Software Design and Analysis
CS-3004

Software Process, Process Models and RUP, Domain Model

Lecture # 04, 05, 06
13, 15, 16 Sep

Rubab Jaffar
rubab.jaffar@nu.edu.pk

Today's Outline

- Software Process
- Software Development approaches
- Process models
- Unified Process

Software Processes & Process Model

- **Software Process** is coherent sets of activities for specifying, designing, implementing and testing software systems.
- ***A (software/system) process model*** *is a description of the sequence of activities carried out in an SE project, and the relative order of these activities*

Plan-driven and Agile Processes(Development Approach)

- Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.
- In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.
- In practice, most practical processes include elements of both plan-driven and agile approaches.
- There are no right or wrong software processes.

What is a Process Model ?

It is a description of

i) <u>what tasks need to be performed</u> in

ii) <u>what sequence</u> under

iii) <u>what conditions</u> by

iv) <u>whom</u> to

achieve the "desired results."

Extending the "Simple" Process

As projects got *larger* and more complex, there is a need

- ❖ clarify and stabilize the requirements
- ❖ test more functionalities
- ❖ design more carefully
- ❖ use more existing software & tools
  - ❖ Database
  - ❖ Network
  - ❖ Code control
- ❖ more people to be involved

***Resulting in* more tasks *and* more people**

Effectiveness of using Correct Process Model

By changing the process model, we can improve:

- *Development speed (time to market)*
- *Product quality*
- *Project visibility*
- *Administrative overhead*
- *Risk exposure*
- *Customer relations, etc.*

Normally, a process model covers the entire lifetime of a product.

From birth of a commercial idea to final installation of last release.

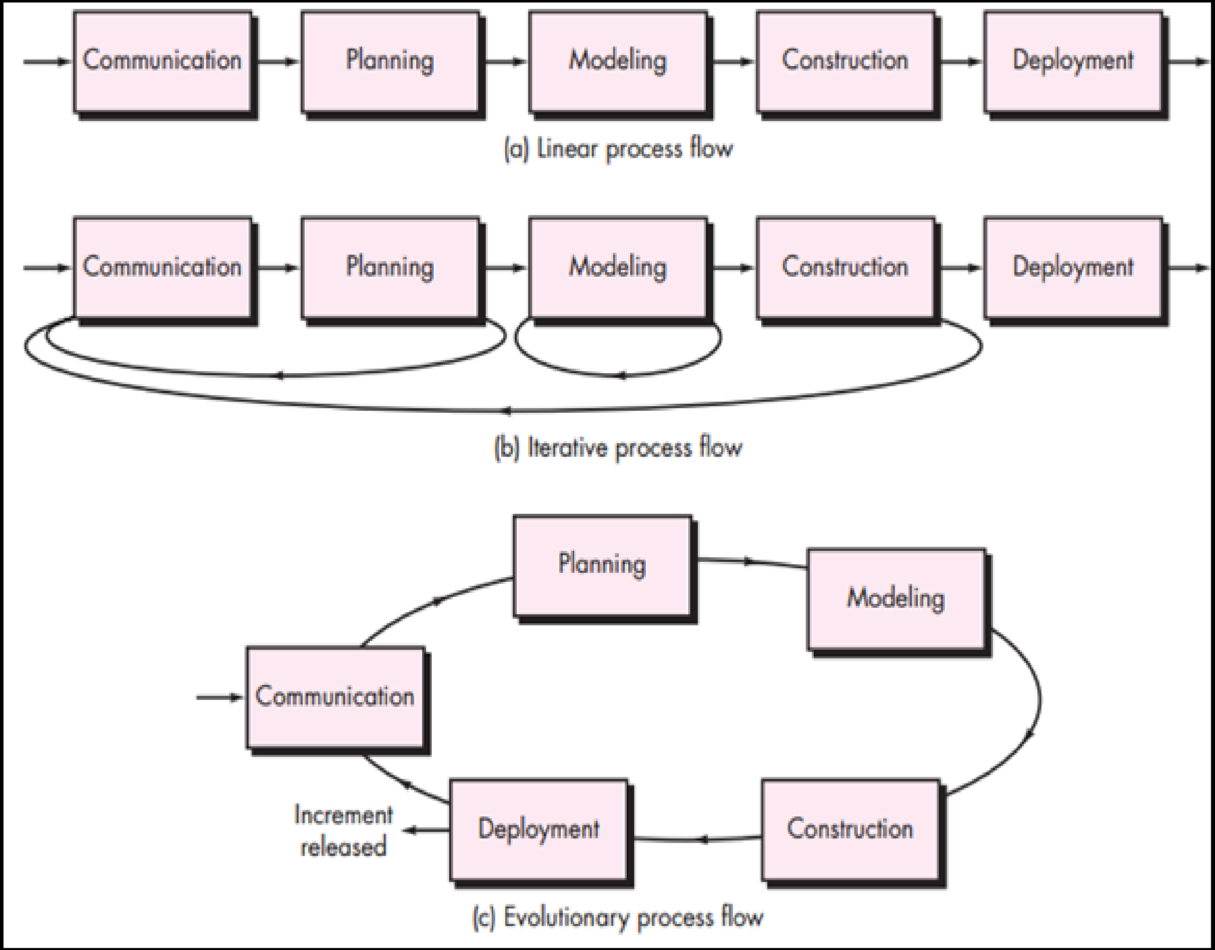
Common Activities of Process Model

Many different software processes but all involve:

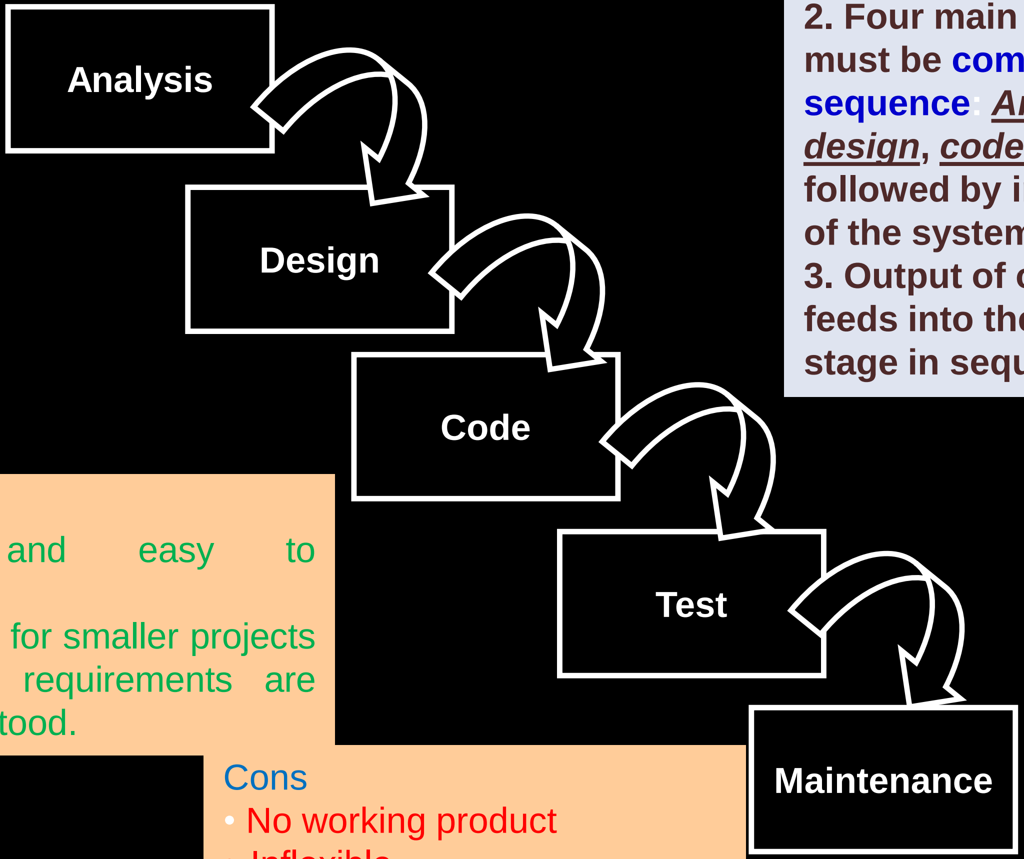
- Specification – defining what the system should do;
- Design and implementation – defining the organization of the system and implementing the system;
- Validation – checking that it does what the customer wants;
- Evolution – changing the system in response to changing customer needs.

Generic Process Flows

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



Waterfall Model

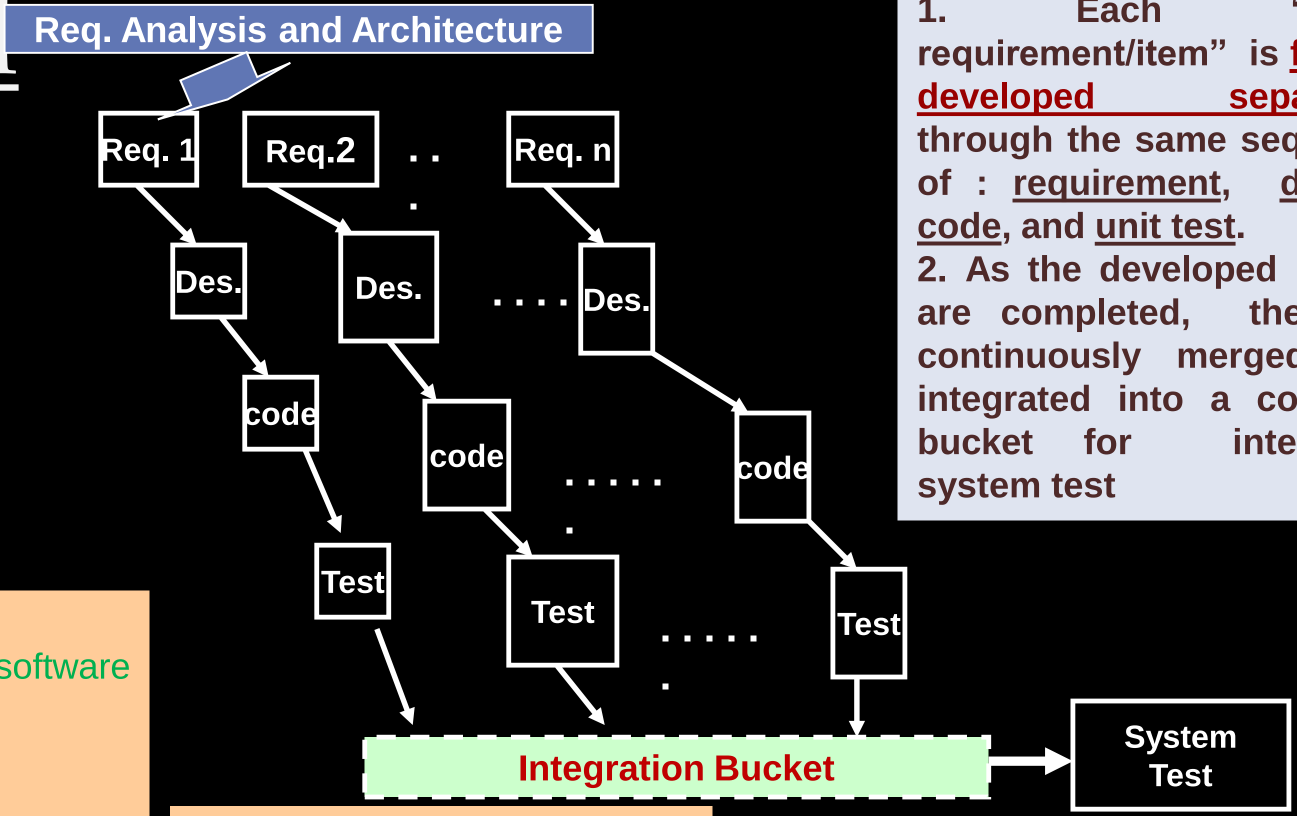


1. Requirements must be specified.
2. Four main tasks must be **completed in sequence**: ***Analysis***, ***design***, ***code***, and ***test,*** followed by integration of the system.
3. Output of one stage feeds into the next stage in sequence

Pros
- Simple and easy to understand.
- Works well for smaller projects where the requirements are well-understood.

