Topics

- State Specifications
- Behavior Specifications
- State Change Specifications

Principles of requirements specification

- Specification models are concerned with
  - State
  - Behavior
  - State change
- State and behavior modeling are conducted in parallel
- The world cannot be understood from a single view
- Visual Modeling Language - UML
State specifications

- **Object state** is determined by the values of its attributes and associations

- **State specification**:
  - Model of data structures
  - Static view on the system
  - Class operations left out in initial specs
  - Emphasis on entity classes ("business objects")

Modeling classes

- Cornerstone of OO development – a system is a set of collaborating (and classified) objects

- Iterative and incremental process

- CASE tool
  - For collaborative development
  - For personal productivity otherwise

Discovering classes

- No two analysts will come up with the identical class models for the same application domain

- Discovering classes
  - Noun phrase
  - Common class patterns
  - Use case driven
  - CRC
  - Mixed
**Noun phrase approach**

- Nouns considered candidate classes
- Three kinds of candidate classes
  - Irrelevant (can be skipped)
  - Relevant
  - Fuzzy
- Assumes that the Requirements Document is complete and correct

**Common class pattern approach**

- Derives candidate classes from the classification theory of objects
- One possible classification pattern:
  - Concept (e.g. Reservation)
  - Events (e.g. Arrival)
  - Organization (e.g. Department)
  - People (e.g. Passenger)
  - Places (e.g. TravelOffice)
- Just a guidance
- Only loosely bound to user requirements
- Possible naming misinterpretations

**Use case driven approach**

- Assumes that:
  - Use Case Diagrams (and possibly some high-level Sequence Diagrams) have been developed
  - Narrative descriptions for each use case exist
- Similar to the noun phrase approach
- Function-driven (problem-driven)
- Relies on the completeness of use case models
CRC approach

- CRC – classes, responsibilities, collaborators
- More than a technique for class discovery
- Animated brainstorming sessions
- Identifies classes from the analysis of how objects collaborate to perform business functions (use cases)
- Suitable also for:
  - Verification of classes discovered with other methods
  - Determination of class properties

Mixed approach

- Perhaps with elements of all four previous approaches
- Middle-out rather than top-down or bottom-up
- One possible scenario:
  - Initial classes – domain knowledge
  - Common class patterns approach to guide
  - Noun phrase approach to add more classes
  - Use case approach to verify
  - CRC to brainstorm

Guidelines for class discovery

- Statement of purpose
- Description for a set of objects
  - Singleton classes
  - Houses a set of attributes
    - Identifying attributes - keys
    - OID
  - Class or attribute?
  - Houses a set of operations (what does the class do?)
Example 4.1 – University Enrolment

Consider the following requirements for the University Enrolment system and identify the candidate classes:

- Each university degree has a number of compulsory courses and a number of elective courses.
- Degree
- Course
- CompulsoryCourse
- ElectiveCourse
- Relevant
- Fuzzy

Example 4.1 – University Enrolment

More requirements:

- Each course is at a given level and has a credit-point value.
- A course can be part of any number of degrees.
- Each degree specifies minimum total credit points value required for degree completion.
- Students may combine course offerings into programs of study suited to their individual needs and leading to the degree in which enrolled.

Example 4.1 – University Enrolment (solution)

<table>
<thead>
<tr>
<th>Relevant classes</th>
<th>Fuzzy classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Course</td>
<td>CompulsoryCourse</td>
</tr>
<tr>
<td>Degree</td>
<td>ElectiveCourse</td>
</tr>
<tr>
<td>Student</td>
<td>Studyprogram</td>
</tr>
<tr>
<td>CourseOffering</td>
<td></td>
</tr>
</tbody>
</table>
Example 4.2 – Video Store

Consider the following requirements for the Video Store system and identify the candidate classes:

- The video store keeps in stock an extensive library of current and popular movie titles. A particular movie may be held on video tapes or disks.

