Structural Modeling

Structural model describes the structure of the data that supports the business processes in an organization. During the analysis phase, the structural model presents the logical organization of data without indicating how the data are stored, created, or manipulated so that analysts can focus on the business without being distracted by technical details. Later, during the design phase, the structural model is updated to reflect exactly how the data will be stored in databases and files.

Objectives

- The structure of data used in the system is represented through Class-Responsibility-Collaboration cards, class diagrams, and object diagrams.
  - Understand the rules and style guidelines for creating CRC cards, class diagrams, and object diagrams.
  - Understand the processes used to create CRC cards, class diagrams, and object diagrams.
  - Be able to create CRC cards, class diagrams, and object diagrams.
  - Understand the relationship between the structural and use-case models.

Structural model

- a formal way of representing the objects that are used and created by a business system.
- illustrates people, places, or things about which information is captured, and how they are related to each other.
- drawn using an iterative process in which the model becomes more detailed and less “conceptual” over time.
- In the analysis phase,
  - analysts draw a conceptual model,
    - which shows the logical organization of the objects without indicating how the objects are stored, created, or manipulated.
  - Because this model is free from any implementation or technical details, the analysts can focus more easily on matching the model to the real business requirements of the system.

During the analysis phase,
- analysts create functional models to represent how the business system will behave.
- At the same time, analysts need to understand the information that is used and created by the business system (e.g., customer information, order information).

In the design phase,
- analysts evolve the conceptual structural model into a design model that reflects how the objects will be organized in databases and files.
- At this point, the model is checked for redundancy and the analysts investigate ways to make the objects easy to retrieve.

Structural Models

Structural model describes the structure of the data that supports the business processes in an organization. During the analysis phase, the structural model presents the logical organization of data without indicating how the data are stored, created, or manipulated so that analysts can focus on the business without being distracted by technical details. Later, during the design phase, the structural model is updated to reflect exactly how the data will be stored in databases and files.
Every time a systems analyst encounters a new problem to solve, (s)he must learn the underlying problem domain.

The goal of the analyst:
- to discover the key objects contained in the problem domain and to build a structural model of the objects.
- OO modeling allows the analyst to reduce the “semantic gap” between the underlying problem domain and the evolving structural model.

However, the real world and the world of software are very different.
- The real world tends to be messy, whereas the world of software must be neat and logical.

As such, an exact mapping between the structural model and the problem domain may not be possible.

Reduce the “semantic gap” between the real world and the world of software

Create a vocabulary for analysts and users

Represent things, ideas, and concepts of importance in the application domain
- the objects, contained in the domain of the problem

At this stage of development, the structural model does not represent software components or classes in an object-oriented programming language,
- even though the structural model does contain analysis classes, attributes, operations, and relationships among the analysis classes.

The structural model at this point should represent the responsibilities of each class and the collaborations between the classes.

Typically, structural models are depicted using
- CRC cards,
- class diagrams,
- object diagrams.

A class
- a general template that we use to create specific instances, or objects, in the application domain.

All objects of a given class are identical in structure and behavior but contain different data in their attributes.

There are two different general kinds of classes of interest during the analysis phase:
- concrete
- abstract

When analysts describe the application domain classes,
- they are referring to concrete classes;

Concrete classes are used to create objects

Abstract classes are useful abstractions.

For example, from an employee class and a customer class, we may identify a generalization of the two classes and name the abstract class person.

We may not actually instantiate the Person class in the system itself, but rather only create and use employees and customers.
An attribute of an analysis class
- represents a piece of information
  - relevant to the description of the class within the application
    domain of the problem being investigated.
  - contains information that the analyst or user feels the
    system should store.
- For example,
  - a possible relevant attribute of an employee class is employee
    name,
  - while one that may not be as relevant is hair color.
- Both describe something about an employee, but hair color
  probably is not all that useful for most business applications.

Only those attributes that are important to the task
should be included in the class.
- One should only add attributes that are primitive or
  atomic types, that is,
  - integers, strings, doubles, date, time, boolean, and so
    on.
- Most complex or compound attributes are really
  placeholders for relationships between classes.
- Therefore, they should be modeled as relationships, not
  attributes.

