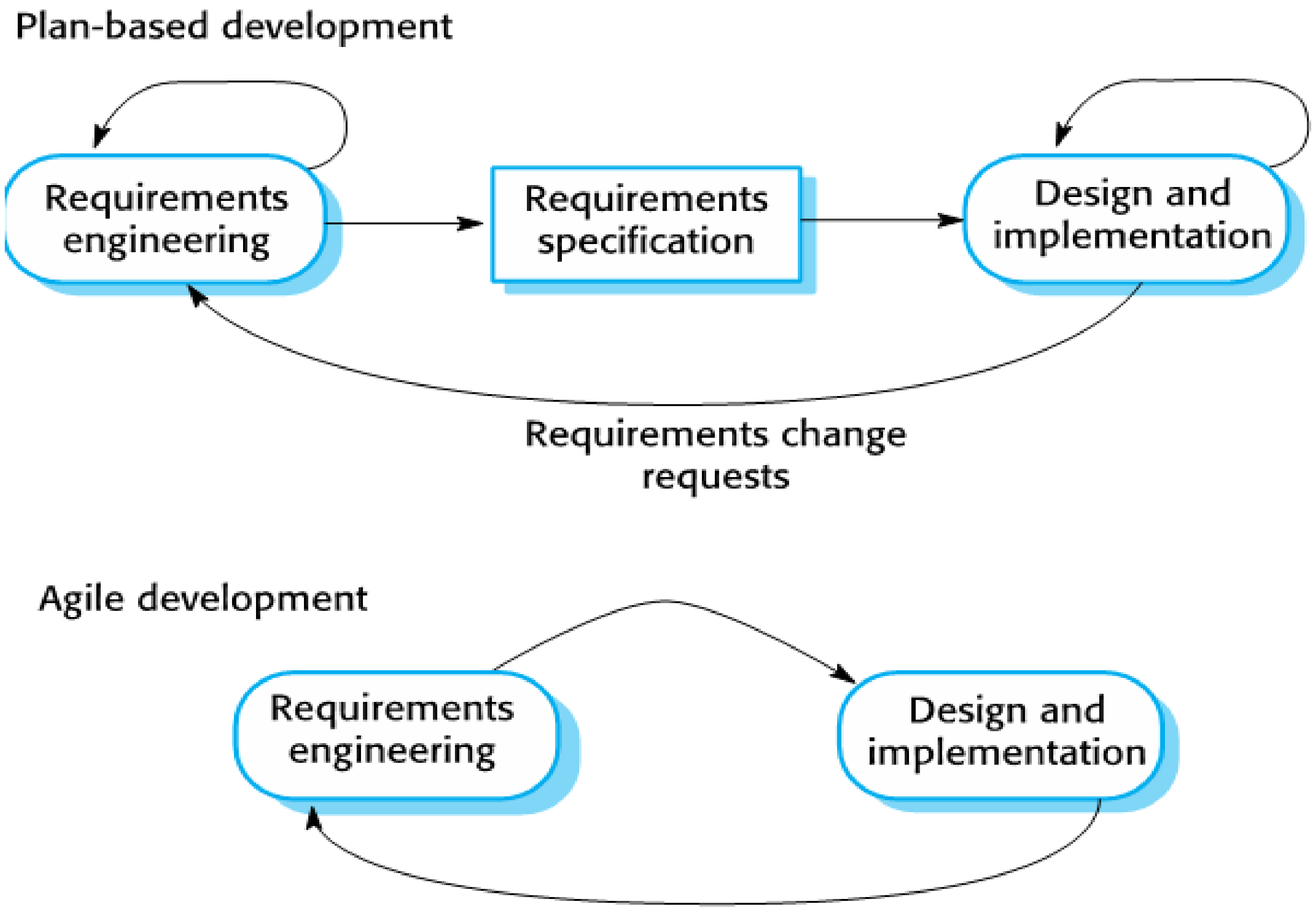
AGILE DEVELOPMENT CHARACTERISTICS

- Program specification, design and implementation are inter-leaved
- The system is developed as a series of versions or increments with stakeholders involved in version specification and evaluation
- Frequent delivery of new versions for evaluation
- Extensive tool support (e.g. automated testing tools) used to support development.
- Minimal documentation – focus on working code

PLAN-DRIVEN AND AGILE DEVELOPMENT

- Plan-driven development
  - A plan-driven approach to software engineering is based around separate development stages with the outputs to be produced at each of these stages planned in advance.
  - Not necessarily waterfall model – plan-driven, incremental development is possible
  - Iteration occurs within activities.
- Agile development
  - Specification, design, implementation and testing are inter-leaved and the outputs from the development process are decided through a process of negotiation during the software development process.

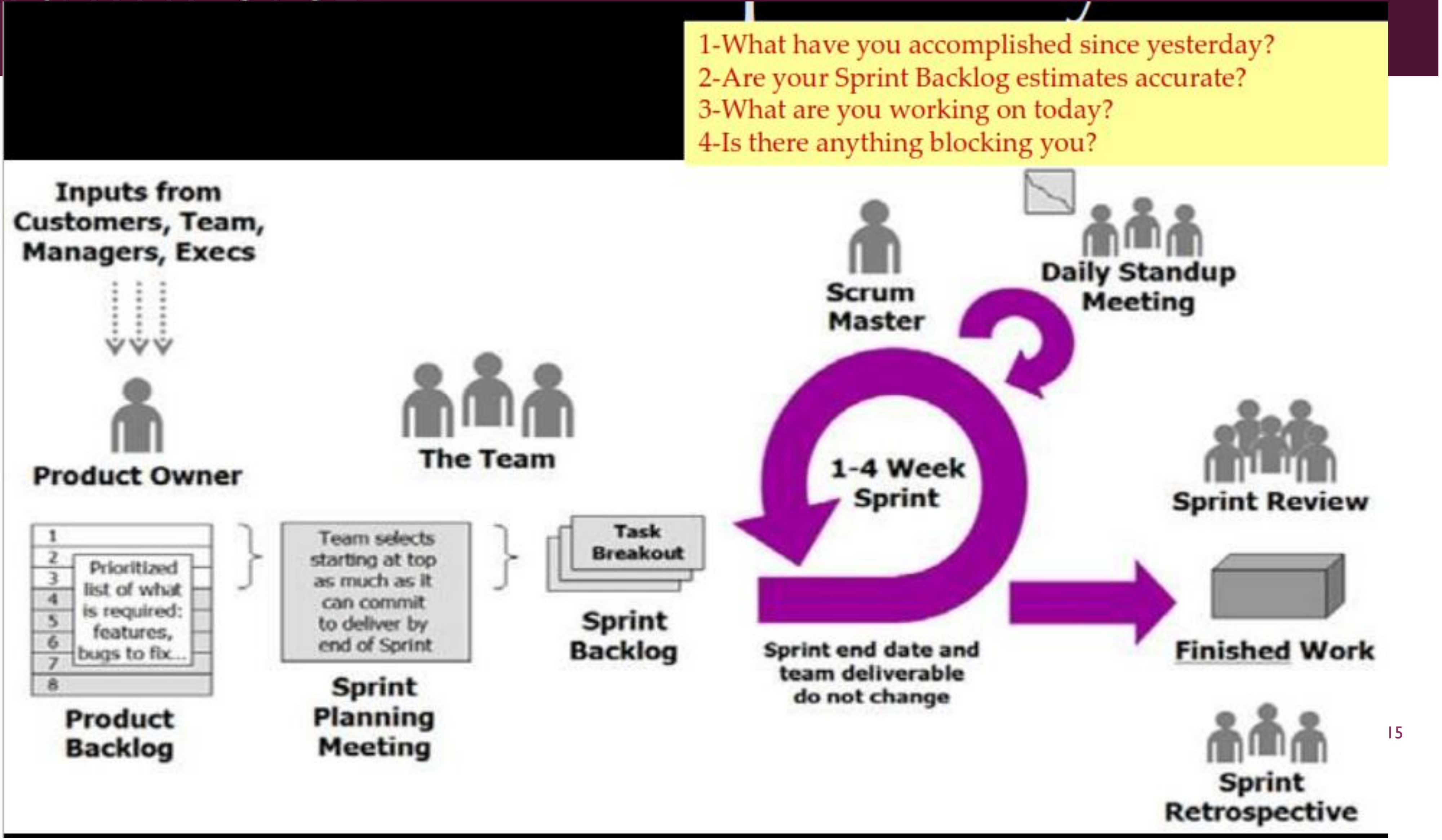
PLAN-DRIVEN AND AGILE DEVELOPMENT


Plan-based development
Requirements engineering
Requirements specification
Design and implementation
Requirements change requests
Agile development
Requirements engineering
Design and implementation

AGILE MODELS

- XP
- Scrum

THE SCRUM SPRINT CYCLE



1-What have you accomplished since yesterday?
2-Are your Sprint Backlog estimates accurate?
3-What are you working on today?
4-Is there anything blocking you?

Inputs from Customers, Team, Managers, Execs
Product Owner
The Team
Scrum Master
Daily Standup Meeting
1-4 Week Sprint
Sprint Review
1
2
3
4
5
6
7
8
Prioritized list of what is required: features, bugs to fix...
Team selects starting at top as much as it can commit to deliver by end of Sprint
Task Breakout
Finished Work
Product Backlog
Sprint Planning Meeting
Sprint Backlog
Sprint end date and team deliverable do not change
Sprint Retrospective

UNIFIED PROCESS MODEL

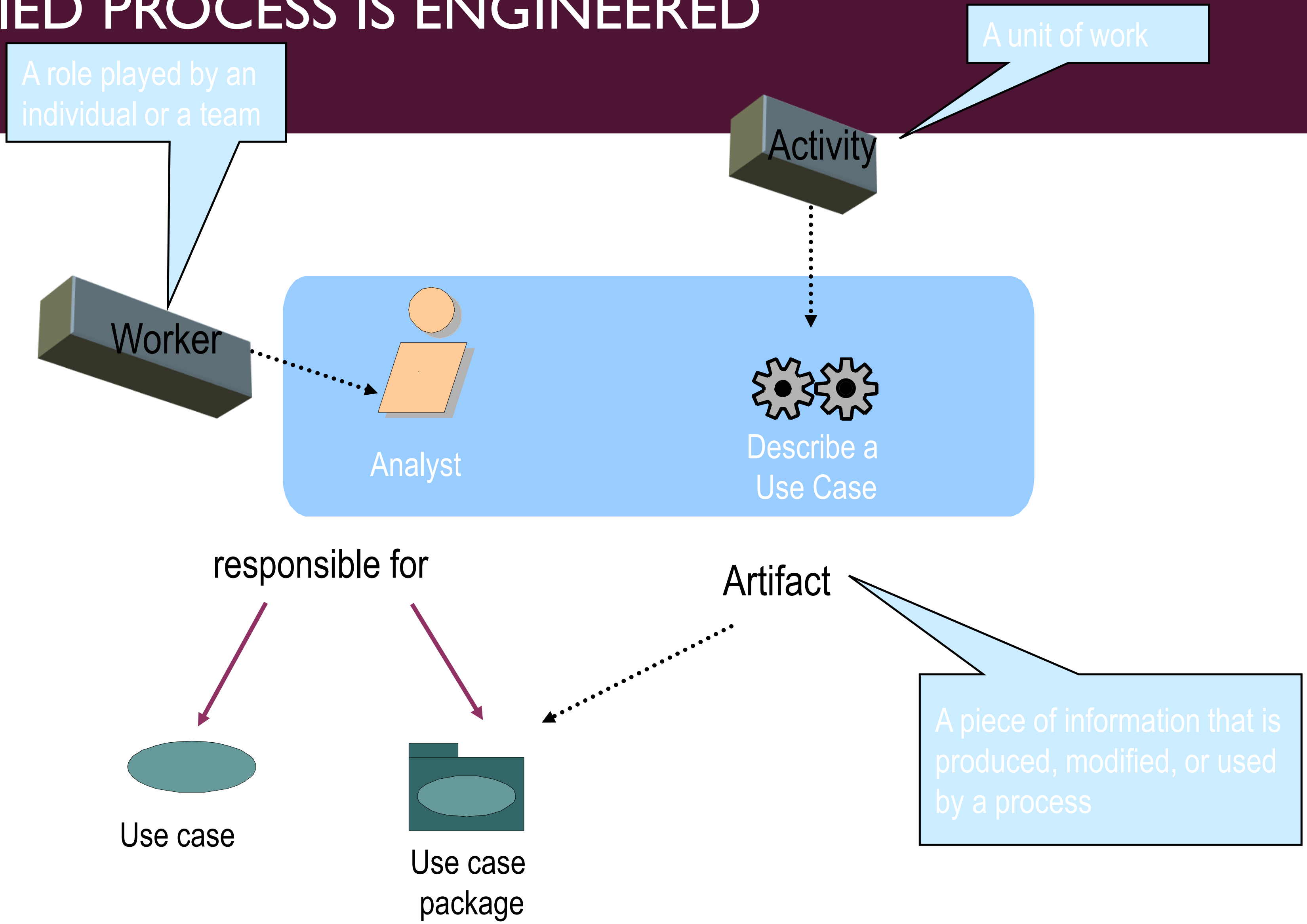
- A process model that was created 1997 to give a framework for Object-oriented Software Engineering
- Iterative, incremental model to adapt to specific project needs
- Risk driven development
- Combining spiral and evolutionary models
- **2D process : phases and workflows**
- **Utilizes Miller's Law**

UNIFIED PROCESS

- The Unified Process is not simply a process, but rather an extensible framework which should be customized for specific organizations or projects.