<table>
<thead>
<tr>
<th>Relevant</th>
<th>Irrelevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovieTitle</td>
<td>VideoStore</td>
</tr>
<tr>
<td>VideoTape</td>
<td>Stock</td>
</tr>
<tr>
<td>VideoDisk</td>
<td>Library</td>
</tr>
</tbody>
</table>

Example 4.2 – Video Store

More requirements:

- Video tapes are in either "Beta" or "VHS" format
- Video disks are in DVD format
- Each movie has a particular rental period (expressed in days), with a rental charge to that period
- The video store must be able to immediately answer any inquiries about a movie's stock availability and how many tapes and/or disks are available for rental
- The current condition of each tape and disk must be known and recorded

Example 4.2 – Video Store (solution)

<table>
<thead>
<tr>
<th>Relevant classes</th>
<th>Fuzzy classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovieTitle</td>
<td>RentalConditions</td>
</tr>
<tr>
<td>VideoMedium</td>
<td></td>
</tr>
<tr>
<td>VideoTape</td>
<td></td>
</tr>
<tr>
<td>VideoDisk</td>
<td></td>
</tr>
<tr>
<td>(or DVDDisk)</td>
<td></td>
</tr>
<tr>
<td>BetaTape</td>
<td></td>
</tr>
<tr>
<td>VHSTape</td>
<td></td>
</tr>
</tbody>
</table>
Example 4.3 – Contact Management

Consider the following requirements for the Contact Management system and identify the candidate classes:

- To "keep in touch" with current and prospective customer base
- To win new contracts
- To store the names, phone numbers, postal and courier addresses, etc. of organizations and contact persons in these organizations
- To schedule tasks and events for the employees with regard to relevant contact persons
- Employees can schedule tasks and events for other employees or for themselves
- A task is a group of events that take place to achieve a result (e.g. to solve customer's problem)
- Typical types of events are: phone call, visit, sending a fax, arranging for training, etc.

Example 4.3 – Contact Management (solution)

### Relevant classes

<table>
<thead>
<tr>
<th>Relevant classes</th>
<th>Fuzzy classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization</td>
<td>CurrentOrg</td>
</tr>
<tr>
<td>Contact</td>
<td>ProspectiveOrg</td>
</tr>
<tr>
<td>Employee</td>
<td>PostalAddress</td>
</tr>
<tr>
<td>Task</td>
<td>CourierAddress</td>
</tr>
<tr>
<td>Event</td>
<td></td>
</tr>
</tbody>
</table>

Specifying classes

- **In Class Diagram**
  - Each class given a name (and possibly a code)
    - Singular noun
      - Recommendation – multiple words joined; each word starting with a capital letter (e.g. PostalAddress)
      - Meaningful
      - Short (less than 30 characters)
  - Class properties to be defined
    - Attributes (initially those that capture interesting object states)
      - Recommendation – small letters; underscore to separate words (e.g. street_name)
    - Operations (can be delayed till later analysis stages or even till design)
Example 4.4 – University Enrolment

Refer to Example 4.1

Consider the following additional requirements from the Requirements Document:

- A student's choice of courses may be restricted by timetable clashes and by limitations on the number of students who can be enrolled in the current course offering.

More requirements:

- A student's proposed program of study is entered on-line enrolment system T
- The system checks the program's consistency and reports any problems
- The problems need to be resolved with the help of an academic adviser
- The final program of study is subject to academic approval by the delegate of the Head of Division and it is then forwarded to the Registrar.

Example 4.4 – University Enrolment (solution)

<table>
<thead>
<tr>
<th>Degree</th>
<th>Course</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree</td>
<td>Course</td>
</tr>
<tr>
<td>Degree</td>
<td>Course</td>
</tr>
<tr>
<td>Degree</td>
<td>Course</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StudyProgram</th>
<th>Class</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>CourseOffering</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student</th>
<th>CourseOffering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>CourseOffering</td>
</tr>
<tr>
<td>Student</td>
<td>CourseOffering</td>
</tr>
<tr>
<td>Student</td>
<td>CourseOffering</td>
</tr>
</tbody>
</table>

MACIASZEK (2001): Req Analysis & Syst Design
Example 4.5 – Video Store

- Refer to Example 4.2
- The additional requirements are:
  - The rental charge differs depending on video medium: tape or disk (but it is the same for the two categories of tapes: Beta and VHS).