Cons
- No working product
- Inflexible

Incremental Model

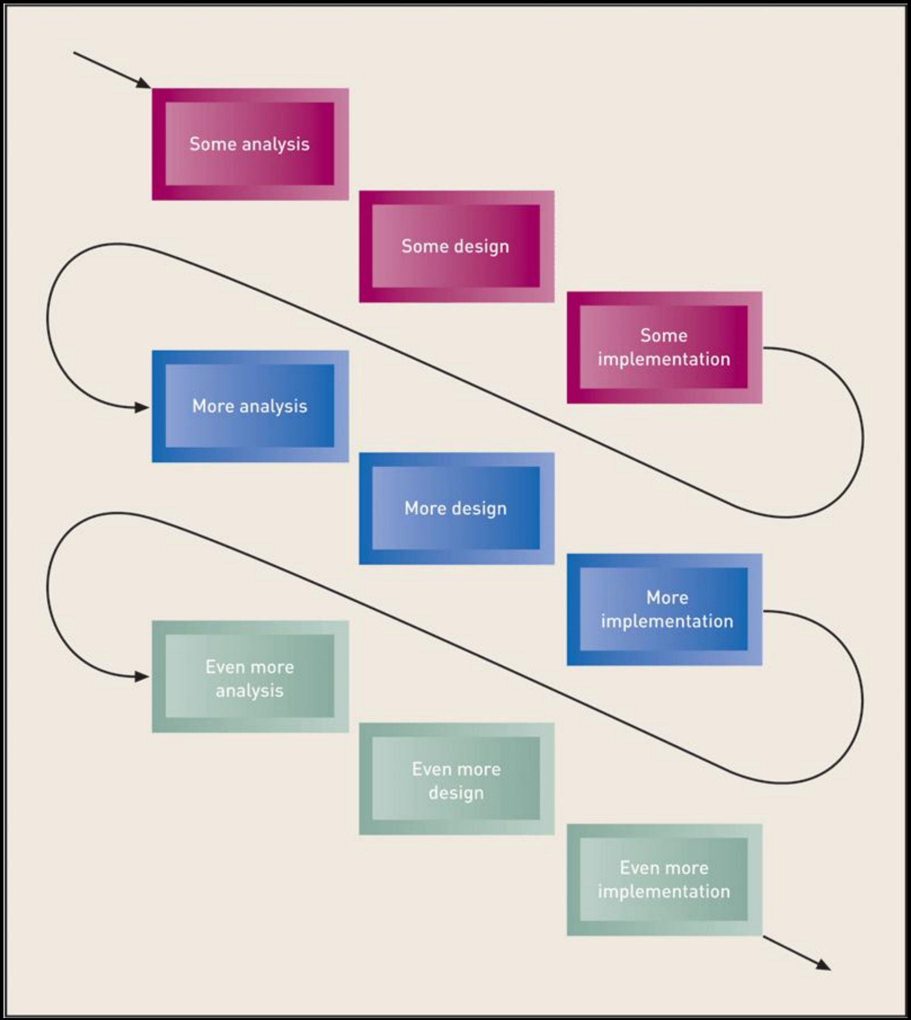


1. Each "major requirement/item" is **further developed separately** through the same sequence of : requirement, design, code, and unit test.
2. As the developed pieces are completed, they are continuously merged and integrated into a common bucket for integrated system test

Pros
- Generates working 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 of System Development Activities



Problems

- Lack of process visibility
- Systems are often poorly structured

Applicability

- For small or medium-size interactive systems
- For parts of large systems
- For short-lifetime systems

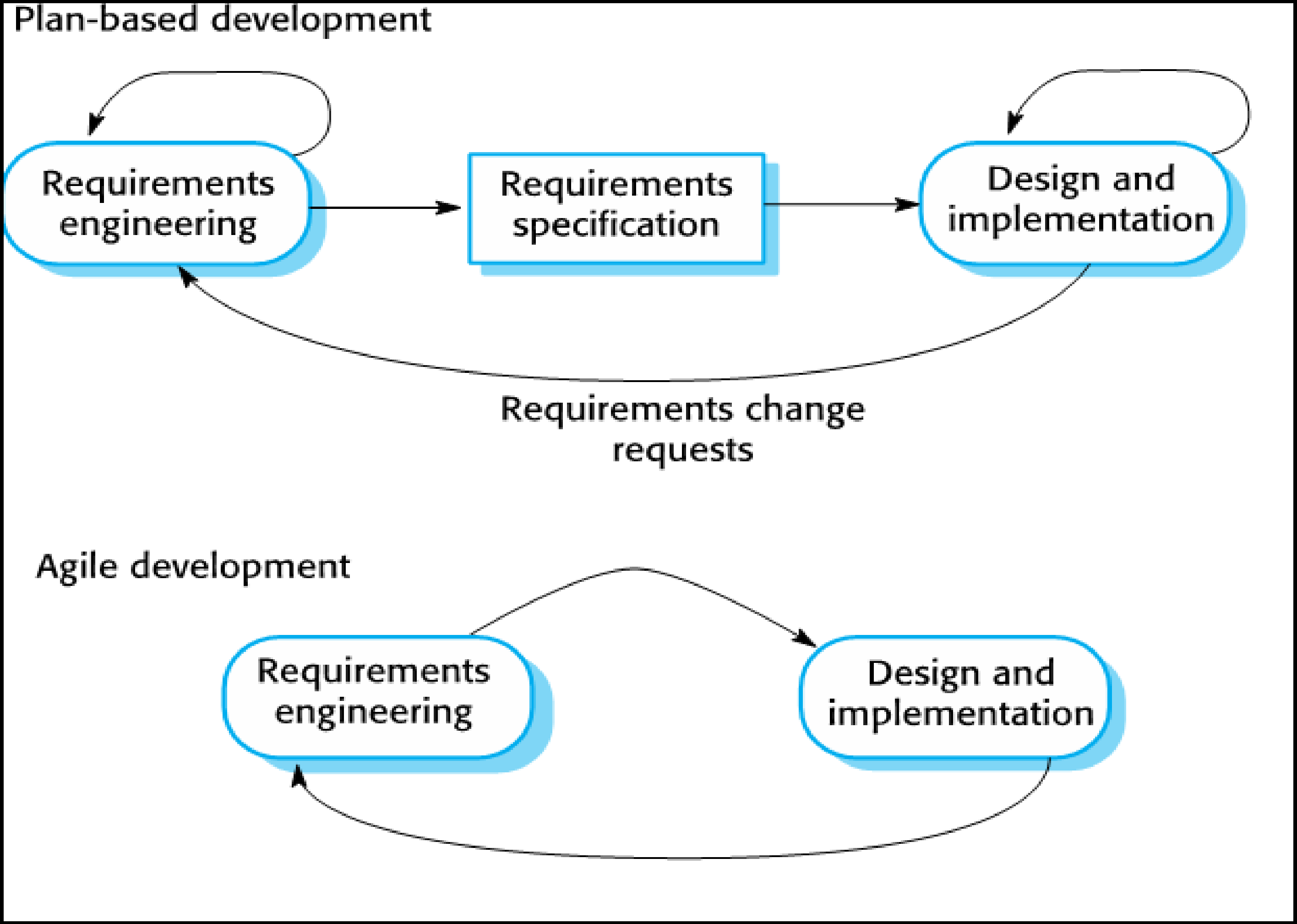
Rapid Software Development

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

Agile models

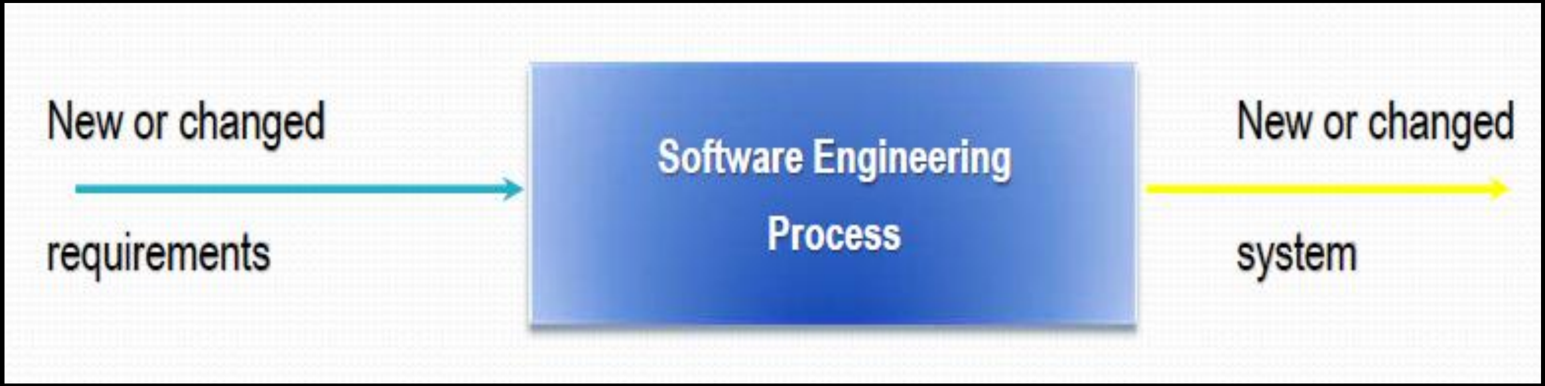
- XP
- Scrum

Today's Outline

- Unified Process (UP)
- UP characteristics
- Unified Process 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’