- The Rational Unified Process is, similarly, a customizable framework. As a result, it is often impossible to say whether a refinement of the process was derived from UP or from RUP, and so the names tend to be used interchangeably.

THE UNIFIED PROCESS IS ENGINEERED


A role played by an individual or a team
A unit of work
Activity
Worker
Analyst
Describe a Use Case
responsible for
Artifact
A piece of information that is produced, modified, or used by a process
Use case
Use case package

BUILDING BLOCKS OF UP

- All aspects of the Rational Unified Process are based on a set of building blocks, which are used to describe what should be produced, who is in charge of producing it, how production will take place, and when production is complete.
- These four building blocks are:

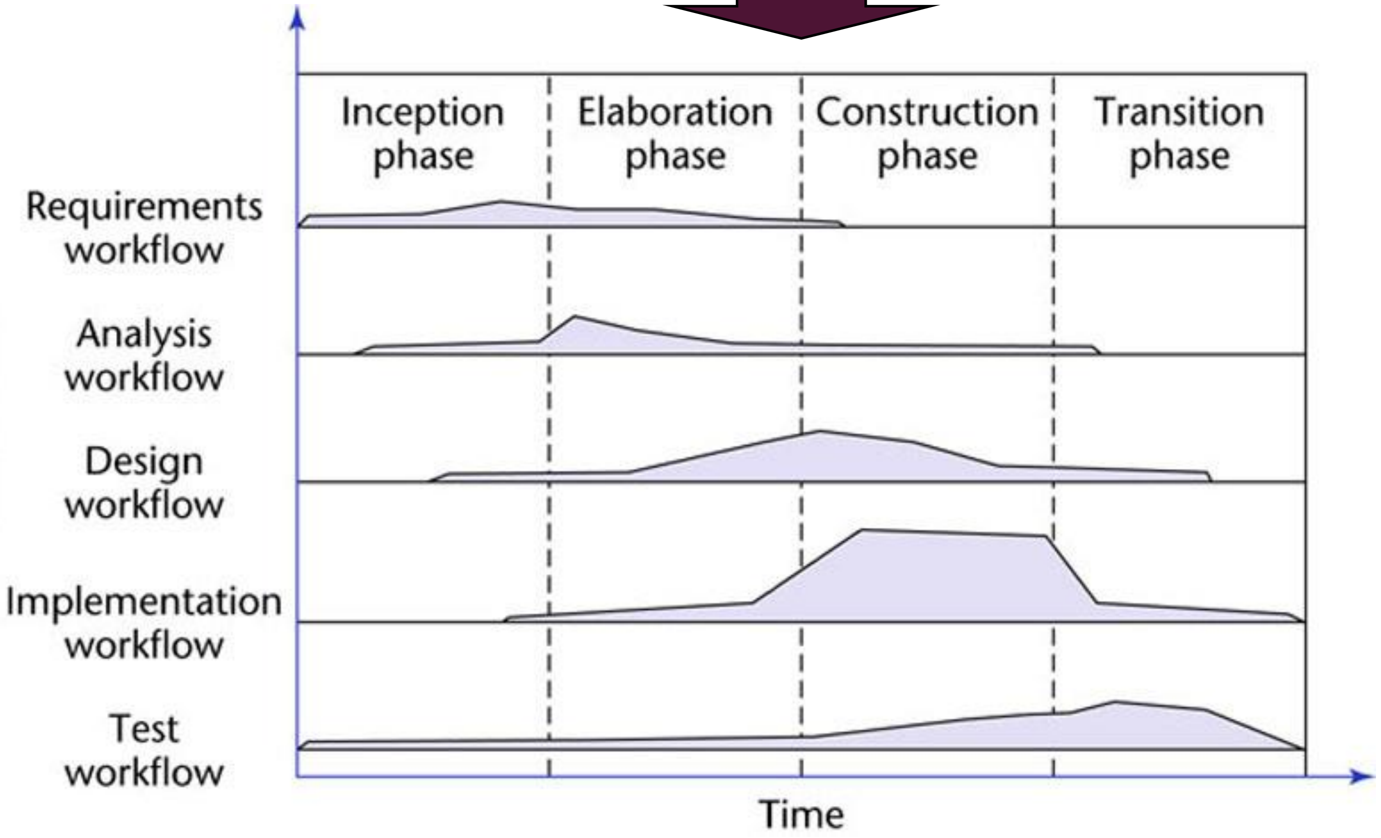
- Workers, the 'Who': The behavior and responsibilities of an individual, or a group of individuals together as a team, working on any activity in order to produce artifacts.
- Activities, the 'How': A unit of work that a worker has to perform. Activities should have a clear purpose, typically by creating or updating artifacts.
- Artifacts, the 'What': An artifact represents any tangible output that emerges from the process; anything from new code functions to documents to additional life cycle models.
- Workflows, the 'When': Represents a diagrammed sequence of activities, in order to produce observable value and artifacts.

THE PHASES/WORKFLOWS OF THE UNIFIED PROCESS

- Phase is Business context of a step

[redacted]

Workflow

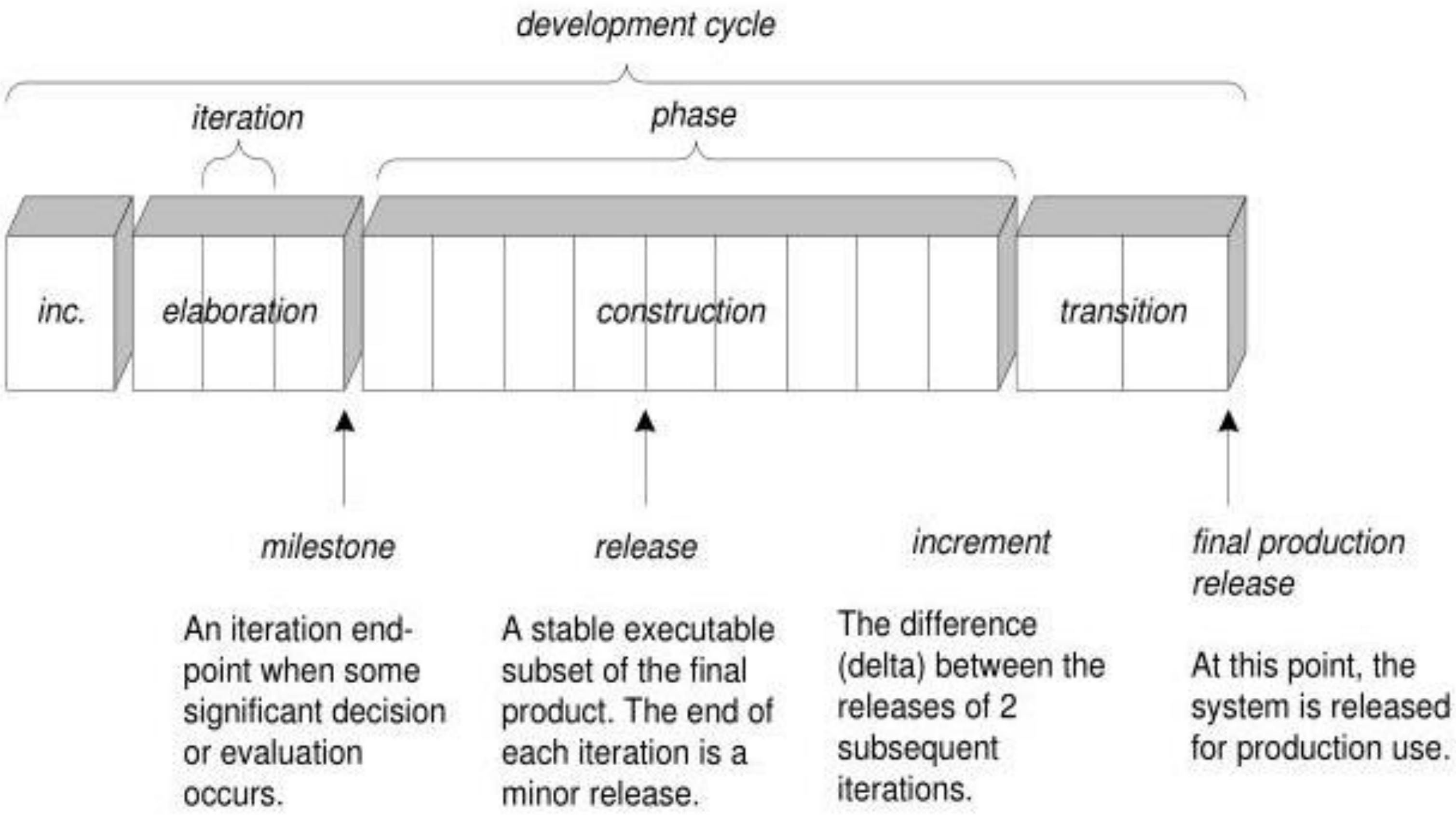


THE UNIFIED PROCESS (UP)

- A software development process describes an approach to building, deploying, and maintaining software.

- UP has emerged as a popular and effective iterative software development process for building OO systems.
  - Rational Unified Process (RUP) is a modified version of the Unified Process, which was modified by Rational Software, is widely practiced and adopted by our industry.

THE UP PHASES



milestone

An iteration end-point when some significant decision or evaluation occurs.

release

A stable executable subset of the final product. The end of each iteration is a minor release.

increment

The difference (delta) between the releases of 2 subsequent iterations.

final production release

At this point, the system is released for production use.

RISK-DRIVEN PLANNING

- Identify and drive down the highest risks
  - Includes the practice of **architecture-centric iterative development**. i.e., early iterations focus on building, testing and stabilizing the core architecture.
  - Tackles the difficult things first

CLIENT-DRIVEN PROGRAMMING

- Builds visible features that the client cares most about.

ADVANTAGES OF UP

- Emphasis on addressing very early high risk areas.
- It does not assume a fixed set of requirements at the inception of the project, but allows to refine the requirements as the project evolves.
- It does not put a strong focus on documents.
- The main focus is the software product and its quality.

UP BEST PRACTICES

- Get high risk and high value first
- Constant user feedback and engagement

- Early cohesive core architecture
- Test early, often and realistically

- Apply use cases where needed
- Do some visual modeling with UML

- Manage requirements and scope creep
- Manage change requests and configuration

UP PHASES

- **Inception**
- Approximate vision, business case, scope, vague estimates.
- Note: Inception is not a requirements phase; rather, it is a kind of feasibility phase, where just enough investigation is done to support a decision to continue or stop.

UP PHASES

- **Elaboration**

- Refined vision, iterative implementation of the core architecture, resolution of high risks, identification of most requirements and scope, more realistic estimates

- Note: Elaboration is not the requirements or design phase; rather, it is a phase where the core architecture is iteratively implemented, and high risk issues are mitigated.