Example 4.5 – Video Store

- More requirements:
  - The system should accommodate future video storage formats in addition to VHS tapes, Beta tapes and DVD disks
  - The employees frequently use a movie code, instead of movie title, to identify the movie
  - The same movie title may have more than one release by different directors

Example 4.5 – Video Store (solution)
Example 4.6 – Contact Management

- Refer to Example 4.3 and consider the following additional information
  - A customer is considered current if there exists a contract with that customer for delivery of our products or services. Contract management is, however, outside the scope of our system.

<table>
<thead>
<tr>
<th>CurrentOrg</th>
<th>ProspectOrg</th>
</tr>
</thead>
</table>

- Fuzzy

Example 4.6 – Contact Management

- More requirements:
  - Reports on contacts based on postal and courier addresses (e.g. find all customers by post code)
  - Date and time of the task creation are recorded
  - The "money value" of a task can be stored
  - Events for the employee are displayed on the employee's screen in the calendar-like pages (one day per page).
    - The priority of each event (low, medium or high) is visually distinguished on the screen
    - Not all events have a "due time" - some are "untimed"
  - Event creation time cannot be changed, but the due time can.
  - Event completion date and time are recorded
  - The system stores identifications of employees who created tasks and events, who are scheduled to do the event ("due employee"), and who completed the event

Example 4.6 – Contact Management (solution)
Example 4.7 - Telemarketing

Consider the following additional information:
- Each campaign
  - Has a title that is generally used for referring to it
  - Has also a unique code for internal reference
  - Runs over a fixed period of time
- Soon after the campaign is closed, the prizes are drawn
  and the holders of winning tickets are advised

More requirements:
- Tickets are uniquely numbered within each campaign
- The total number of tickets in a campaign, number of tickets sold so far, and the current status of each ticket are known (e.g. available, ordered, paid for, prize winner)
- To determine the performance of the society’s telemarketers, the duration of calls and the successful call outcomes (i.e. resulting in ordered tickets) are recorded
- Extensive information about supporters is maintained
  - Contact details (address, phone number, etc.)
  - Historical details such as the first and most recent dates when a supporter had participated in a campaign
  - Any known supporter’s preferences and constraints (e.g. times not to call, usual credit card number)

More requirements:
- Telemarketing calls are made according to their priorities
  - Calls which are unanswered or where an answering machine was found, are rescheduled
  - Times of repeat calls are alternated
  - Number of repeat calls is limited
    - Limits may be different for different call types (e.g. a normal “solicitation” call may have different limit than a call to remind a supporter of an outstanding payment)
  - Call outcomes are categorized - success (i.e. tickets ordered), no success, call back later, no answer, engaged, answering machine, fax machine, wrong number, disconnected.
Example 4.7 – Telemarketing (solution)

Campaign
- campaign_code: String
- campaign_title: String
- date_start: Date
- date_close: Date
- date_drawn: Date
- num_tickets: Integer
- num_tickets_sold: Integer

Supporter
- supporter_id: String
- supporter_name: String
- phone_number: String
- mailing_address: String
- date_first: Date
- date_last: Date
- campaign_count: Integer
- preferred_hours: String
- credit_card_number: String

Telemarketer
- telemarketer_id: String
- telemarketer_name: String
- average_per_hour: Double
- success_per_hour: Double

Prize
- prize_descr: String
- prize_value: Currency
- prize_ranking: Integer

CallOutcome
- start_time: Date
- end_time: Date

CallScheduled
- phone_number: String
- priority: String
- attempt_number: Integer

Telemarketing
- ticket_number: String
- ticket_value: Currency
- ticket_status: String

Discovering associations
- Side effect of discovering classes
- Some attributes are associations
  - “Dry-run” of use cases to discover more associations
  - Avoid ternary associations
  - Cycles of associations that do not commute

Specifying associations
- Naming associations
  - Recommendation – small letters; underscore to separate words (e.g. emp_task)
- Naming association roles
- Determining multiplicity
  - Lower and/or upper multiplicity bounds can be omitted initially
  - Rolenames for recursive associations
Example 4.8 – Contact Management

Refer to Examples 4.3 and 4.6 - specify associations
Consider, for example, the requirement:
The system allows producing various reports on our contacts based on postal and courier addresses.

