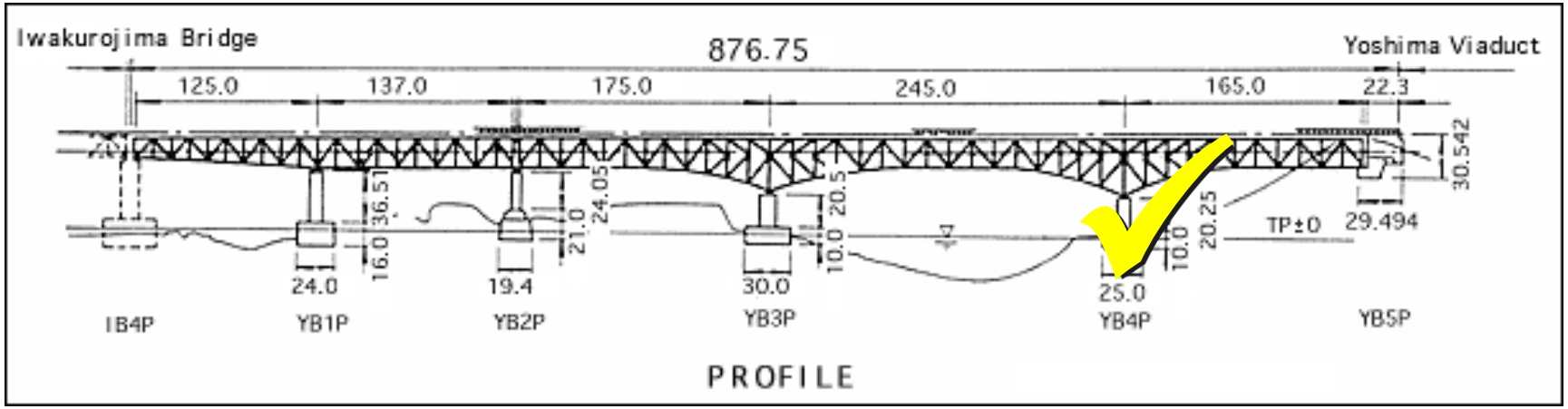
Software Modeling and Unified Modeling Language

What is modeling?

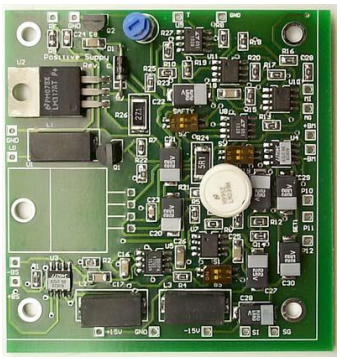
- Before they build the real thing...

…they first build models..and then learn from them

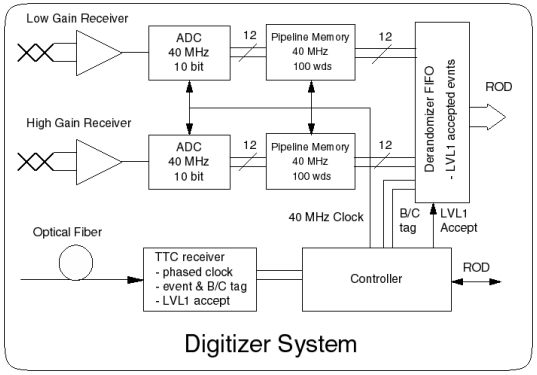


Engineering Models

- Engineering model:
  A reduced representation of some system



Modeled system



Model

Purpose:

- ***To help us understand a complex problem or solution***
- ***To communicate ideas about a problem or solution***
- ***To drive implementation***

What is modeling?

- Modeling consists of building an abstraction of reality.
- Abstractions are simplifications because:
  - They ignore irrelevant details and
  - They only represent the relevant details.
- What is *relevant* or *irrelevant* depends on the purpose of the model.

Why do we *model*?

- It is cheaper to mess up a large project model than it is to mess up the actual product.
- To comprehend complex systems
  - In their entirety
- Models allow big issues to be caught early.
  - Really big issues, the costly ones.
    - Are lives at stake?
    - Recurring maintenance costs after product delivery
      - Are many times the big cost of a project
      - Support
      - Bug fixing
      - Downtime

Why model software?

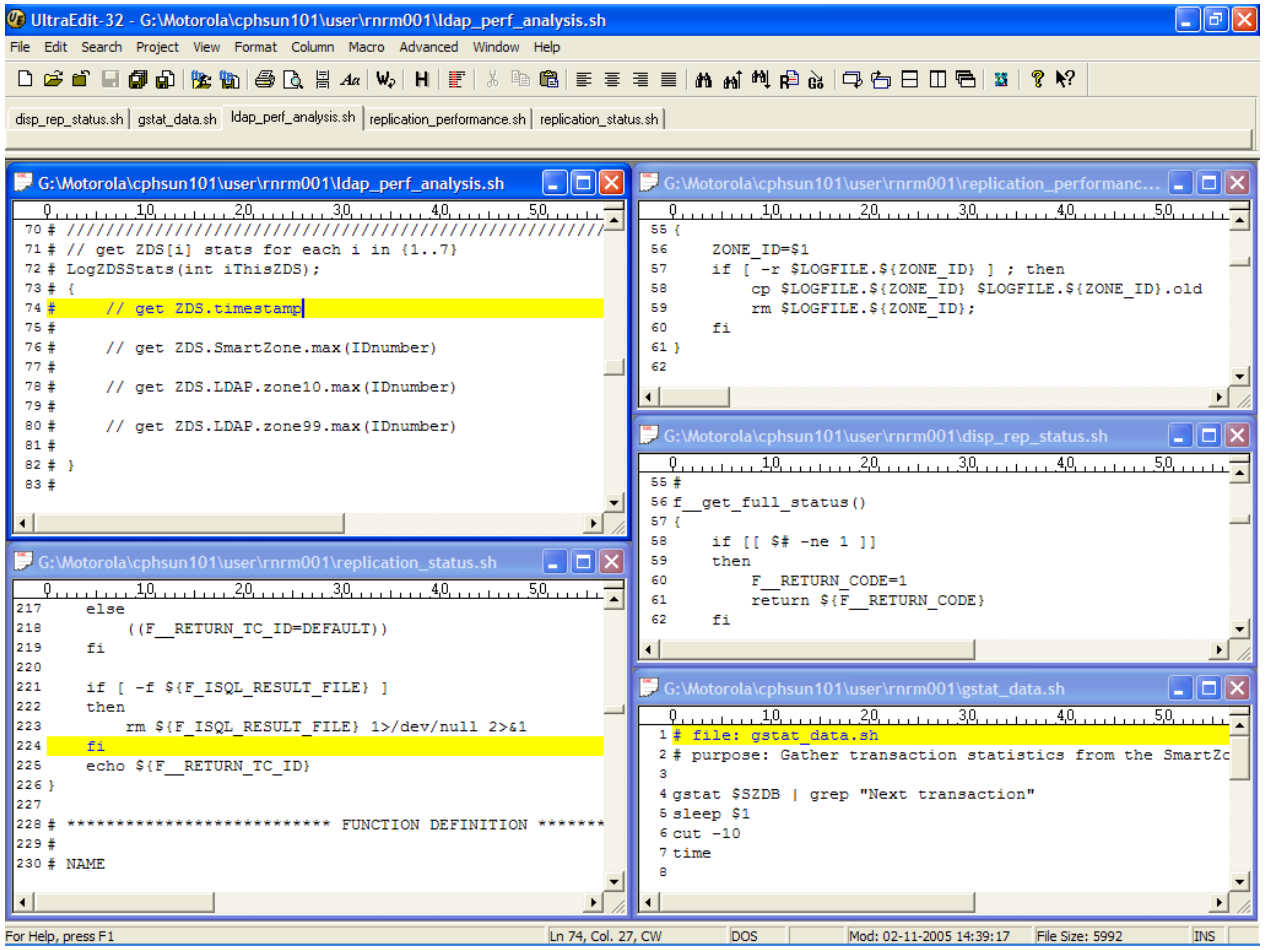
Why model software?

- Software is getting increasingly more complex
  - Windows XP > 40m lines of code
  - A single programmer cannot manage this amount of code in its entirety.
- Code is not easily understandable by developers who did not write it
- We need simpler representations for complex systems
  - Modeling is a mean for dealing with complexity

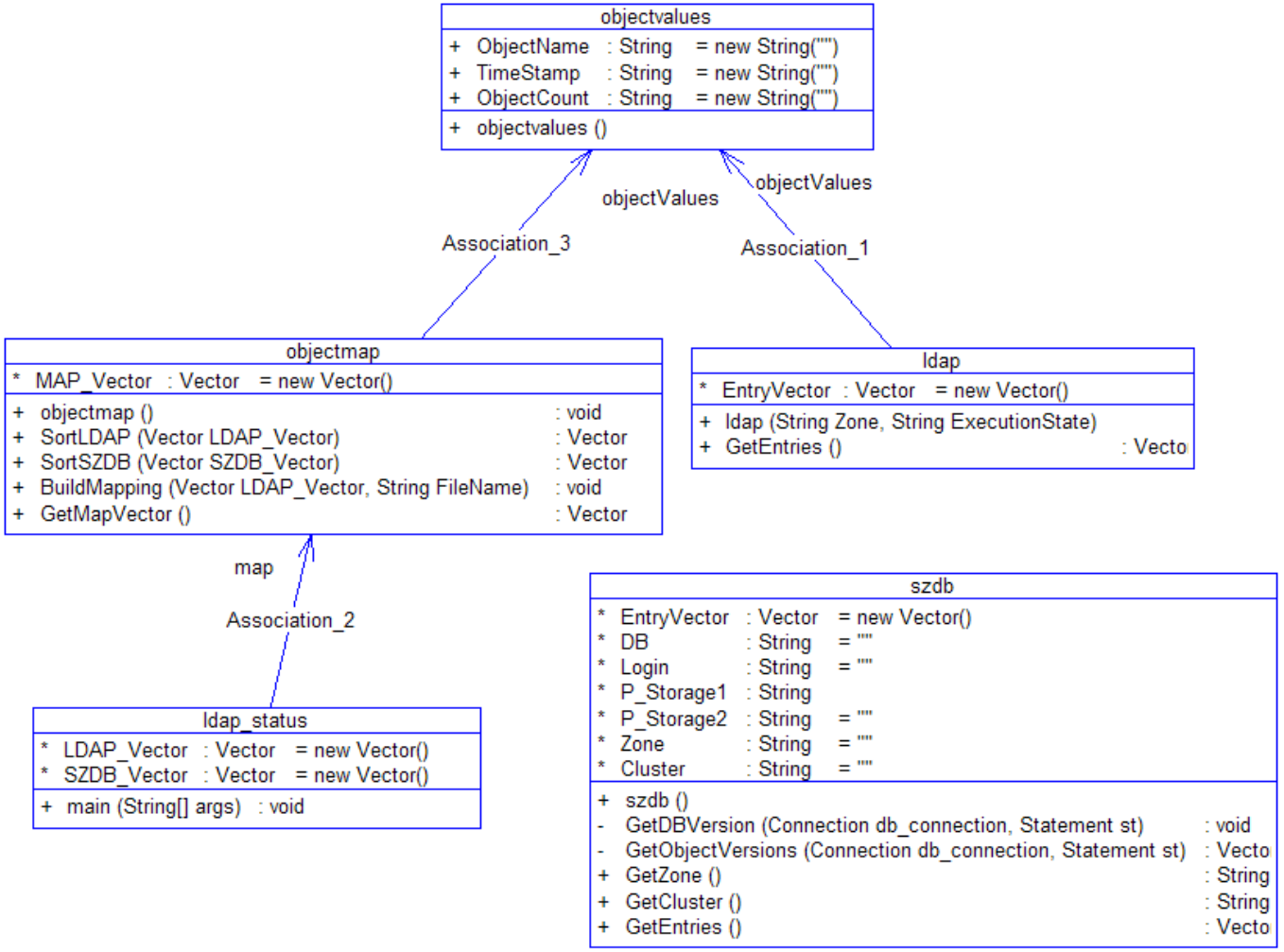
What is Visual Modeling?