An operation, or service, of an analysis class is
- where the behavior of the class will be defined.
- Action that instances/objects can take
  - For example,
    - classes are normally required to provide the means to create
      instances, delete instances, access individual attribute values,
      set individual attribute values, access individual relationship
      values, and remove individual relationship values.
    - However, at this point in the development of the evolving
      system, the analyst should avoid cluttering up the definition
      of the class with these basic types of operations and only
      focus on relevant problem domain-specific operations.

Many different types of relationships can be defined,
- but all can be classified into three basic categories of
data abstraction mechanisms:
  - generalization relationships,
  - Aggregation relationships,
  - association relationships.
- These data abstraction mechanisms allow the analyst
  to focus on the important dimensions while ignoring
  nonessential dimensions.
- Like attributes, the analyst must be careful to include
  only relevant relationships.

The generalization abstraction enables the analyst to
create classes that inherit attributes and operations of
other classes.
The analyst creates a superclass
- contains the basic attributes and operations that will be
  used in several subclasses.
  - inherit the attributes and operations of their superclass and
    can also contain attributes and operations that are unique just
    to them.
  - For example, a customer class and an employee class can be
    generalized into a person class by extracting the attributes and
    operations they have in common and placing them into the new
    superclass, Person.

In this way, the analyst can reduce the redundancy in
the class definitions
  - so that the common elements are defined once and then
    reused in the subclasses.
- Generalization is represented with the a-kind-of
  relationship, so that we say that an employee is an
  a-kind-of person.
- The analyst also can use the flip side of
generalization, specialization, to uncover additional
classes by allowing new subclasses to be created from
an existing class.
Generalization Relationships

- For example, an employee class can be specialized into a secretary class and an engineer class.
- Furthermore, generalization relationships between classes can be combined to form generalization hierarchies.
- Based on the previous examples, a secretary class and an engineer class can be subclasses of an employee class, which in turn could be a subclass of a person class.
- This would be read as, “a secretary and an engineer are a-kind-of employee and a customer and an employee are a-kind-of person.”

Aggregation Relationships

- Aggregation or composition relates parts to wholes.
- Many different types of aggregation or composition relationships have been proposed in data modeling, knowledge representation, and linguistics.
  - For example, a-part-of (logically or physically), a-member-of (as in set membership), contained in, related to, associated with.
- All aggregation relationships relate parts to wholes or parts to assemblies.

Aggregation Relationships

- Aggregation relationships can also be combined into aggregation hierarchies.
  - For example, a piston is a-part-of an engine, while an engine is a-part-of a car.
- Aggregation relationships are bidirectional in nature. The flip side of aggregation is decomposition.
  - The analyst can use decomposition to uncover parts of a class that should be modeled separately.
    - For example, if a door and engine are a-part-of a car, then a car has parts door and engine.
  - The analyst can bounce around between the various parts to uncover new parts.
    - For example, the analyst can ask, “What other parts are there to a car?” or “To which other assemblies can a door belong?”

Association Relationships

- There are other types of relationships that do not fit neatly into a generalization (a-kind-of) or aggregation (a-part-of) framework.
- These relationships are usually a weaker form of the aggregation relationship.
  - For example, a patient schedules an appointment.
    - It could be argued that a patient is a-part-of an appointment.
  - However, there is a clear semantic difference between this type of relationship and one that models the relationship between doors and cars or even workers and unions.
- As such, they are simply considered to be associations between instances of classes.

Your Turn

- What classes, attributes, and operations that would be required to describe the process of registration for campus housing?

CLASS-RESPONSIBILITY-COLLABORATION CARDS

CRC cards, used to document responsibilities collaborations of a class.
Responsibilities and Collaborations

• The responsibilities of a class can be divided into two separate types:
  – knowing responsibilities
  – doing responsibilities

• Knowing responsibilities
  – those things that an instance of a class must be capable of knowing.
  – An instance of a class typically knows the values of its attributes and its relationships.

• Doing responsibilities
  – those things that an instance of a class must be capable of doing.
  – An instance of a class can execute its operations, or it can request a second instance that it knows about to execute one of its operations on behalf of the first instance.

Responsibilities and Collaborations

• A client object
  – an instance of a class that sends a request to an instance of another class for an operation to be executed.

• A server object
  – the instance that receives the request from the client object.

• A contract
  – formalizes the interactions between the client and server objects.

