CS211 Lecture: Identifying Objects, Classes, and Responsibilities; CRC Cards

last revised July 28, 2003

Objectives:

1. To show how to use CRC cards to identify objects and find responsibilities

Materials:

1. ATM System example on the web.
2. Handouts from previous session on analysis, including Session Use Case
3. Supply of 4x6 cards for CRC cards
4. Handout of Use Cases and Analysis Classes for CS112 Address Book problem
5. Handout of CRC cards for CS112 Address Book problem

I. Introduction

A. We have seen that, regardless of what software development model is followed, certain tasks will need to be done as part of the development process per se - whether all at once, iteratively, or incrementally.

1. Analysis. The goal of this task is to understand the problem.

2. Design. The goal of this task is to develop the overall structure of a solution to the problem in terms of individual, buildable components and their relationships to one another.

3. Implementation. The goal of this task is to actually build the system as designed.

4. Quality Assurance The goal of this task is to ensure that the individual components and the system as a whole do what they are supposed to do (which involves identifying their shortcomings and fixing them.)

5. Deployment / Maintenance: Actually using the software and making changes as necessary.

B. Today’s lecture begins our discussion with the transition between analysis and the design tasks.

C. In traditional software development methodologies, the line between analysis and design is fairly sharp, because very distinct notations are used for documentation at each stage. In OO methodologies, the line is less
distinct, because similar notations can be used at both stages. The seamlessness of OO tools at the transitions between the various phases of the software lifecycle is a strength of the OO paradigm.

D. The approach that we will take to design is called a *use-case driven approach*, because we will use the use cases identified during analysis to drive the design process. Our approach will be as follows:

1. Develop the class structure for the system
   
   a) Identify the classes that need to be part of the system.
   
   b) Assign responsibilities to each class. Each responsibility that must be fulfilled to accomplish the use cases must be assigned to some class - the focus of this set of lectures.
   
   c) Identify the relationships between various classes - a topic we will begin to deal with in this set of lectures, and then will pursue further in the next set of lectures.

2. In a subsequent set of lectures, we will deal with the process of detailed design of the various classes. For each class, we must determine:
   
   a) Attributes comprising the state of objects of that class.
   
   b) Interactions between objects of that class and other objects (of the same or different classes).
   
   c) Operations that can be performed by objects of that class.

3. Since large systems may include hundreds or thousands of classes, some partitioning of classes into subsystems (packages) is often necessary. This is a portion of the design process we will not discuss until later, though.

E. As we do the design, we will often discover the need for additional objects and classes, to facilitate the implementation of the objects we discovered during analysis. (Booch et. al. observe that there may be a 5:1 ratio between classes discovered at analysis time and classes ultimately needed to implement a system.)
II. Approaches to Identifying Classes

A. Domain Analysis - and, in particular, the analysis class diagram - will give us a sense of what some of the key classes in the system will be.

B. One additional approach to identifying classes that is sometimes simplistic, but yet is often useful, is called noun extraction. The basic idea is this: read over the system requirements/use case flows of events, and note the nouns that appear.

1. Some of the nouns that appear - especially the ones that appear frequently - will turn out to refer to objects that need to be represented by classes in the final system.

2. Other nouns will turn out to be attributes of objects, rather than objects in their own right. An important skill to develop is being able to distinguish the two. Recall that objects have three essential characteristics:

   \textit{ASK}

   a) Identity.
   b) State (often complex - i.e. involving more than a simple value).
   c) Behavior

3. It is also important to note that some nouns may refer to entities outside the scope of the system (actors) which will not need to be represented by classes inside the system.

4. \textit{Example}: Perform noun extraction using the flow of events for the Session use case.

\textit{NOTE:}

a) Some nouns that probably \textbf{will} become classes:

(1) Session
(2) ATM Card
(3) Card Reader
(4) Console used by customer to enter information
(5) Transaction

b) Some nouns that probably \textbf{will not} become classes:

(1) Customer - WHY? Outside the scope of this system - an actor
(2) PIN - WHY? A simple value - lacks complex state, behavior, or identity.

C. It is important to recognize that identifying classes is not something we do once and then never change. As the design process proceeds, we should be prepared to:

1. Add additional classes that we discover the need for

2. Reconfigure classes identified previously as we develop a clearer sense of what their responsibilities will be.

III. Assigning Responsibilities to Classes: CRC Cards

A. Once we have some notion of the key classes that the objects comprising the system will belong to, we can begin determining what responsibilities each class will fulfill.