- A visual model presents 1+ graphical displays of the Model
  - Uses a *standard* set of graphical elements.
- A visual model provides better communication between project team members
  - Analysts
  - Developers
  - Managers
  - Testers
  - Users
- Visual creatures can better comprehend *conceptual complexity*
  - When it is presented visually as opposed to
    - textual presentation
    - orally spoken

Textual Modelling



Visual Modeling



Characteristics of Useful Models

- Abstract
  - Emphasize important aspects while removing irrelevant ones
- Understandable
  - Expressed in a form that is readily understood by observers
- Accurate
  - Faithfully represents the modeled system
- Predictive
  - Can be used to derive correct conclusions about the modeled system
- Inexpensive
  - Much cheaper to construct and study than the modeled system

To be useful, engineering models must have <u>all</u> of these characteristics!

How Models are Used

- To detect errors and omissions in designs before committing full resources to full implementation
  - Through (formal) analysis and experimentation
  - Investigate and compare alternative solutions
  - Minimize engineering risk
- To communicate with stakeholders
  - Clients, users, implementers, testers, documenters, etc.
- To drive implementation

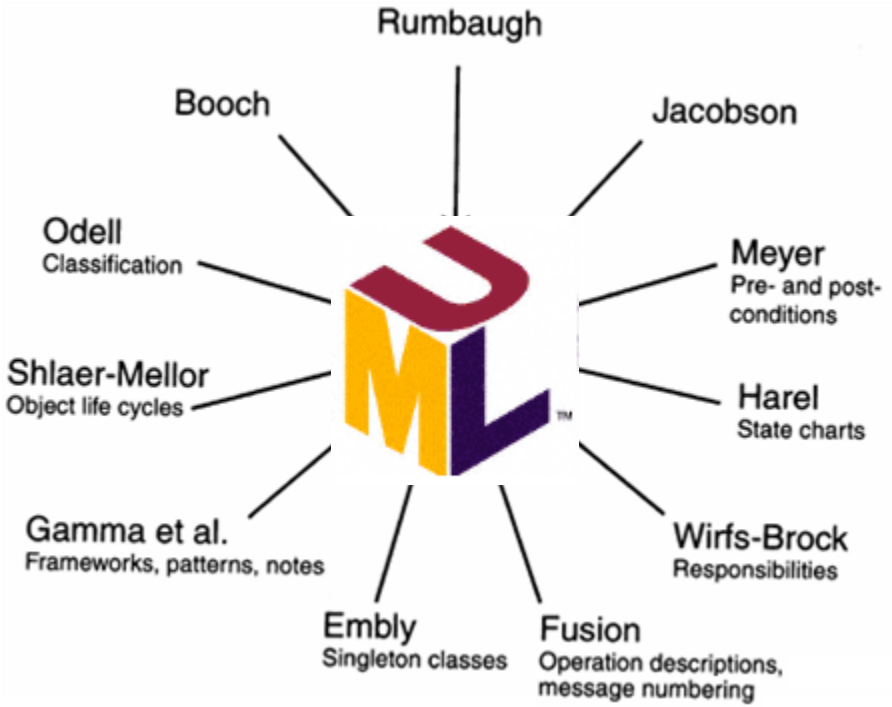
What is UML?

- UML is a product
  - From *Object Management Group, Inc.* http://omg.org/
  - Published as a standard, therefore it's ”Open Source”
- UML is the standard modeling language widely used in industry

Language = Syntax * Semantics

- Syntax ➔ the symbols of the language (words, pictures)
- Semantics ➔ the meanings of the symbols
- ”Language” ➔
  - Allows a tool to have *model checking ability*
  - Ensures a certain level of correctness

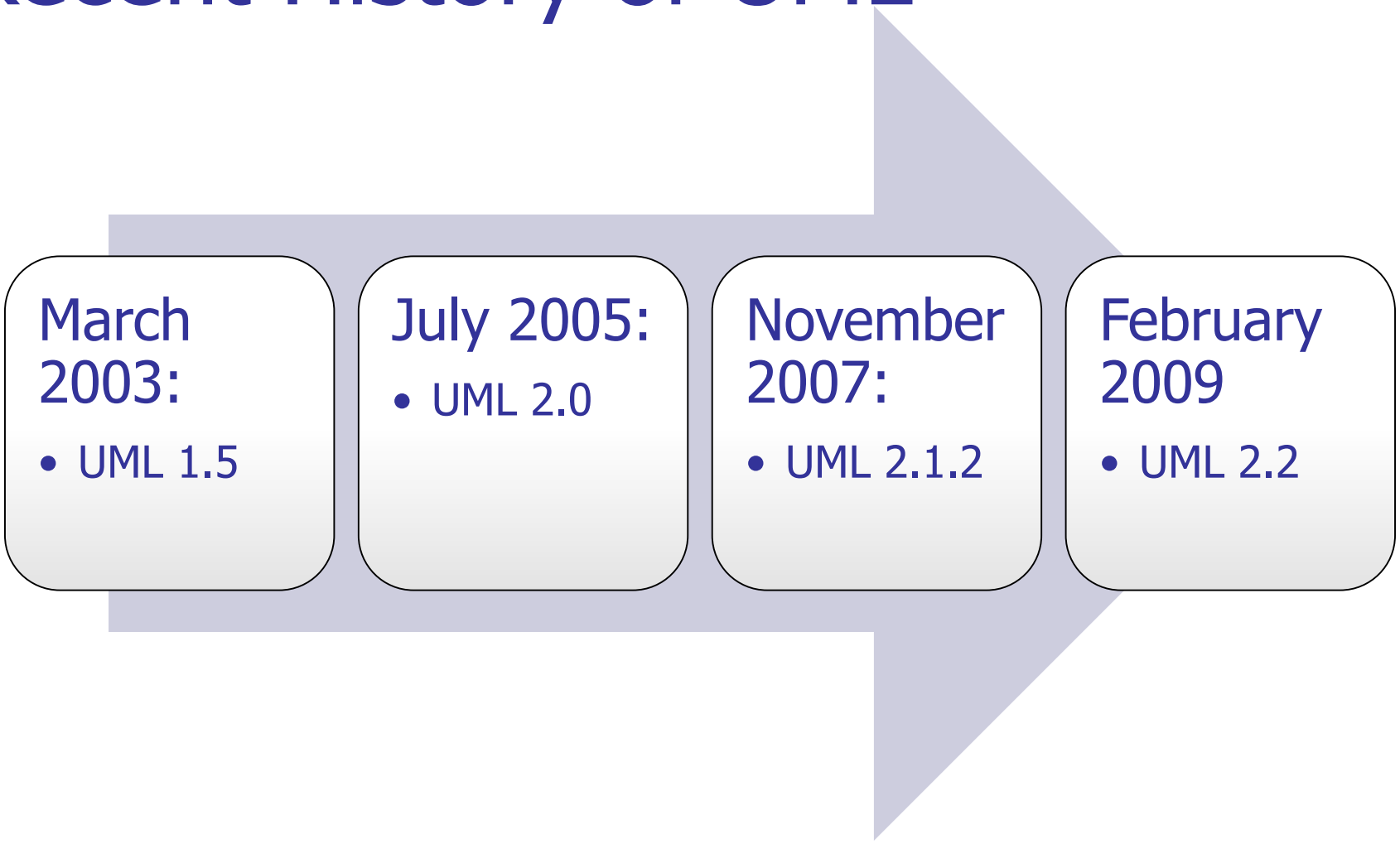
The Method Wars



UML Today

- UML notation won the Method Wars
  - An alliance of popular technologies
- UML is published and approved by the ISO as an international standard
  - ISO/IEC 19501 (UML v1.4.2)
- Current publicly available version is v2.2
- Benefits from standardization
  - Common interoperability
  - Collaboration among competitors!

Recent History of UML



UML Version Differences

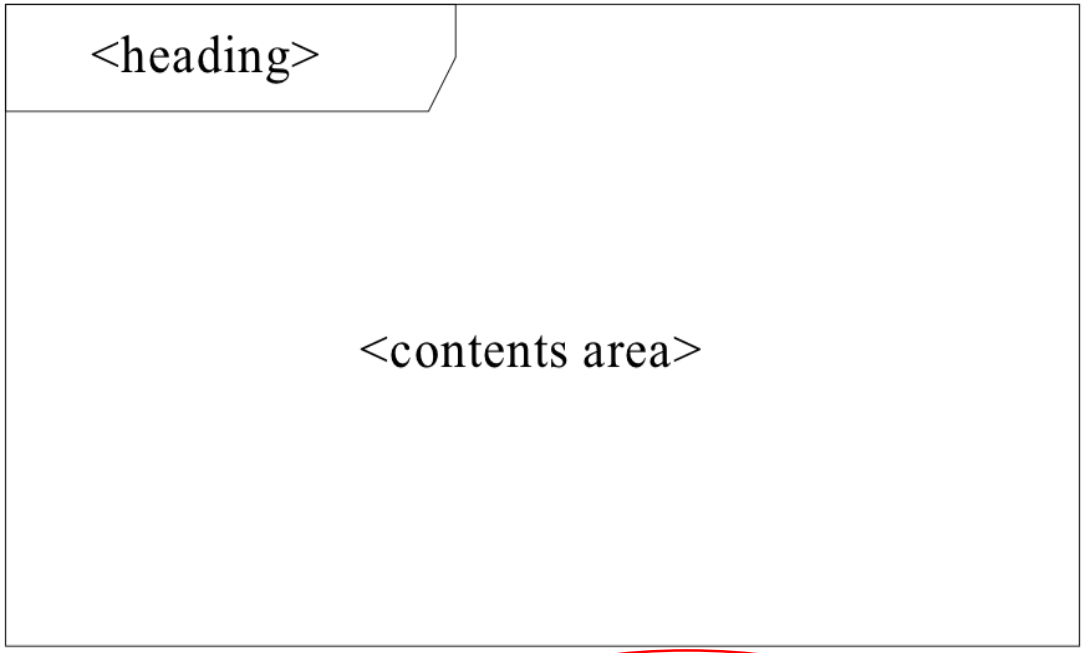
- In UML v2
  - Messages can be lost, and can be found.
  - Sequential diagrams can capture conditional and looping logic.
  - Discrete value lists used for multiplicities has been eliminated
    - {1,3..5,7,13}

Changes in UML 2

- Frames and nesting
- Changes in diagram notation:
  - Activity diagram
  - Deployment diagram
  - Sequence diagram
- Profiles and stereotypes
- New diagram types:
  - Composite structure diagrams
  - Timing diagrams

Notation for all UML 2 Diagrams

- Mandatory is now the **contents area**
- Optional: The contents area can be surrounded with **frame** and a **heading**



Frame

<heading> ::= [<diagram kind>]<name>[<parameters>]

Diagram Kinds

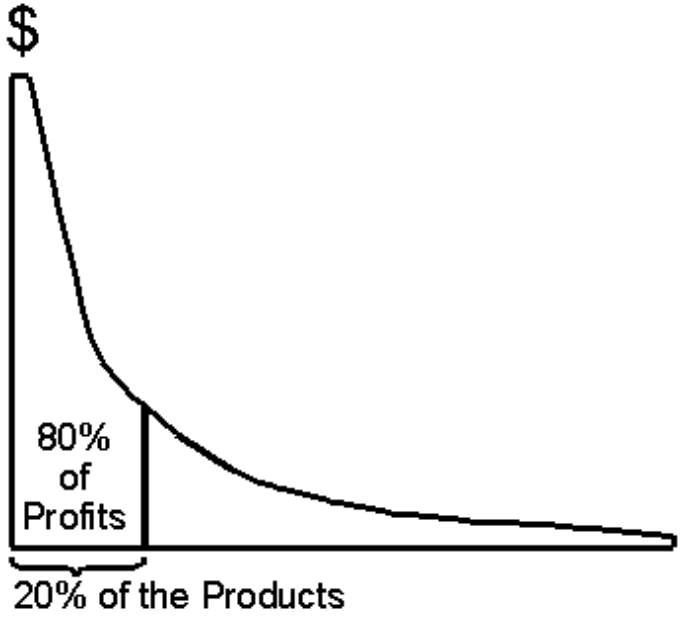
Activity diagram
Class diagram
Component diagram
Interaction diagram
Package
state machine diagram
Use case diagram