• For example,
  – A patient makes an appointment with a doctor. This sets up an obligation for both the patient and doctor to appear at the appointed time. Otherwise, consequences, such as billing the patient for the appointment regardless of whether the patient appears, can be dealt out. The contract should also spell out what the benefits of the contract will be, such as a treatment being prescribed for whatever ails the patient and a payment to the doctor for the services provided.

Responsibilities and Collaborations

• They then can either ask questions of themselves or be asked questions by other members of the development team.

• For example,
  – Who or what are you?
  – What do you know?
  – What can you do?

• The answers to the questions are then used to add detail to the evolving CRC cards.

Responsibilities and Collaborations

• The structural model describes the objects necessary to support the business processes modeled by the use cases.

• Most use cases involve a set of several classes.

• This set of classes forms a collaboration.

• Collaborations allow the analyst to think in terms of
  – clients,
  – servers,
  – contracts.

Elements of a CRC Card

• The set of CRC cards contains all the information necessary
  – to build a logical structural model of the problem under investigation.

• Each card captures and describes the essential elements of a class.

• The front of the card allows the recording of the
  – class’s name,      – list of associated use cases,
  – ID,               – responsibilities,
  – type,             – collaborators,
  – description,
Elements of a CRC Card

- The name of a class should be a noun
  - but not a proper noun such as the name of a specific person or thing.
- In later stages of development, it is important to be able to trace back design decisions to specific requirements.
  - In conjunction with the list of associated use cases, the ID number for each class can be used to accomplish this.
- The description is simply a brief statement that can be used as a textual definition for the class.
- The responsibilities of the class tend to be the operations that the class must contain,
  - that is, the doing responsibilities.

Elements of a CRC Card

- The back of a CRC card contains the attributes and relationships of the class.
- The attributes of the class represent the knowing responsibilities that each instance of the class will have to meet.
  - Typically, the data type of each attribute is listed with the name of the attribute:
    - for example, the amount attribute is double, and the insurance carrier is text.
- Three types of relationships are captured at this point in time:
  - generalization,
  - aggregation,
  - other associations.
- CRC cards are used to document the essential properties of a class.
  - However, once the cards are filled out, the analyst can use the cards and anthropomorphism in roleplaying to uncover missing properties.

Example CRC Card (front)

Example CRC Card (back)

Class Diagram Syntax

A CLASS
  Represents a kind of person, place, or thing about which the system will need to capture and store information
  Has a name (e.g., doctor and related to its type compartment
  Has a list of attributes in its attribute compartment
  Has a list of operations in its behavior compartment
  Has an identifier that is unique to all classes

AN ATTRIBUTE:
  Represents a property that describes an instance of an object
  Can be defined twice or often attributes, shown by placing a class before the attribute name

AN OPERATION:
  Represents the service or function that a class can perform
  Can be classified as a constructor, query, or update operation
  Includes parameters that may contain parameters or information needed to perform the operation

AN ASSOCIATION:
  Represents a relationship between multiple classes, or a class and a bundle
  Is labeled and a role or name to a role name, which better represents the relationship
  Can exist between zero or one classes
  Contains multiplicity symbols, which represent the minimum and maximum instances a class instance can be associated with the related class instance
Elements of a Class Diagram

- Following figure shows a class diagram that was created to reflect the classes and relationships that are needed for the set of use cases that describe the appointment system discussed previously.

Elements of a Class Diagram

- **Class**
  - The main building block of a class diagram.
  - Stores and manages information in the system.
- **During analysis, classes refer to the**
  - People, places, events, things about which the system will capture information.
- **Later, during design and implementation, classes can refer to**
  - Implementation-specific artifacts
    - Like windows, forms, and other objects used to build the system.
- **Each class is drawn using a three-part rectangle with**
  - The class’s name at the top,
  - Attributes in the middle,
  - Operations at the bottom.

Elements of a Class Diagram

- **For example,**
  - You should be able to identify Person, Doctor, Patient, Medical History, Appointment, Bill, and Symptom as some of the classes in the previous figure.
- **The attributes of a class and their values define**
  - The state of each object that is created from the class.
- **The behavior is represented by the operations.**
- **Attributes**
  - Properties of the class about which we want to capture information
    - Person class in the previous figure contains the attributes lastname, firstname, address, phone, and birthdate.

