COMP 3700: Software Modeling and Design

(Domain Analysis)

Topics

- Process Overview
  - Development Stages and Lifecycle
- System Conception
  - Preparing a Problem Statement
- Domain Analysis
- Application Analysis

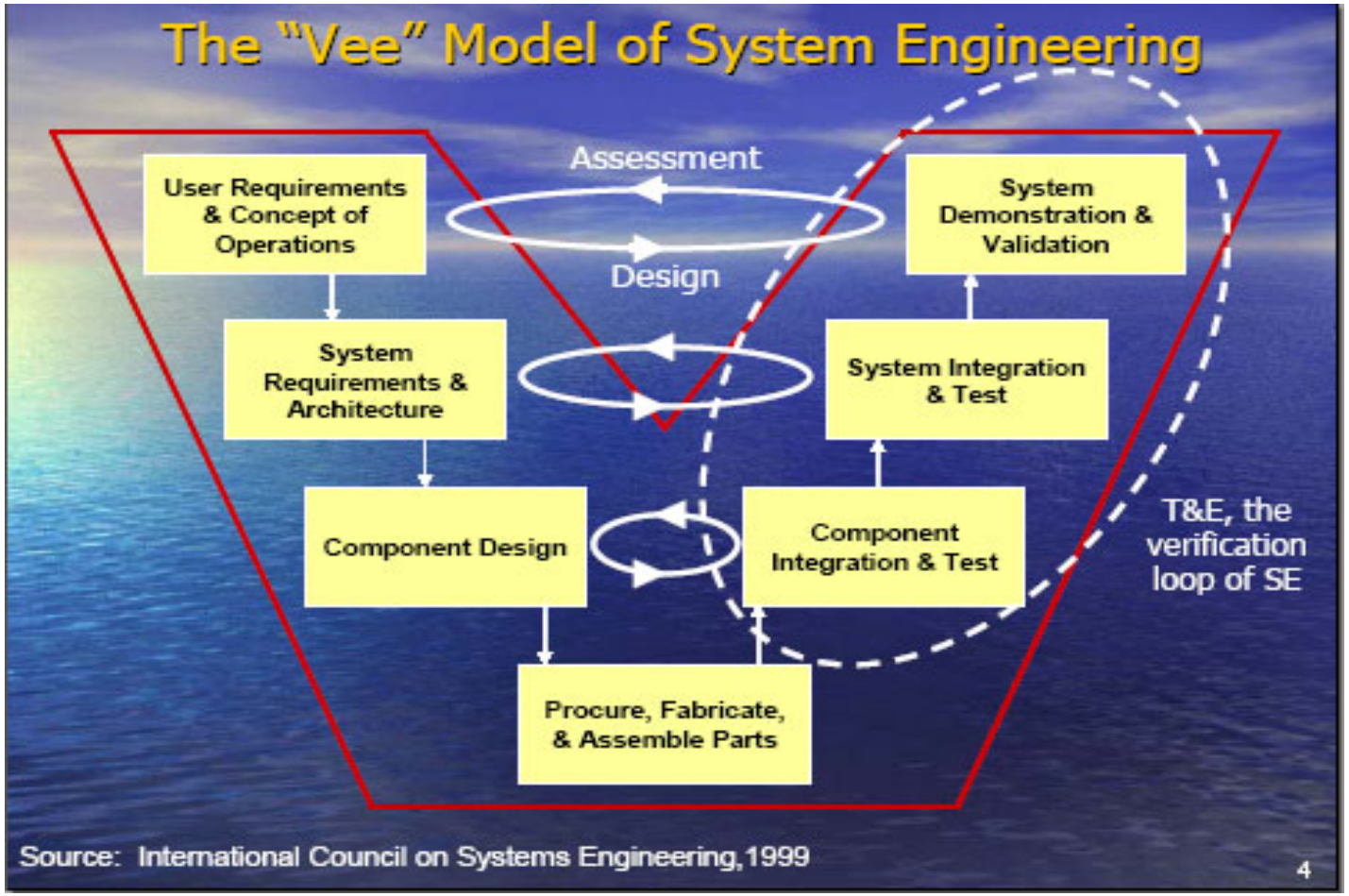
Development Stages

- **System conception** – Conceive an application and formulate tentative requirements.
- **Analysis** – Understand requirements by constructing models
  - Specify *what* needs to be done, not *how* it is done.
- **System design** – Devise the architecture and object interaction design.
- **Class design** – augment and adjust entity representations from analysis
  - Algorithms, data structures, class operations.

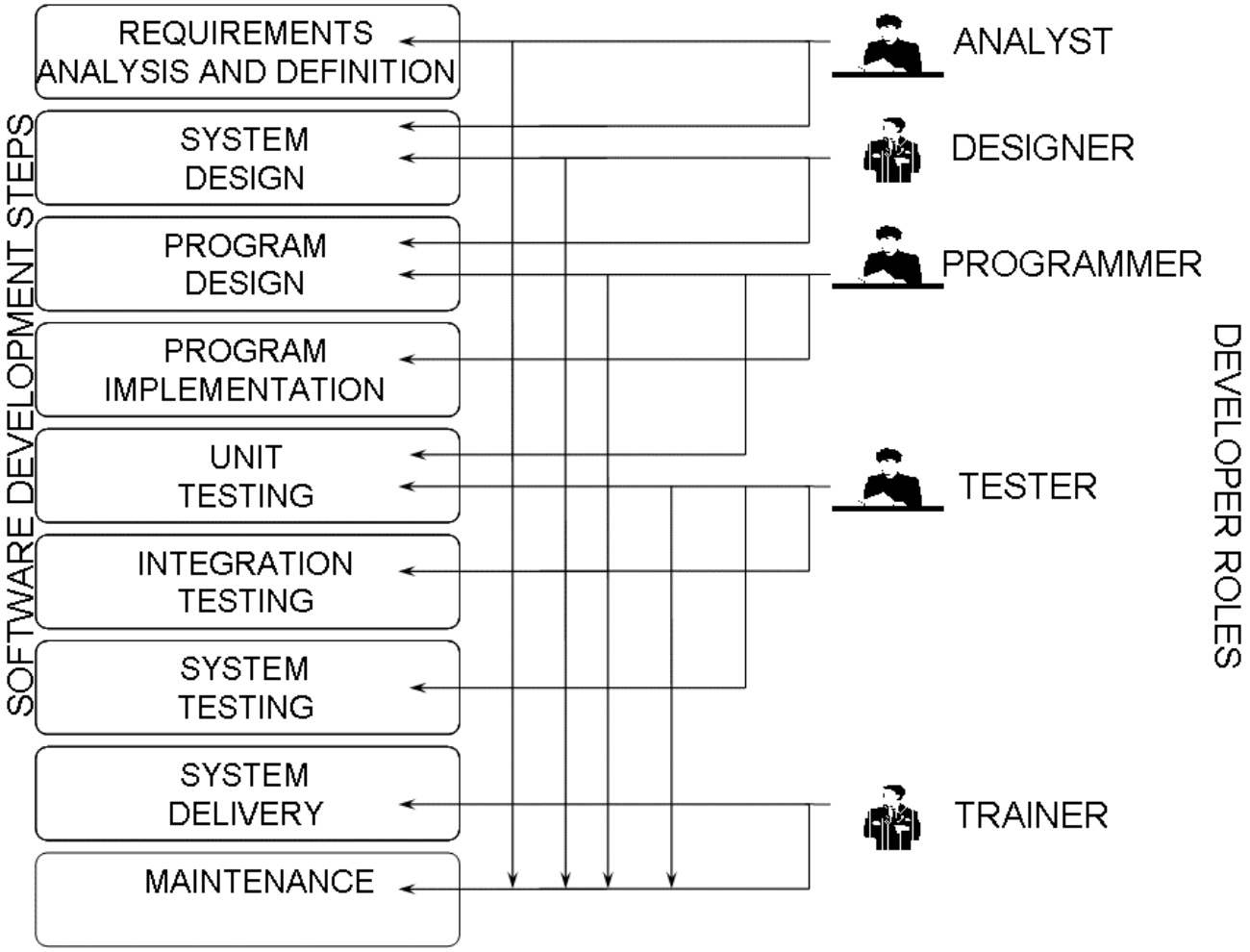
Development Stages

- **Implementation** – Translate the design into programming code and data structures.
- **Testing** – Ensure that the application is suitable for actual use and it satisfies the requirements.
- **Deployment** – Place the application in the field.
- **Maintenance** – Preserve the long-term viability of the application.