- The following abbreviations are usually used for diagram frames:
- **act** (for activity frames)
- **cmp** (for component frames)
- **sd** (for interaction frames)
- **pkg** (for package frames)
- **stm** (for state machine frames)
- **uc** (for use case frames)

Why are interaction frames abbreviated with “sd”?
Historical: sequence diagram

UML: 80-20 rule

- You can solve 80% of the modeling problems by using 20 %
- We teach you those 20%
- 80-20 rule: Pareto princi

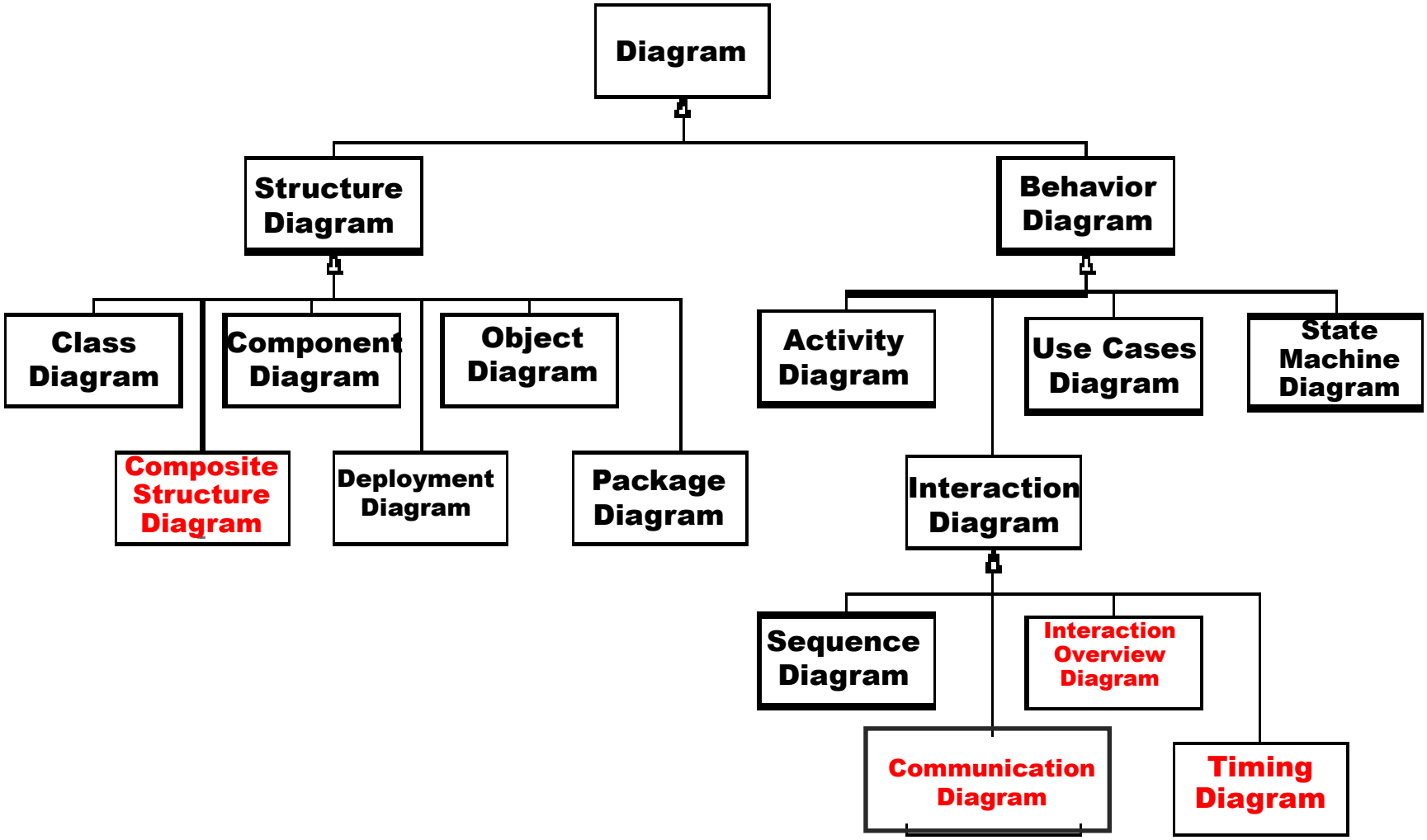


Vilfredo Pareto, 1848-1923
Introduced the concept of Pareto Efficiency,
Founder of the field of microeconomics.

UML Models are composed of UML Diagrams

- How is a model represented?
  - By using one or more diagrams
    - or other models.
- Models are composed of one or more *UML diagrams*.
- UML diagrams
  - Present a particular view of a model
    - Through a picture
  - Express an unambiguous *concept*
    - unambiguous
  - Can each be in a different development state
    - Depends upon the amount of *time* invested into the model.

UML 2.0 Diagram types



UML Diagram Classifications

- Static Diagrams (structure)
  - Class Diagram
  - Package Diagram
  - Component Diagram
  - Structure Diagram
  - Deployment Diagram
- Dynamic Diagrams (behavior)
  - Use Case Diagram
  - Interaction Diagram
  - Communication Diagram (also Collaboration Diagram in UMLv1.4)
  - Statechart Diagram
  - Activity Diagram

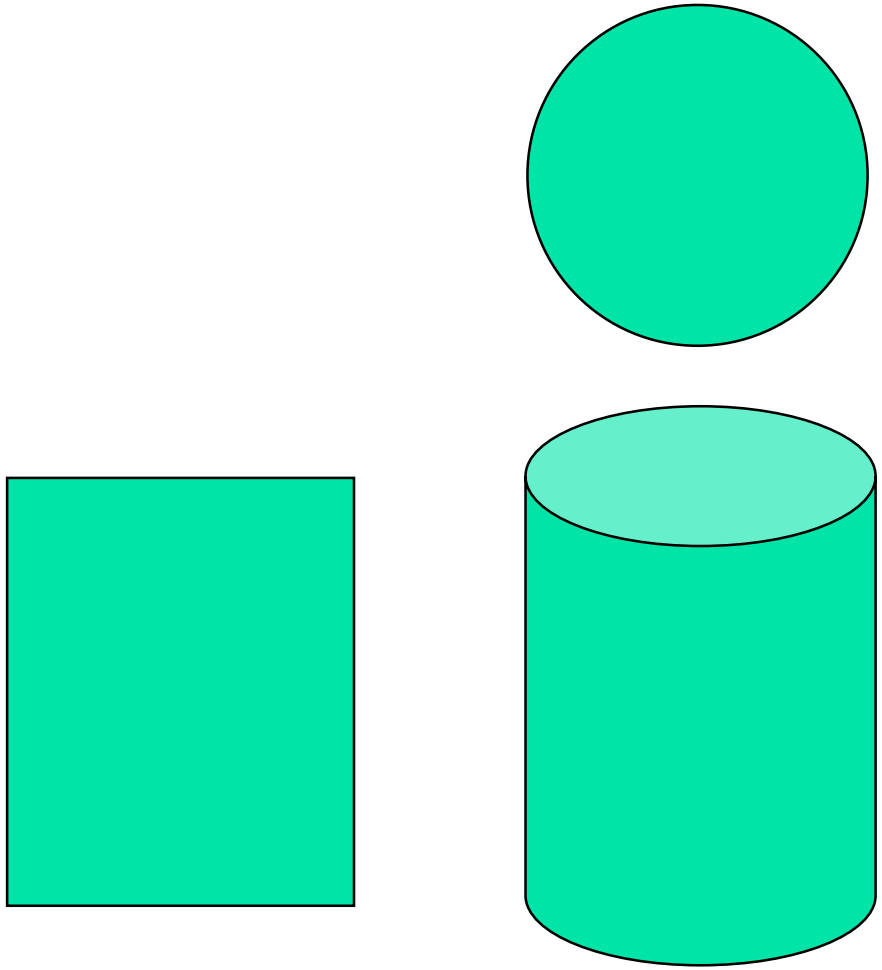
Why so many diagrams?

- A system typically has many different stakeholders
- Your objective is to communicate *clearly* with every type of stakeholder.
- Each stakeholder has his own particular interests
  - ...in specific aspects of the same system
  - ...in specific aspects of the same diagram!
- Each diagram is a different ”view” of the same model.
- Conscientious system design involves all possible viewpoints
- The more diagrams you can provide
  - the more clear the system becomes
  - the more you will be able to better defend resulting conclusions

Different Stakeholders

- Customers
- Domain Experts
- Business Analyst
- Designers
- Marketing Team
- Sales Team
- Product manager
- Programmers
- Database Administrators
- ...etc.

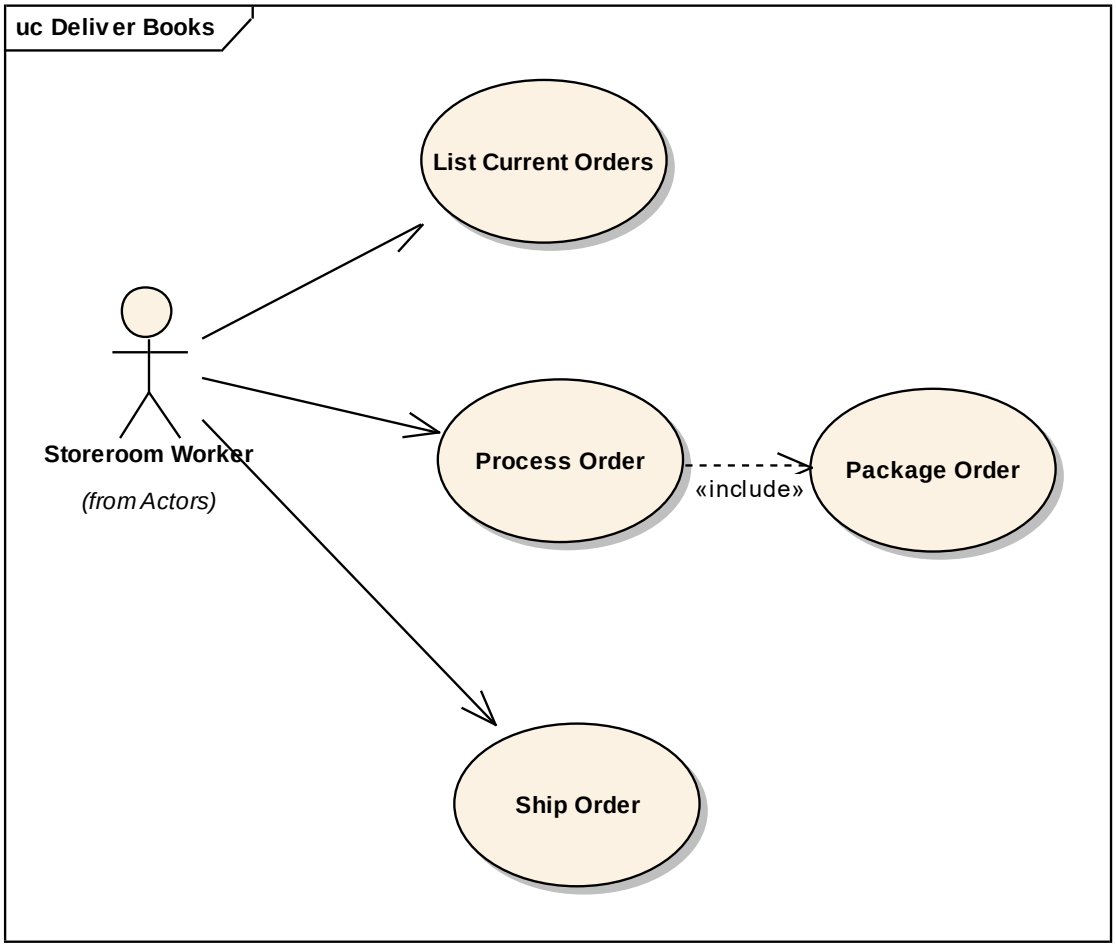
Get two views of the same object!