Elements of a Class Diagram

- You may want to store derived attributes,
  - Attributes that can be calculated or derived
    - Denoted by placing a slash (/) before the attribute’s name
      - Notice how the Person class contains a derived attribute called “/age,” which can be derived by subtracting the patient’s birthdate from the current date.
- It is also possible to show **visibility** of the attribute on the diagram.
  - Visibility relates to the level of information hiding to be enforced for the attribute.
  - The visibility of an attribute can be
    - Public (+)
    - Protected (#)
    - Private (-)

Elements of a Class Diagram

- A **public attribute**
  - Attribute that is not hidden from any other object.
  - As such, other objects can modify its value.
- A **protected attribute**
  - Attribute that is hidden from all other classes except its immediate subclasses.
- A **private attribute**
  - Attribute that is hidden from all other classes.
- The default visibility for an attribute is normally private.
Elements of a Class Diagram

- **Operations**
  - actions or functions that a class can perform.
  - The functions that are available to all classes
    - e.g., create a new instance, return a value for a particular attribute, set a value for a particular attribute, or delete an instance are not explicitly shown within the class rectangle.
    - Instead, only those operations that are unique to the class are included,
      - such as the “cancel without notice” operation in the Appointment class and the “generate cancellation fee” operation in the Bill class in the previous figure.

- **Elements of a Class Diagram**

- **A class can contain three kinds of operations:**
  - constructor,
  - query,
  - update.

  - A constructor operation
    - creates a new instance of a class.
      - For example,
        - the patient class may have a method called “insert ()" that creates a new patient instance as patients are entered into the system.
      - If a method is available to all classes (e.g., create a new instance), it is not explicitly shown on the class diagram,
        - so typically you will not see constructor methods explicitly on the class diagram.

  - A query operation
    - makes information about the state of an object available to other objects,
      - but it will not alter the object in any way.
      - For instance,
        - the “calculate last visit ()” operation that determines when a patient last visited the doctor’s office will result in the object being accessed by the system,
          - but it will not make any change to its information.
        - If a query method merely asks for information from attributes in the class (e.g., a patient’s name, address, or phone),
          - then it is not shown on the diagram because we assume that all objects have operations that produce the values of their attributes.

- **An update operation**
  - will change the value of some or all of the object’s attributes, which may result in a change in the object’s state.
  - Consider changing the status of a patient from new to current with a method called “change status ()”, or associating a patient with a particular appointment with “schedule appointment (appointment).”

- **Relationships**
  - A primary purpose of the class diagram is to show the relationships, or associations, that classes have with one another.
    - depicted on the diagram by drawing lines between classes.
  - When multiple classes share a relationship (or a class shares a relationship with itself),
    - a line is drawn and labeled with either the name of the relationship or the roles that the classes play in the relationship.
      - For example, in the previous figure the two classes Patient and Appointment are associated with one another whenever a patient schedules an appointment.
      - Thus, a line labeled “schedules” connects patient and appointment, representing exactly how the two classes are related to each other.
Elements of a Class Diagram

- Sometimes a class is related to itself,
  – as in the case of a patient being the primary insurance carrier for
    other patients (e.g., their spouse, children).
- In the previous figure, notice that a line was drawn between the Patient class
  and itself, and is called “primary insurance carrier” to depict the role that
  the class plays in the relationship.
- Notice that a plus “+” sign is placed before the label to communicate that it is
  a role as opposed to the name of the relationship.
- When labeling an association, use either a relationship name or
  a role name (not both),
  – whichever communicates a more thorough understanding of the
    model.

Elements of a Class Diagram

- Relationships also have multiplicity,
  – documents how an instance of an object can be associated with
    other instances.
- Numbers are placed on the association path to denote the
  minimum and maximum instances that can be related through
  the association in the format
  \[\text{minimum number . . maximum number}\].
  – The numbers specify the relationship from the class at the far end
    of the relationship line to the end with the number.
  - For example, in the figure, there is a “0 . . *” on the appointment end of the
    “patient schedules appointment” relationship, meaning that a patient can be
    associated with zero through many different appointments.
  - At the patient end of this same relationship there is a “1,” meaning that an
    appointment must be associated with one and only one (1) patient.

Elements of a Class Diagram

- There are times when a relationship itself has associated
  properties,
  – especially when its classes share a many-to-many relationship.
  – In these cases, a class called an association class is formed,
  - association class
    has its own attributes and operations.
  - It is shown as a rectangle attached by a dashed line to the
    association path, and the rectangle’s name matches the label of
    the association.
  - Most often, classes are related through a “normal” association;
    – however, two special cases of an association will appear quite
    often:
      - generalization
      - aggregation.