Unified Process Model

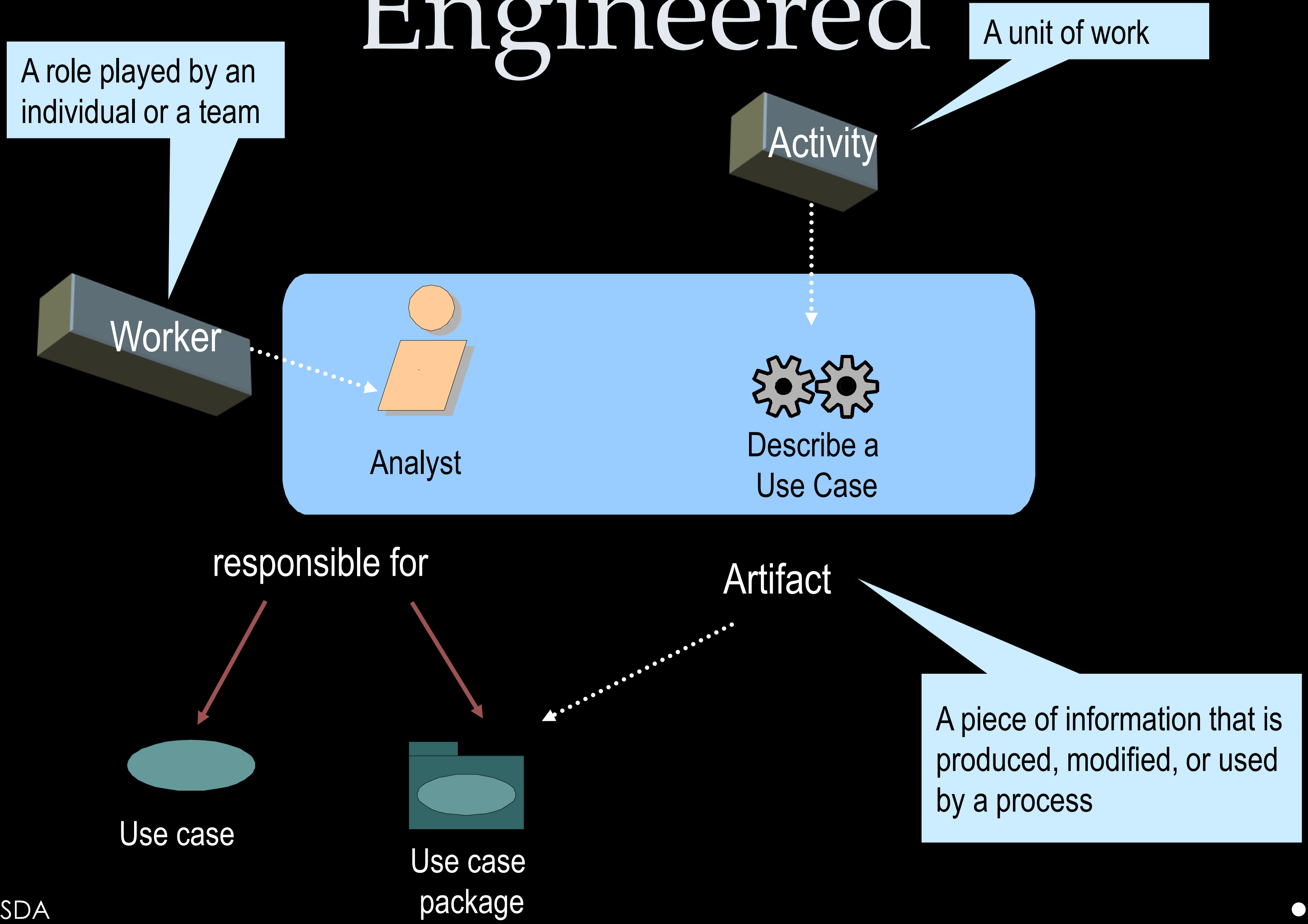
- A process model that was created 1997 to give a framework for Object-oriented Software Engineering
- Iterative, incremental model to adapt to specific project needs
- Risk driven development
- Combining spiral and evolutionary models
- **2D process : phases and workflows**
- **Utilizes Miller's Law**

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

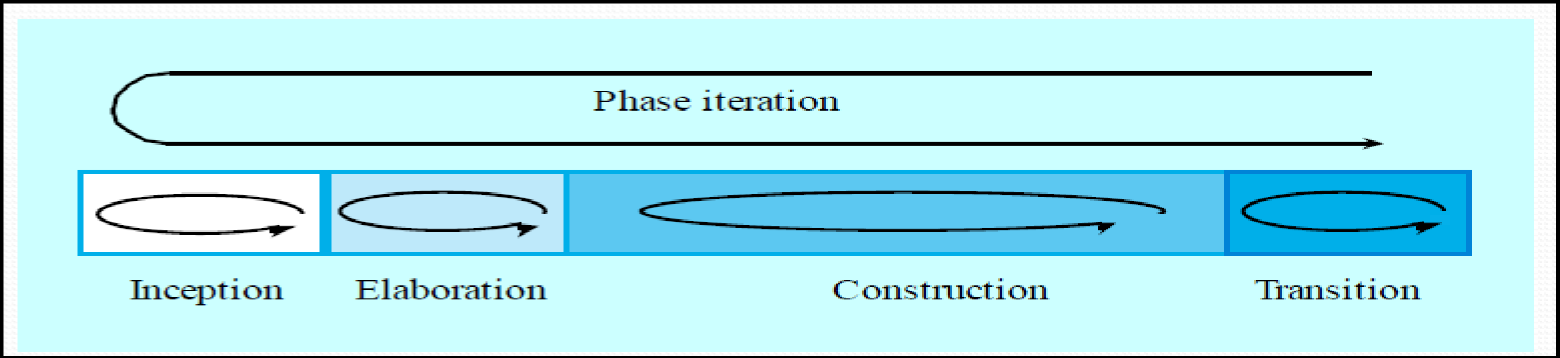

A role played by an individual or a team
A unit of work
Activity
Worker
Analyst
Describe a Use Case
responsible for
Artifact
A piece of information that is produced, modified, or used by a process
Use case
Use case package

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 is 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.

Basic Characteristics of Unified Process



- Object-oriented
- Use-case driven
- Architecture centric
- Risk focused
- Iteration and incrementation

UP Characteristics

- **Object-oriented**
  - o Utilizes object oriented technologies.
  - o Classes are extracted during OOA and designed during OOD.
- **Use-case driven**
  - o Utilizes use case model to describe complete functionality of the system
  - o This practice reinforces the fundamental notion that a system must conform to the needs of the users, instead of your users conforming to the system.

UP Characteristics

Architecture centric

- Focus core architecture in the early iterations
- In earliest iterations, get high valued requirements
- View of the whole design with the important characteristics made more visible
- Expressed with class diagram

Risk-focused:

- The Unified Process requires the project team to focus on addressing the most critical risks early in the project life cycle. The deliverables of each iteration, especially in the Elaboration phase, must be selected in order to ensure that the greatest risks are addressed first.

UP Characteristics

Iterative and incremental

- Way to divide the work

- Iterations are steps in the process, and increments are growth of the product

- The basic software development process is iterative
  - Each successive version is intended to be closer to its target than its predecessor

Unified Process Model

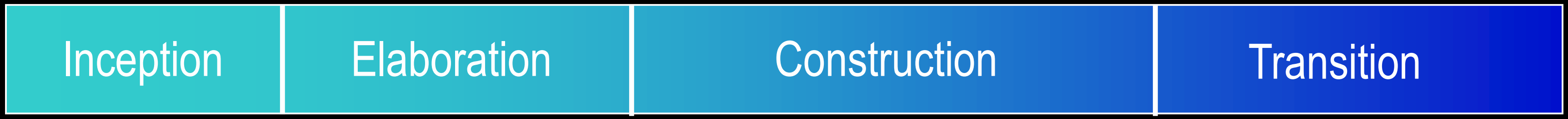
- Integrating two seemingly contradicting insights:
  - o Definitive activities and deliverables as in the Waterfall Model.
  - o Iterative and incremental processes.

- A project is split into several *phases:*
  - o Each phase is split into several *iterations.(2 to 6 weeks)*
  - o Each iteration consists of the traditional process activities, known as *workflow*.

- Each workflow places *different* emphasis on the activities depending on the current iteration.

- Risk analysis is performed in each iteration.

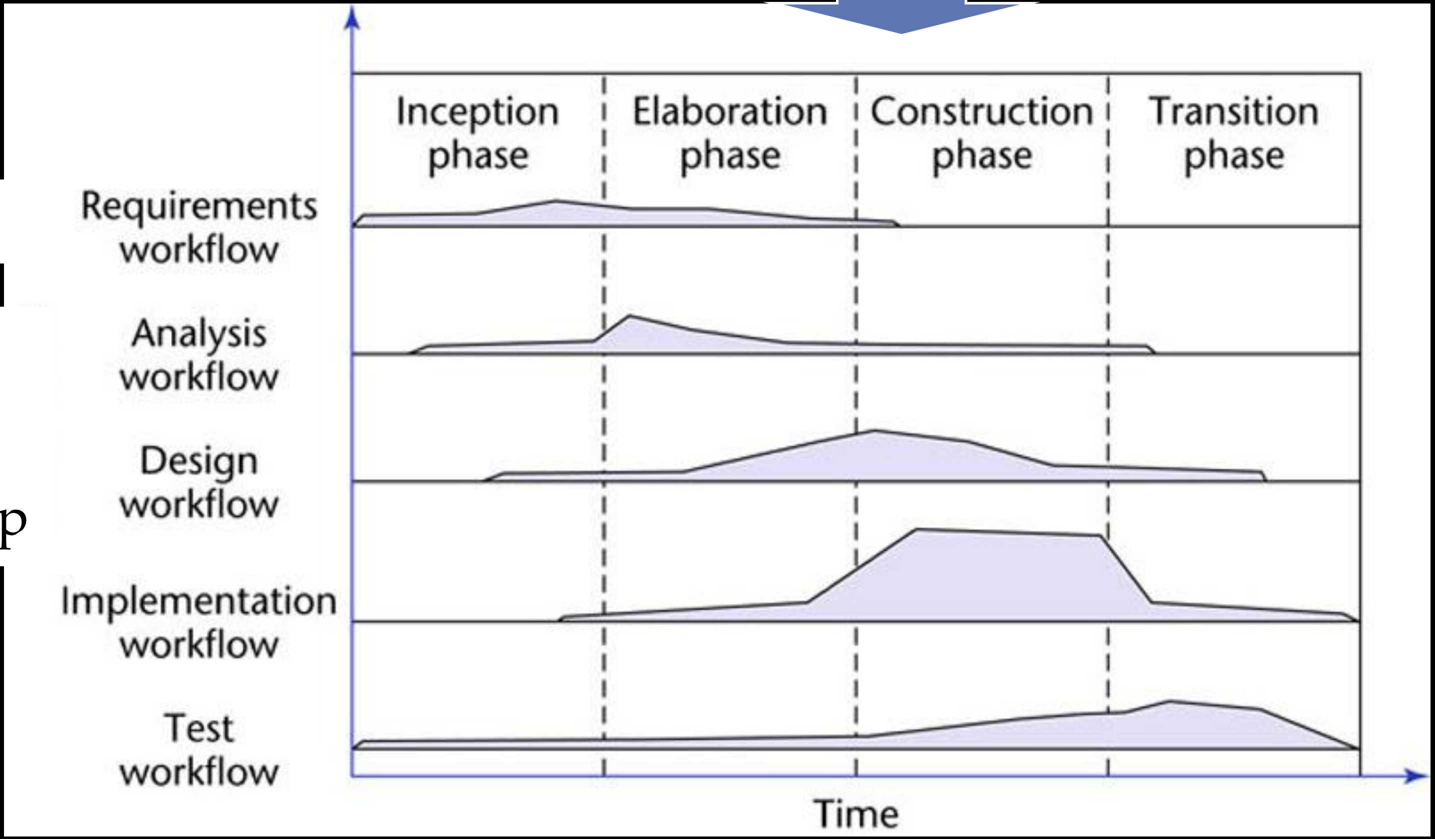
Unified Process Phases



- Inception
    - Establish the business case for the system, define risks, obtain 10% of the requirements, estimate next phase effort.
- Elaboration
    - Develop an understanding of the problem domain and the system architecture, risk significant portions may be coded/tested, 80% major requirements identified.
- Construction
    - System design, programming and testing. Building the remaining system in short iterations.
- Transition
    - Deploy the system in its operating environment. Deliver releases for feedback and deployment.