UML First Pass

- Use case Diagrams
  - Describe the functional behavior of the system as seen by the user.
- Class diagrams
  - Describe the static structure of the system: Objects, Attributes, Associations
- Sequence diagrams
  - Describe the dynamic behavior between actors and the system and between objects of the system
- Statechart diagrams
  - Describe the dynamic behavior of an individual object (essentially a finite state automaton)
- Activity Diagrams
  - Model the dynamic behavior of a system, in particular the workflow (essentially a flowchart)
- Deployment Diagrams
- Requirement Diagrams

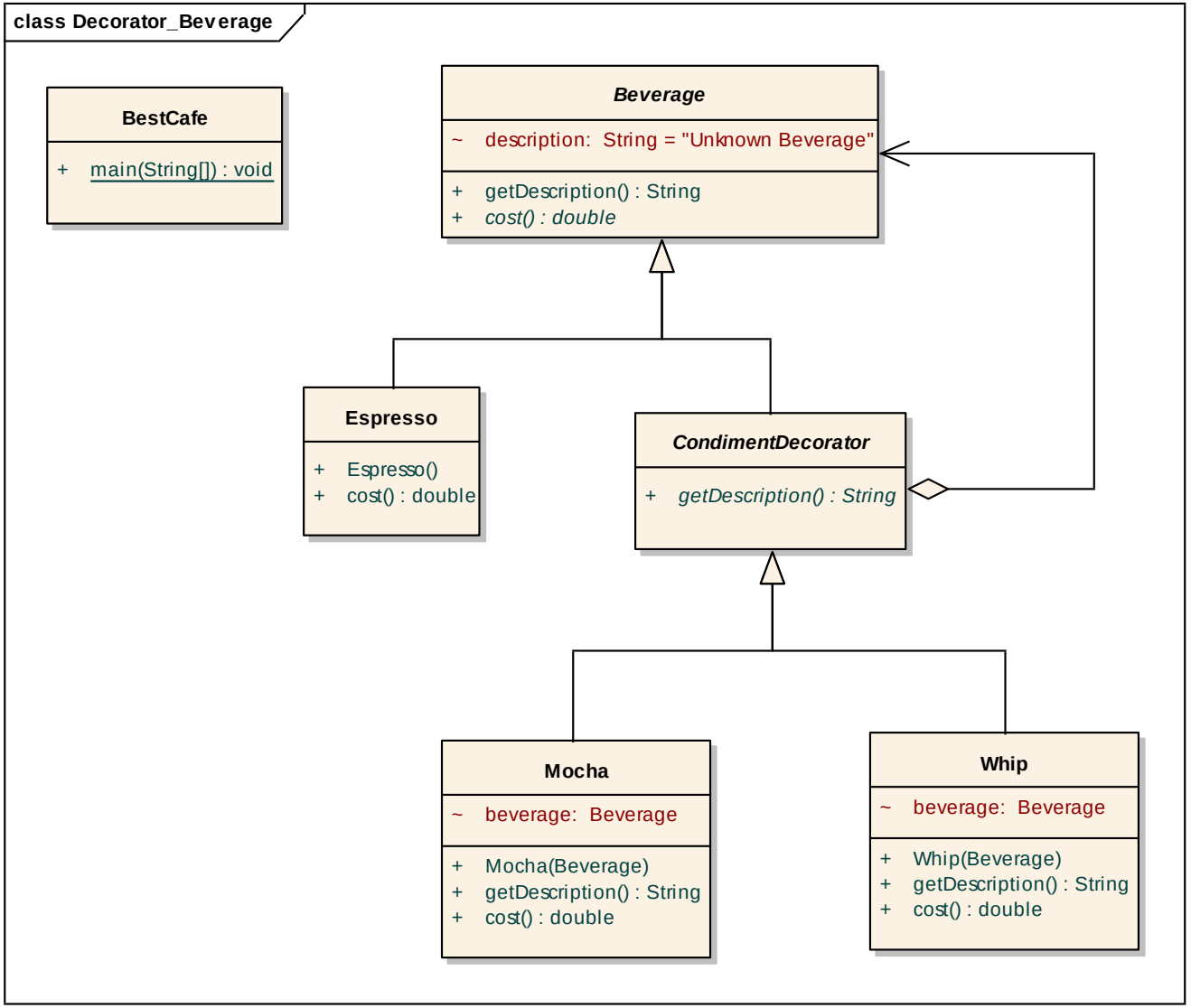
UML first pass: Use case diagrams



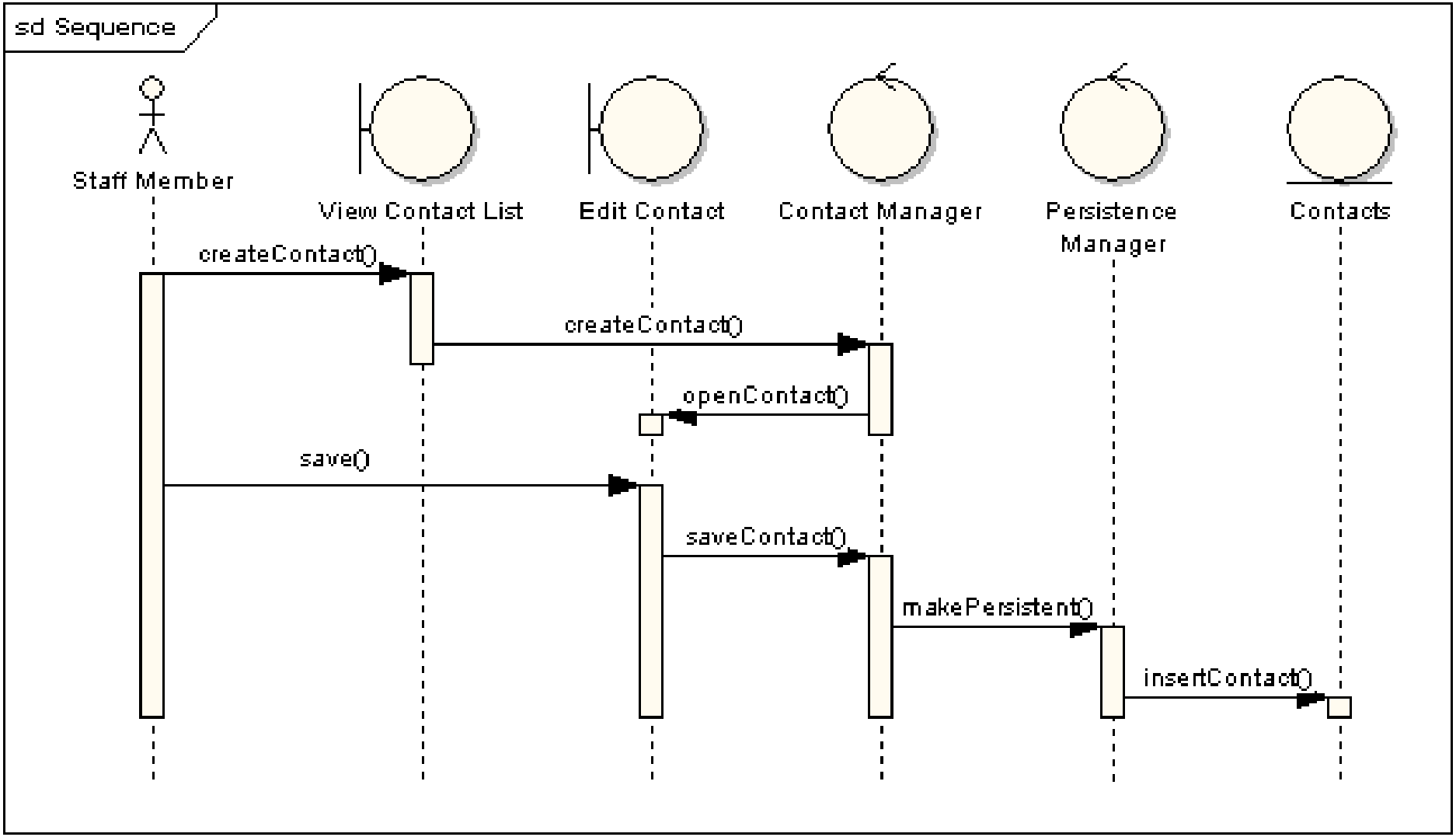
Use case diagrams represent the functionality of the system from user's point of view