PostalAddress
- street : String
- po_box : String
- city : String
- state : String
- post_code : String
- country : String

CourierAddress
- street_and_directions : String
- city : String
- state : String
- country : String

Contact
- contact_id : Integer
- family_name : String
- first_name : String
- phone : String
- fax : String
- email : String

Example 4.8 – Contact Management (solution – 1)

PostalAddress
- street : String
- po_box : String
- city : String
- state : String
- post_code : String
- country : String

CourierAddress
- street_and_directions : String
- city : String
- state : String
- country : String

Organization
- organization_id : Integer
- organization_name : String
- phone : String
- fax : String
- email : String
- is_current : Boolean

Contact
- contact_id : Integer
- family_name : String
- first_name : String
- phone : String
- fax : String
- email : String

Example 4.8 – Contact Management (solution – 2)

Task
- description : String
- created_dt : Date
- value : Currency

Event
- description : String
- created_dt : Date
- due_dt : Date
- completed_dt : Date
- priority : Byte

Task
- task

Employee
- employee_id : String
- family_name : String
- first_name : String
- middle_name : String

Employee
- employee

Contact
- contact

Task
- task

Employee
- employee

Contact
- contact
Modeling aggregation

- Four semantics for aggregation possible
  - ExclusiveOwns (e.g. Book has Chapter)
    - Existence dependency
    - Transitivity
    - Asymmetricity
    - Fixed property
  - Owns (e.g. Car has Tire)
    - No fixed property
  - Has (e.g. Division has Department)
    - No existence dependency
    - No fixed property
  - Member (e.g. Meeting has Chairperson)
    - No special properties except membership

Discovering aggregation

- Discovered in parallel with discovery of associations
- The litmus test phrases
  - "has"
  - "is-part-of"
- Can relate more than two classes

Specifying aggregation

- UML supports
  - Aggregation
    - By-reference semantics
    - Hollow diamond
    - Corresponds to Has and Member aggregations
  - Composition
    - By-value semantics
    - Solid diamond
    - Corresponds to ExclusiveOwns and Owns aggregations
Example 4.9 – University Enrolment

Refer to Examples 4.1 and 4.5.
Consider the following additional requirements:

- The student’s academic record to be available on demand.
- The record to include information about the student’s grades in each course that the student enrolled in (and has not withdrawn without penalty).
- Each course has one academic in charge of a course, but additional academics may also teach in it.
  - There may be a different academic in charge of a course each semester.
  - There may be different academics for each course each semester.

Example 4.9 – University Enrolment (solution)

Course
- `<<PK>>` course_code : String
- `<<CK>>` course_name : String
- credit_points : Integer

Student
- `<<PK>>` student_id : String
- student_name : String
- current_fees : Money

AcademicInCharge
- course_code : String
- student_id : String

CourseOffering
- `<<PK>>` year : Date
- semester : Integer
- enrolment_quota : Integer

0..* takes
0..* takes_courseOffering
0..* has_student

Modeling generalization

- Common features abstracted into a more generic class.
- Subclasses inherit (reuse) superclass features.
- Substitutability – subclass object is a legal value for a superclass variable (e.g., a variable holding Fruit objects can have an Apple object as its value).
- Polymorphism – the same operation can have different implementations in different classes.
- Abstract operation – implementation provided in subclasses.
- Abstract class – class with no direct instance objects.
  - A class with an abstract operation is abstract.
Discovering and specifying generalization

- Some discovered in parallel with discovery of associations
- The litmus test phrases
  - “can-be”
  - “is-a-kind-of”
- Multiple inheritance possible
- Solid line with an arrowhead pointing to the superclass