UP PHASES

- **Construction**
- Iterative implementation of the remaining lower risk and easier elements, and preparation for deployment.
- Note: Construction is not the implementation phase

UP PHASES

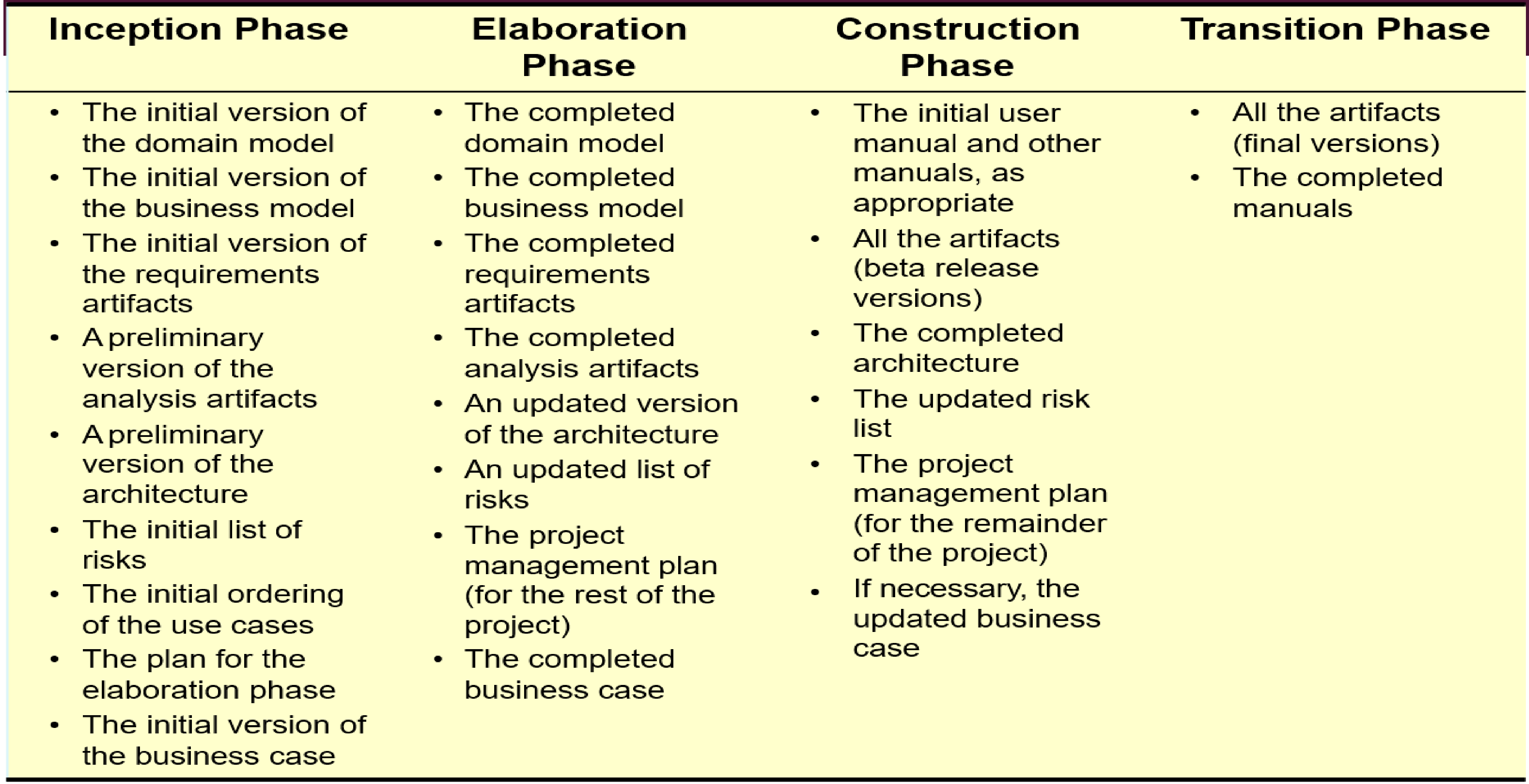
- **Transition**

Beta tests, deployment

UNIFIED PROCESS PHASES

- Inception
  - Define **business case, risks, 10% requirements** identified, estimate next phase effort.
- Elaboration
  - **Understanding of problem / architecture, risk significant units** are **coded/tested**, **80%** requirements identified.
- Construction
  - System design, **programming and testing**.
- Transition
  - **Deploy** the system in its operating environment.

PHASE DELIVERABLES



Inception Phase	Elaboration Phase	Construction Phase	Transition Phase
• The initial version of the domain model • The initial version of the business model • The initial version of the requirements artifacts • A preliminary version of the analysis artifacts • A preliminary version of the architecture • The initial list of risks • The initial ordering of the use cases • The plan for the elaboration phase • The initial version of the business case	• The completed domain model • The completed business model • The completed requirements artifacts • The completed analysis artifacts • An updated version of the architecture • An updated list of risks • The project management plan (for the rest of the project) • The completed business case	• The initial user manual and other manuals, as appropriate • All the artifacts (beta release versions) • The completed architecture • The updated risk list • The project management plan (for the remainder of the project) • If necessary, the updated business case	• All the artifacts (final versions) • The completed manuals

EXAMPLE ROLES IN UP

- **Stake Holder**: customer, product manager, etc.
- **Software Architect**: established and maintains architectural vision
- **Process Engineer**: leads definition and refinement of Development Case
- **Graphic Artist**: assists in user interface design, etc.

PLANNING WORKFLOW

- Define scope of Project
- Define scope of next iteration
- Identify Stakeholders
- Capture Stakeholders expectation
- Build team
- Assess Risks
- Plan work for the iteration
- Plan work for Project
- Develop Criteria for iteration/project closure/success
- UML concepts used: initial Business Model, using class diagram

THE UML

WHAT IS UML?

- The Unified Modeling Language (UML) is a language for **specifying, visualizing, constructing, and documenting** the artifacts of software systems, as well as for business modeling and other non-software systems.
- Visual

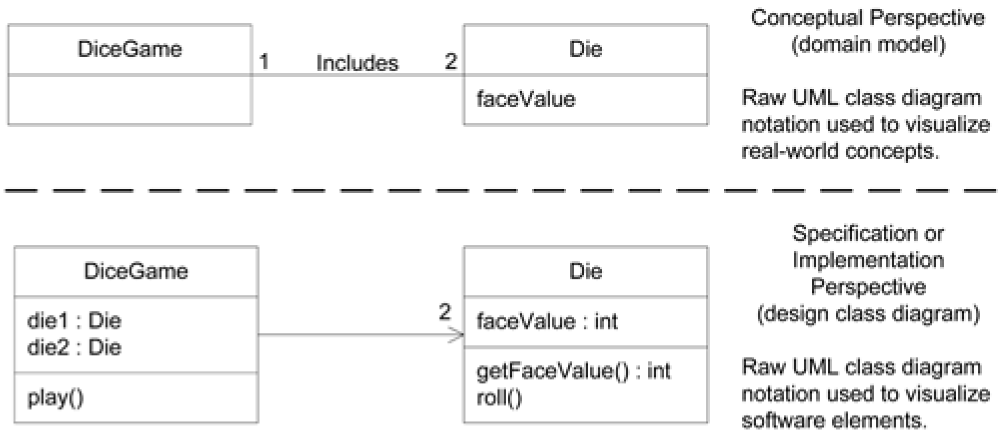
THREE WAYS TO APPLY UML

- UML as a Sketch
- UML as a Blueprint
- UML as a Programming Language

What does Agile Modeling emphasize?

THE PERSPECTIVES TO APPLY UML

- Conceptual Perspective
- Specification (Software) Perspective
- Implementation (Software) Perspective



Conceptual Perspective
(domain model)

Raw UML class diagram notation used to visualize real-world concepts.

Specification or Implementation Perspective
(design class diagram)

Raw UML class diagram notation used to visualize software elements.

"CLASS" IN DIFFERENT PERSPECTIVES

- Conceptual Class
- Specification Class
- Implementation Class

UML AND “SILVER BULLET” THINKING

DO UML DIAGRAMS REALLY MAKE THINGS BETTER?

UML AS A SKETCH DOMAIN MODEL

DOMAIN MODELLING

- The end goal of object-oriented analysis and design is the construction of the classes that will be used to implement the desired software system.
- Domain modeling is a first step in that direction.
- Why: Domain modeling helps us to identify the relevant concepts and ideas of a domain
- When: Domain modeling is done during object-oriented analysis
- Guideline: Only create a domain model for the tasks at hand

WHAT IS A DOMAIN MODEL?

- ***Problem domain :*** *area (**scope**)of application that needed to be investigated to solve the problem*

- ***Domain Model** : Illustrates **meaningful conceptual objects** in problem domain.*

- *So domain model are conceptual objects of the area of application to be investigated*

- The Domain Model illustrates noteworthy concepts in a domain

DOMAIN MODEL REPRESENTATION

- Captures the most important types of objects in a system.
- *A domain model is a visual representation of* ***real world concepts*** *(real-situation objects), that could be :* ***idea, thing*** *,* ***event*** *or* ***object****…..etc .*
  - ➢ **Business objects** - represent things that are manipulated in the business e.g. **Order**.
  - ➢ ***Real world objects*** – things that the business *keeps track of* e.g. ***Contact , book***.
  - ➢ ***Events*** *that come to light* **-** e.g. ***sale***.

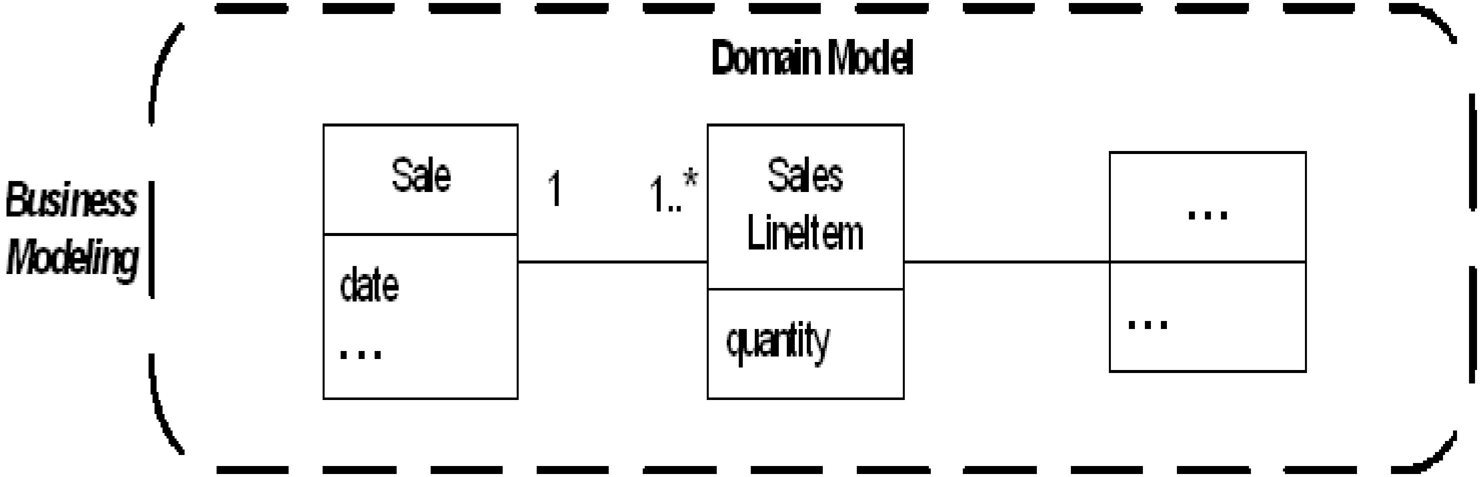
DOMAIN MODEL REPRESENTATION

Domain Model may contain :

- Domain **objects** (conceptual classes)
  - **Attributes** of domain objects
- **Associations** between domain objects
  - **Multiplicity**

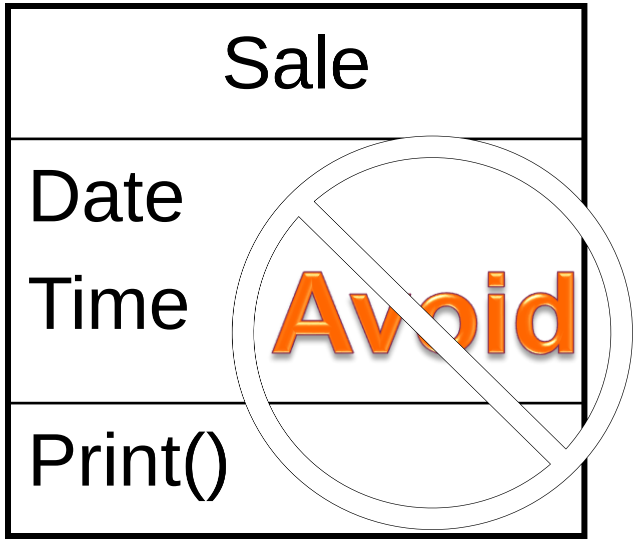
DOMAIN MODEL - UML NOTATION

- Illustrated using a set of domain objects (conceptual classes) ***with no operations*** (*no* ***responsibility*** *assigned yet , **this will be assigned during design**).*