UML first pass: Class diagrams

Class diagrams represent the structure of the system

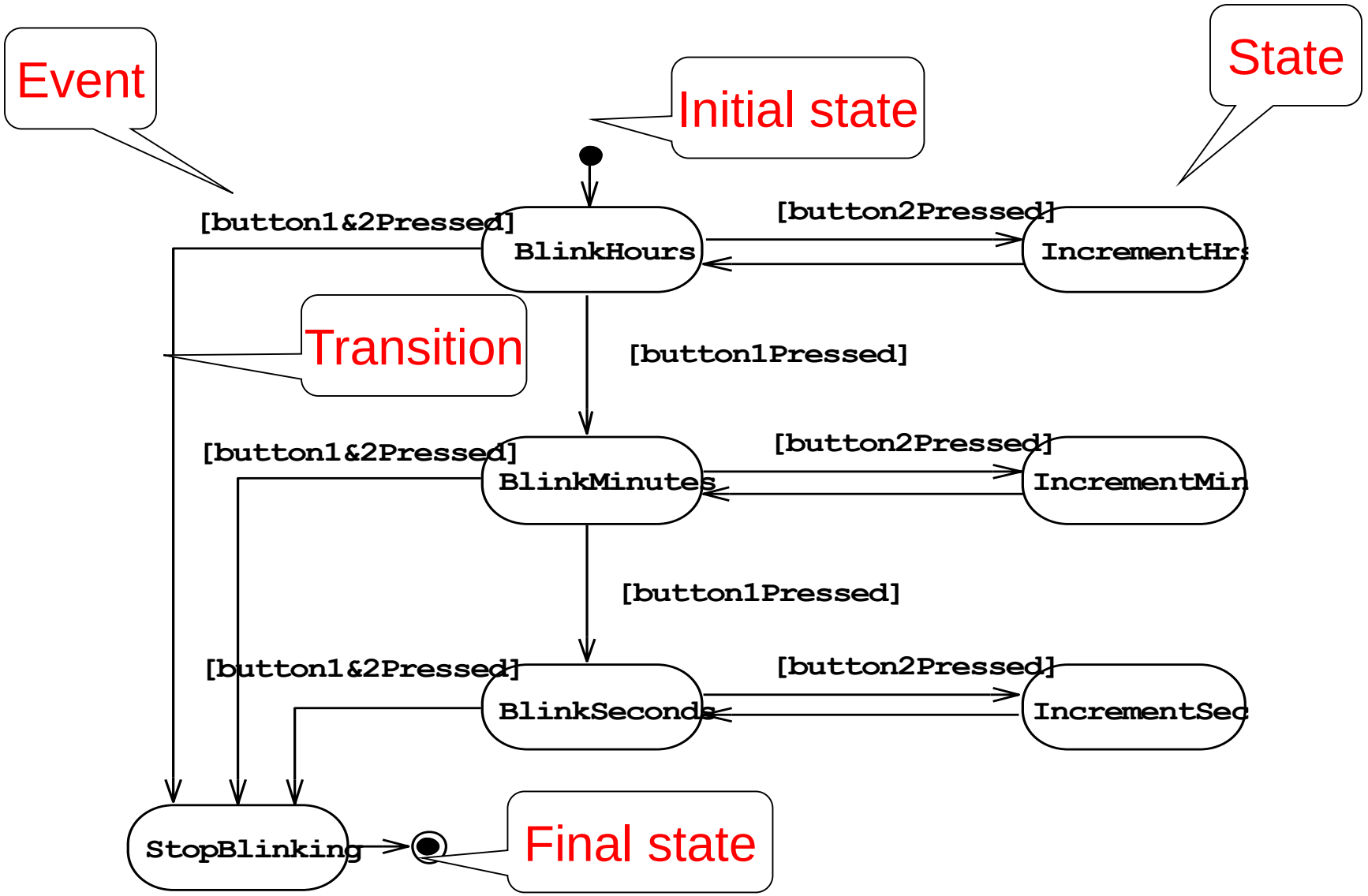


UML first pass: Sequence diagram



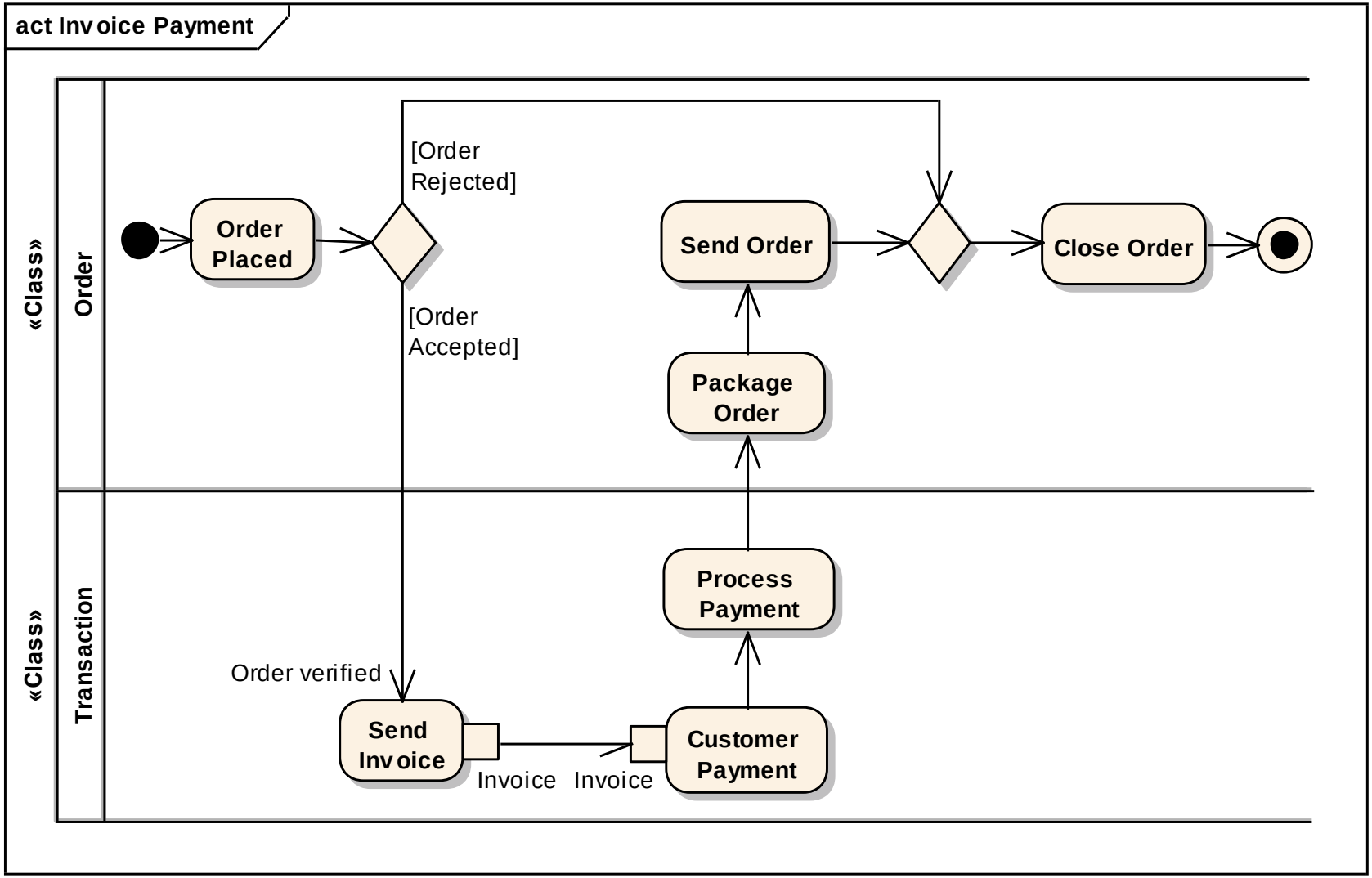
Sequence diagrams represent the behavior as interactions

UML first pass: Statechart diagrams

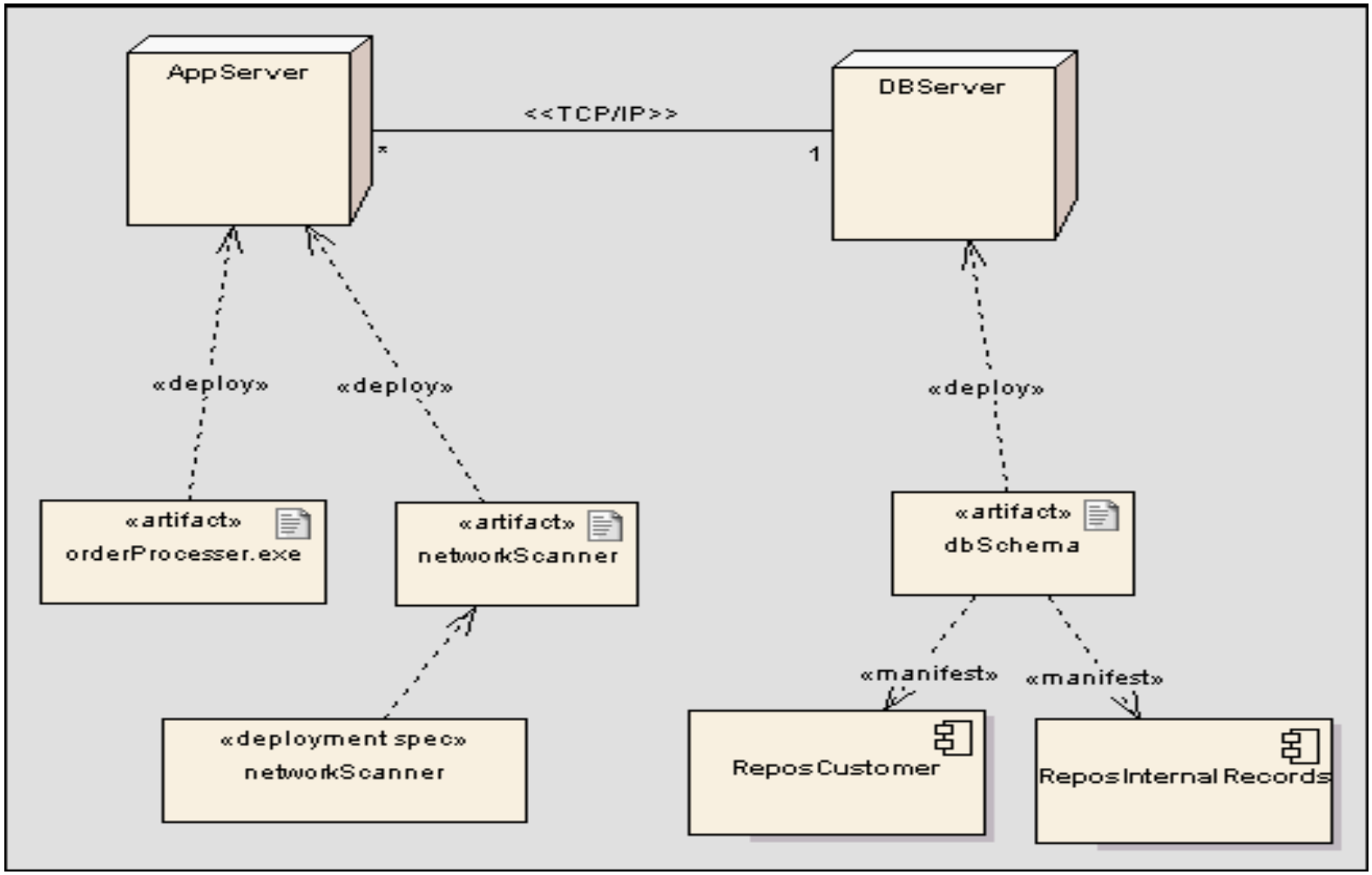


Represent behavior as states and transitions

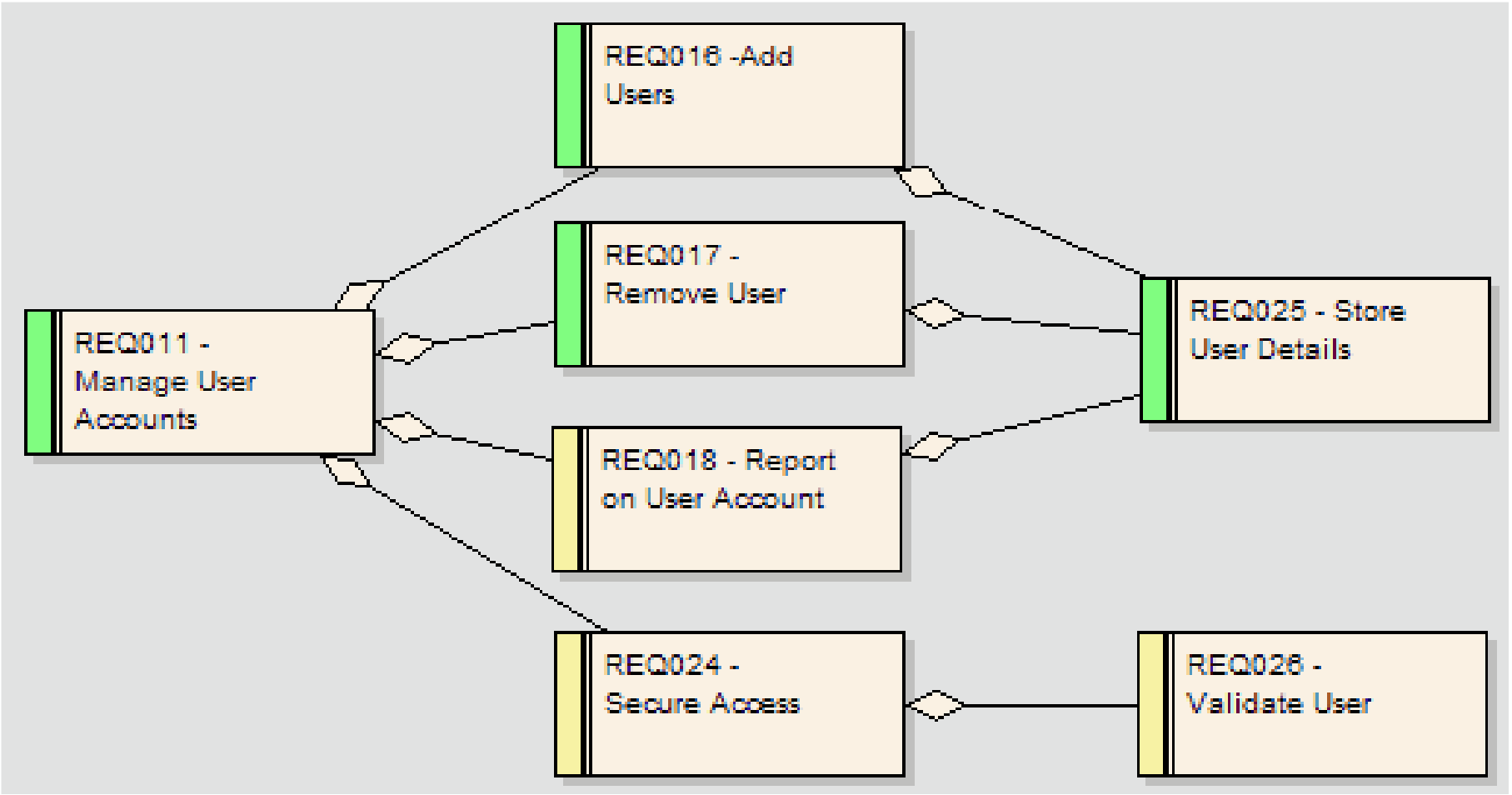
Activity Diagram



Deployment Diagrams:



Requirement Diagrams:



UML Tools

- PowerPoint
- Visio
- Rational Rose
- Eclipse
- Telelogic Tau
- Enterprise Architect

PowerPoint

- Great communication tool, but primarily ”output only”
- Can benefit from Visual Basic scripting?
- Does not enforce any UML standards

Visio

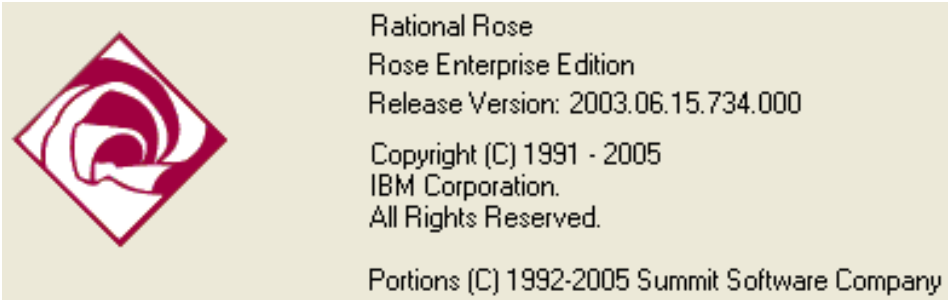
- Similar to, but more ”precise” than PowerPoint
- Standard Edition
  - Can use stencil
    - Available from http://www.phruby.com/stencildownload.html
    - Free, but does not enforce any UML standards
- Professional Edition
  - Handles UML natively
- Output only
- I was disappointed when I couldn’t export the model

Rational Rose

- Rational's flagship product
- Getting old; Last release version dated 2003
- Supports subset of UML v1.4.2
- Supports XMI
- Generates code from the model: C++, Java, Ada

What is Rational Rose?

- Generates code
  - Ada 83
  - Ada 95
  - C++ (ANSI, Visual)
  - CORBA
  - Java
  - COM
  - Visual Basic
  - XML
- Supports databases
  - Generation
  - Reverse engineering

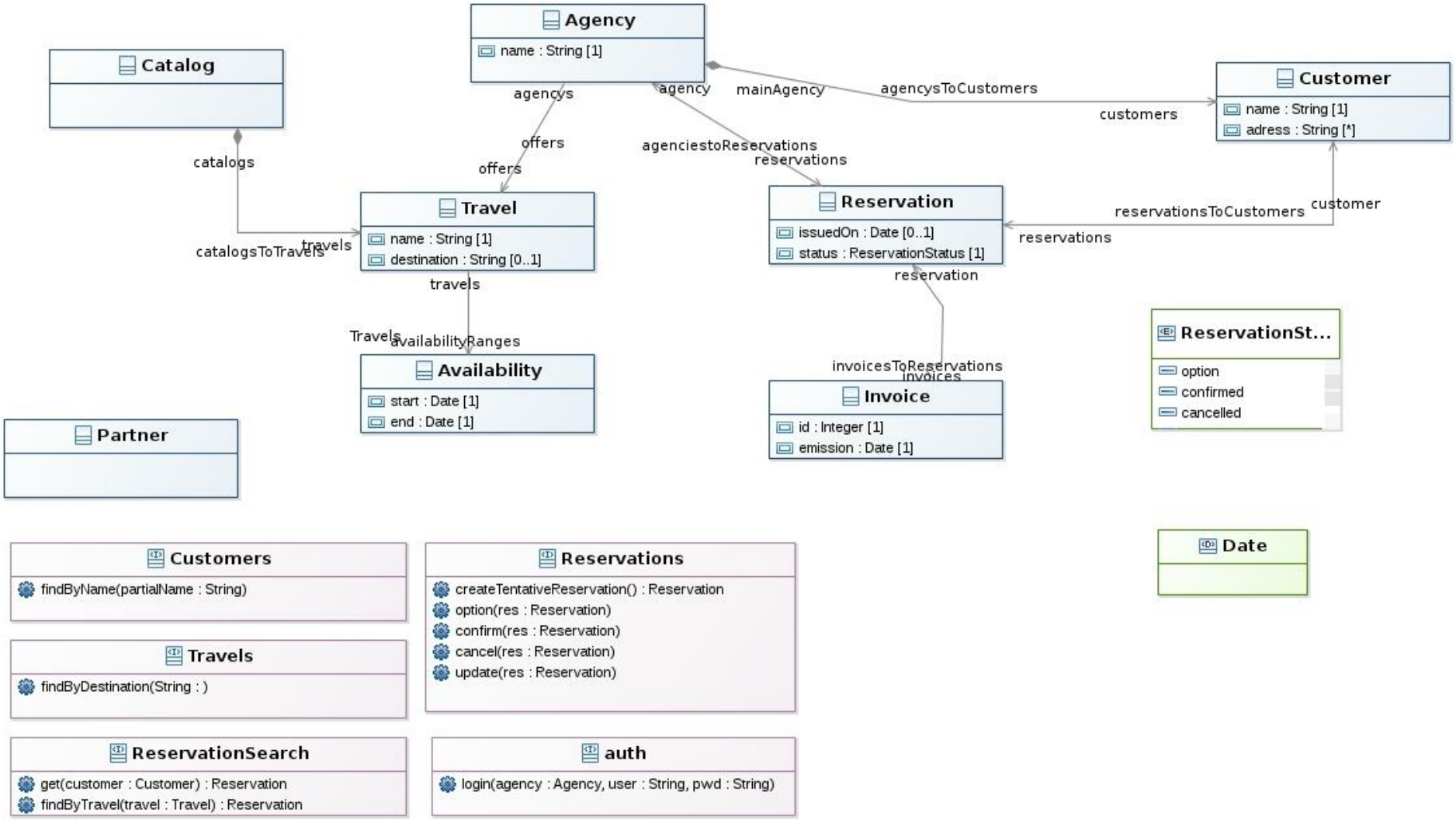


♦ Rational Rose ”supports” UML v1.4

Eclipse

- http://www.eclipse.org/
- IDE using the plug-in concept (UML Designer)
- Open Source
- IBM's recommended path from Rational Rose

Eclipse – UML Designer example



Requirements and Use Case Modeling

What is a requirement?

- The requirements are the descriptions of the system services and constraints that are generated during the requirements engineering process.

- It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

Types of requirement

- User requirements
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.

- System requirements
  - A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

User and System Requirement

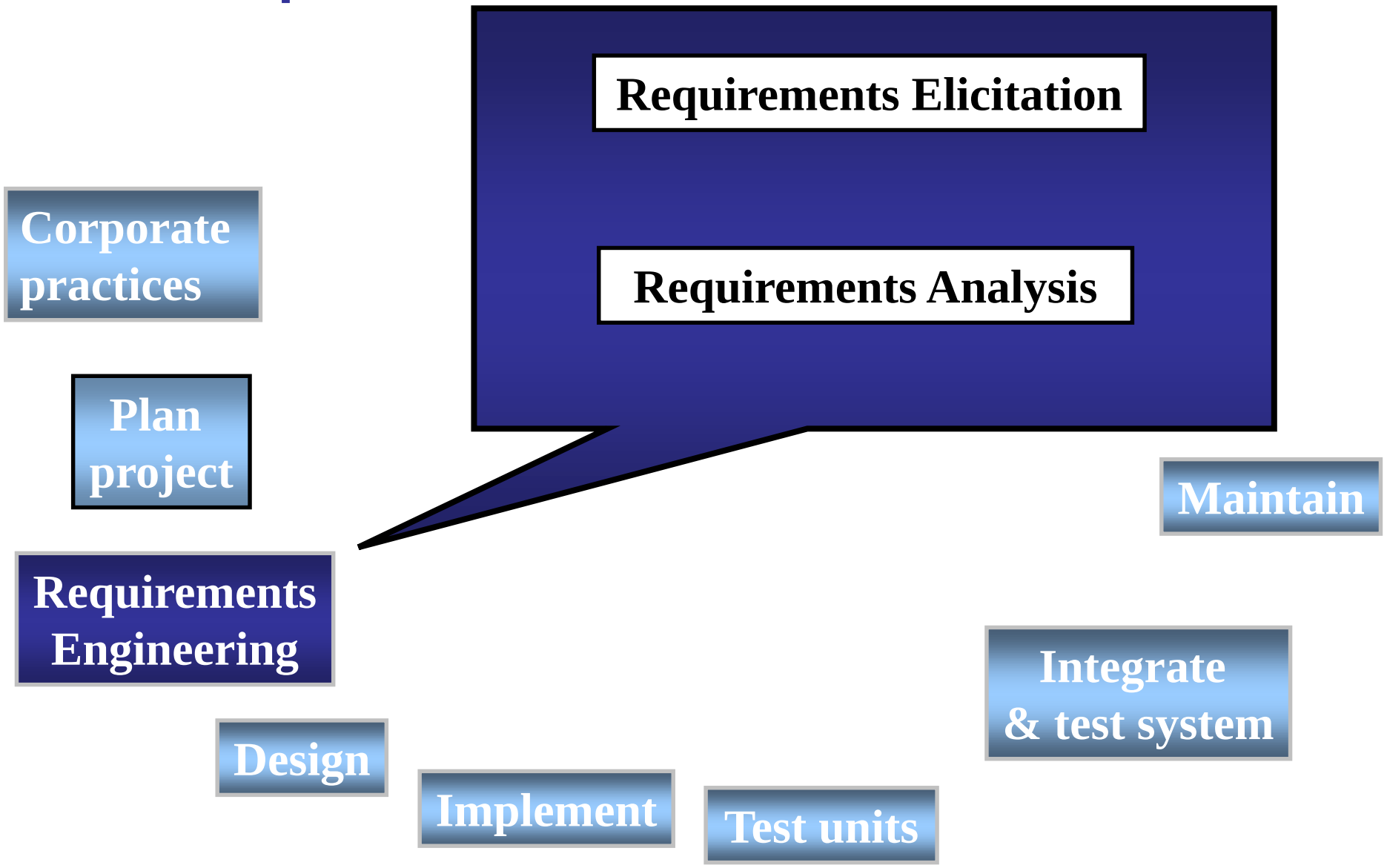
- User requirement
  - 1. The 'system' shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.
- System Requirement
  - 1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
  - 1.2 The system shall automatically generate the report for printing after 5:30pm on the last working day of the month.
  - 1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
  - 1.4 If drugs are available in different dose units (e.g. 10mg, 20mg, etc), separate reports shall be created for each dose unit.
  - 1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.
  - …