The Phases/Workflows of the Unified Process

- Phase is Business context of a step



Workflow is Technical context of a step

Unified Process – Inception Activities

Inception

Project Initiation

- Start with an idea
- Specify the end-product vision
- Analyze the project to assess scope
- Work the business case for the project including overall costs and schedule, and known risks
- Identify Stakeholders
- Obtain funding

Unified Process – Elaboration Activities

Elaboration

- The goal of the elaboration phase is to baseline the most significant requirements.

- **Elaboration Essential Activities**

- Analyze the problem domain.
- Refine the Vision to understand the most critical Use Cases
- Create and baseline iteration plans for construction phase.
- Refine component architecture and decide build/buy/reuse
- Develop a project plan and schedule.
- Mitigate high-risk elements identified in the previous phase.

Unified Process – Construction Activities

Construction

- The goal of the construction phase is to clarify the remaining requirements and complete the development of the first operational quality version of the software product.

- Construction Essential Activities
  - Complete component development and testing (Integrate all remaining components and features into the product
  - Assure resource management control and process optimization

Unified Process – Transition Objectives

Transition

- Deploy the system in its operating environment. Deliver releases for feedback and deployment.
- The focus of the Transition Phase is to ensure that software is available for its end users and meets their needs.
- Transition Objectives
    - Assess deployment baselines against acceptance criteria
- Primary deliverable
    - Final product onto a production platform
- Other deliverables
    - All the artifacts (final versions)
    - Completed manual

Six best "must" UP practices

1. Time-boxed iterations

2. *Strive for cohesive architecture* and reuse existing components: e.g.
    1. core architecture developed by small, co-located team
    2. then early team members divide into sub-project leaders

3. Continuously verify quality: *test early & often*, and realistically by integrating all software at each iteration

Six best "must" UP practices

4. Visual modeling: prior to programming, do at least some visual modeling to explore creative design ideas
5. Manage requirements: find, organize, and track requirements through skillful means

6. Manage change:
   - disciplined configuration management protocol, version control,
   - change request protocol
   - baselined releases at iteration ends

Unified Process Workflows (6 core workflows)

- Business Modeling Workflow: During this workflow, the business context (scope) of the project should be outlined.
- Requirements Workflow: Used to define all potential requirements of the project, throughout the software development life cycle.
- Analysis & Design Workflow: Once the requirements workflow is complete, the analysis and design phase takes those requirements and transforms them into a design that can be properly implemented.
- Implementation Workflow: This is where the majority of actual coding takes place, implementing and organizing all the code into layers that make up the whole of the system.
- Test Workflow: Testing of all kinds takes place within this workflow.
- Deployment Workflow: Finally, the deployment workflow constitutes the entire delivery and release process, ensuring that the software reaches the customer as expected.

Unified Process Workflows (3 core supporting workflows)

- Project Management Workflow: Where all activities dealing with project management take place, from pushing design objectives to managing risk to overcoming delivery constraints.

- Configuration & Change Management Workflow: Used to describe the various artifacts produced by the development team, ideally ensuring that there is minimal overlap or wasted efforts performing similar activities that result in identical or conflicting artifacts.

- Environment Workflow: Finally, this workflow handles the setup and management of all software development environments throughout the team, including the processes, as well as the tools, that are to be used throughout the software development life cycle.

Case Study: Applying the Rational Unified Process: A Web Service Sample

- http://www.differnet.com/crose/CR_Files/CRose-RUPSample/RUPSample-CR1.pdf

Today's Outline

- Domain Modelling
- How to create domain model?
- Example
- Case Study

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 <u>*keeps track of*</u> e.g. **Contact , book**.
  - ***Events*** *that come to light* - e.g. ***sale***.

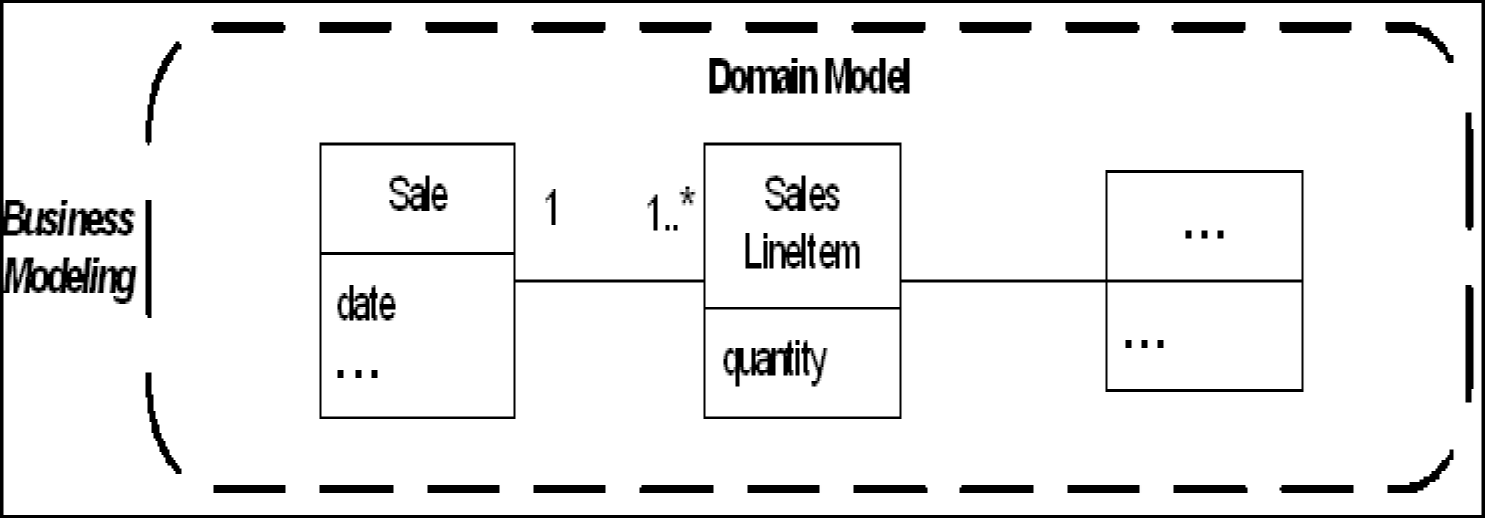
Domain Model Representation

Domain Model may contain :