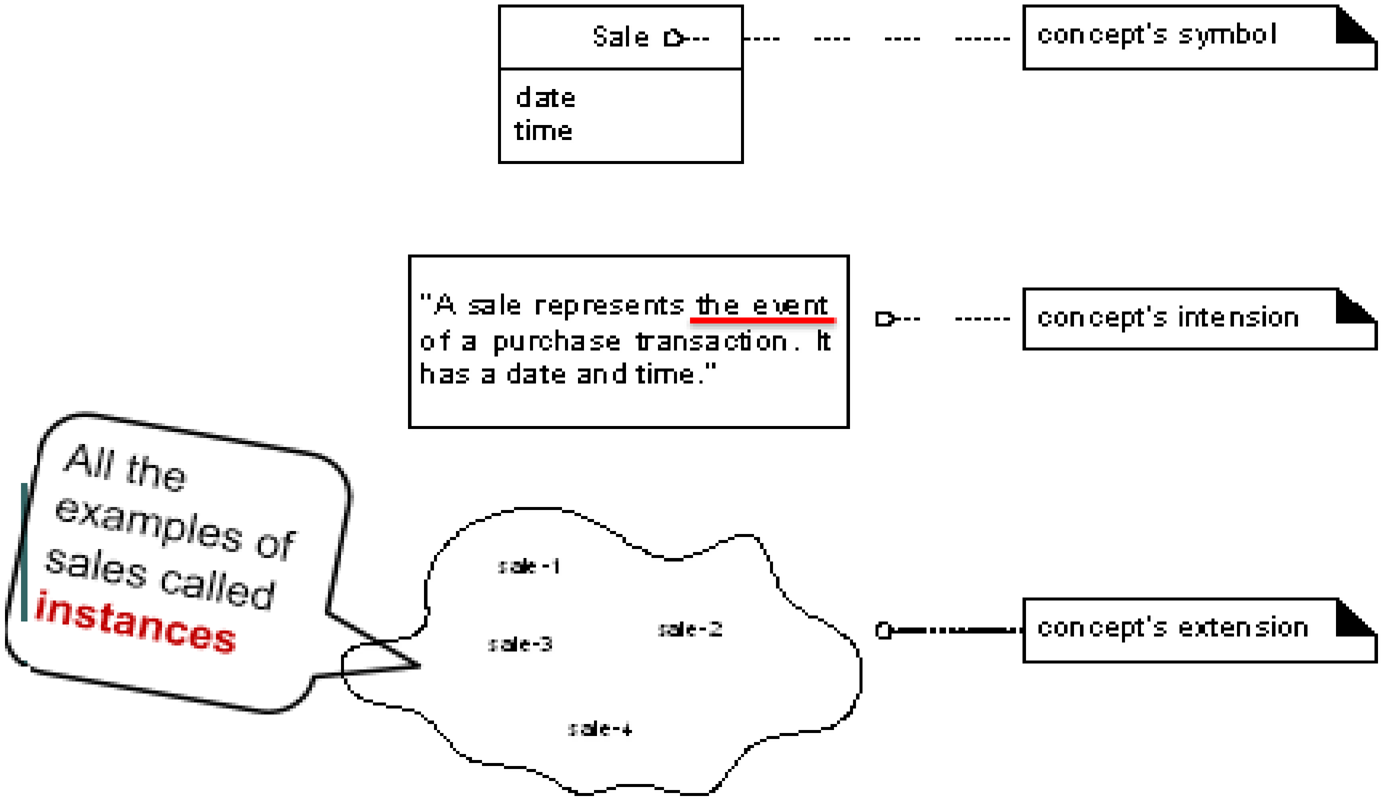


A DOMAIN MODEL IS NOT A SOFTWARE DOCUMENT

Object **responsibilities** is not part of the domain model. (***But to consider during Design***)

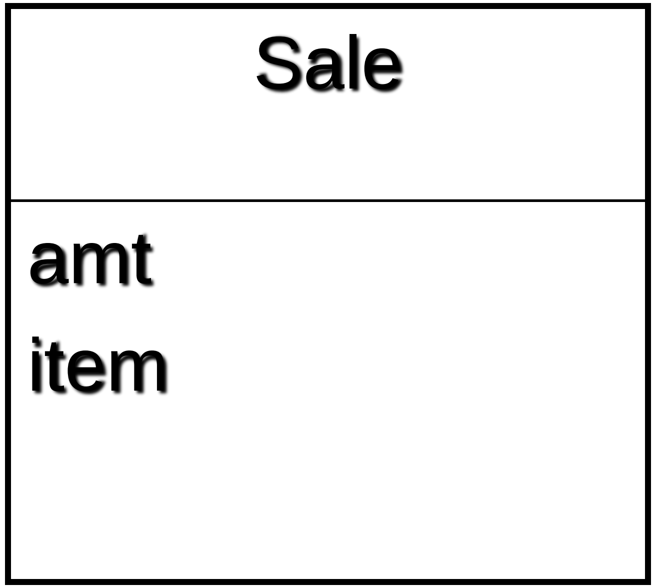


SYMBOL, INTENSION AND EXTENSION.


Sale
date
time
concept's symbol
"A sale represents the event of a purchase transaction. It has a date and time."
concept's intension
All the examples of sales called instances
sale-1
sale-3
sale-2
sale-4
concept's extension

A DOMAIN MODEL IS CONCEPTUAL, NOT A SOFTWARE ARTIFACT

Conceptual Class:



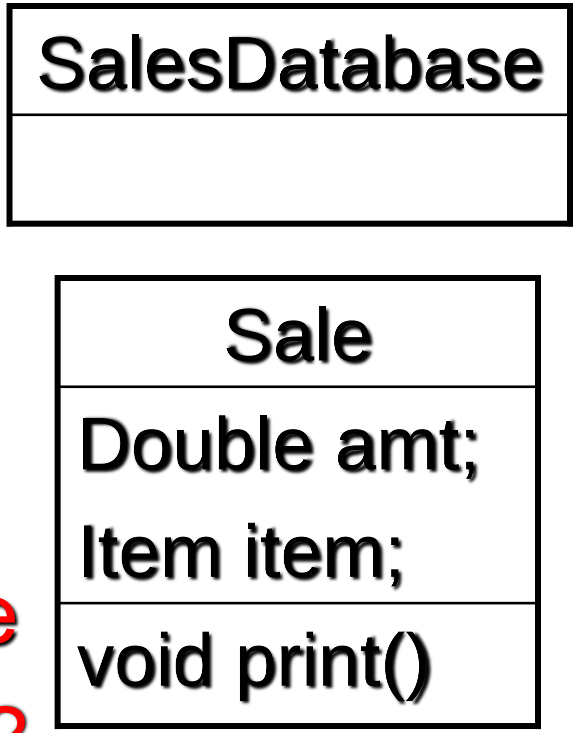
Sale

amt

item

vs.

Software Artifacts:



SalesDatabase

Sale

Double amt;

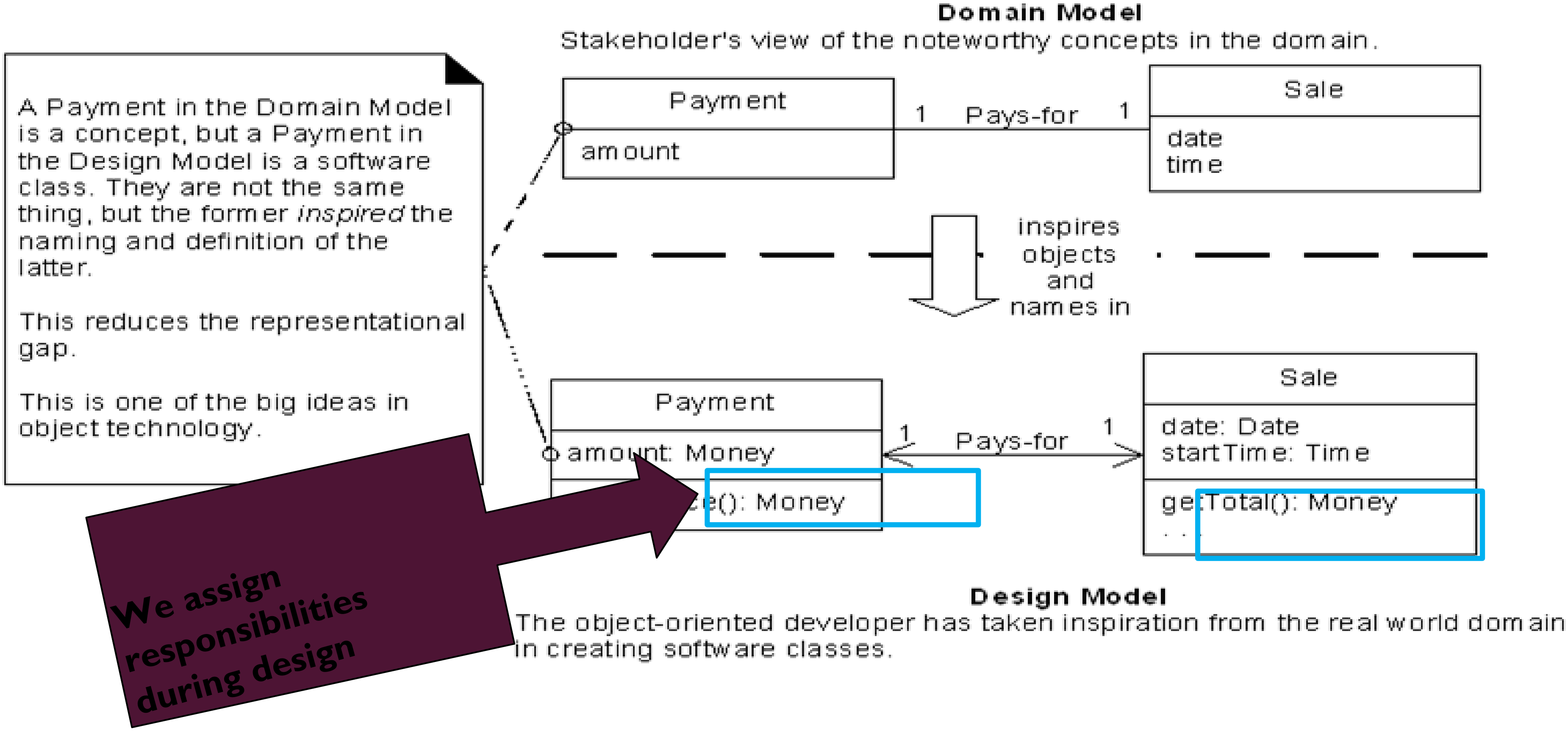
Item item;

void print()

What's the difference?

Why Create Domain model?

Answer : Get inspiration to create software classes



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.

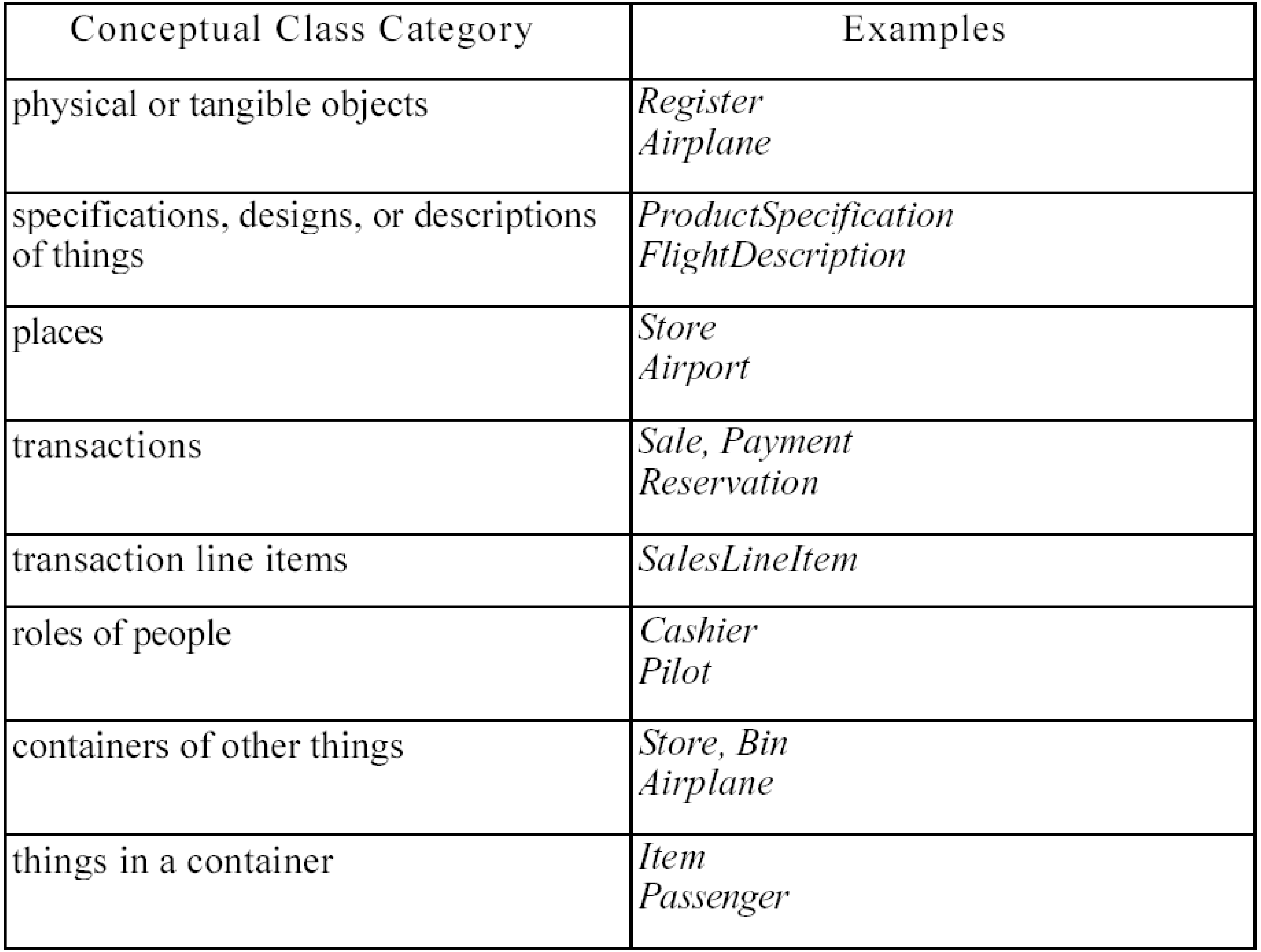
This reduces the representational gap.