B. One tool that we can use to help us do this is called CRC Cards (CLASS, RESPONSIBILITY, COLLABORATOR). CRC cards are not a formal part of UML, but are commonly used as a vehicle for doing design that is then documented using UML diagrams.

1. A CRC card is a card (generally about 4 x 6) containing at the top the name of a class, followed by two parallel lists.

   a) The list on the left hand side lists the responsibilities of the class.
   b) The list on the right hand side lists the other classes (if any) with which this class must collaborate to carry out each task.

2. To get started, we can create a CRC card for each of the classes we discovered during the analysis phase. As we discover the need for additional classes, we can create additional CRC cards.

**EXAMPLE:** We will develop a partial set of CRC cards for the ATM example in class. For the portion we are doing, we will need cards for the following classes:

   a) ATM
   b) Session
   c) Transaction
   d) CardReader
   e) CustomerConsole
   f) Card
3. A typical way to use CRC cards is to "walk through" the various use cases, identifying tasks that need to be performed and assigning the responsibility for each to an appropriate class, by recording it in the “responsibility” column of the appropriate card.

   a) The use case itself is made a responsibility of some class.

   b) The classes that are called upon to perform specific responsibilities as part of the use case become collaborators, noted in the “Collaborators” column of the card for the class that is responsible for the use case.

   c) In addition, each collaborator class gets one or more responsibilities listed in the “Responsibilities” column of its card - which may in turn, lead to identifying further collaborators it needs, etc.

4. The key question to ask for each operation we find in the use cases is "what class should be responsible for this?" Often there will be more than one possible answer, so the different alternatives need to be examined carefully before a choice is made.

5. This process lends itself particularly well to a group of people working together, with individual members of the group role-playing various classes. (Remember, in an OO system the basic computational model is one of different objects sending messages to each other. We represent this by having the person who is role playing a class that needs some task perform asking the representative of an appropriate collaborating class to perform it.)

C. EXAMPLE: Walk through session use case.

   Ask several students to role play the various classes. Fill in CRC cards on board as classes get responsibilities or collaborators.

   NOTE AT OUTSET: There is no one best way to make the responsibility assignments. I made certain choices in developing the example, and we will work with those so that everything hangs together.

   1. The use case flow of events for this case begins “A session is started when a customer inserts an ATM card into the card reader slot of the machine..."
a) An obvious assignment of responsibilities is to have a Session object that is responsible for performing the Session use case.

(Note on card)

b) However, the Session object cannot be responsible for starting the session use case.  WHY?

ASK

A session object is not even created until the use case is begun. Thus, at the very beginning of the use case, there is no session object in existence as yet!

c) So what class should be responsible for starting a session when the card is inserted?

ASK - be sure to get both ATM and CardReader

For our purposes, we will make the CardReader responsible to tell the ATM that a card has been inserted. Then, the ATM will be responsible for actually creating the session.

Put responsibility to inform ATM on CardReader card, with ATM as collaborator; and give ATM responsibility to start a session when card is inserted on ATM card.

d) What class(es) does ATM need as collaborators for this task?

ASK - be sure to get:

(1) Session (The Session constructor is used to actually create the Session object.)

(2) CustomerConsole (for message telling user to insert card)
   Enter the above on card for ATM

(3) Note that CardReader is not made a collaborator of ATM, but rather the other way around - CardReader makes use of a service to ATM (responding to insertion of the card.)

e) What other classes get responsibilities as a result of this?

ASK - NOTE ON CARDS
(1) Session has already been given the responsibility of performing the Session use case - otherwise, we would have to add that to its card now.

(2) CustomerConsole is made responsible for displaying a message to the customer.

2. At this point, we continue the flow of events in the use case, understanding that the newly-created Session object is now responsible for carrying the use case out, making use of other classes as needed.

3. The first thing that must happen is that the card must actually be read. Continuing with the use case flow of events: “The ATM pulls the card into the machine and reads it.”

What collaborators does Session use to get this job done?

a) (class) Card - when we read the card, we create a Card object that contains information about it.

   Card is added as a collaborator of Session, and gets a responsibility on its own CRC card - to represent information about a customer’s ATM card.

b) CardReader (to read actual information from the card).

   Note: The flow of events says “the ATM pulls the card ...”; but in the design, we make this a responsibility of a component part of the ATM - the card reader - not of the ATM itself. From the perspective of one using the system, it looks like the ATM is reading the card - but from the vantage point of design, the actual task is given to the card reader.