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

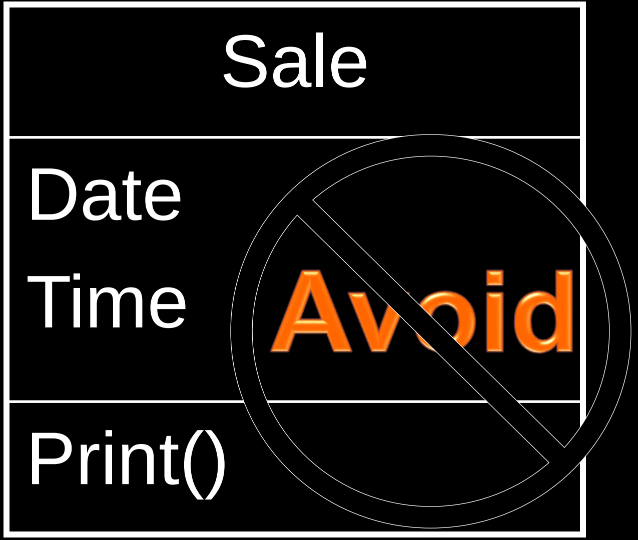
Domain Model - UML Notation

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

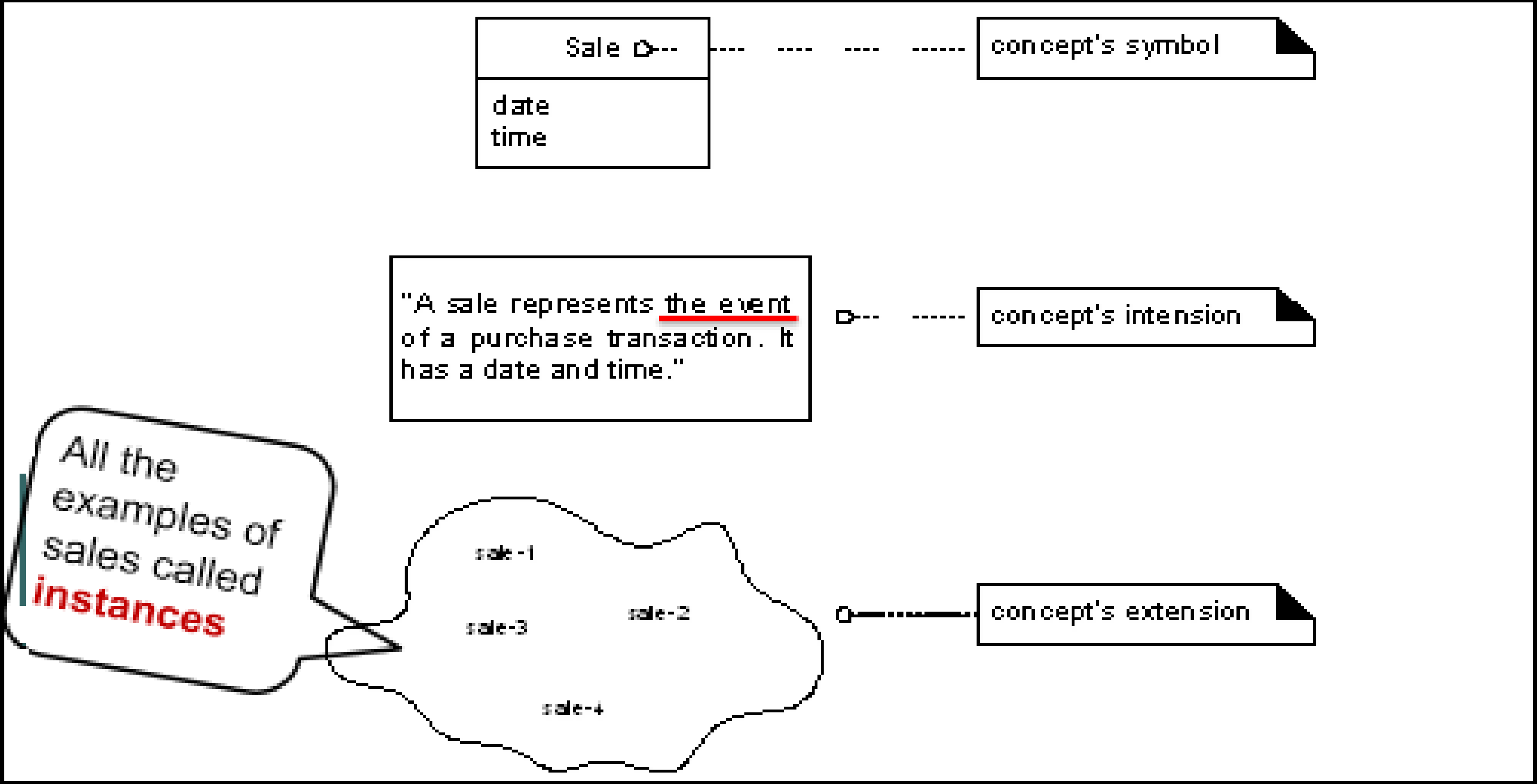


A Domain Model is not a Software document

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



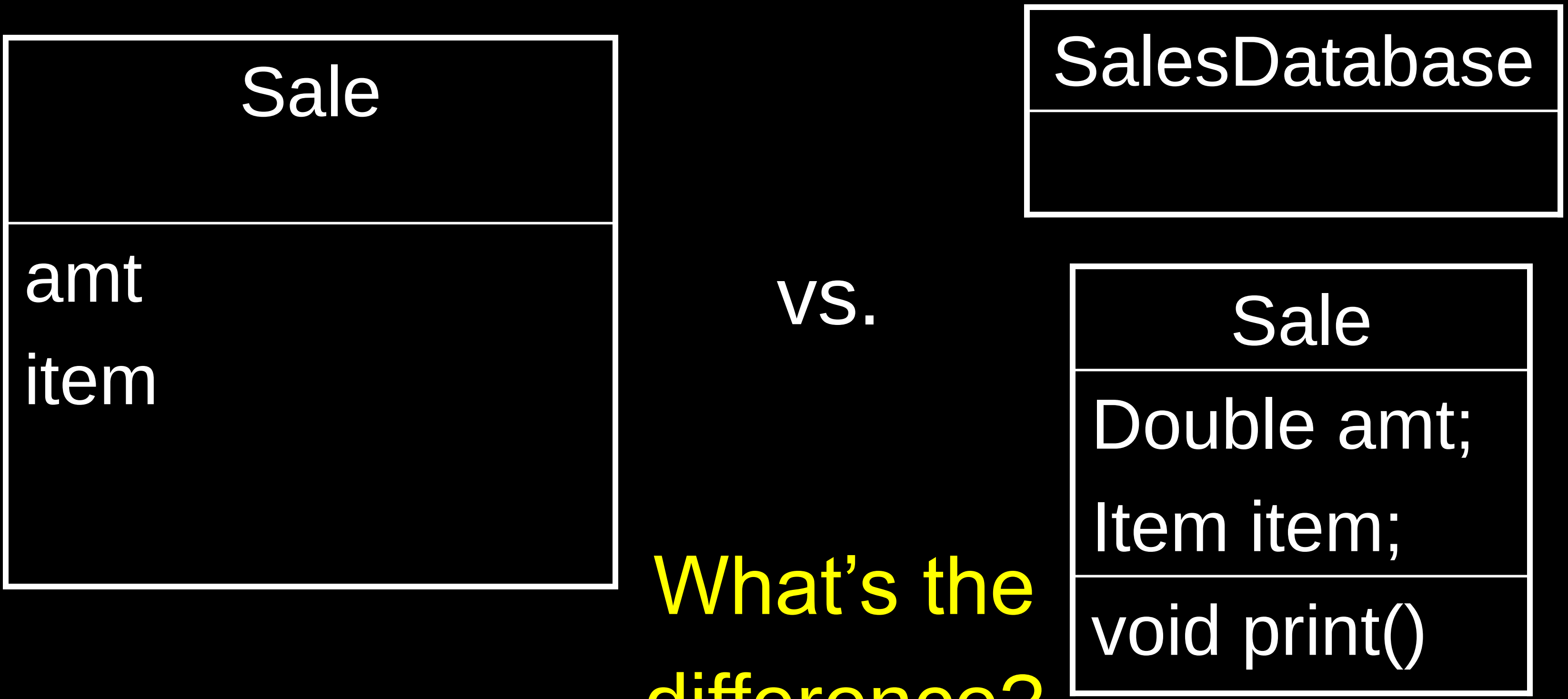
Symbol, Intension and Extension.


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-2
sale-3
sale-4
concept's extension

A Domain Model is Conceptual, not a Software Artifact

Conceptual Class:

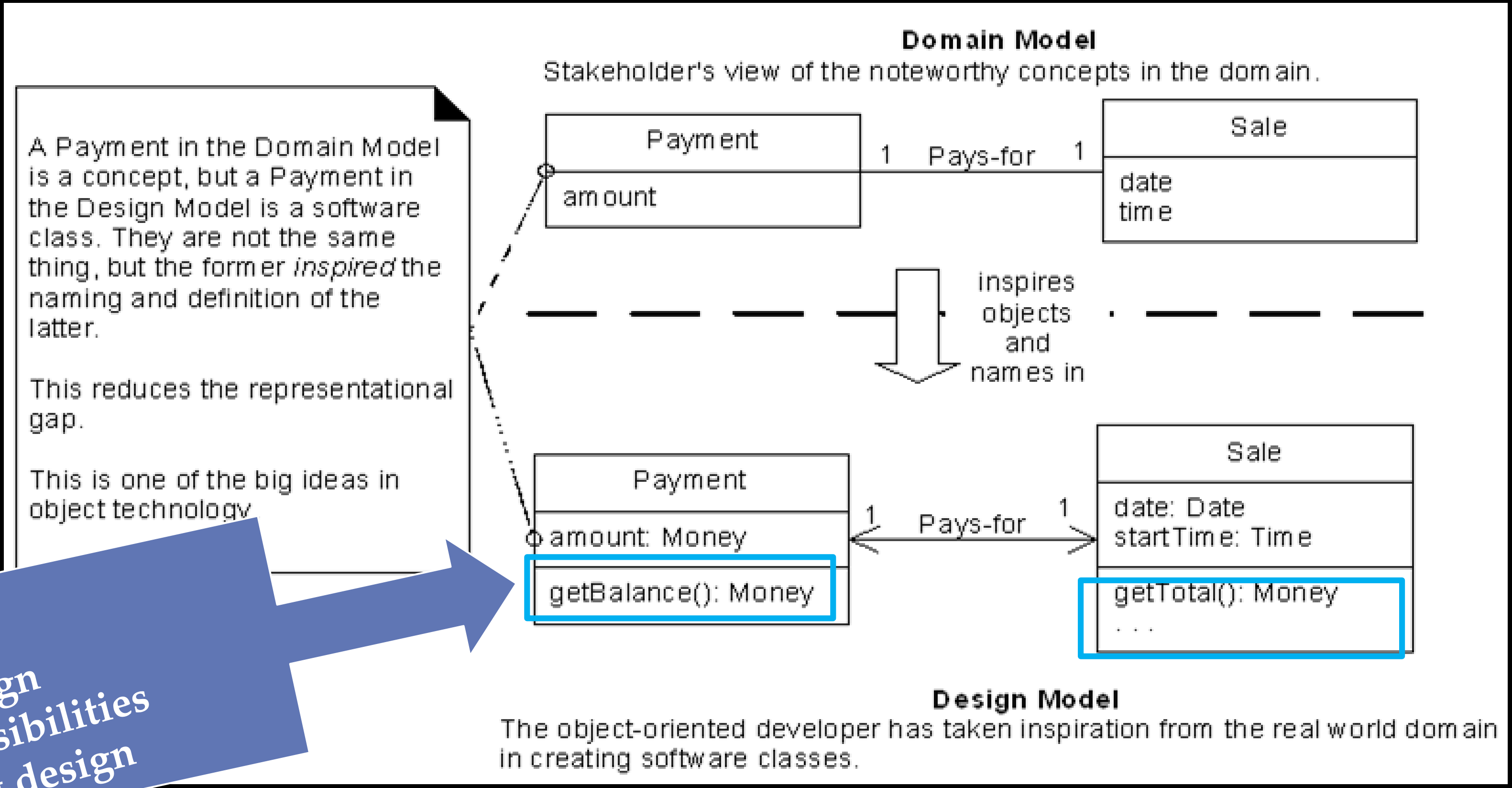
Software Artifacts:



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.
Payment
amount
1
Pays-for
1
Sale
date
time
inspires objects and names in
Payment
amount: Money
getBalance(): Money
1
Pays-for
1
Sale
date: Date
startTime: Time
getTotal(): Money
. . .
Design Model
The object-oriented developer has taken inspiration from the real world domain in creating software classes.

We assign responsibilities during design

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.