This is one of the big ideas in object technology.

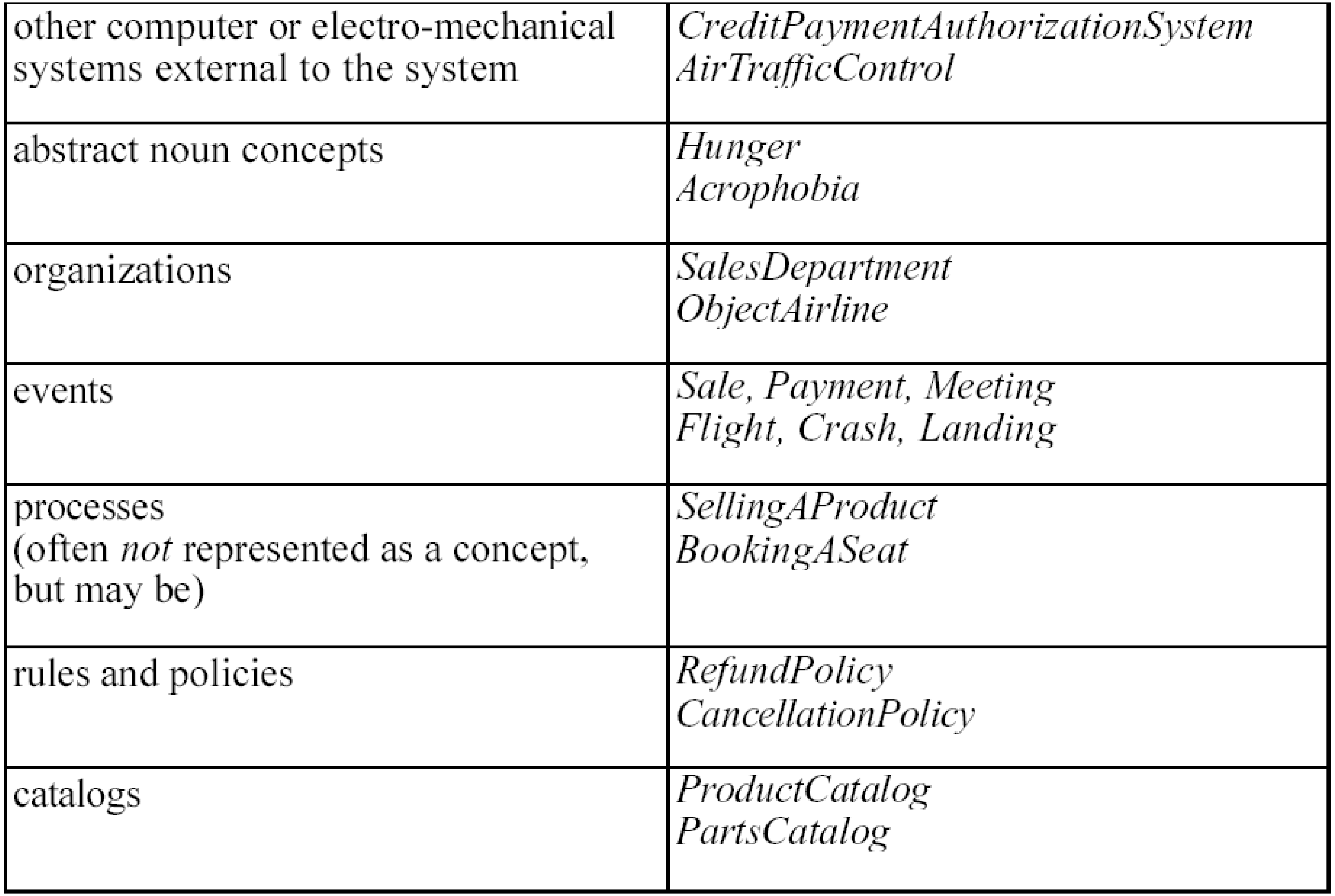
We assign responsibilities during design

Design Model

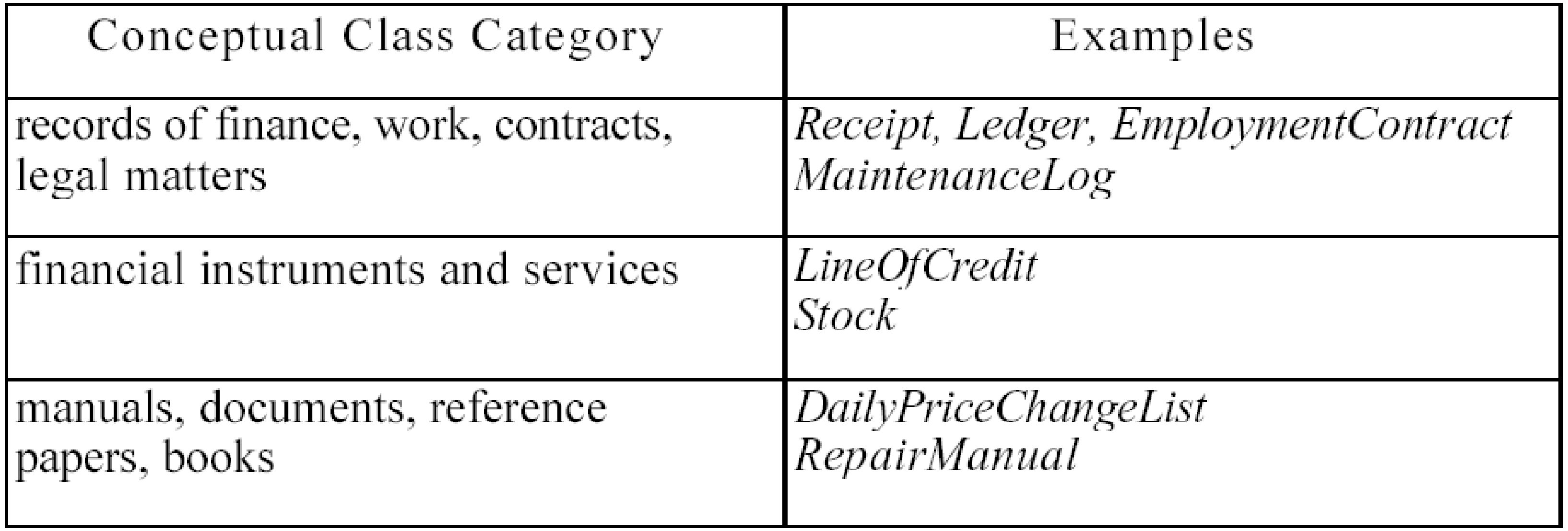
The object-oriented developer has taken inspiration from the real world domain in creating software classes.



Conceptual Class Category	Examples
physical or tangible objects	*Register* *Airplane*
specifications, designs, or descriptions of things	*ProductSpecification* *FlightDescription*
places	*Store* *Airport*
transactions	*Sale, Payment* *Reservation*
transaction line items	*SalesLineItem*
roles of people	*Cashier* *Pilot*
containers of other things	*Store, Bin* *Airplane*
things in a container	*Item* *Passenger*



other computer or electro-mechanical systems external to the system	*CreditPaymentAuthorizationSystem* *AirTrafficControl*
abstract noun concepts	*Hunger* *Acrophobia*
organizations	*SalesDepartment* *ObjectAirline*
events	*Sale, Payment, Meeting* *Flight, Crash, Landing*
processes (often *not* represented as a concept, but may be)	*SellingAProduct* *BookingASeat*
rules and policies	*RefundPolicy* *CancellationPolicy*
catalogs	*ProductCatalog* *PartsCatalog*



Conceptual Class Category	Examples
records of finance, work, contracts, legal matters	*Receipt, Ledger, EmploymentContract* *MaintenanceLog*
financial instruments and services	*LineOfCredit* *Stock*
manuals, documents, reference papers, books	*DailyPriceChangeList* *RepairManual*

CHARACTERISTICS OF DOMAIN MODELING

- Visual representation of conceptual classes.
- Associations and relationships between concepts (e.g Payment PAYS-FOR Sales).
- Attributes for information content (e.g. Sale records DATE and TIME).
- Does not include operations / functions.
- Does not describe software classes.
- Does not describe software responsibilities.

STEPSTOCREATEA DOMAIN MODEL

1. Create User Stories

2. Identify candidate conceptual classes

3. Draw them in a UML domain model

4.Add associations necessary to record the relationships that must be retained

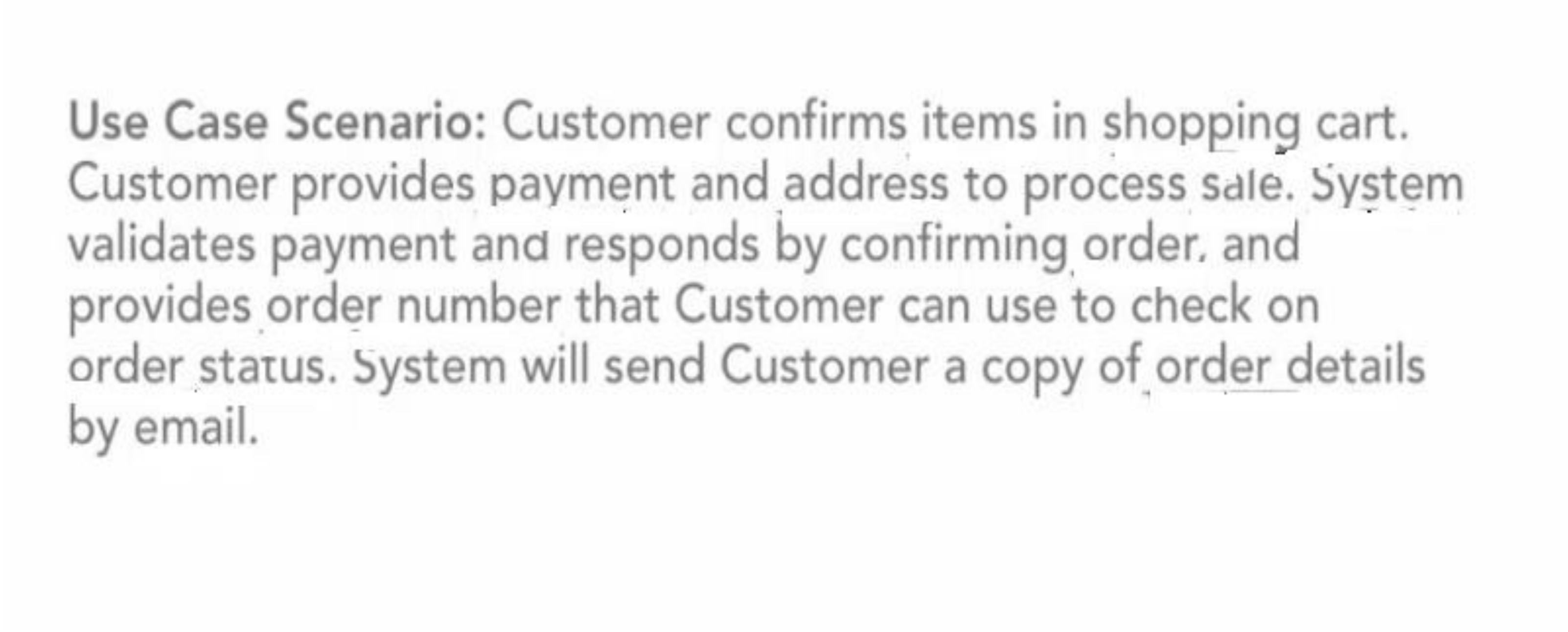
5.Add attributes necessary for information to be preserved

6. Use existing names for things, the vocabulary of the domain

USER STORIES

- A User Story is one or more sentences in the everyday or business language of the end user or user of a system that captures what a user does or needs to do as part of his or her job function.
- It captures the 'who', 'what' and 'why' of a requirement in a simple, concise way, often limited in detail by what can be hand-written on a small paper note card.
- User stories are the descriptions of the domain that could be :
  - The problem definition.
  - The Scope.
  - The vision.

USER STORIES



Use Case Scenario: Customer confirms items in shopping cart. Customer provides payment and address to process sale. System validates payment and responds by confirming order, and provides order number that Customer can use to check on order status. System will send Customer a copy of order details by email.