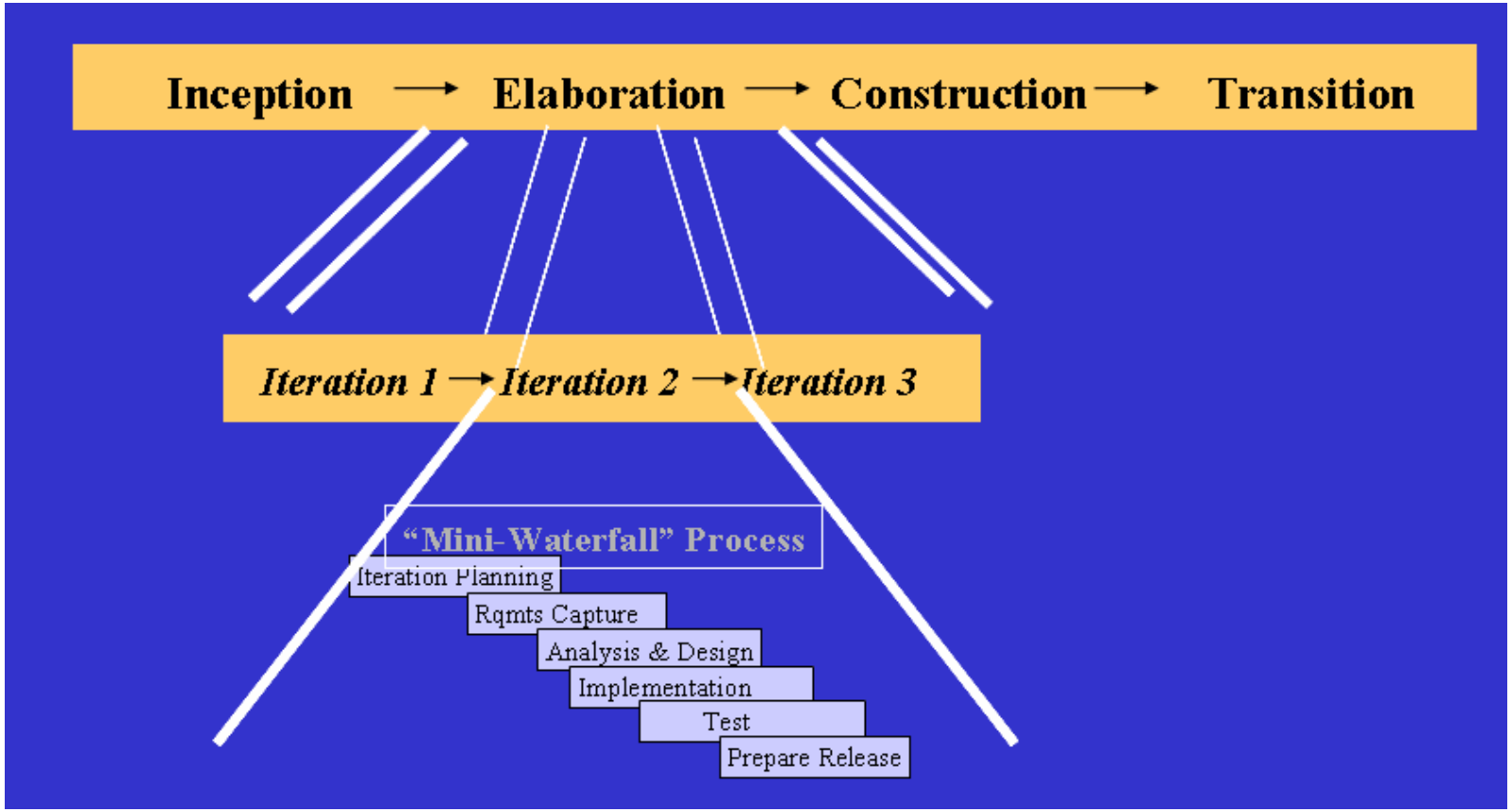
Development Life Cycle



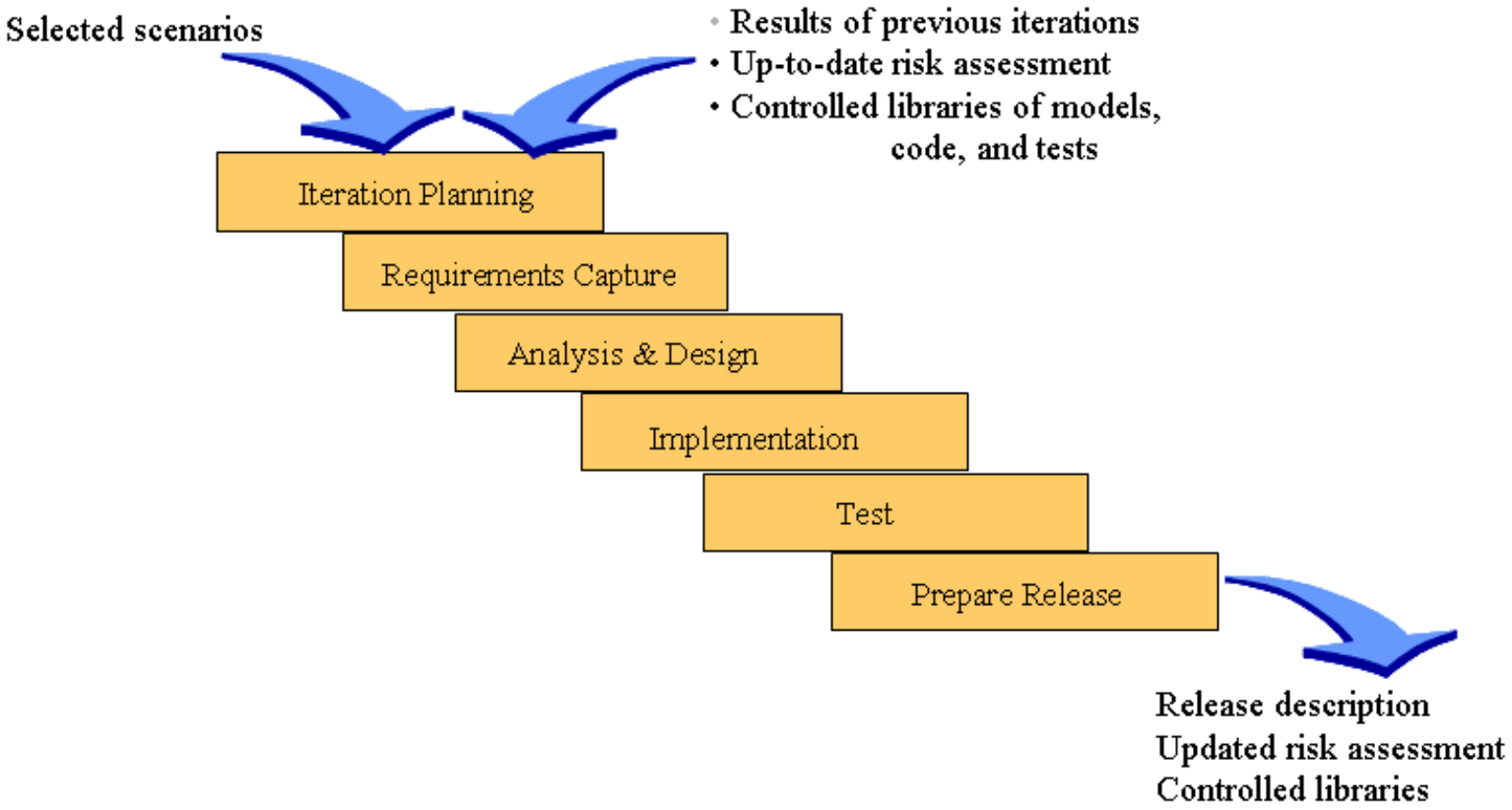
Waterfall Lifecycle



Incremental Iterative Lifecycle



Incremental Iterative Lifecycle



System Conception

- **The purpose of system conception –**
  - **What need does the proposed system meet?**
  - **Can it be developed at a reasonable cost?**
  - **Will the demand for the result justify the cost for development?**
- **What is the vision and business case for the project?**
- **Is it feasible?**
- **Buy and/or build?**
- **Proceed or stop?**

Vision – Problem Statement

Problem Statement

The problem of	*[describe the problem]*
affects	*[the stakeholders affected by the problem]*
the impact of which is	*[what is the impact of the problem?]*
a successful solution would be	*[list some key benefits of a successful solution]*

Vision – Position Statement

Position Statement

For	*[target customer]*
Who	*[statement of the need or opportunity]*
The (product name)	*is a [product category]*
That	*[statement of key benefit; that is, the compelling reason to buy]*
Unlike	*[primary competitive alternative]*
Our product	*[statement of primary differentiation]*

Vision – Stakeholders

Stakeholders

Name	Description	Responsibilities