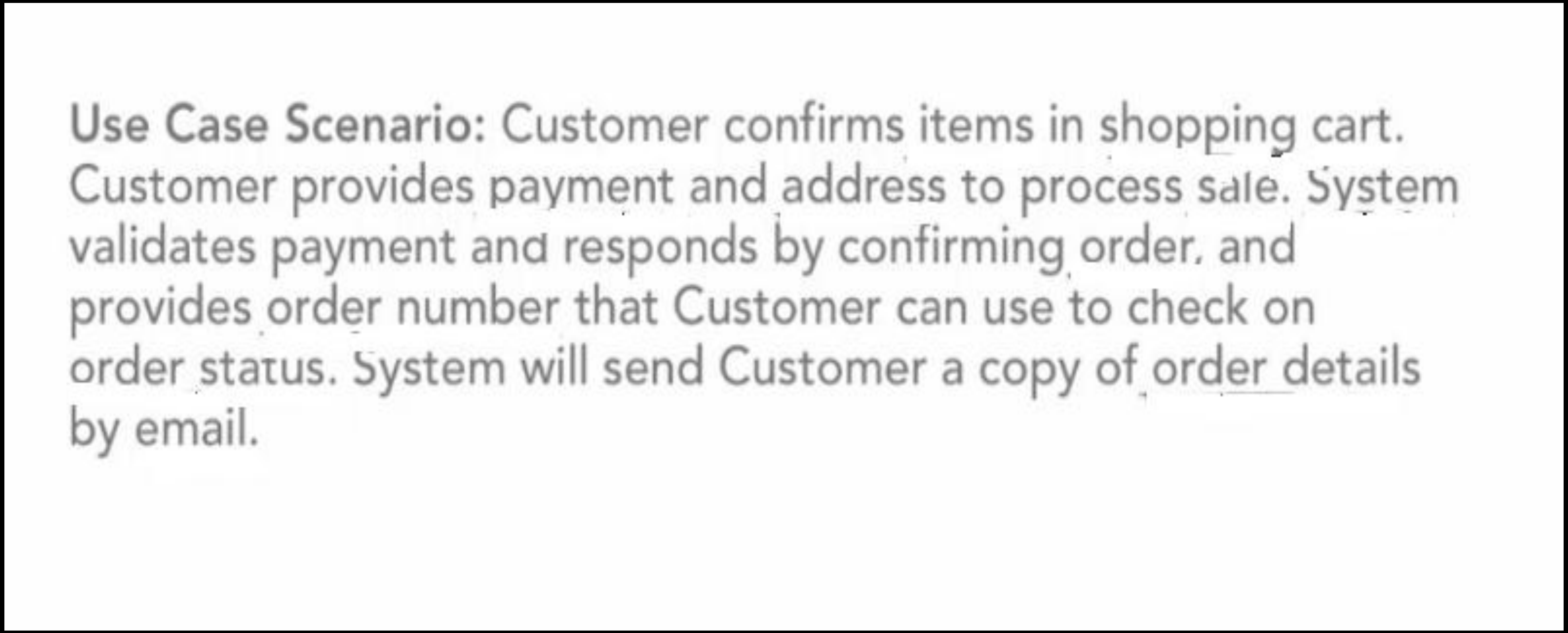
Steps To Create A Domain Model

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,

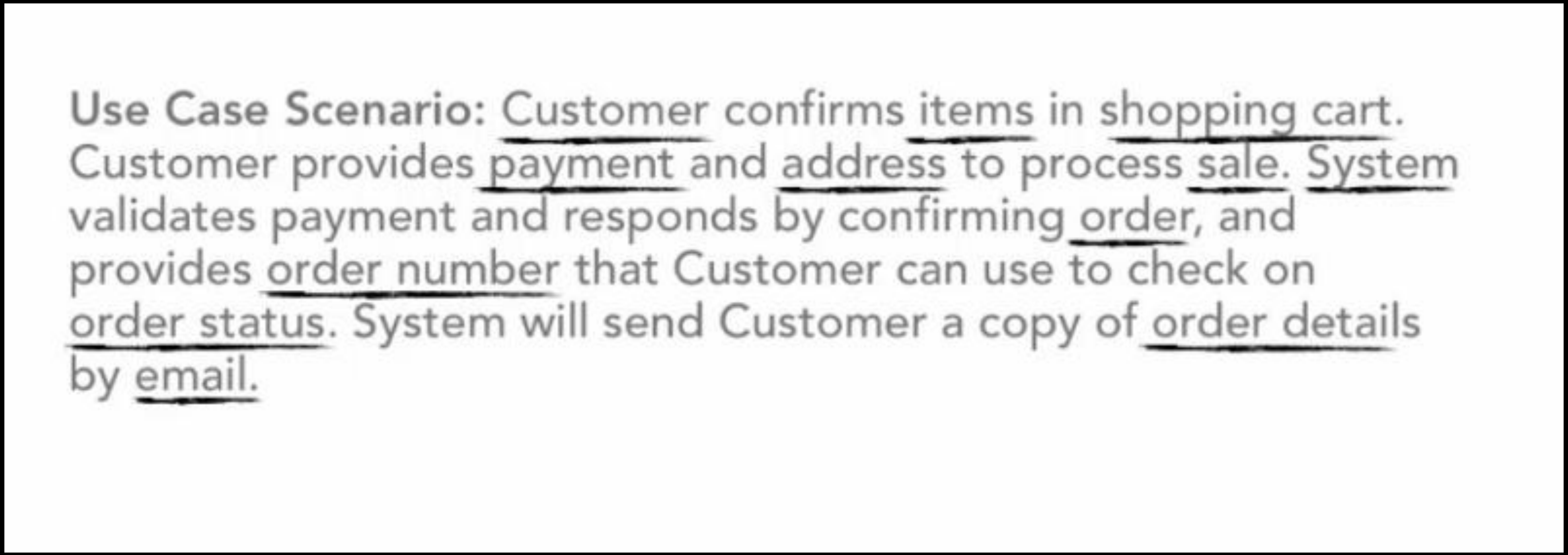
the account balance is updated. The ATM prints the customer's account balance on a receipt.

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

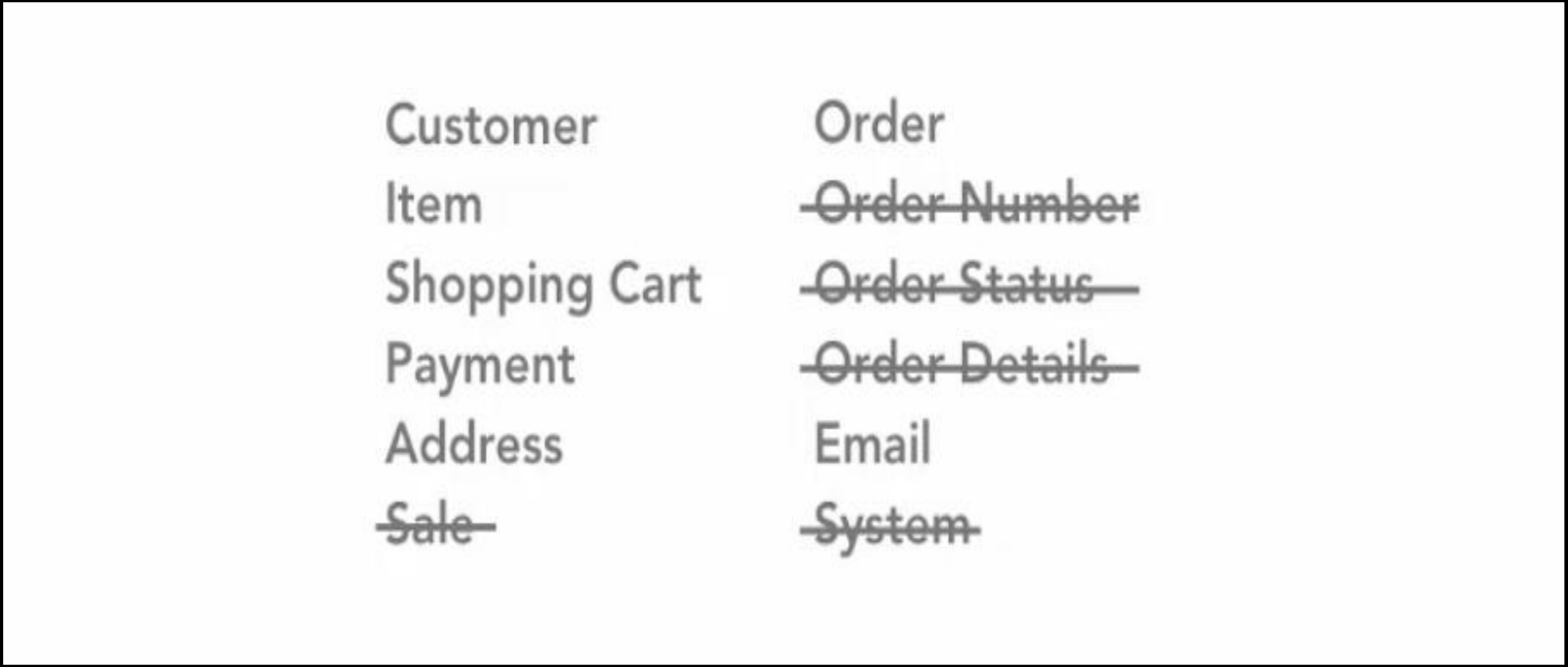


Use Case Scenario: Customer confirms items in shopping cart. Customer provides payment and address to process sale. System validates payment and responds by confirming order, and provides order number that Customer can use to check on order status. System will send Customer a copy of order details by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