IDENTIFY OBJECTS: NOUN PHRASE IDENTIFICATION

- *Identify Nouns and Noun Phrases in textual descriptions of the domain.*
- *However, Words may be ambiguous (such as : System)*
- *Different phrases may represent the same concepts.*
- *Noun phrases may also be attributes or parameters rather than classes:*
  - *If it stores state information or it has multiple behaviors, then it's a class*
  - *If it's just a number or a string, then it's probably an attribute*

NOUN PHRASE IDENTIFICATION

- Consider the following problem description, analyzed for Subjects,Verbs, Objects:

The ATM verifies whether the customer's card number and PIN are correct.

If it is, then the customer can check the account balance, deposit cash, and withdraw cash.

Checking the balance simply displays the account balance.

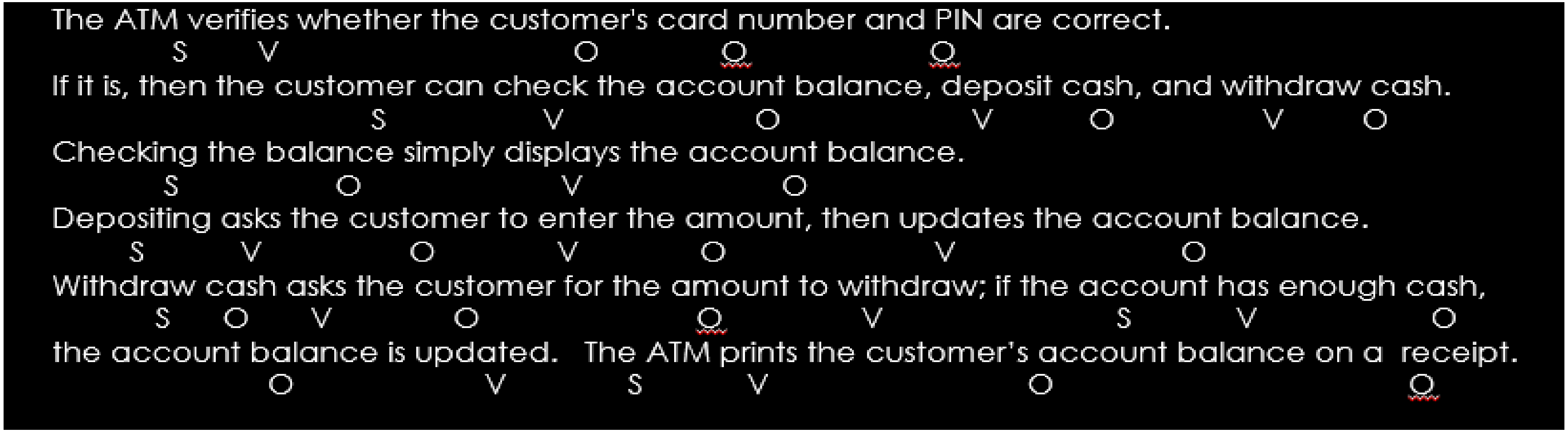
Depositing asks the customer to enter the amount, then updates the account balance.

Withdraw cash asks the customer for the amount to withdraw; if the account has enough cash,

the account balance is updated. The ATM prints the customer's account balance on a receipt.

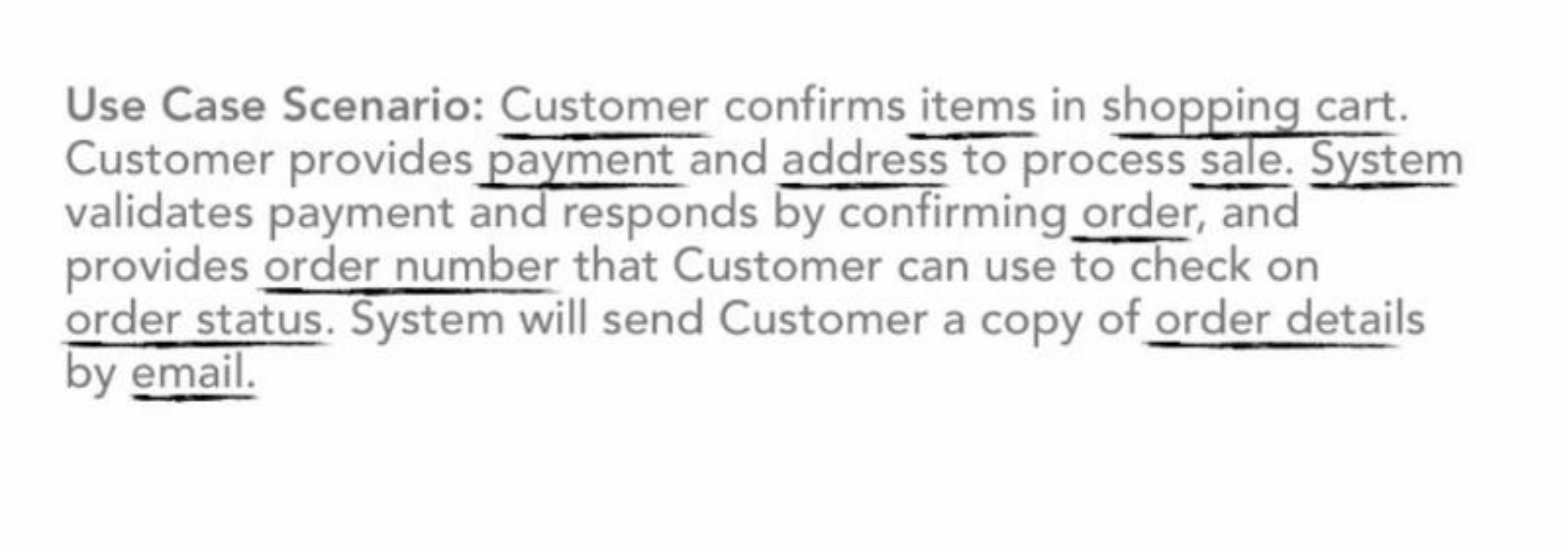
NOUN PHRASE IDENTIFICATION

- Consider the following problem description, analyzed for Subjects,Verbs, Objects:



```
The ATM verifies whether the customer's card number and PIN are correct.
        S     V                   O         O            O
If it is, then the customer can check the account balance, deposit cash, and withdraw cash.
                     S          V             O             V       O          V      O
Checking the balance simply displays the account balance.
       S          O             V             O
Depositing asks the customer to enter the amount, then updates the account balance.
     S      V          O        V         O             V               O
Withdraw cash asks the customer for the amount to withdraw; if the account has enough cash,
       S    O    V         O              O          V               S       V           O
the account balance is updated.   The ATM prints the customer's account balance on a  receipt.
              O             V       S       V                 O                       O
```

IDENTIFY OBJECTS

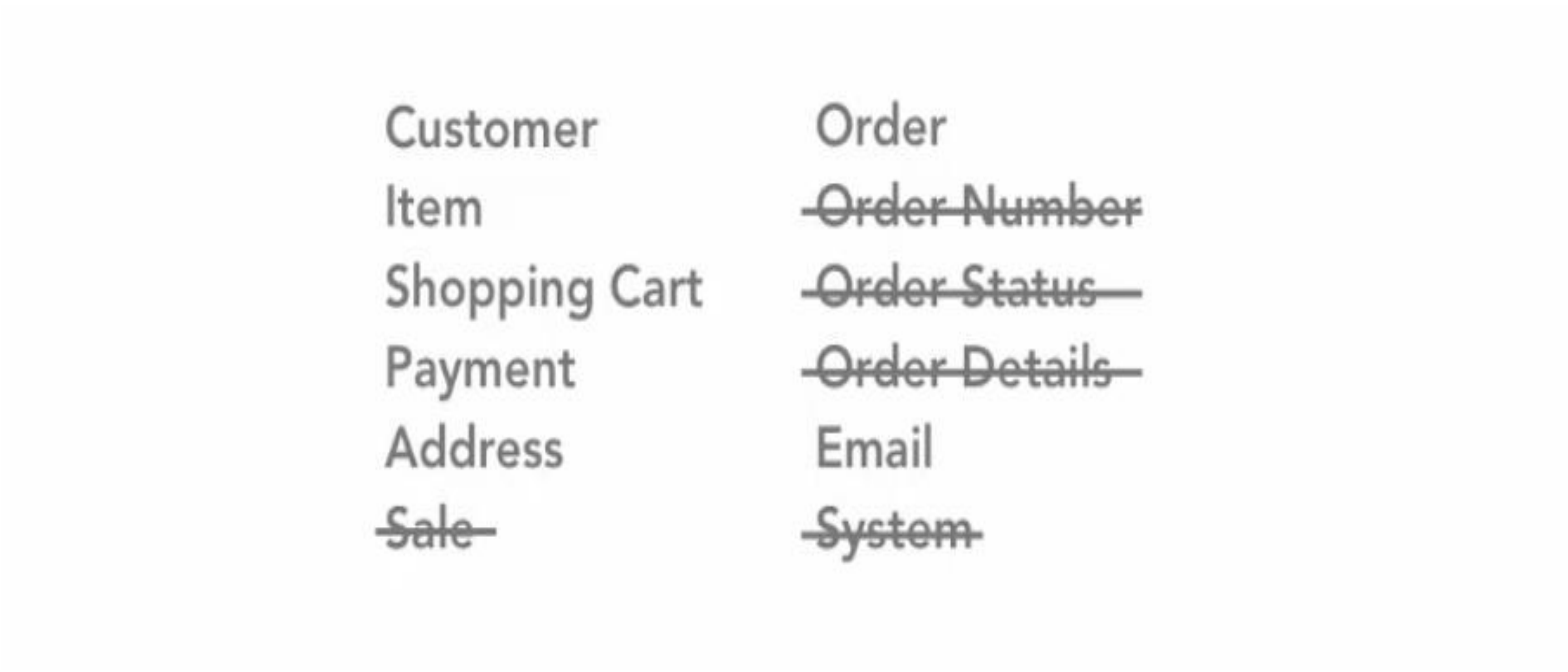


Use Case Scenario: Customer confirms items in shopping cart. Customer provides payment and address to process sale. System validates payment and responds by confirming order, and provides order number that Customer can use to check on order status. System will send Customer a copy of order details by email.

IDENTIFICATION OF CONCEPTUAL CLASSES

- *Identify candidate conceptual classes*
- *Go through them and :*
  - ❑ ***Exclude irrelevant features and duplications***
  - ❑ ***Do not add things that are outside the scope*** *(outside the application area of investigation)*

REFINE OBJECTS



Customer

Item

Shopping Cart

Payment

Address

~~Sale~~

Order

~~Order Number~~

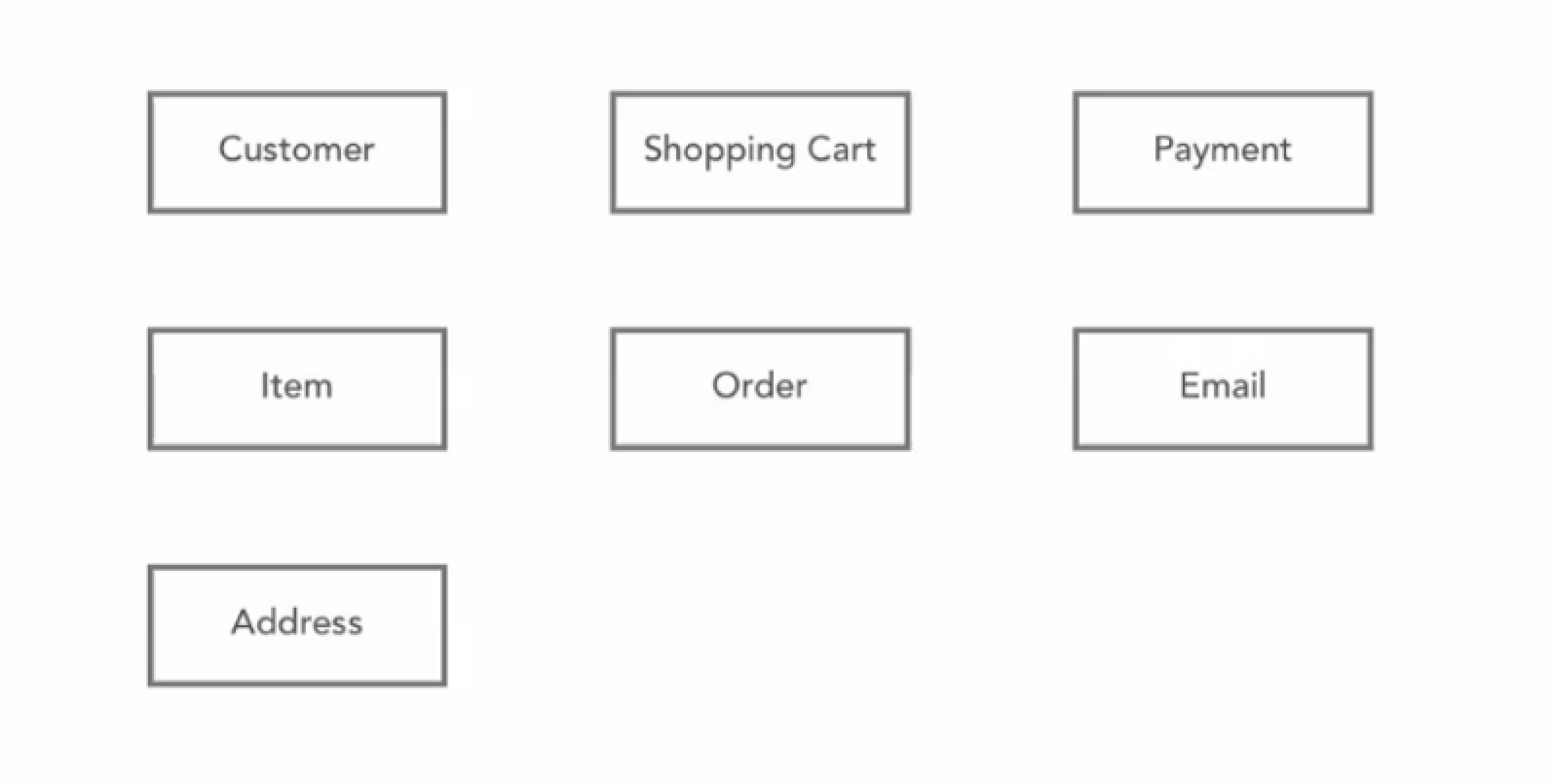
~~Order Status~~

~~Order Details~~

Email

~~System~~

DRAWING OBJECTS



Customer

Shopping Cart

Payment

Item

Order

Email

Address

ATTRIBUTES

- *A logical data value of an object.*
- *Imply a need to remember information.*
  - Sale needs a **dateTime** attributte
  - Store needs a **name** and **address**
  - Cashier needs an **ID**