Example 4.10 – Video Store

- Refer to Examples 4.2 and 4.5.
- The classes identified in Example 4.5 imply a generalization hierarchy rooted at the class `VideoMedium`
- Extend the model to include relationships between classes, and specify generalization relationships
- Assume that the Video Store needs to know if a `VideoTape` is a brand new tape or if it was already taped over (this can be captured by an attribute `is_taped_over`)
- Assume also that the storage capacity of a `VideoDisk` allows holding multiple versions of the same movie, each in a different language or with different endings

Example 4.10 – Video Store (solution)
Modeling and specifying objects

- Only to exemplify
  - To illustrate complex relationships between objects
  - To demonstrate changes to objects over time
  - To illustrate object collaboration

Example 4.11 – University Enrolment

Show few objects representing the classes in Example 4.9

Don Donaldson: Student
COMP224: AcademicRecord
COMP325: Course
COMP326: AcademicRecord
COMP225: Course
2000 Sem2: CourseOffering
Rick Richards: AcademicInCharge

Behavior specification

- Depicted in use cases
- Determines which classes are involved in execution of use cases
  - Main class operations identified
  - Message passing between objects captured
  - Control classes and boundary classes considered
- Computations modeled in Activity Diagrams
- Interactions modeled in Sequence Diagrams or Collaboration Diagrams
Modeling use cases

- Complete piece of functionality
  - Main flow
  - Subflows
  - Alternate flows
- Piece of externally visible functionality
- Orthogonal piece of functionality
- Piece of functionality initiated by an actor
- Piece of functionality that delivers an identifiable value to an actor

Discovering use cases

- Discovered from
  - Requirements identified in the Requirements Document
  - Actors and their purpose in the system
- Questions to ask
  - What are the main tasks performed by each actor?
  - Will an actor access or modify information in the system?
  - Will an actor inform the system about any changes in other systems?
  - Should an actor be informed about unexpected changes in the system?

Specifying use cases

- Actors
- Use cases
- Four kinds of relationships
  - Association (between actor and use case)
  - Include (stereotyped with the word: «include»)
    - Included use case is always necessary for the completion of the activating use case
  - Extend (stereotyped with the word: «extend»)
    - Another use is activated occasionally at specific extension point
  - Generalization
- Relationships to be used with restrain
Example 4.12 – University Enrolment

- Data Entry Person
- Student
  - Provide Examination Results
  - Enter Program of Study
  - Validate Program of Study
  - Provide Enrolment Instructions
- Registrar Office
- Student Office
  - Validate Program of Study
  - Provide Enrolment Instructions

Example 4.13 – Contact Management

- Employee
  - Maintain Organization
  - Customer Services Manager
  - Complete Event
  - Create Task
  - Schedule Event

Example 4.14 – Video Store

- Scanning Device
  - Rent Video
  - Reserve Video
  - Maintain Customer
  - Answer Enquiry
  - Return Video
- Employee
  - Rent Video
  - Reserve Video
  - Order Video
Example 4.14 – Video Store (Rent Video)

**Brief Description**
A customer wishes to rent a video tape or disk that is picked from the store's shelves or that has been previously reserved by the customer. Provided the customer has a non-delinquent account, the tape is rented out once the payment has been received. If the tape is not returned in a timely fashion, an overdue notice is mailed to the customer.

**Actors**
Employee, Scanning Device

**Preconditions**
Video tape or disk is available to be hired. Customer has a membership card. Scanner devices work correctly. Employee at the front desk knows how to use the system.

**Main Flow**
Videos are rented out and the database is updated accordingly.

**Main Postconditions**
A customer does not have a membership card. In this case, the Maintain Customer use case may be activated to issue a new card.

**Alternative Flows**
A customer does not have a membership card. In this case, the Maintain Customer use case may be activated to issue a new card.

