Suggestions for Doing a Problem Domain Analysis

From Object-Oriented Modeling and Design, Chapter 8, James Rumbaugh (1991) with a few additions.

**Identifying Classes**

The first step in constructing an object model is to identify relevant objects in the problem domain. Objects include physical entities, such as houses, employees, and machines, as well as concepts, such as trajectories, seating assignments, and payment schedules. Generalize the objects into classes. For example, the problem domain may contain a customer and checkout clerk; these can be generalized to a single class called Person. **Classes often correspond to nouns.**

Next, discard unnecessary and incorrect classes while keeping the right classes (if in doubt, keep a class – it can be discarded at a later time)

- **redundant classes**: if two classes express the same information, the most descriptive name should be kept.
- **irrelevant classes**: if a class has little or nothing to do with the problem, it should be eliminated.
- **vague classes**: a class should be specific; drop or rename tentative classes that have ill-defined boundaries or are too broad in scope.
- **attributes**: (often correspond to nouns or noun phrases) simple, one-dimensional quantities such as name, age, and weight should be implemented as attributes. More complex, multidimensional quantities such as a student's schedule should be modeled as a separate class and related to other classes through association or aggregation. Other situations are more ambiguous and the determination is based on usage. For example, an address may be implemented as a series of attributes: street, city, and zip code. On the other hand, if an address must *do something*, the should be implemented as a separate class with operations or behaviors.

**Identifying Associations**

Associations often correspond to stative verbs or verb phrases in. Here is a list of clues (i.e., verbs and verb phrases) that will help identify associations.

- physical location (*next two, part of, contained in*)
- directed action (*drives*)
- communication (*talks to*)
- ownership (*has, part of, contains*)
- satisfaction of some condition (*works for, married to, manages, oversees*)

Extract all the candidates from the problem statement and get them down on paper first, don't try
to refine things too early. Don't spend too much time trying to distinguish between association, aggregation, and composition. Use whichever seems most natural at the time and move on; if the relationship becomes more clear later, you can modify the model at that time.

Discard unnecessary or incorrect associations:

- **Associations between eliminated classes**: if one of the classes in the association has been eliminated, then the association must be eliminated or restated in terms of other classes.
- **Irrelevant associations**: eliminate any associations that are outside the problem domain.
- **Actions**: an association should describe a structural property of the problem domain and not a transient event.
- **Misnamed associations**: don’t say how or why a situation came about, say what it is.

**Identifying Attributes**

Attributes usually correspond to noun is followed by possessive phrases, such as "the color of the car" or "the position of the cursor." Adjectives often represent specific enumerated attribute values, such as red, on, or expired. (Enumerated types are also often called symbolic constants or named constants.)

Keeping the right attributes

- **objects**: if the independent existence of an entity is important, rather than just its value, then it is an object and should be represented by a class.

**Refining Inheritance**

The next step is to organize classes by using inheritance to share a common structure (i.e., attributes and operations shared by two or more classes). Inheritance can be added in two directions

- **generalizing common aspects of existing classes into a superclass (bottom-up)**
  - Discover inheritance from the bottom up by searching for classes with similar attributes, associations, or operations.
  - Sometimes the superclass may not really exist in the problem domain; in this case it may be useful to invent an abstract superclass.
- **refining existing classes into specialized subclasses (top-down)**
  - Top-down specializations are often apparent from the problem domain. Look for noun phrases composed of various adjectives on the class name.