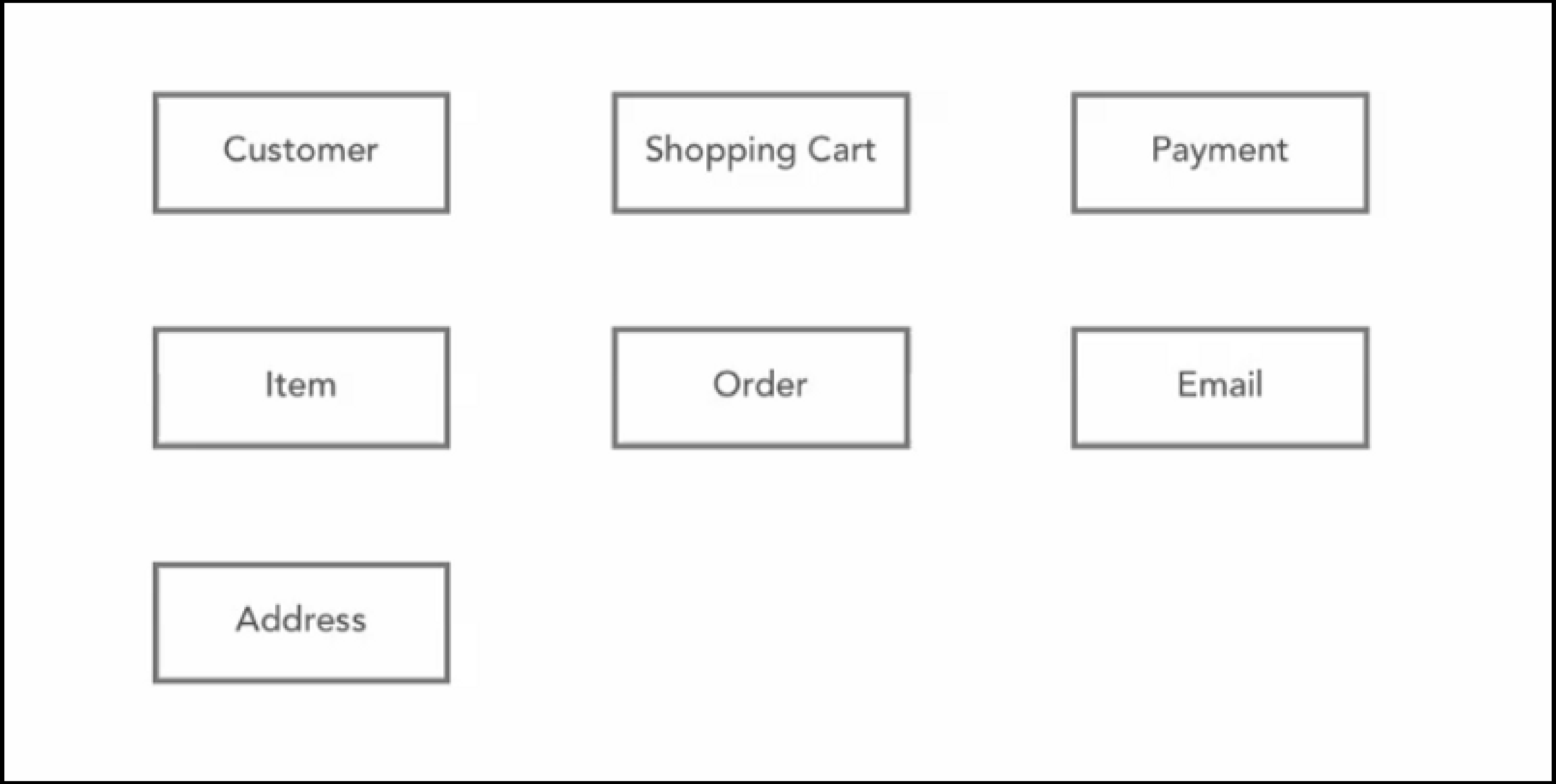
Refine Objects



Customer
Item
Shopping Cart
Payment
Address
~~Sale~~

Order
~~Order Number~~
~~Order Status~~
~~Order Details~~
Email
~~System~~

Drawing Objects



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

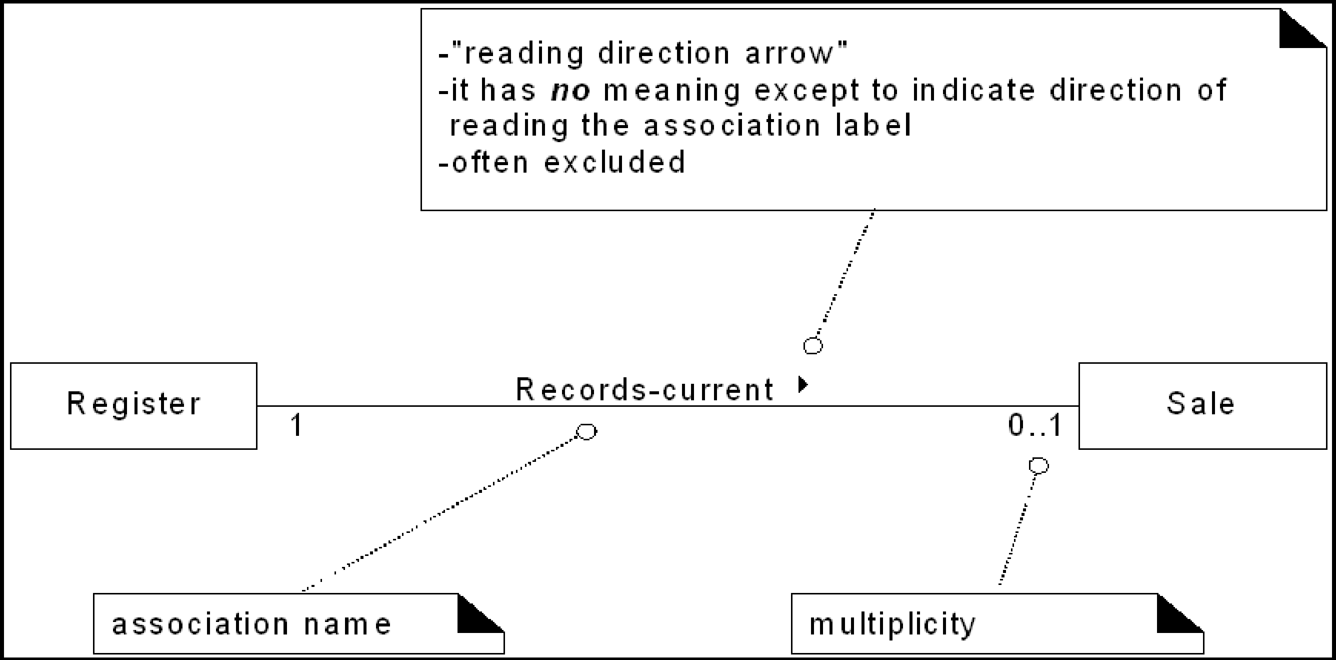
- ***If we do not think of a thing as a number or text in the real world, then it is probably a conceptual class.***
- ***If it takes up space, then it is likely a conceptual class.***

***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
1
0..1
Sale
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

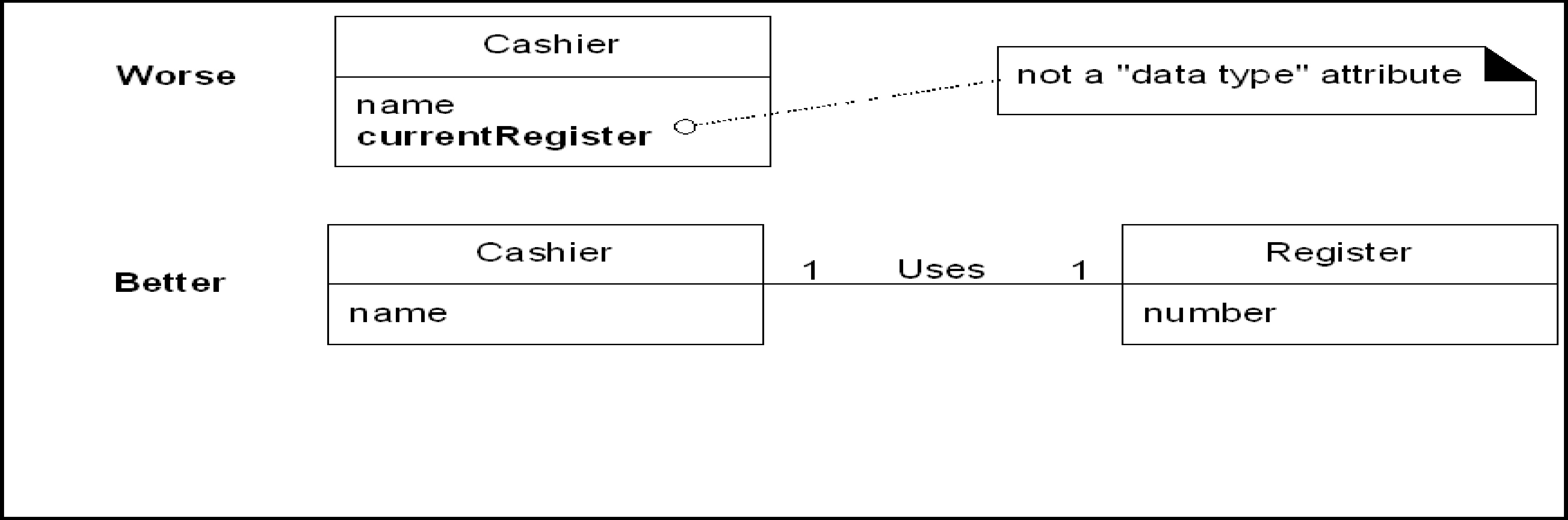
- A communicate with B
    - Customer - Cashier
- A is related to a transaction B
    - Customer - Payment
- A is a transaction related to another transaction B .
    - Payment - Sale
- A is owned by B
    - Register - Store

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

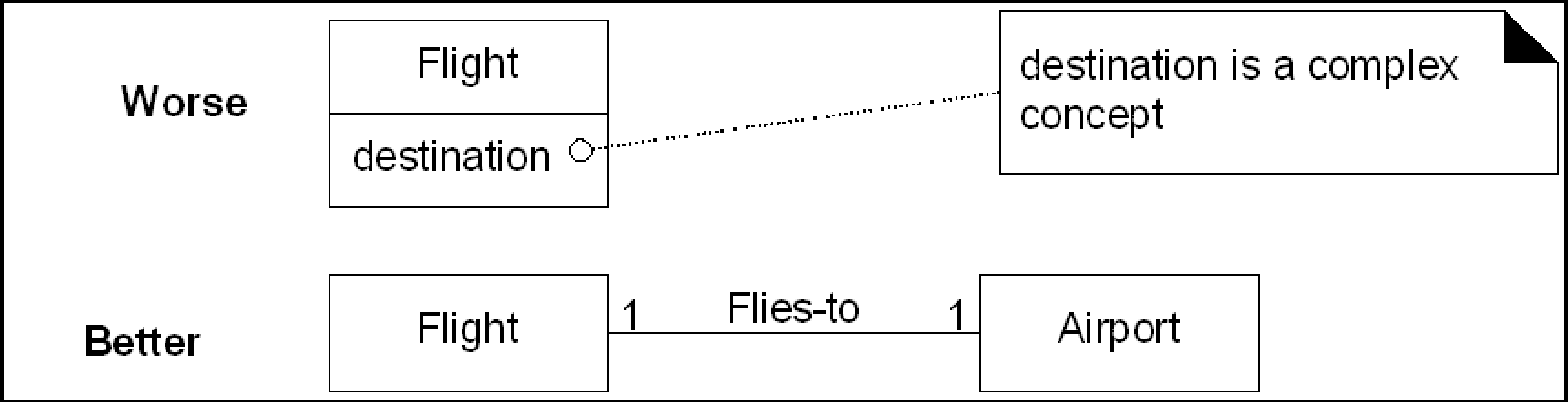
- To avoid:
  - ***Avoid showing redundant or derivable associations***
  - ***Do not overwhelm the domain model with associations not strongly required***

Association or attribute ?



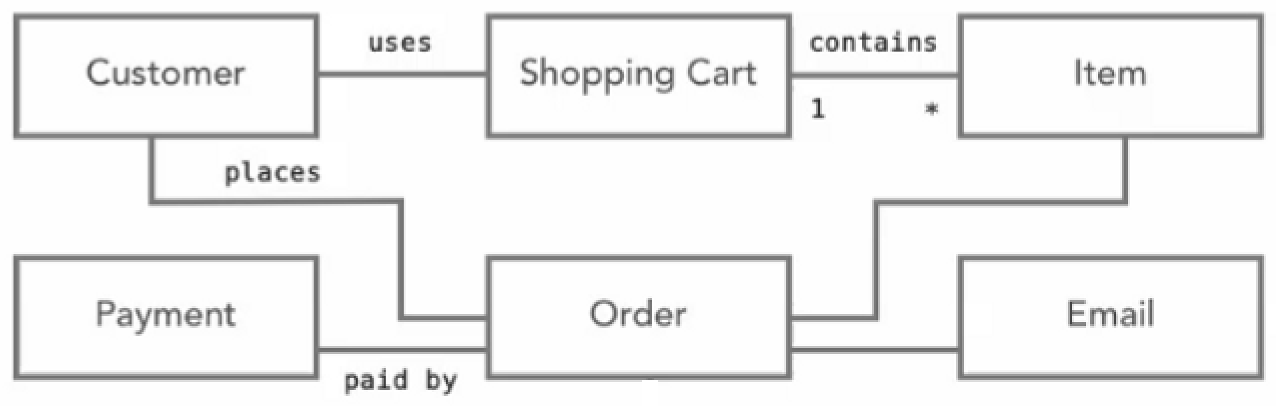
- *Most attribute type should be "**primitive**" data type, such as: numbers , string or boolean (true or false)*
- *Attribute should not be a complex domain concept(Sale , Airport)*
- *CurrentRegister is of type "Register", so expressed with an association*

Association or attribute ?