A COMMON MISTAKE WHEN MODELING THE DOMAIN- CLASSES OR ATTRIBUTES?

Rule

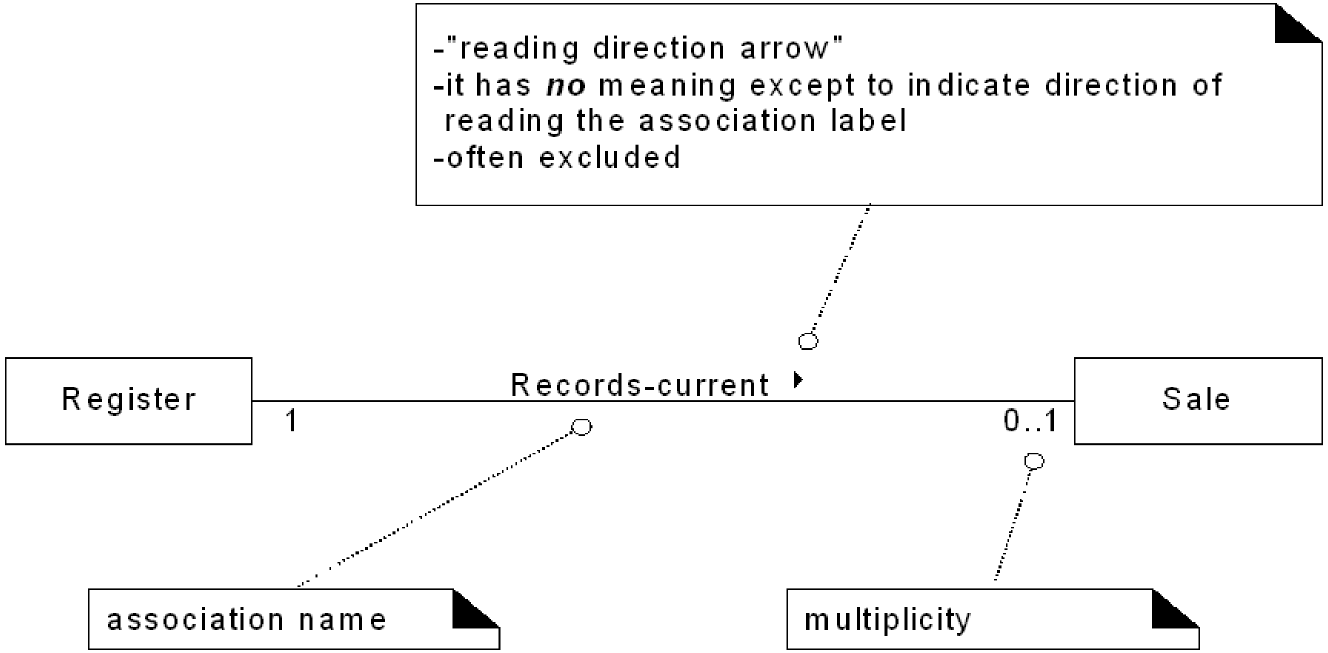
- *If we do not think of a thing as a number or text in the real world, then it is probably a conceptual class.*
- *If it takes up space, then it is likely a conceptual class.*

Examples:

- *Is a store an attribute of a Sale ?*
- *Is a destination an attribute of a flight ?*

IDENTIFYING OBJECT RELATIONSHIPS-ASSOCIATIONS

- Relationship between classes (more precisely, between instances of those classes)indicating some meaningful and interesting connection


-"reading direction arrow"
-it has *no* meaning except to indicate direction of reading the association label
-often excluded
Register
Records-current ▸
Sale
1
0..1
association name
multiplicity

COMMON ASSOCIATION LIST

- A is a physical part of B .
  - Wing - Airplane
- A is a logical part of B
  - SalesLineItem - Sale
- A physical contained in B
  - Register-Sale
- A is a logical contained in B
  - ItemDescription - Catalog
- A is a description of B .
  - ItemDescription - Item
- A is a member of B
  - Cashier – Store

COMMON ASSOCIATION LIST

- A uses or manage B
  - Cashier-Register
- A is an event related to B
  - Sale- Customer
- A is recorded in B
  - Salel-Register
- A is an organization subunit of B.
  - Departement - Store

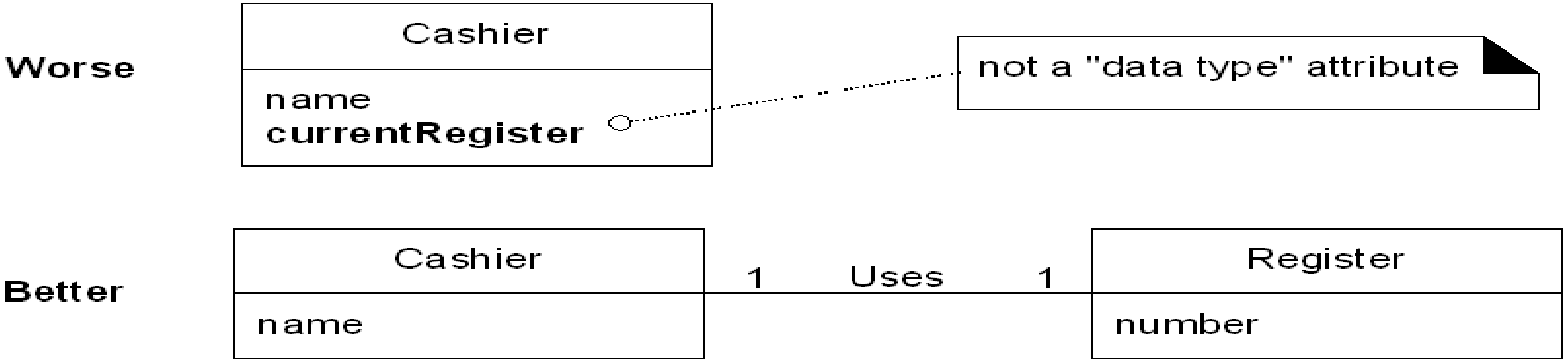
COMMON ASSOCIATION LIST

- A communicate with B
  - Customer - Cashier
- A is related to a transaction B
  - Customer - Payment
- A is a transaction related to another transaction B .
  - Payment - Sale
- A is owned by B
  - Register - Store

HIGH PRIORITY ASSOCIATION

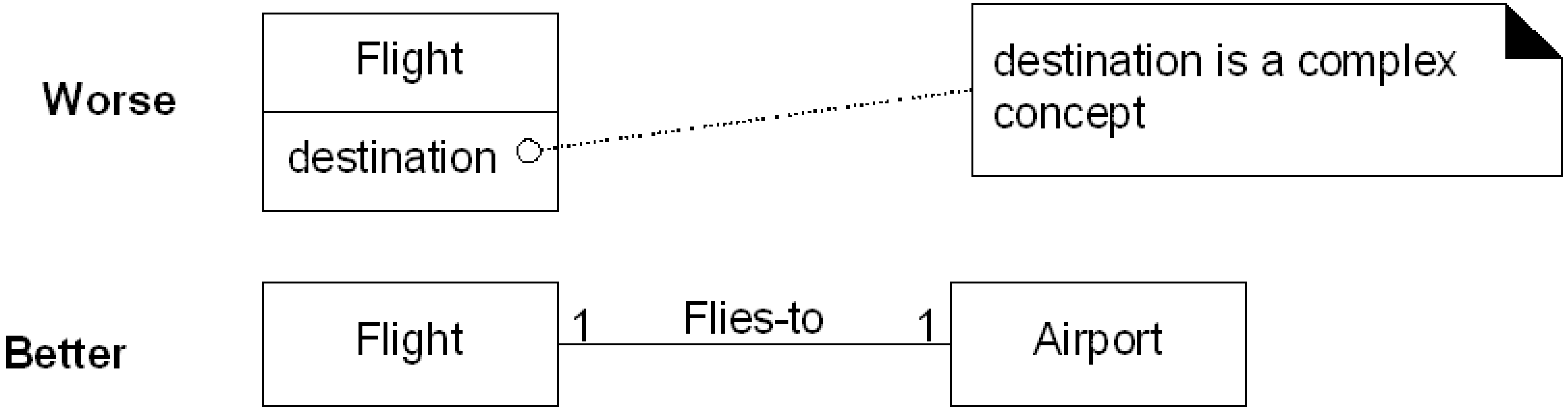
  - A is a physical or logical part of B
  - A is physically or logically contained in/on B
  - A is recorded in B
- To avoid:
  - ***Avoid showing redundant or derivable associations***
  - ***Do not overwhelm the domain model with associations not strongly required***

ASSOCIATION OR ATTRIBUTE ?



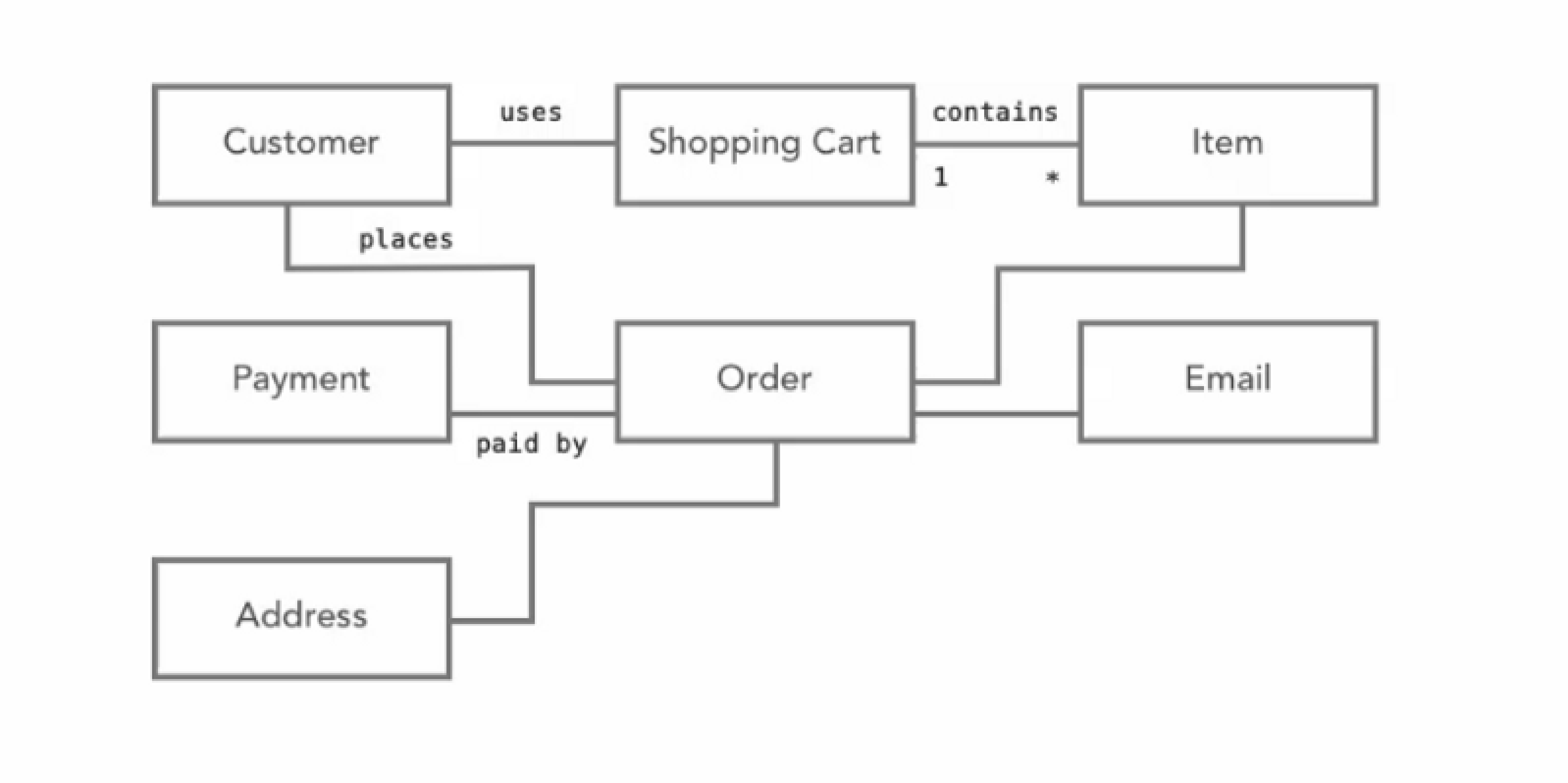
- ❑ *Most attribute type should be “**primitive**” data type, such as: numbers , string or boolean (true or false)*
- ❑ *Attribute should not be a complex domain concept(Sale ,Airport)*
- ❑ *CurrentRegister is of type “Register”, so expressed with an association*

Association or attribute ?