   (1) This gives rise to CardReader being a collaborator of Session (add to card).

   (2) This gives rise to a responsibility of CardReader (to actually read the card.). For this responsibility, CardReader also makes use of Card as a collaborator. (In fact, it creates the Card object which it then gives to the session.)

c) A design decision that I made in this system is to give the ATM object responsibility for providing access to its component parts when Sessions and Transactions need this access. (e.g. a Session
object asks the ATM object to give it a reference to the CardReader object).

(1) This makes ATM a collaborator of Session. (Add to CRC)

(2) This gives ATM a responsibility - to provide access to component parts. (Add to CRC)

4. What if the card proves to be unreadable? The flow of events says three things must occur. “(If the reader cannot read the card due to improper insertion or a damaged stripe, the card is ejected, an error screen is displayed, and the session is aborted.)”

a) Who should be responsible for ejecting the bad card?

   ASK - This one’s pretty clear - the CardReader!

   Note this responsibility on its card.

b) Who should be responsible for telling the user the card is bad?

   ASK - Again - obvious - the CustomerConsole

   Since the customer console has already been given a responsibility for displaying messages to the customer, no new responsibility needs to be added here.

c) The aborting of the Session is easy: the relevant method just terminates.

5. Now the flow of events goes on to say “The customer is asked to enter his/her PIN”.

a) What class(es) does the Session need as collaborator(s)?

   ASK

   CustomerConsole

b) Add CustomerConsole as a collaborator for Session, and add responsibility to read a PIN as a responsibility of CustomerConsole.

6. The flow of events continues by saying that the customer “is then allowed to perform one or more transactions, choosing from a menu of possible types of transaction in each case.”
a) What class should be responsible for offering the customer the list of choices?

ASK

(1) Could be the Session.

(2) Could be class Transaction. We will go this route, since this puts knowledge about the possible types of transactions in this class (which needs to have it anyway) without burdening Session with this knowledge.

Add responsibility to Transaction.

b) What collaborators does Transaction need for this task?

(a) CustomerConsole
(b) ATM (to provide access to console)
(c) Constructors of appropriate subclass: Withdrawal, Deposit, Transfer, Inquiry)

Note on card for Transaction.

Add accept choice from a menu as a responsibility of CustomerConsole.

c) Since performing a transaction use case has a separate flow of events, we will defer developing details until later, but will note this as a responsibility of Transaction now.

Add to CRC card

7. The flow of events continues: “After each transaction, the customer is asked whether he/she would like to perform another.”

We will fold this into the transaction use case responsibility.

8. The flow of events continues: “When the customer is through performing transactions, the card is ejected from the machine and the session ends.”

We have already made ejecting a card a responsibility of CardReader
9. The flow of events ends by saying “If a transaction is aborted due to too many invalid PIN entries, the session is also aborted, with the card being retained in the machine.

This adds a “retain card” responsibility to CardReader.

D. SHOW ON THE WEB - my CRC cards for ATM system - GO OVER

IV. A Class Exercise:

A. We will now do a class exercise in which we will generate CRC cards for the AddressBook problem done in CS112.

B. Preparation:

1. HANDOUT: Use cases and analysis classes for this problem (Go over)

2. DISTRIBUTE 4X6 CARDS

3. Assign people to work in groups of three:

a) Assign each team one of the three uses cases.

b) roles as follows

   (1) The appropriate boundary classes (MainWindow, AddDialog, EditDialog, or ConfirmDialog - same person but separate cards)
   (2) Controller class - one person, one card
   (3) The two entity classes (AddressBook and Person - same person but separate cards)

C. Directions:

Act out the assigned scenario, with the person responsible for the currently active class holding up the card and speaking for it - e.g. in the example we just did, when the Session needs to read a PIN, the person responsible for the Session card would hold it up and say: "I need a PIN. CustomerConsole, read one for me". Then the person responsible for the CustomerConsole card would hold it up and say "I am reading a PIN from the customer"... As a consequence, CustomerConsole would appear on the Session card as a collaborator, and "get a PIN from customer" would appear on the CustomerConsole card as a responsibility.

NOTE: Assume that the MainWindow is responsible for initially notifying the Controller when a button is clicked, and that the Controller is then
responsible for performing the use case, calling on other classes as collaborators as needed.

D. Allow time for work, then have teams share results. Get one team to “act out” each use case.

E. HANDOUT AT END: My CRC cards for this problem