[Name the stakeholder type.]	*[Briefly describe the stakeholder.]*	*[Summarize the stakeholder's key responsibilities with regard to the system being developed; that is, their interest as a stakeholder. For example, this stakeholder:* *ensures that the system will be maintainable* *ensures that there will be a market demand for the product's features* *monitors the project's progress* *approves funding* *and so forth]*

Vision – User Environment

- Detail the working environment of the target user. Here are some suggestions:
  - Number of people involved in completing the task? Is this changing?
  - How long is a task cycle? Amount of time spent in each activity? Is this changing?
  - Any unique environmental constraints: mobile, outdoors, in-flight, and so on?
  - Which system platforms are in use today? Future platforms?
  - What other applications are in use? Does your application need to integrate with them?
  - This is where extracts from the Business Model could be included to outline the task and roles involved, and so on.]

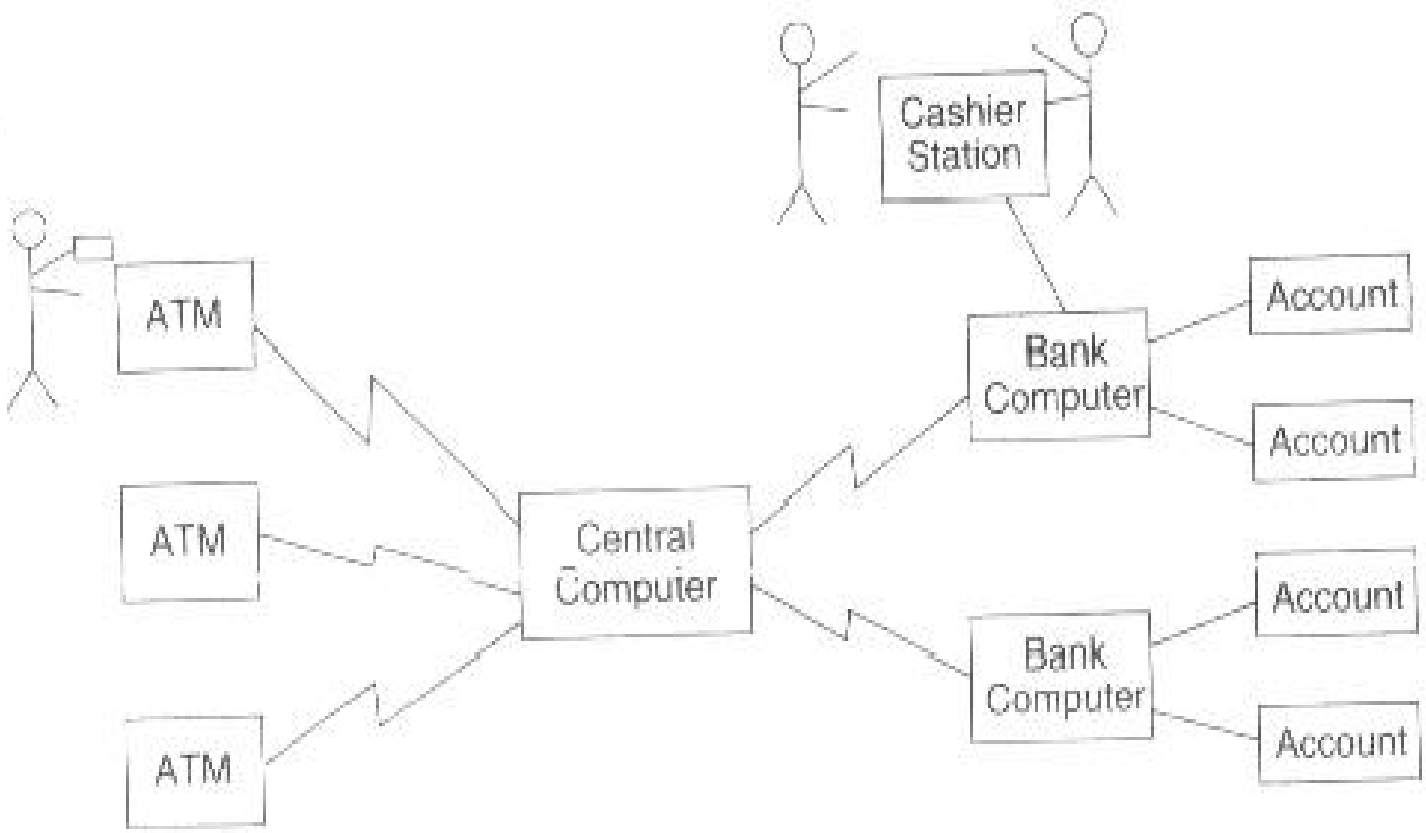
Vision – The Rest

- Please see the
  - Product Overview
    - Product Perspective
    - Assumptions and Dependencies
    - Needs and Features
    - Alternatives and Competition
  - Other Product Requirements

Elaborating a Concept

- Who is the application for?
- What problems it will solve?
- Where will it be used?
- When is it needed?
- Why is it needed?
- How will it work?