Elements of a Class Diagram

- Generalization
  - shows that one class (subclass) inherits from another class
    (superclass),
  – meaning that the properties and operations of the superclass are
    also valid for objects of the subclass.
  - The generalization path is shown with a solid line from the
    subclass to the superclass and a hollow arrow pointing at the
    superclass.
  – For example,
    - the figure tells us that doctors, nurses, and administrative personnel are all
      kinds of employees and that these employees and patients are kinds of
      persons.
    - The generalization relationship occurs when you need to use
      words such as “is a-kind-of” to describe the relationship.

Elements of a Class Diagram

- Aggregation
  - used when classes actually comprise other classes.
  – For example,
    - think about a doctor’s office that has decided to create health care teams that
      include doctors, nurses, and administrative personnel.
    - As patients enter the office, they are assigned to a health care team that cares
      for their needs during their visits.
    - A diamond is placed nearest the class representing the aggregation
      (health care team), and lines are drawn from the arrow to connect
      the classes that serve as its parts (doctors, nurses, and
      administrative staff).
  - Typically, you can identify these kinds of associations when you
    need to use words like “is a part of” or “is made up of” to
    describe the relationship.
• When a class diagram is fully populated with all of the classes and relationships for a real world system, the class diagram can become very complex. When this occurs, it is sometimes necessary to simplify the diagram. Using a view mechanism can simplify the class diagram. Views – developed originally with relational database management systems – a subset of the information contained in the database.

• In this case, the view would be a useful subset of the class diagram, such as a use-case view that shows only the classes and relationships that are relevant to a particular use case. A second view could be to show only a particular type of relationship: – aggregation, association, or generalization. A third type of view is to restrict the information shown with each class, – for example, only show the name of the class, the name and attributes, or the name and operations. These view mechanisms can be combined to further simplify the diagram.

• A second approach to simplify a class diagram through the use of packages (i.e., logical groups of classes). – To make the diagrams easier to read and to keep the models at a reasonable level of complexity, you can group the classes together into packages. Packages – general constructs that can be applied to any of the elements in UML models. – are used to simplify use-case diagrams. In the case of class diagrams, it is simple to sort the classes into groups based on the relationships they share.

• An object diagram – an instantiation of all or part of a class diagram. Instantiation means to create an instance of the class with a set of appropriate attribute values. Object diagrams can be very useful when one is trying to uncover details of a class. – it is easier to think in terms of concrete objects (instances) rather than abstractions of objects (classes). For example, in the following figure, a portion of the class diagram in the previous figure has been copied and instantiated. The top part of the figure simply is a copy of a small view of the overall class diagram. The lower portion is the object diagram that instantiates that subset of classes.

• By reviewing the actual instances involved, John Doe, Appt1, and Dr. Smith, we may discover additional relevant attributes, relationships, and/or operations or possibly misplaced attributes, relationships, and/or operations. For example, – an appointment has a reason attribute. – Upon closer examination, the reason attribute may have been better modeled as an association with the Symptom class. – Currently, the Symptom class is associated with the Patient class. – After reviewing the object diagram, this seems to be in error. – As such, we should modify the class diagram to reflect this new understanding of the problem.
Creating a structural model is an iterative process whereby the analyst makes a rough cut of the model then refines it over time. Structural models can become quite complex. In fact, some systems have class diagrams that contain hundreds of classes.

**Object Identification**

- **Textual Analysis**
  - an analysis of the text in the use-case descriptions.
  - The analyst starts by reviewing the use-case descriptions and the use-case diagrams.
  - The text in the descriptions is examined to identify potential objects, attributes, operations, and relationships.
  - The nouns in the use case suggest possible classes.
  - The verbs suggest possible operations.
  - Following figure presents a summary of guidelines.
  - The textual analysis of use-case descriptions has been criticized as being too simple,
    - but because its primary purpose is to create an initial roughcut structural model, its simplicity is a major advantage.

- **Common Object List**
  - a list of objects that are common to the business domain of the system.
  - In addition to looking at the specific use cases,
    - the analyst also thinks about the business separately from the use cases.
  - Several categories of objects have been found to help the analyst in creation of the list,
    - such as
      * physical or tangible things,
      * incidents,
      * roles
      * interactions.