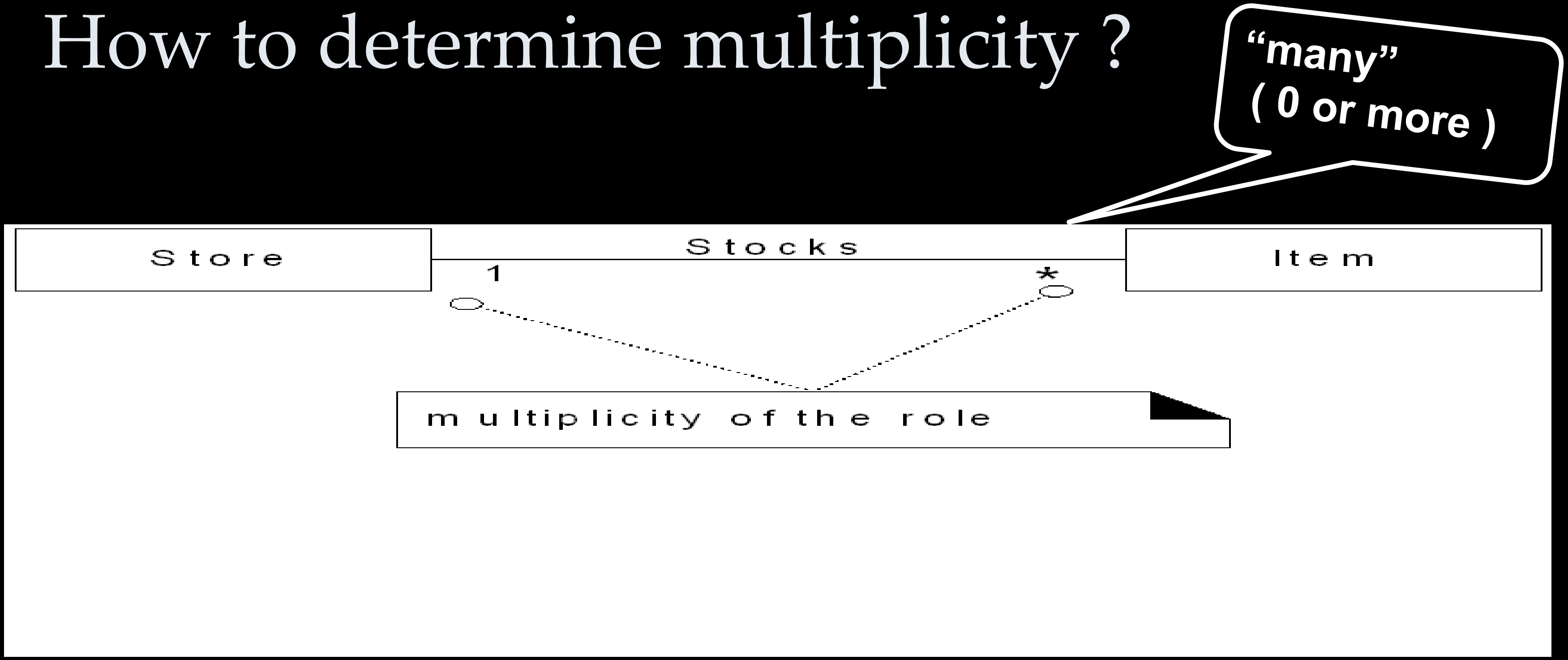


A destination airport is ***not a string***, it is a complex thing that occupies many square kilometers of space. So "Airport" should be related to "Flight" via an association, not with attribute

Identifying Object Relationships

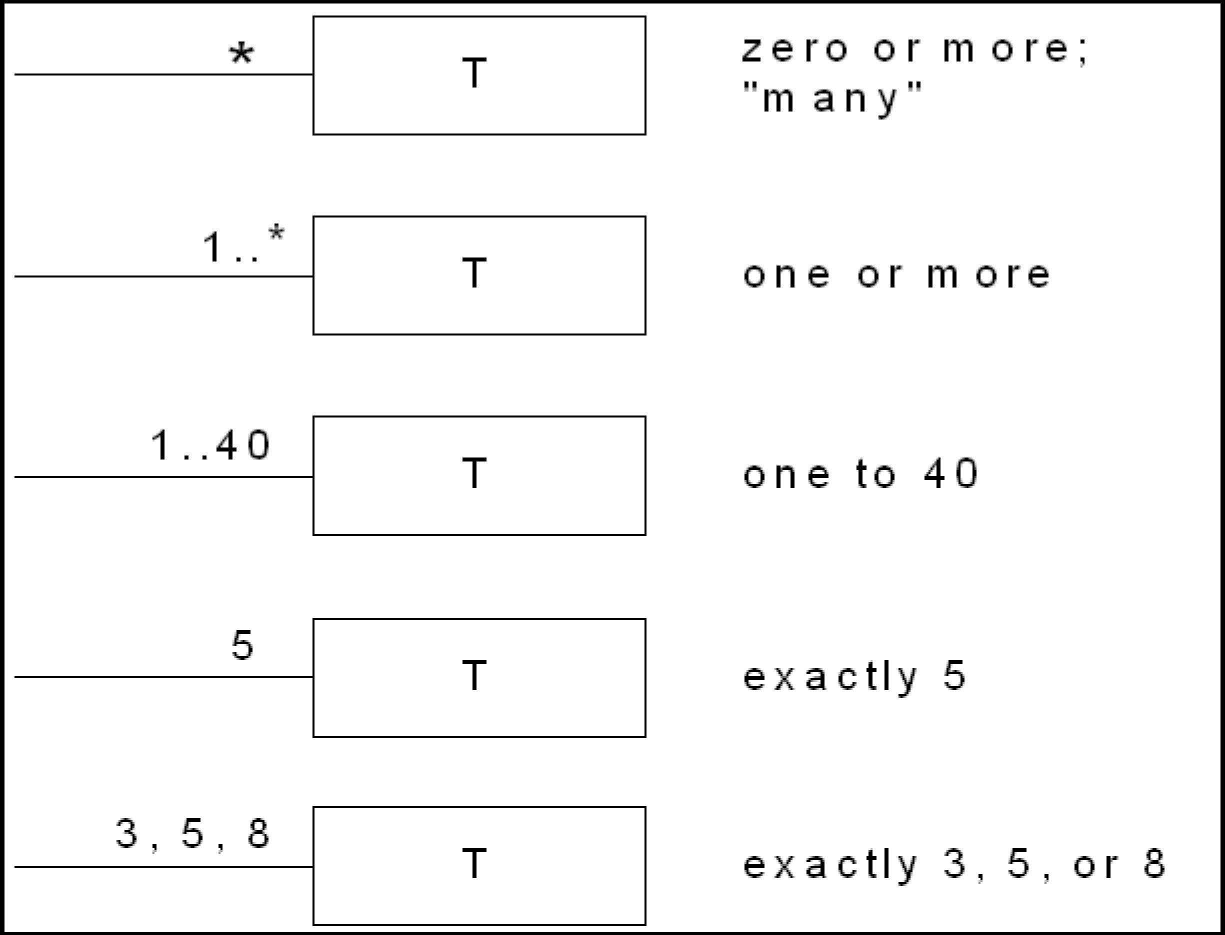


How to determine multiplicity ?



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

Multiplicity



How to create a domain model

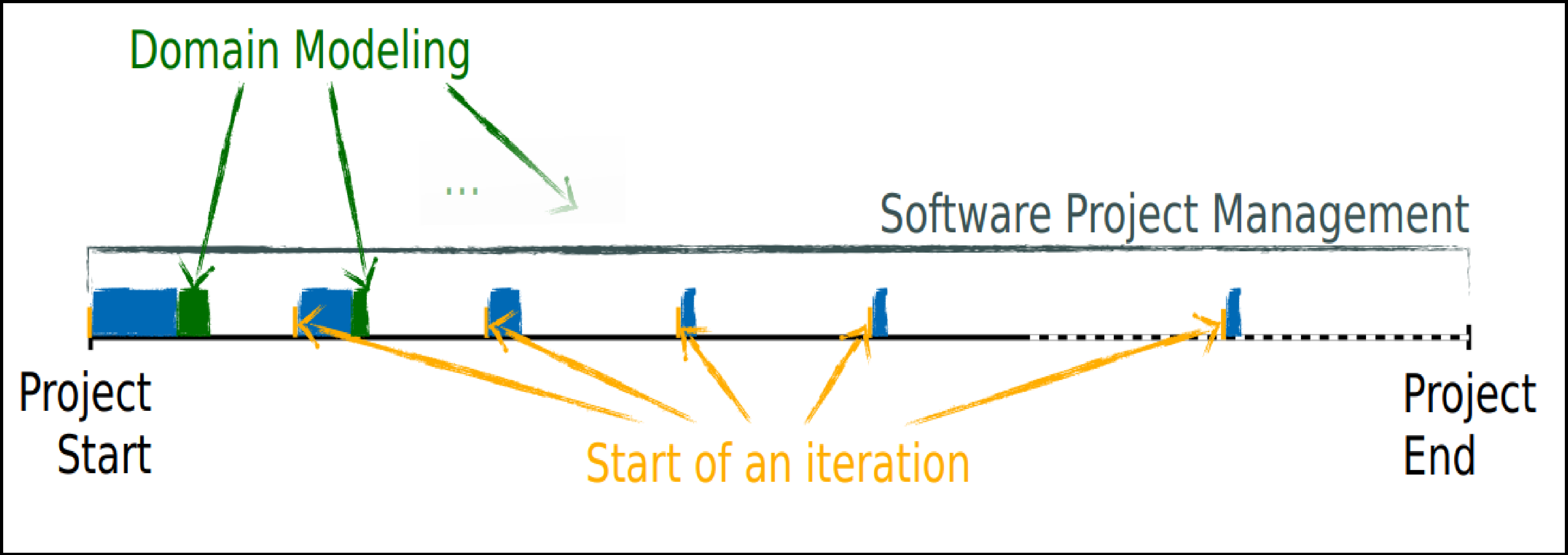
- *Identify candidate conceptual classes*
- *Go through them*
  - ***Exclude irrelevant features and duplications***
  - ***Do not add things that are outside the scope***
- *Draw them as classes in a UML class diagram*
- *Add associations necessary to record the relationship that must be retained*
-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



Example: Scenario (or Basic Flow) of POS:

1. Customer arrives at POS checkout with goods and/or services to purchase.

2. Cashier starts a new sale.

3. Cashier enters item identifier.

4. System records sale line item and presents item description, price, and running total.

Price calculated from a set of price rules.

Cashier repeats steps 3-4 until indicates done.

5. System presents total with taxes calculated.

6. Cashier tells Customer the total, and asks for payment.

7. Customer pays and System handles payment.

8. System logs completed sale and sends sale and payment information to the external

accounting system (for accounting and commissions) and Inventory system (to update inventory).

9. System presents receipt.

10. Customer leaves with receipt and goods (if any).

- **Main Success Scenario (or Basic Flow) of POS:**

1. Customer arrives at POS checkout with goods and/or services to purchase.

2. Cashier starts a new sale.

3. Cashier enters item identifier.

4. System records sale line item and presents item description, price, and running total.

Price calculated from a set of price rules.

Cashier repeats steps 3-4 until indicates done.

5. System presents total with taxes calculated.

6. Cashier tells Customer the total, and asks for payment.

7. Customer pays and System handles payment.

8. System logs completed sale and sends sale and payment information to the external accounting system (for accounting and commissions) and Inventory system (to update inventory).

9. System presents receipt.

10. Customer leaves with receipt and goods (if any).

That is all