The ATM Case Study – Concept Statement



The ATM Case Study – Concept Statement

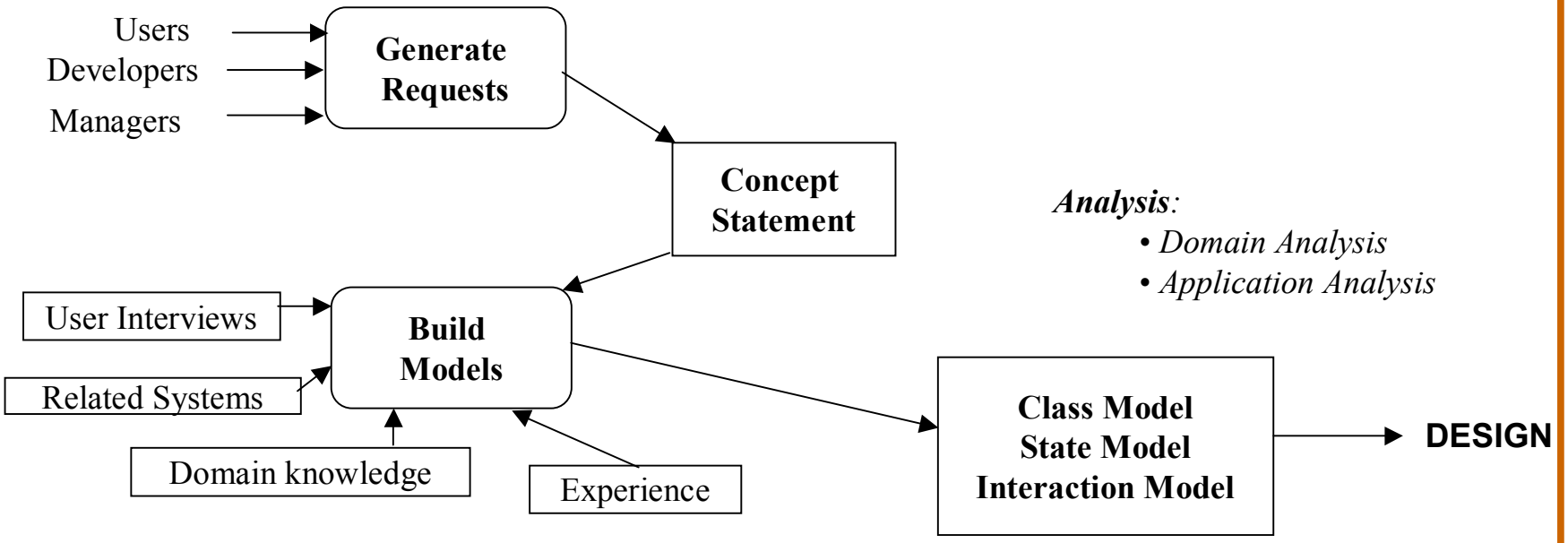
Design the software to support a computerized banking network including both human cashiers and automatic teller machines (ATMs) to be shared by a consortium of banks. Each bank provides its own computer to maintain its own accounts and process transactions against them. Cashier stations are owned by individual banks and communicate directly with their own bank's computers. Human cashiers enter account and transaction data.

Automatic teller machines communicate with a central computer that clears transactions with the appropriate banks. An automatic teller machine accepts a cash card, interacts with the user, communicates with the central system to carry out the transaction, dispenses cash, and prints receipts. The system requires appropriate recordkeeping and security provisions. The system must handle concurrent accesses to the same account correctly.

The banks will provide their own software for their own computers; you are to design the software for the ATMs and the network. The cost of the shared system will be apportioned to the banks according to the number of customers with cash cards.

Domain Analysis

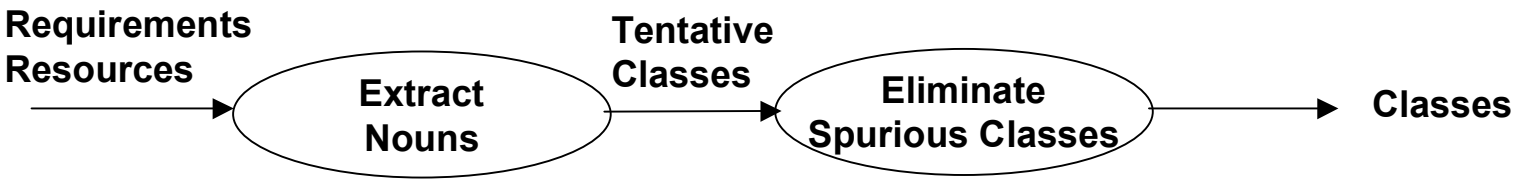
- **Domain analysis is concerned with devising a precise, concise, and understandable model of the real world.**
- **Analysis starts with the *concept statement* generated during system conception.**



Class Domain Model

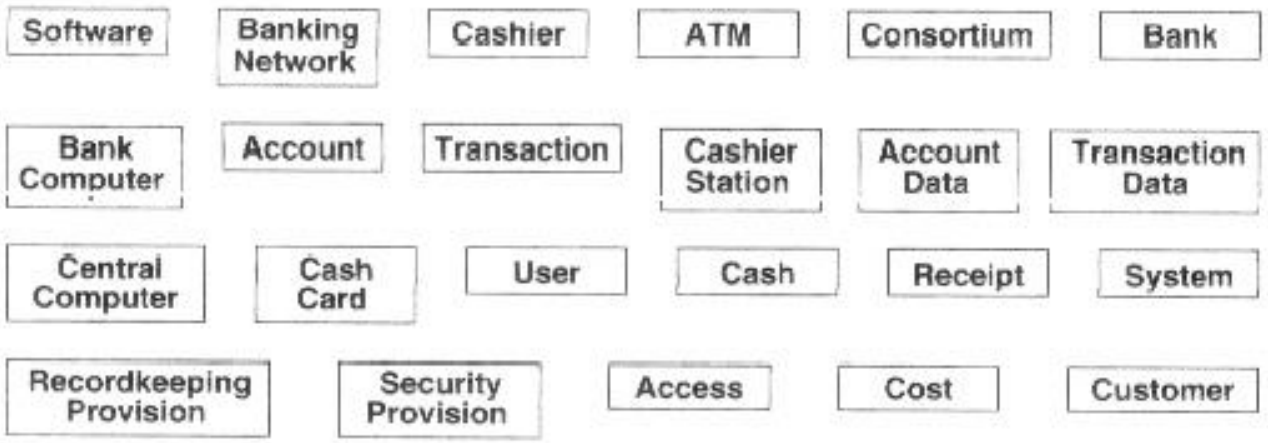
- **First step:** Construct the class domain model that shows the static structure of the system – Real word classes and their relations to each other.
- You need to perform the following:
  - Find classes
  - Prepare a data dictionary
  - Find associations
  - Find attributes
  - Organize and simplify classes using inheritance
  - Verify that access paths exist for likely queries
  - Iterate and refine the query
  - Group classes into packages

Finding Classes



- "A reservation system to sell tickets to performances at various theatres".
- ***Reservation***, ***System***, ***Ticket***, ***Performance***, ***Theater***