- **Tangible things** in the business domain
  - include books, desks, chairs, and office equipment.

- **Incidents**
  - events that occur in the business domain, such as meetings, flights, performances, or accidents.

- **Reviewing the use cases** can readily identify the roles that
  - the people “play” in the problem, such as doctor, nurse, patient, or receptionist.

- **An interaction**
  - a transaction that takes place in the business domain, such as a sales transaction.

- **Other types of objects** that can be identified include places, containers, organizations, business records, catalogs, and policies.

**Textual Analysis Guidelines**

- A common or improper noun implies a class of objects.
- A proper noun or direct reference implies an instance of a class.
- A collective noun implies a class of objects made up of groups of instances of another class.
- An adjective implies an attribute of an object.
- A doing verb implies an operation.
- An object implies a classification relationship between an object and its class.
- A being verb implies an aggregation or association relationship.
- A receiving verb implies an operation.
- An instance implies an exception.
- A predicate or descriptive phrase implies an operation.
- An adverb implies an attribute of a relationship or an operation.

Patterns

- The idea of using patterns is relatively new in object-oriented systems development.
- A pattern can be defined as a useful group of collaborating classes that provide a solution to a commonly occurring problem.
- Since patterns provide a solution to commonly occurring problems, they are reusable.
- Many different types of patterns have been proposed, ranging from high-level business oriented patterns to more low-level design patterns.

Object Identification

- If you are developing a business information system in one of these business domains, then the patterns developed for that domain may be a very useful starting point in identifying needed classes and their attributes, operations, and relationships.
- Following figure lists some common business domains for which patterns have been developed, as well as their source.

<table>
<thead>
<tr>
<th>Business Domains</th>
<th>Sources of Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accounting</td>
<td>3, 4</td>
</tr>
<tr>
<td>Acros-Role</td>
<td>2</td>
</tr>
<tr>
<td>Assembly-Plan</td>
<td>1</td>
</tr>
<tr>
<td>Container-Current</td>
<td>1</td>
</tr>
<tr>
<td>Contract</td>
<td>2, 4</td>
</tr>
<tr>
<td>Document</td>
<td>2, 4</td>
</tr>
<tr>
<td>Employment</td>
<td>2, 4</td>
</tr>
<tr>
<td>Financial Derivative Contracts</td>
<td>3</td>
</tr>
<tr>
<td>Geographic Location</td>
<td>2, 4</td>
</tr>
<tr>
<td>Group-Member</td>
<td>1</td>
</tr>
<tr>
<td>Interaction</td>
<td>1</td>
</tr>
<tr>
<td>Material Requirements Planning</td>
<td>4</td>
</tr>
<tr>
<td>Organization and Party</td>
<td>2, 3</td>
</tr>
<tr>
<td>Plan</td>
<td>1, 3</td>
</tr>
<tr>
<td>Process Manufacturing</td>
<td>4</td>
</tr>
<tr>
<td>Trading</td>
<td>3</td>
</tr>
<tr>
<td>Transactions</td>
<td>1, 4</td>
</tr>
</tbody>
</table>


Steps for Object Identification and Structural Modeling

1. Create CRC cards by performing textual analysis on the use-cases.
2. Brainstorm additional candidate classes, attributes, operations, and relationships by using the common object list approach.
3. Role-play each use-case using the CRC cards.
4. Create the class diagram based on the CRC cards.
5. Review the structural model for missing and/or unnecessary classes, attributes, operations, and relationships.
6. Incorporate useful patterns.
7. Review the structural model.

Building CRC Cards and Class Diagrams

- Although CRC cards and class diagrams can be used to describe both the As-Is and To-Be structural model of the evolving system, they are most often used for the To-Be model.
- There are many different ways to identify a set of candidate objects and to create CRC cards and class diagrams.
- Today most object identification begins with the use cases identified for the problem.
- Processes used to create the structural model of the problem are listed in the next slide.

Create CRC Cards
- This is done by performing textual analysis on the use case descriptions.
- The normal flow of events, subflows, and alternate/exceptional flows written as part of the use case were written in a special form called Subject Verb Direct object Preposition Indirect object (SVDPI).
- By writing use-case events in this form, it is easier to use the guidelines for use-case analysis to identify the objects.
Building CRC Cards and Class Diagrams

- Reviewing the primary actors, stakeholders, and interests, and giving a brief description of each use case allow additional candidate objects to be identified.
- Furthermore, it is useful to go back and review the original requirements to look for information that was not included in the text of the use cases.
- Record all information uncovered for each candidate object on a CRC card.