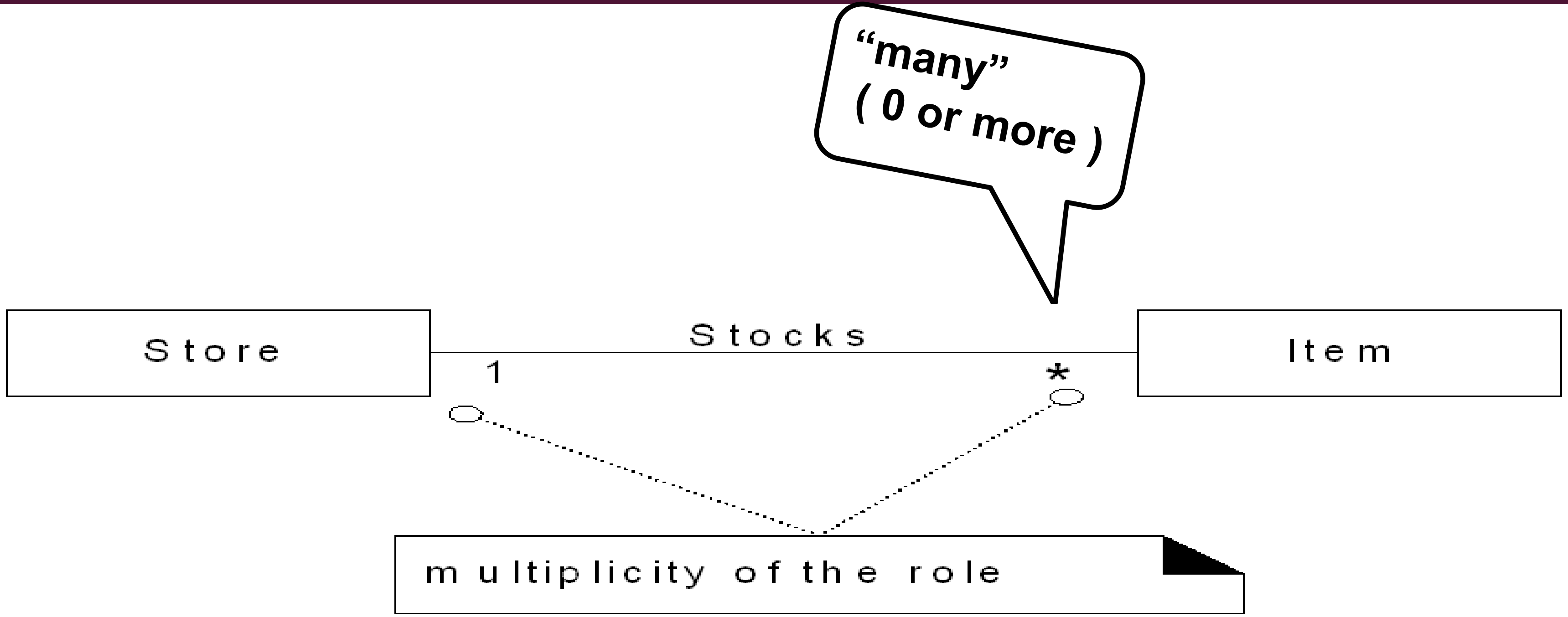


A destination airport is ***not a string***, it is a complex thing that occupies many square kilometers of space. So "Airport" should be related to "Flight" via an association , not with attribute

IDENTIFYING OBJECT RELATIONSHIPS



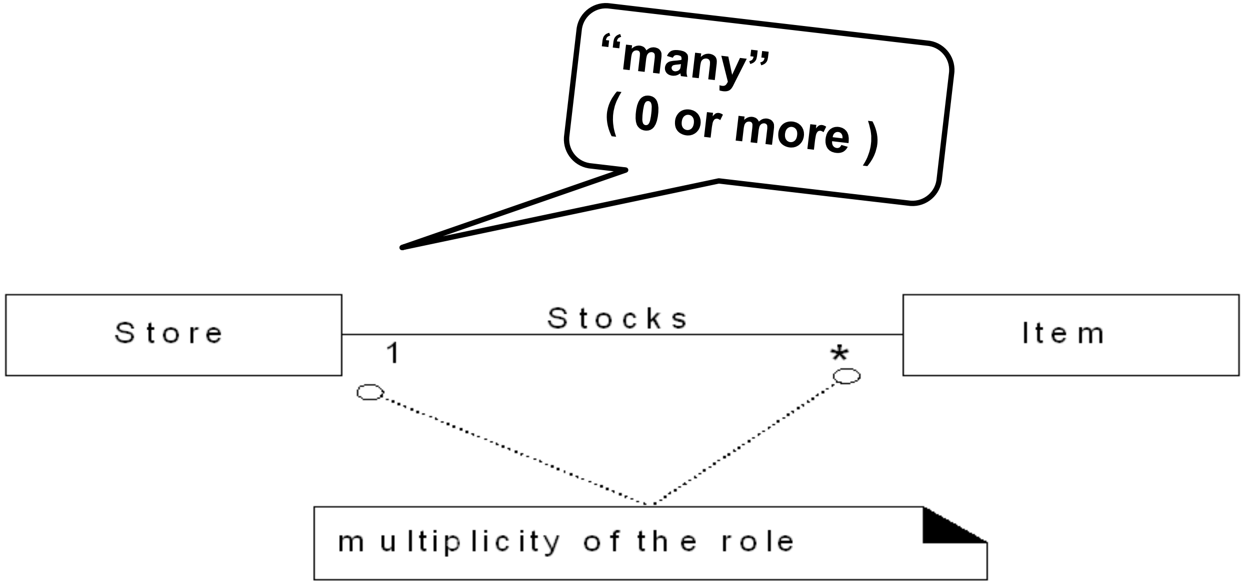
Multiplicity



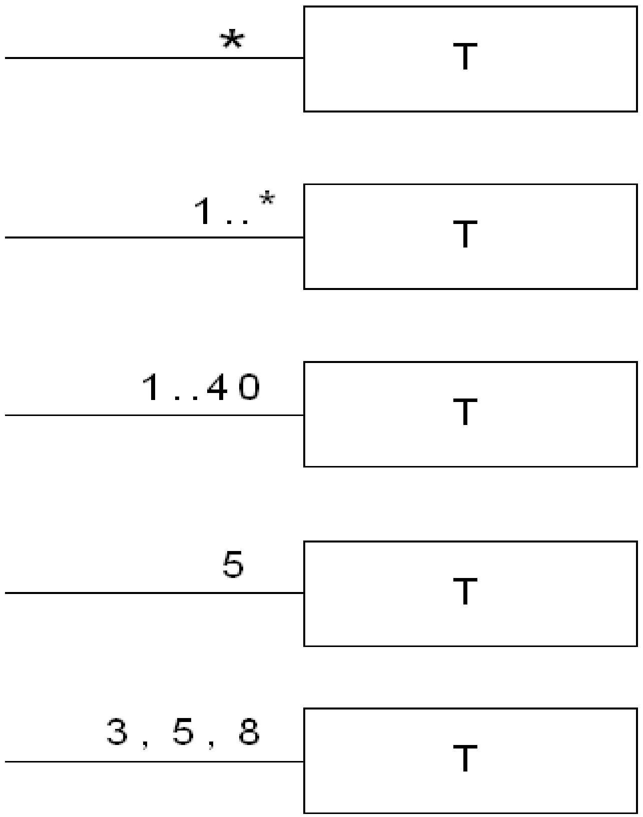
- ***Multiplicity*** *indicates how many instances can be validly associated with another instance, at a particular moment, rather than over a span of time.*

How to determine multiplicity ?

- *Ask these 2 questions :*
    - *store may stock how many item ?*
    - *item may be stocked in how many stores ?*



Multiplicity



*	T	zero or more; "many"
1..*	T	one or more
1..40	T	one to 40
5	T	exactly 5
3, 5, 8	T	exactly 3, 5, or 8

HOW TO CREATE A DOMAIN MODEL

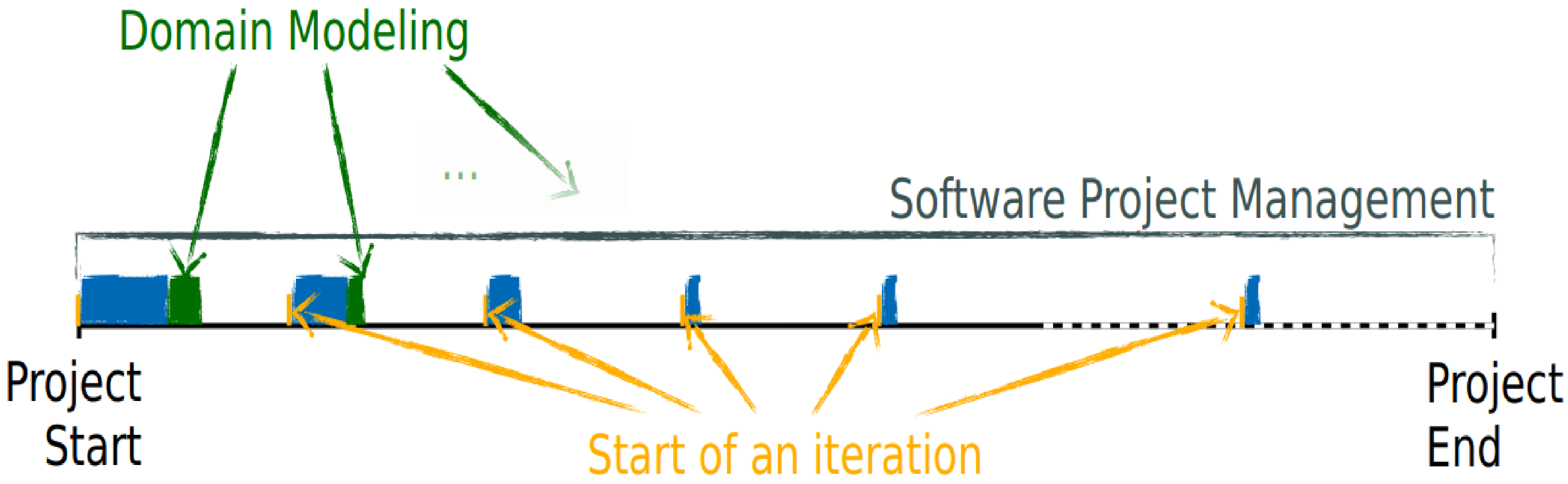
- *Identify candidate conceptual classes*
- *Go through them*
  - ***Exclude irrelevant features and duplications***
  - ***Do not add things that are outside the scope***
- *Draw them as classes in a UML class diagram*
- *Add associations necessary to record the relationship that must be retained*
- *Add attributes necessary for information to be preserved*

BUT REMEMBER

- *There is no such thing as a single correct domain model. All models* ***are approximations of the domain*** *we are attempting to understand.*

- *We* ***incrementally evolve a domain model*** *over several iterations on attempts to capture all possible conceptual classes and relationships.*

THE GOAL OF THIS LECTURE IS TO ENABLE YOU TO SYSTEMATICALLY CARRY OUT SMALL(ER) SOFTWARE PROJECTS THAT PRODUCE WELL-DESIGNED SOFTWARE.



That is all