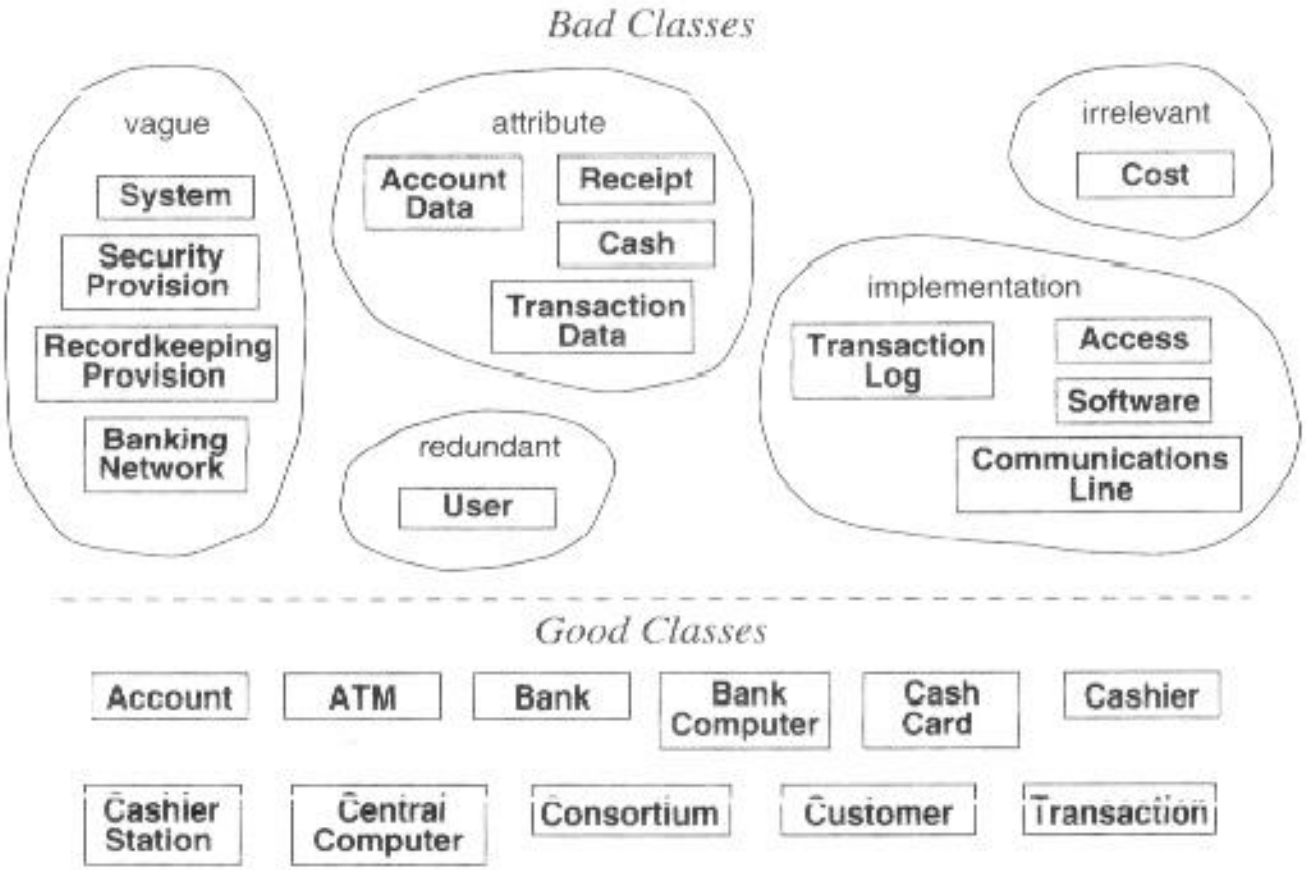
ATM Classes



ATM Classes Extracted from the Problem Statement Nouns

ATM Classes Extracted from the Knowledge of the Problem Domain

Keeping the Right Classes



Keeping the Right Classes

- **Redundant classes – If two classes express the same concept, you should keep the most descriptive name.**
  - ***Customer* and *User* are redundant; retain *Customer* because it is more descriptive.**
- **Irrelevant classes – A class has little or nothing to do with the problem, eliminate it.**
  - **Apportioning *Cost* is outside the scope of the ATM software.**
- **Vague classes – A class should be specific (not broad).**
  - ***Recordkeeping Provision* is vague and is handled by *Transaction*.**

Keeping the Right Classes

- **Attributes – Names that primarily describe individual objects should be restated as attributes.**
  - **Does the independent existence of a property important?**
  - **ATM Example: *AccountData* probably describes an account.**
- **Operations – A name describes an operation that is applied to objects.**
  - **A *Call* may be a sequence of actions – become a part of the state model**
  - **A *Call* may also be a class in a billing system with attributes *date, time, origin, destination.***

Keeping the Right Classes

- **Roles – The name of a class should reflect its intrinsic nature, not the role it plays.**
  - **For a car manufacturer database system *Owner* would be a poor name; *Customer* or possibly *Person* is better.**
- **Implementation Constructs – Eliminate constructs extraneous to the real-world.**
  - **ATM Example: *CommunicationLine* is the physical implementation of a communication link.**

Preparing a Data Dictionary

- Isolated words have too many interpretations.
- Prepare a data dictionary for all modeling elements.
  - Write a paragraph to precisely describe each class.
  - Describe the scope of the class in the context of the problem.
- Example:
  - **Account:** A single account at a bank against which transactions can be applied. Accounts may be of various types, such as checking or savings. A customer can hold more than one account.
  - **Bank:** A financial institution that holds accounts for customers and issues cash cards authorizing access to accounts over the ATM network.

ATM Example

Verb phrases

Banking network includes cashier stations and ATMs
Consortium shares ATMs
Bank provides bank computer
Bank computer maintains accounts
Bank computer processes transaction against account
Bank owns cashier station
Cashier station communicates with bank computer
Cashier enters transaction for account
ATMs communicate with central computer about transaction
Central computer clears transaction with bank
ATM accepts cash card
ATM interacts with user
ATM dispenses cash
ATM prints receipts
System handles concurrent access
Banks provide software
Cost apportioned to banks

Implicit verb phrases

Consortium consists of banks
Bank holds account
Consortium owns central computer
System provides recordkeeping
System provides security
Customers have cash cards

Knowledge of problem domain

Cash card accesses accounts
Bank employs cashiers

Keeping the Right Associations

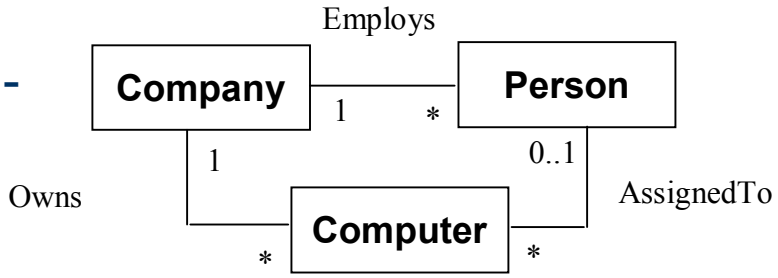
- **Associations between eliminated classes – If certain classes are eliminated, then the relationships they entail are not useful.**
  - **ATM Example: *ATM dispenses cash*; *ATM prints receipts*; *System provides recordkeeping* etc.**
- **Irrelevant or implementation associations - Eliminate associations that are outside the problem domain or deal with implementation constructs.**
  - **ATM Example: *System handles concurrent access* is an implementation concept. [Real-world objects are inherently concurrent; it is the implementation of the access algorithm that must be concurrent].**

Keeping the Right Associations

- **Actions – An association should describe a structural property of the application domain, not a transient event.**
  - ***ATM Example: ATM accepts cash card* describes part of the interaction cycle between customer and ATM – not a persistent connection.**
- **Ternary associations – Decompose associations among three or more classes into binary associations.**
  - **ATM Example: *Bank computer processes transactions against account* can be broken into *Bank computer processes transactions* and *Transaction concerns account.***