Building CRC Cards and Class Diagrams

- Examine Common Object Lists
  - The second step to brainstorm additional candidate objects, attributes, operations, and relationships using the common object list approach.
    - What are the tangible things associated with the problem?
    - What are the roles played by the people in the problem domain?
    - What incidents and interactions take place in the problem domain?

Building CRC Cards and Class Diagrams

- Many of these questions already have partial answers to them.
  - For example,
    - the primary actors and stakeholders are the roles that are played by the people in the problem domain.
  - However, it is possible to uncover additional roles that were not thought of previously.
- As in the previous step, be sure to record all information uncovered onto the CRC cards.
  - This includes any modifications uncovered for any previously identified candidate objects and any information regarding any new candidate objects identified.

Building CRC Cards and Class Diagrams

- Role-Play the CRC Cards
  - The third step to role-play each use case using the CRC cards.
  - Each CRC Card should be assigned to an individual, who will perform the operations for the class on the CRC card.
    - As the “performers” play out their roles, the “system” will tend to break down.
    - When this occurs, additional objects, attributes, operations, or relationships will be identified.
  - Again, like the previous steps, anytime any new information is discovered, new CRC cards are created or modifications to existing CRC cards are made.

Building CRC Cards and Class Diagrams

- Create the Class Diagram
  - The fourth step to create the class diagram based on the CRC cards.
    - This is equivalent to creating the use-case diagram from the use case descriptions.
  - Information contained on the CRC cards is simply transferred to the class diagrams.
  - The responsibilities are transferred as operations, and the attributes as attributes, and the relationships are drawn as generalization, aggregation, or association relationships.

Building CRC Cards and Class Diagrams

- However, the class diagram also requires that the visibility of the attributes and operations be known.
  - As a general rule of thumb, attributes are private and operations are public.
  - Therefore, unless the analyst has a good reason to change the default visibility of these properties, then the defaults should be accepted.
  - Finally, the analyst should examine the model for additional opportunities to use aggregation or generalization relationships.
  - As in the previous steps, any and all changes must be recorded on the CRC cards.
Building CRC Cards and Class Diagrams

• **Review the Class Diagram**
  • The fifth step
    – to review the structural model for missing and/or unnecessary classes, attributes, operations, and relationships.
  • Up until this step, the focus of the process has been on adding information to the evolving model.
  • At this point in time, the focus begins to switch from only adding information to also challenging the reasons for including the information contained in the model.

Building CRC Cards and Class Diagrams

• **Incorporate Patterns**
  • The sixth step
    – to incorporate useful patterns into the evolving structural model.
  • A useful pattern is one that would allow the analyst to more fully describe the underlying domain of the problem being investigated.
  • After identifying the useful patterns, the analyst incorporates the identified patterns into the class diagram and modifies the affected CRC cards.
    – This includes adding and removing classes, attributes, operations, and/or relationships.

Building CRC Cards and Class Diagrams

• **Review the Model**
  • The final step
    – to review the structural model, including both the CRC cards and the class diagram.
  • This is best accomplished during a formal review meeting using a walkthrough approach
    – in which an analyst presents the model to a team of developers and users.
  • The analyst “walks through” the model explaining each part of the model and all of the reasoning that went into the decision to include each class in the structural model.

Building CRC Cards and Class Diagrams

• **This explanation includes justifications for the attributes, operations, and relationships associated with the classes.**
• **Finally, each class should be linked back to at least one use case;**
  – otherwise, the purpose of including the class in the structural model will not be understood.
• **It is often best for the review team to also include people outside the development team that produced the model**
  – because these individuals can bring a fresh perspective to the model and uncover missing objects.

Summary

• **CRC cards** capture the essential elements of a class.
• **Class and object diagrams** show the underlying structure of an object-oriented system.
• Constructing the structural model is an iterative process involving: textual analysis, brainstorming objects, role playing, creating the diagrams, and incorporating useful patterns.

Expanding the Domain

• A quirky and interesting tutorial regarding CRC cards can be found at:
  • [http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/crc_b/](http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/crc_b/)