Requirements Engineering

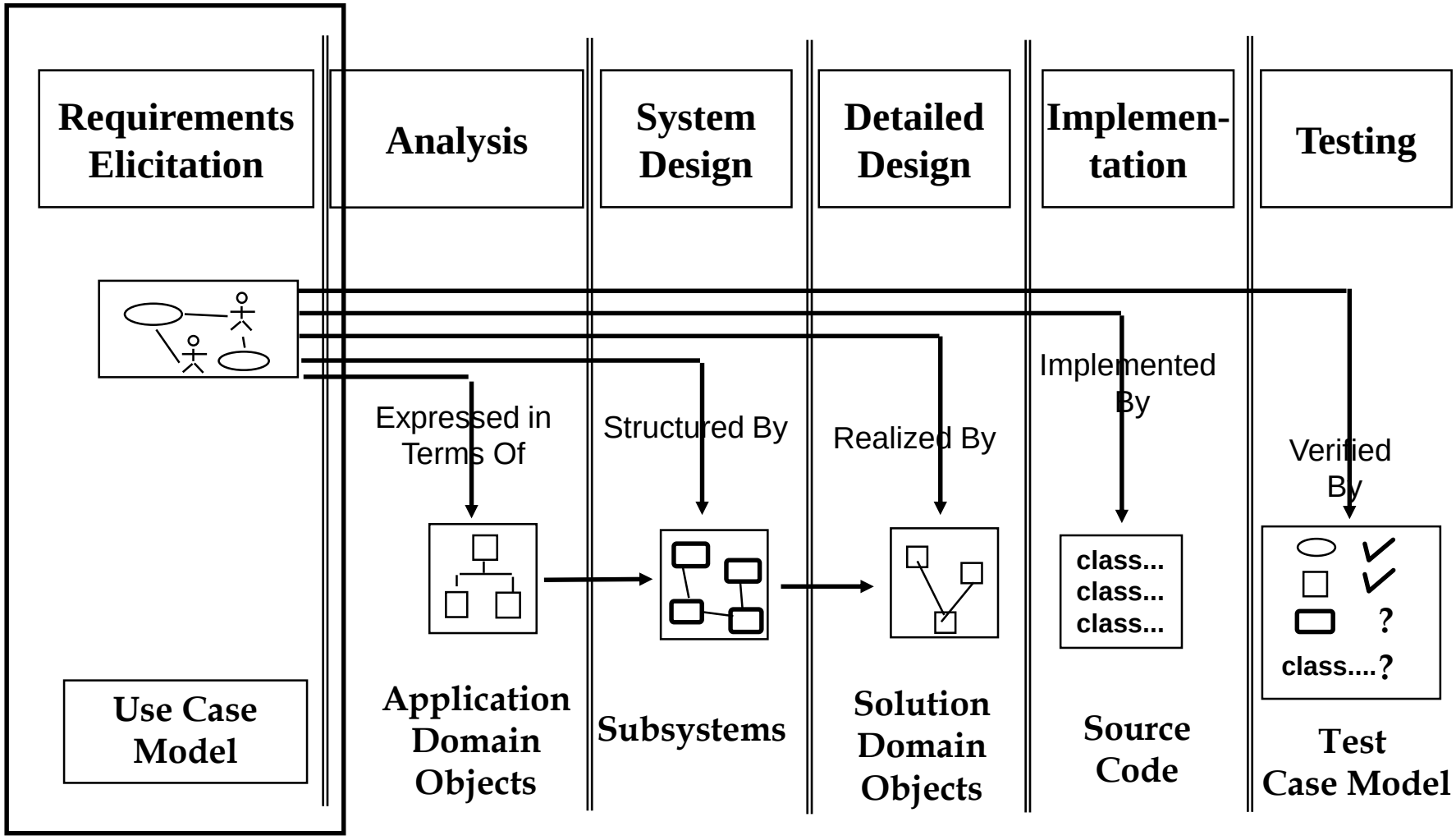
- assist software engineering better understand the problem they will work to solve
  - understand business impacts
  - understand what customer wants
  - understand end-user interaction with system
  - understand project scope for preparing project plan

- has been probably the most neglected activity of software engineering
  - considerable research and empirical studies has been done of late, introducing effective approaches and schemes toward bettering the requirements engineering activity

Requirements Phase



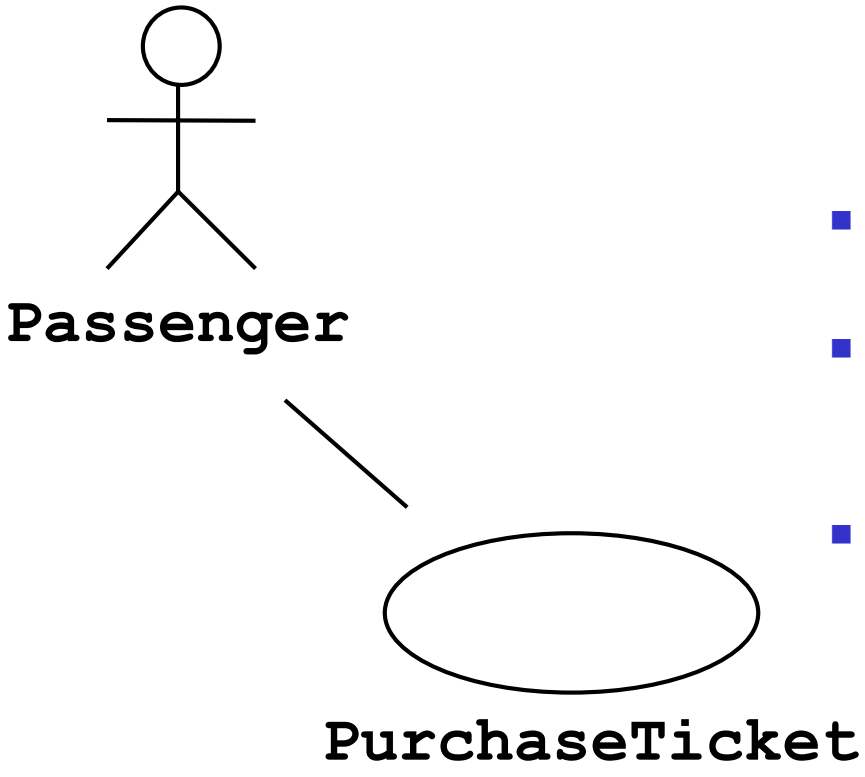
Software Lifecycle Activities



Requirements Elicitation

- Very challenging activity
- Requires collaboration of people with different backgrounds
  - Users with application domain knowledge
  - Developer with solution domain knowledge (design knowledge, implementation knowledge)
- Bridging the gap between user and developer:
  - ***Scenarios:*** Example of the use of the system in terms of a series of interactions with between the user and the system
  - ***Use cases:*** Abstraction that describes a class of scenarios

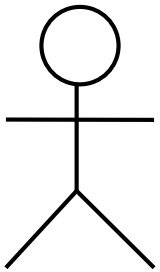
Use Case Diagrams



- Used during requirements elicitation to represent external behavior

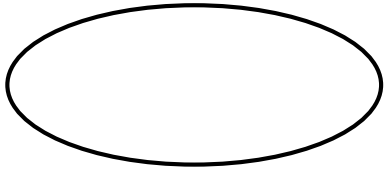
- ***Actors*** represent roles, that is, a type of user of the system
- ***Use cases*** represent a sequence of interaction for a type of functionality
- The use case model is the set of all use cases. It is a complete description of the functionality of the system and its environment

Actors



- An actor models an external entity which communicates with the system:
  - User
  - External system
  - Physical environment
- An actor has a unique name and an optional description.
- Examples:
  - Passenger: A person in the train
  - GPS satellite: Provides the system with GPS coordinates

Use Case



PurchaseTicket

A use case represents a class of functionality provided by the system as an event flow.

A use case consists of:

- Unique name
- Participating actors
- Entry conditions
- Flow of events
- Exit conditions
- Special requirements

Use Case Diagram: Example

Name: `Purchase ticket`

Participating actor:
`Passenger`

Entry condition:

- `Passenger` standing in front of ticket distributor.
- `Passenger` has sufficient money to purchase ticket.

Exit condition:

- `Passenger` has ticket.

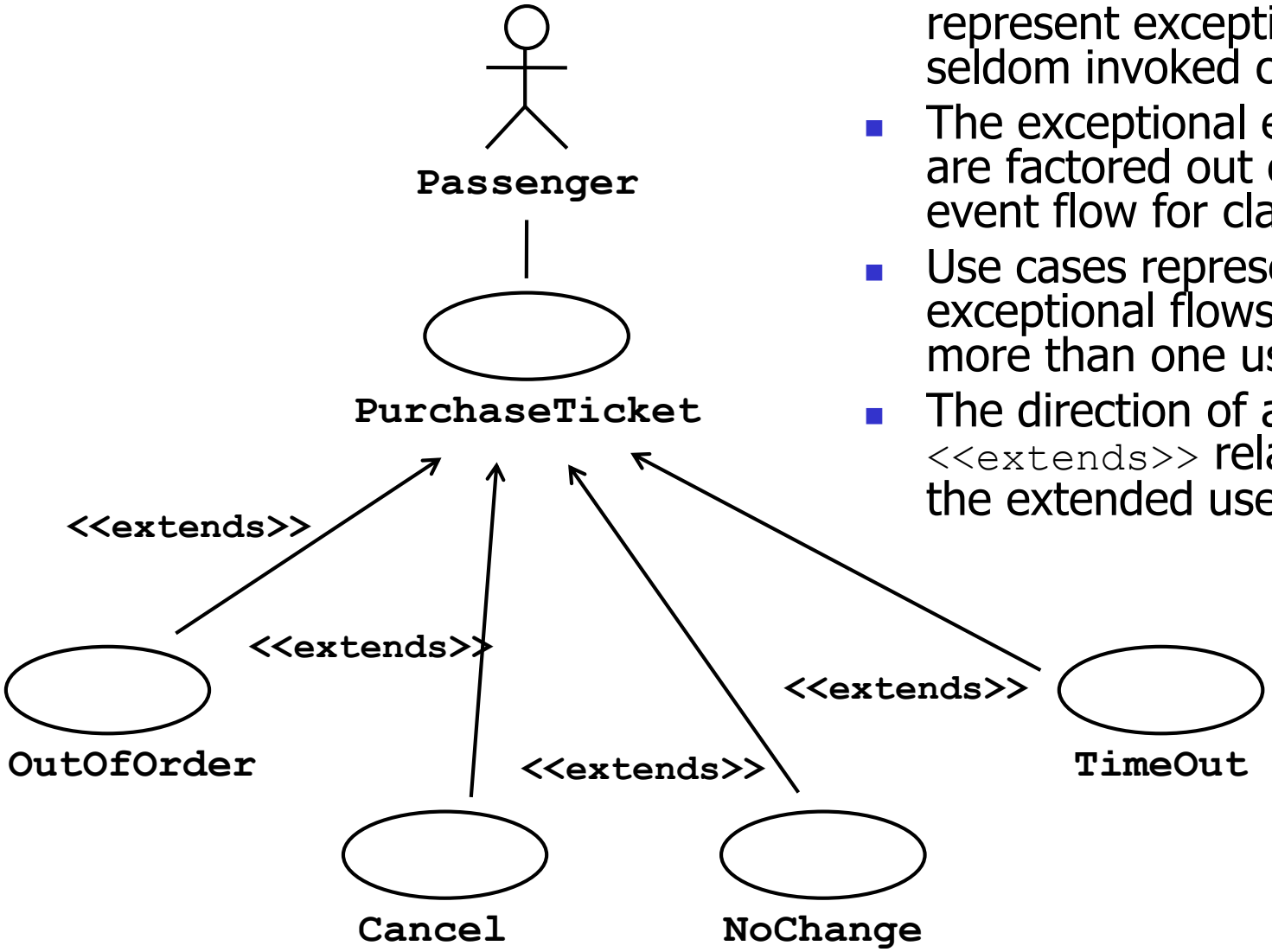
Event flow:

1. `Passenger` selects the number of zones to be traveled.
2. Distributor displays the amount due.
3. `Passenger` inserts money, of at least the amount due.
4. Distributor returns change.
5. Distributor issues ticket.