Keeping the Right Associations

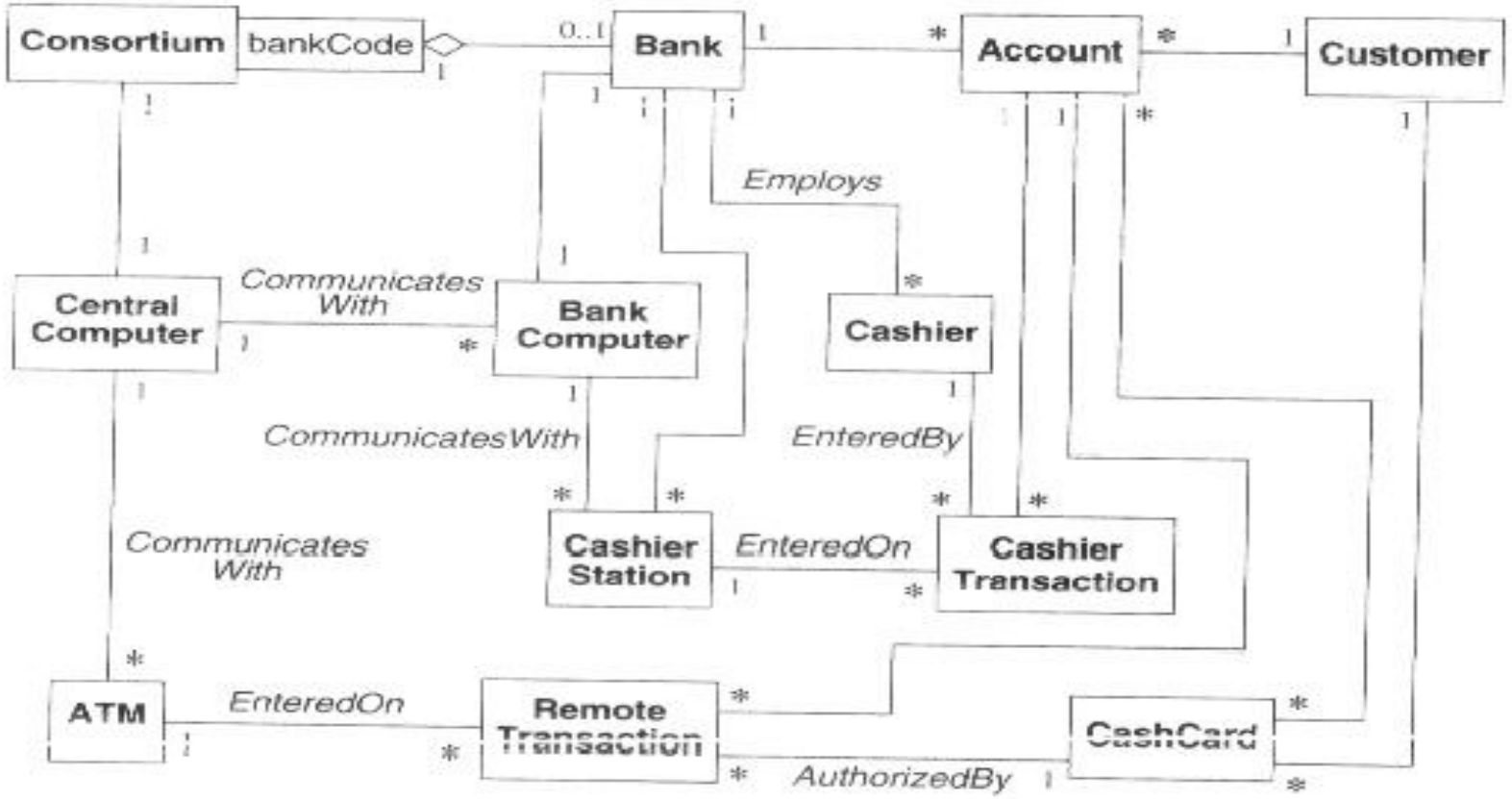
- **Derived Associations – Omit associations that can be defined in terms of other associations (avoid redundancy)**
  - **Example: *GrandParentOf* can be defined in terms of a pair of *ParentOf*.**
  - **Multiple paths between classes sometimes indicate derived associations.**
    - ***Consortium shares ATMs* is a composition of *Consortium owns Central Computer* and *Central Computer communicates with ATMs*.**
  - **Nonredundant associations -**



Keeping the Right Associations

- Misnamed Association – Don't say how or why a situation came about, say what it is.
  - ATM Example: *Bank computer maintains accounts* is a statement of activity; rephrase as *Bank holds account*
- Add association end names
- Qualified associations – Use qualifiers to reduce multiplicity and to uniquely identify entities.
- Specify Multiplicity – For multiplicity values of "many" consider if a qualifier is needed; and check if objects are ordered.
- Add any missing associations
- Use aggregation/composition to depict containment associations.

Keeping the Right Associations



Initial Class Diagram for ATM System

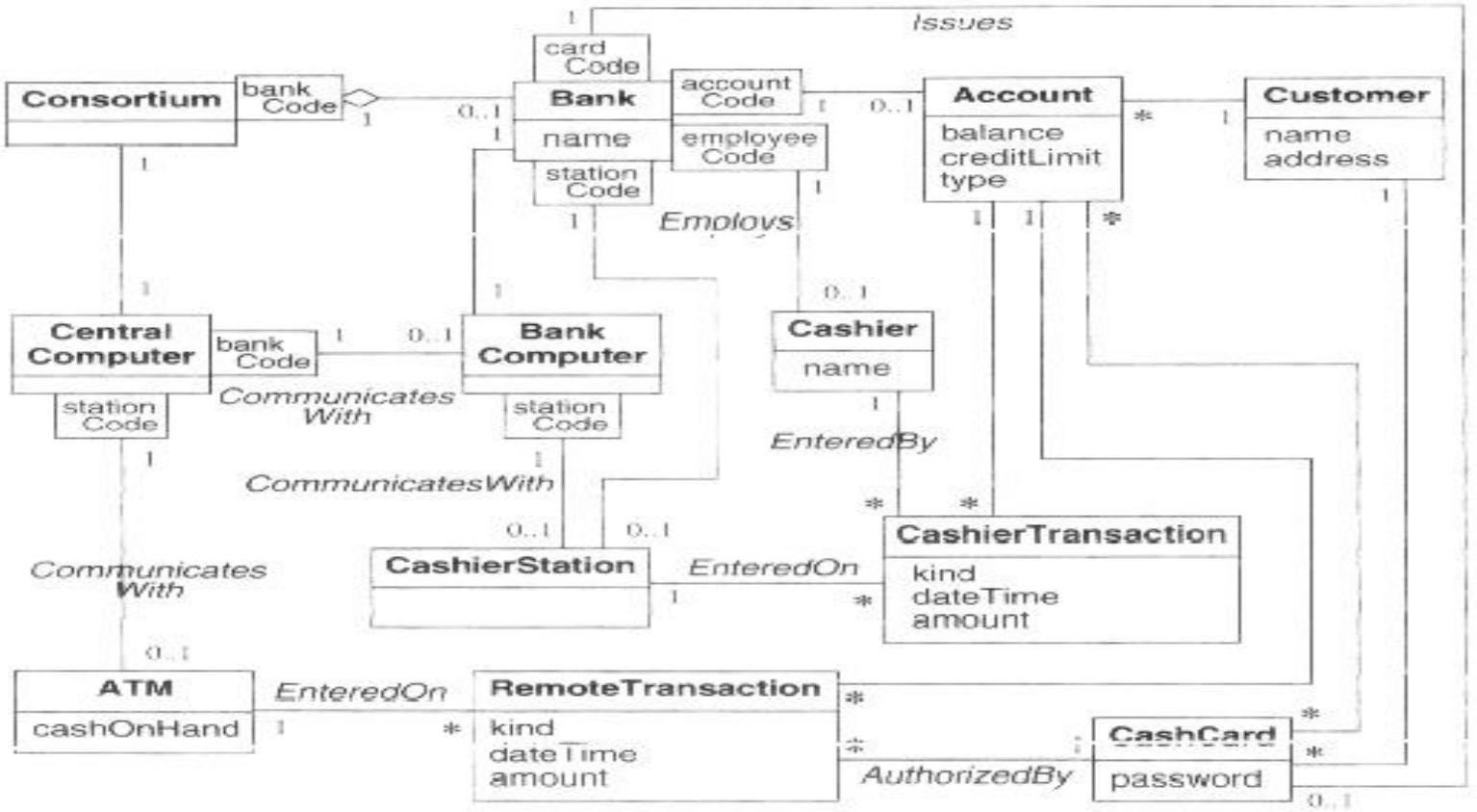
Finding Attributes

- Attributes are data properties of individual objects.
- Attributes correspond to nouns followed by possessive phrases.
  - Example: The *color* of the car; the *position* of the cursor.
- Attributes are less likely to be fully described in the problem statement.
  - You need to draw on your knowledge of the application domain .
  - You can also find attributes in the artifacts of related systems.
- Good news: Attributes seldom affect the basic structure of the problem.
- Omit derived attributes. Example: *Age* is derived from *birthdate* and *currentdate*.