**Postconditions**
Videos are rented out and the database is updated accordingly.
Example 4.15 – Telemarketing (solution)

Modeling activities

- Activity Diagrams
- Flow of logic
  - Sequential control
  - Concurrent control
- Can be used at different levels of abstraction
  - To define execution of a use case
  - To define execution of an operation

Discovering and specifying activities

- The execution proceeds from one activity state to the next
- An activity state completes when its computation is completed
- Activities can be discovered from the narrative specifications of use cases
- Activities are connected by transition lines
- Synchronization bars (fork and re-join)
- Branch diamonds (branch and merge)
- External events not normally modeled on activity graphs
Example 4.16 – Video Store (solution)

Customer Card

Scan Video Medium

Verify Customer

Initiate Rent Transaction

Remove One Medium Request

Accept Payment

Update Stock

Print Receipt

Commit Rent Transaction

[is unreliable]

[is delinquent]

[deposit refused]

> 8 videos

Add deposit

Modeling interactions

- Sequence Diagrams
  - Show an exchange of messages between objects arranged in a time sequence
  - More useful in analysis

- Collaboration Diagrams
  - Emphasize the relationships between objects along which the messages are exchanged
  - More useful in design

- Can be used to determine operations in classes

Message sequences

- Activities in Activity Diagrams are mapped to messages to Sequence Diagrams

- Message can be a:
  - Signal
    - Denotes asynchronous inter-object communication
    - The sender continues executing after sending the signal message
  - Call
    - Denotes synchronous invocation of an operation
    - The return message can return some values to the caller or it can just acknowledge that the operation completed
Example 4.17 – University Enrolment

Determined by the set of operations that the class offers as its service

In analysis
- Signature of each operation is defined
  - Operation name
  - List of formal arguments
  - Return type

In design
- Algorithm of a method that implements the operation is defined

Operation can have
- Instance scope
- Class (static) scope (\$ in front of operation name)

Discovering class operations

From Sequence Diagrams
- Message to an object must be serviced by an operation in that object

From expected object responsibilities, including the CRUD operations
- Create – a new object instance
- Read – the state of an object
- Update – the state of an object
- Delete – i.e. destroy itself
Example 4.18 – UE (solution)

- Refer to Examples 4.9 and 4.17 and to the classes `Course` and `CourseOffering`
- Derive operations from the Sequence Diagram and add them to the classes `Course` and `CourseOffering`

```
Course
<PK> course_code : String
<CK> course_name : String
credit_points : Integer
crs_off : set<CourseOffering>
areYouOpen(out c_check)
```

```
CourseOffering
<AN> year : Date
semester : Integer
enrolment_quota : Integer
std : list<Student>
crs : Course
areYouOpen(out c_check)
addStudent(stdOID)
```

State change specifications

- Statechart Diagrams
- For each class that exhibits an interesting dynamic behavior
  - Changes to some attributes signify state changes

Specifying object states

- State transition fires when a certain event occurs or a certain condition is satisfied
  - transition line does not have to be labeled with an event name
  - condition itself (written in square brackets) can fire the transition
- Transition can be triggered by
  - Signal event
  - Call event
  - Change event
  - Time event
Example 4.19 – Video Store

<table>
<thead>
<tr>
<th>MovieTitle</th>
<th>Available</th>
<th>Not in Stock</th>
<th>put on shelf/quantity</th>
<th>out of stock/quantity</th>
<th>in Stock</th>
<th>ordered/quantity</th>
<th>replenish stock/quantity</th>
<th>ordered/insufficient stock</th>
<th>ordered/insufficient stock</th>
<th>order item/quantity</th>
<th>Reserved</th>
<th>(no more reserved)</th>
<th>update number reserved</th>
<th>Not Reserved</th>
</tr>
</thead>
</table>

Summary

- **State specifications** describe the IS world from the static perspective of classes, their attribute content and their relationships.
  - There are many methods of class discovery.
    - Class diagrams visualize classes and relationships: associations, aggregations, and generalizations.
- **Behavioral specifications** describe the IS world from the operational (functional) perspective.
  - Use case diagrams provide simple visualization – each use case is given narrative specification.
  - Other behavioral diagrams include activity diagrams, interaction diagrams, and addition of operations to classes.
- **State change specifications** describe the IS world from the dynamic perspective.
  - Statechart diagrams allow modeling of state changes.