Uses Cases can be related

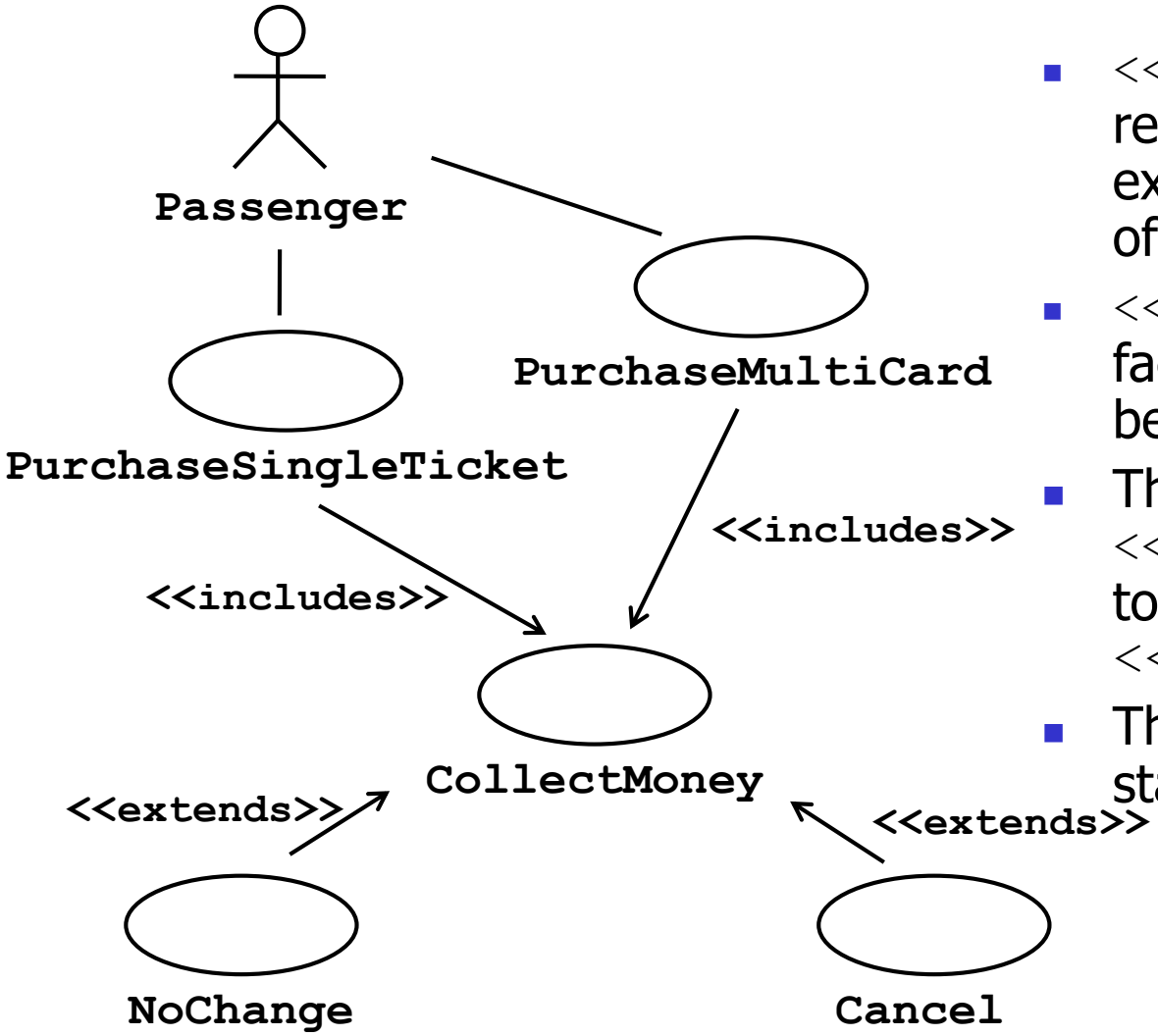
- Extends Relationship
  - To represent seldom invoked use cases or exceptional functionality
- Includes Relationship
  - To represent functional behavior common to more than one use case.

The <<extends>> Relationship



- <<extends>> relationships represent exceptional or seldom invoked cases.
- The exceptional event flows are factored out of the main event flow for clarity.
- Use cases representing exceptional flows can extend more than one use case.
- The direction of a <<extends>> relationship is to the extended use case

The <<includes>> Relationship

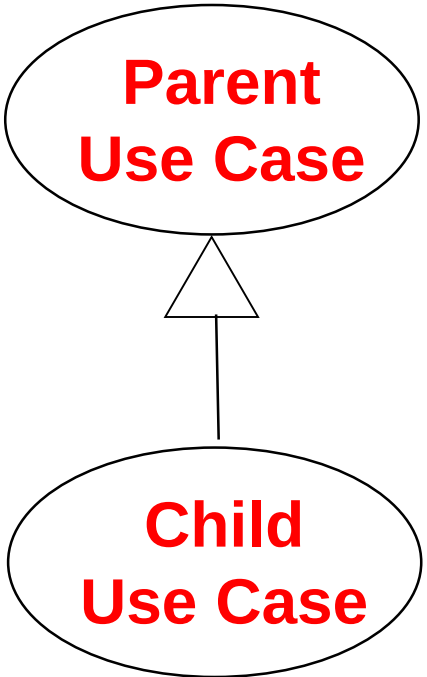


- <<includes>> relationship represents that one use case explicitly includes the behavior of another use case.
- <<includes>> behavior is factored out for reuse, not because it is an exception.
- The direction of a <<includes>> relationship is to the using use case (unlike <<extends>> relationships).
- The Included use case doesn't stand alone

Use Case Generalization

- Generalization Refers to a relationship between a general use case and a more specific version of that use case

- Sub use case as being a “kind of” the super use case



Use Case Example

Draw a use case diagram for a ticket distributor for a train system. The system includes two actors: a traveler, who purchases different types of tickets, and a central computer system, which maintains a reference database for the tariff. Use cases should include:

BuyOneWayTicket

BuyWeeklyCard

BuyMonthlyCard

UpdateTariff

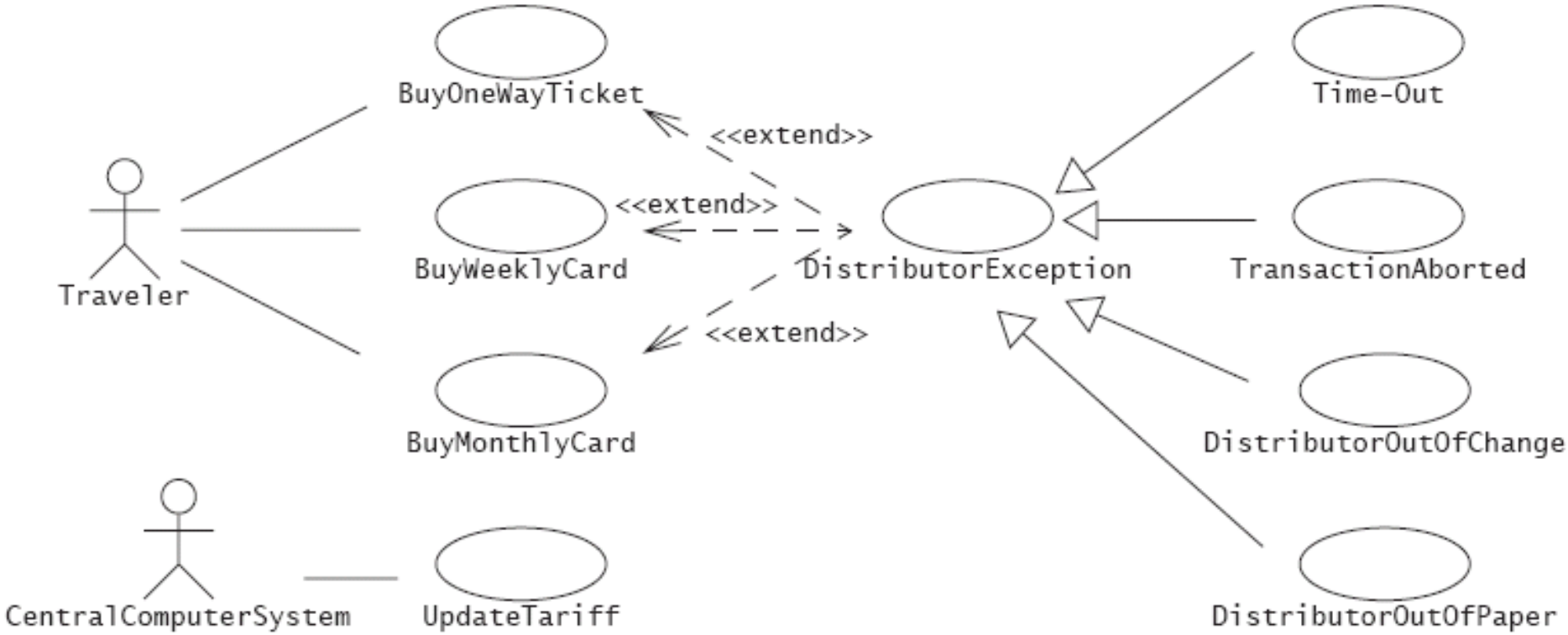
Also include the following exceptional cases:

Time-Out (i.e., traveler took too long to insert the right amount),

TransactionAborted (i.e., traveler selected the cancel button without completing the transaction),

DistributorOutOfChange

DistributorOutOfPaper


BuyOneWayTicket
<<extend>>
Time-Out
Traveler
<<extend>>
BuyWeeklyCard
DistributorException
TransactionAborted
<<extend>>
BuyMonthlyCard
DistributorOutOfChange
CentralComputerSystem
UpdateTariff
DistributorOutOfPaper

Requirement Specification: Types of Requirements

- Functional requirements:
  - Describe the interactions between the system and its environment independent from implementation
  - Examples:
    - The ATM should provide a Cancel button
- Nonfunctional requirements:
  - User visible aspects of the system not directly related to functional behavior.
  - Examples:
    - The response time must be less than 1 second
    - The server must be available 24 hours a day

Nonfunctional Requirement

- Usability: is the ease with which a user can learn to operate, prepare inputs for and interpret outputs of a system. Often, clients address usability issues by requiring the developer to follow user interface guidelines on color, logos, and fonts

- Reliability: the ability of a system to perform its required functions under stated conditions for a specified period of time
  - Robustness: The degree to which a system can function correctly in the presence of invalid inputs or stressful environment conditions
  - Safety: A measure of the absence of catastrophic consequences to the environment

Nonfunctional Requirement (con.)

- Performance: Concerned with quantifiable attributes of the system
  - Response time: how quickly the system react
  - Throughput: how much work the system can accomplish within a specified amount of time
  - Availability: The degree to which a system is operational and accessible when required for use
  - Accuracy
- Supportability: Concerned with the ease of changes to the system after deployment
  - Adaptability: The ability to change the system to deal with additional application domain
  - Maintainability: the ability to change the system to deal with new technology or to fix defect
  - Portability: the ease with which a system can be transferred from one HW/SW to another

Exercise

1. The MTTF of the system shall be larger than 240 hours
2. When user click on show status button, the result shall be shown in 1 second.
3. The warning message should be in red
4. The system should display time correctly in all 24 time zones
5. System should be able to upgrade

Reliability

Performance

Supportability

Usability

Quality of Requirements

Good Requirements

- Must state only one item
  - A current value representing a deviation >= 5% from the nominal value shall cause a display on the operator monitor and if its deviation >= 10% from the nominal value, it shall cause an alarm to sound.

Implementation free
- Shall just state what is needed, not how should it be done.

Feasible
- Achievable, Attainable

Complete
- Shall not require further amplification

Consistent
- Non-contradicting

Example of an Unintended Feature

From the News: London underground train leaves station without driver!

What happened?

- A passenger door was stuck and did not close
- The driver left his train to close the passenger door
  - He left the driver door open
  - He relied on the specification that said the train does not move if at least one door is open
- When he shut the passenger door, the train left the station without him
  - The driver door was not treated as a door in the source code!

Quality of Requirements

Good Requirements

Unambiguous

- One and only one interpretation
  - Use of standard constructs like "Shall" statements.
  - Min and Max, No less than or no greater than etc.
  - Words to avoid: "flexible", "fault tolerant", "high fidelity", "adaptable", "rapid or fast", "adequate", "user friendly", "support", "maximize" and "minimize"

.

Verifiable, Testable

- Not general, Quantitative
  - The system shall not crash during normal operation.

Functionally Complete

- No missing requirement
  - Hard to achieve. Iterative refinement.

.

Example of an Ambiguous Specification

During a laser experiment, a laser beam was directed from earth to a mirror on the Space Shuttle Discovery

The laser beam was supposed to be reflected back towards a mountain top 10,023 feet high

The operator entered the elevation as “10023”

The light beam never hit the mountain top
What was the problem?

The computer interpreted the number in miles...

Requirements Tools

- Tool support for managing requirements:
  - Store requirements in a shared repository
  - Provide multi-user access
  - Automatically create a system specification document from the repository
  - Allow change management
  - Provide traceability throughout the project lifecycle
- RequisitPro from IBM
  - https://www.ibm.com/support/knowledgecenter/en/SSSHCT
- DOORS
  - http://www-03.ibm.com/software/products/en/ratidoor
- Enterprise Architect