Keeping the Right Attributes

- **Objects – If the independent existence of an element is important, rather than just its value, then it is an object.**
- **Qualifiers – If the value of an attribute is context-dependent, then consider restating the attribute as a qualifier.**
  - **Example: *employeeNumber* is not a unique property of a person with two jobs; it qualifies the association *Company employs Person*.**
- **ATM Example:**
  - ***Bank issues cash Card : CardCode* is a qualifier on this association.**
  - ***bankCode* is the qualifier of *Bank* with respect to *Consortium*.**

Keeping the Right Attributes



ATM Class Model with Attributes

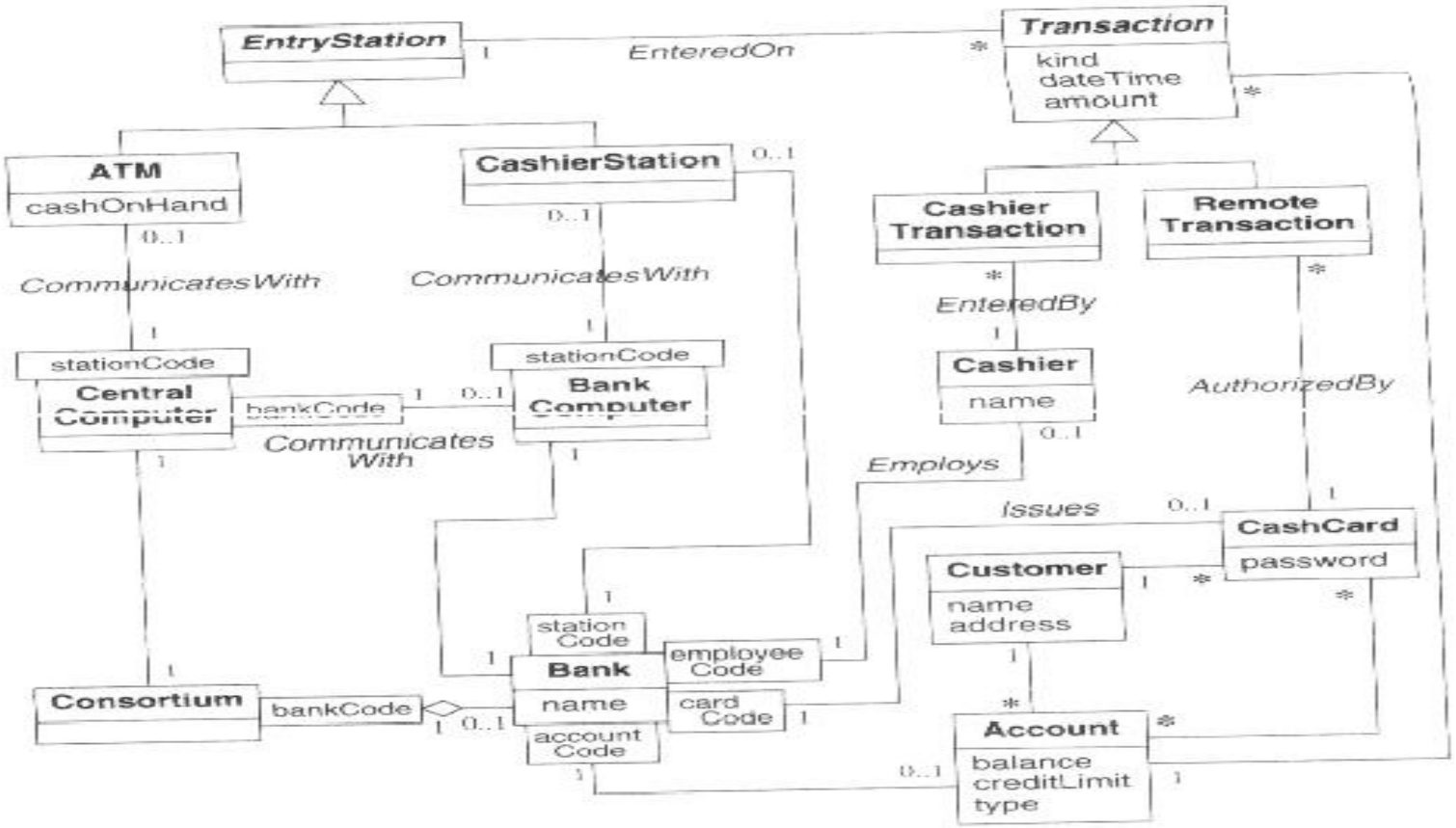
Refining with Inheritance

- The next step is to organize classes to share common structure.
- **Bottom-up generalization**
  - Search for classes with similar attributes, associations, and operations.
  - ATM Example: *RemoteTransaction* and *CashierTransaction* are similar, and can be generalized by *Transaction*.
- **Top-down specialization**
  - Top-down specializations are apparent from the application domain.
  - Look for noun phrases composed of various adjectives on the class name.
  - Example: *fixed* menu, *pop-up* menu, *sliding* menu.

Refining with Inheritance

- Similar associations – When the same association appears more than once, generalize the associated classes.
  - ATM example: *Transaction* is entered on both *Cashier Station* and *ATM*; *EntryStation* generalizes *CashierStation* and *ATM*.
- **Testing Access Paths**
  - ATM Example:
    - A cash card does not uniquely identify an account
    - The user must choose an account; but if the user supplies the account type each card can access at most one account.
    - Alternative is to require customers remember account numbers.

Refining with Inheritance

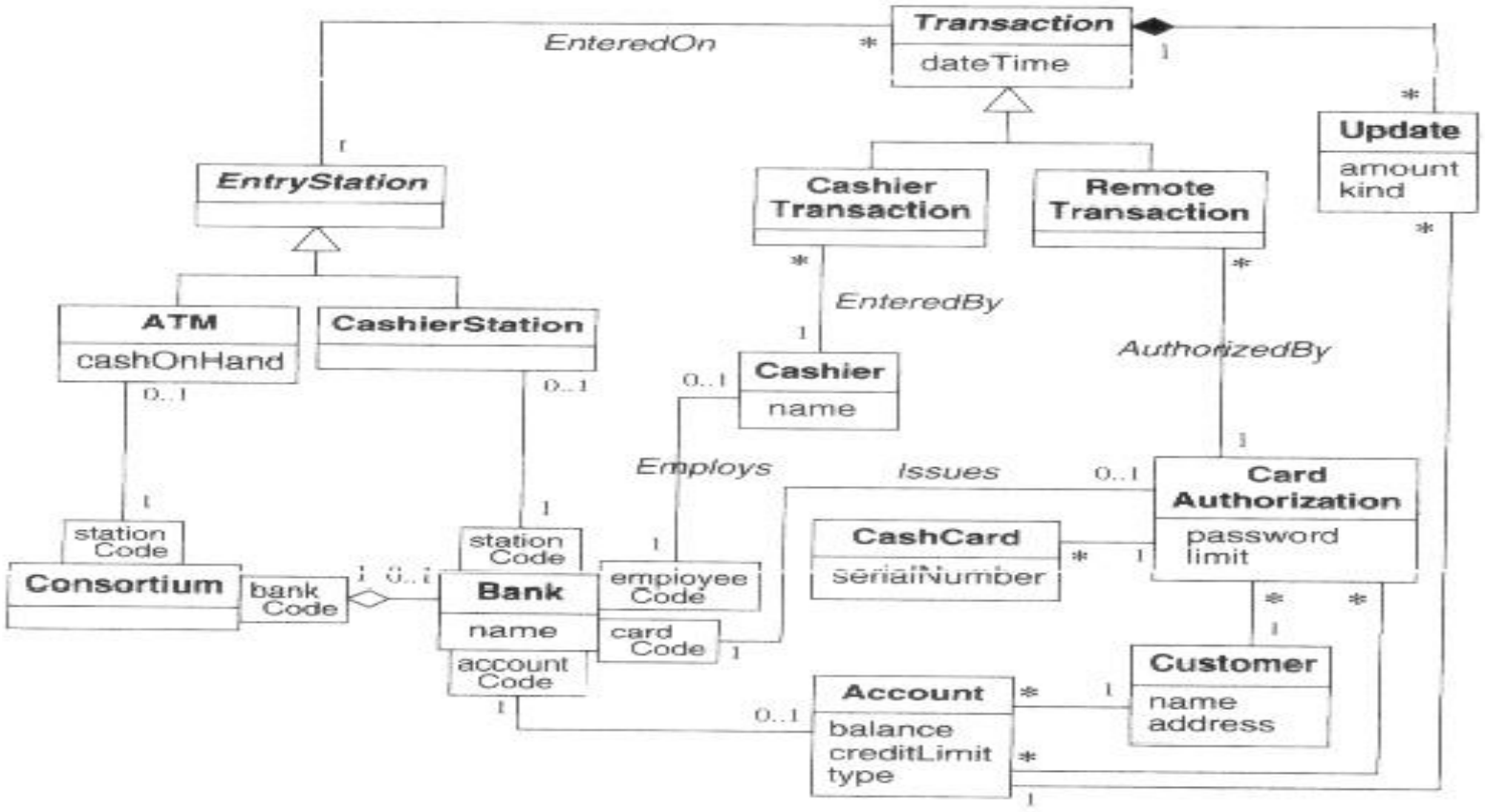


Iterating a Class Model

- A class model is rarely correct after a first pass.

Problem	Solution
Asymmetries in associations &generalizations	Add new classes by analogy
Disparate attributes	Split a class to assure coherency.
Difficulty in generalization	Split it up the class to fit in cleanly
Duplicate associations	Generalize to create superclass and unite
Missing access paths	Add new associations to answer queries

Iterating a Class Model



Grouping Classes into Packages

- The last step in class domain modeling is to group classes into packages. A package is a group of elements (classes, associations, generalizations) with a common theme.
- To assign classes to packages look for cut points.
  - Cut point is a class that is the sole connection between two otherwise disconnected parts of a model.
- ATM Example:
  - Tellers: [cashier, entry station, cashier station, ATM]
  - Accounts: [account, cash card, card authorization, customer, transaction, cashier transaction, remote transaction.
  - Banks: [consortium, bank]

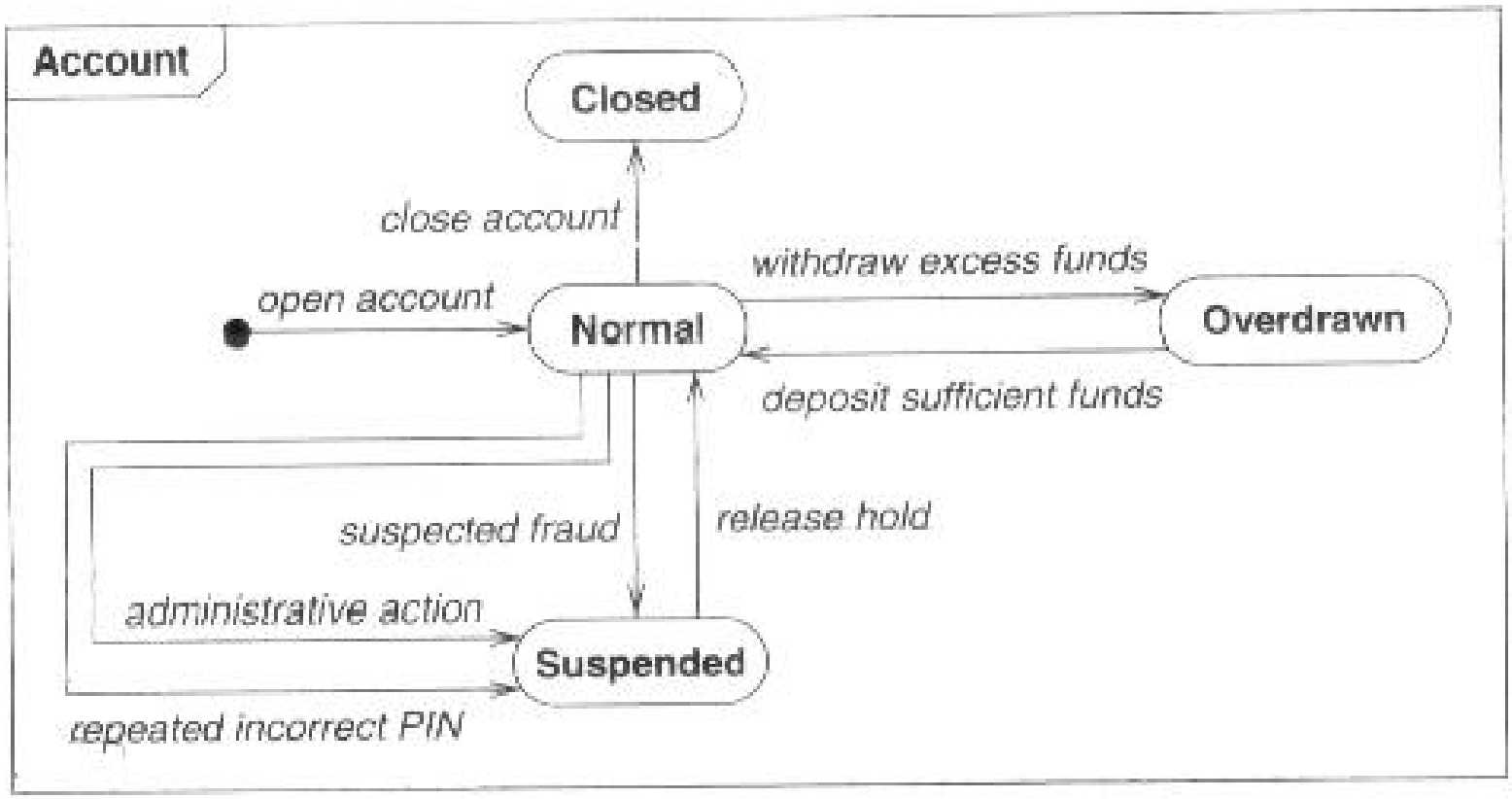
Domain State Model

- Identify domain classes with state-dependent behavior.
- Find states
- Find events
- Build state diagrams
- Evaluate state diagrams

Identifying Classes with States

- Look for classes that can be characterized by a progressive history or that exhibit cyclic behavior.
- Identify the life cycle of an object.
- ATM Example:
  - *Account* is an important business concept.
- List the states for each class.
  - Account states: *Normal, Closed, Suspended, Overdrawn*
- List the events
  - Account events: *close account, withdraw excess funds, repeated incorrect PIN, suspected fraud,* and *administrative action.*

Account Class - State Chart



Domain Interaction Model

- Interaction model is seldom important for domain analysis.
- During domain analysis the emphasis is on key concepts and structural relations.
- Interaction model is important aspect of application analysis.