CRC Modeling:
Bridging the Communication Gap Between Developers and Users

An AmbySoft Inc. White Paper

Scott W. Ambler
Software Process Mentor
AmbySoft Inc.

Material for this paper has be excerpted from:

The Object Primer: The Application Developer's Guide to Object-Orientation

- and -

Process Patterns: Building Large-Scale Systems Using Object Technology

This Version: November 29, 1998

Table of Contents

This white paper describes the CRC (class responsibility collaborator) modeling process for identifying user requirements. CRC modeling is an effective, low-tech method for developers and users to work closely together to identify and understand business requirements.

Class Responsibility Collaborator Cards

A CRC card (Beck & Cunningham, 1989; Ambler, 1995) is a standard index card that has been divided into three sections, as shown below in Figure 1.

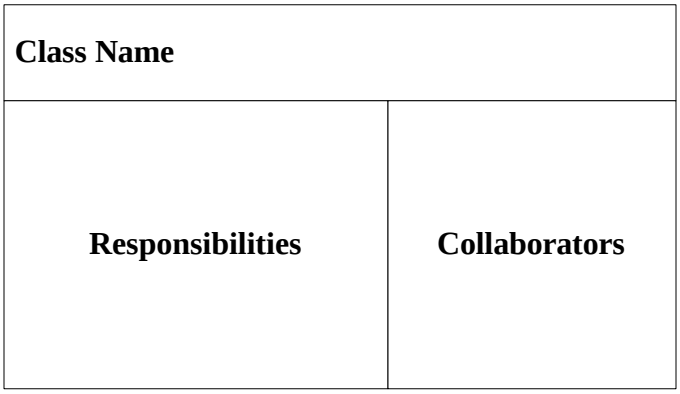

Figure 1. CRC card layout.

A class represents a collection of similar objects. An object is a person, place, thing, event, concept, screen, or report that is relevant to the system at hand. For example, Figure 2 shows a shipping/inventory control system with the classes such as **Inventory Item**, **Order**, **Order Item**, **Customer**, and **Surface Address**. The name of the class appears across the top of the card.

A responsibility is anything that a class knows or does. For example, customers have names, customer numbers, and phone numbers. These are the things that a customer knows. Customers also order products, cancel orders, and make payments. These are the things that a customer does. The things that a class knows and does constitute its responsibilities. Responsibilities are shown on the left hand column of a CRC card.

Sometimes a class will have a responsibility to fulfill, but will not have enough information to do it. When this happens it has to collaborate with other classes to get the job done. For example, an **Order** object has the responsibility to calculate it's total. Although it knows about the **Order Item** objects that are a part of the order, it doesn't know how many items were ordered (**Order Item** knows this) nor does it know the price of the item (**Inventory Item** knows this). To calculate the order total, the **Order** object collaborates with each **Order Item** object to calculate its own total, and then adds up all the totals to calculate the overall total. For each **Order Item** to calculate its individual total, it has to collaborate with **Inventory Item** to determine the cost of the ordered item, multiplying it by the number ordered (which it does know). The collaborators of a class are shown in the right-hand column of a CRC card.

CRC Models

A CRC model is a collection of CRC cards that represent whole or part of an application or problem domain. The most common use for CRC models, the one that this white paper addresses, is to gather and define the user requirements for an object-oriented application[1]. Figure 2 presents an example CRC model for a shipping/inventory control system, showing the CRC cards as they would be placed on a desk or work table. Note the placement of the cards: Cards that collaborate with one another are close to each other, cards that don't collaborate are not near each other.

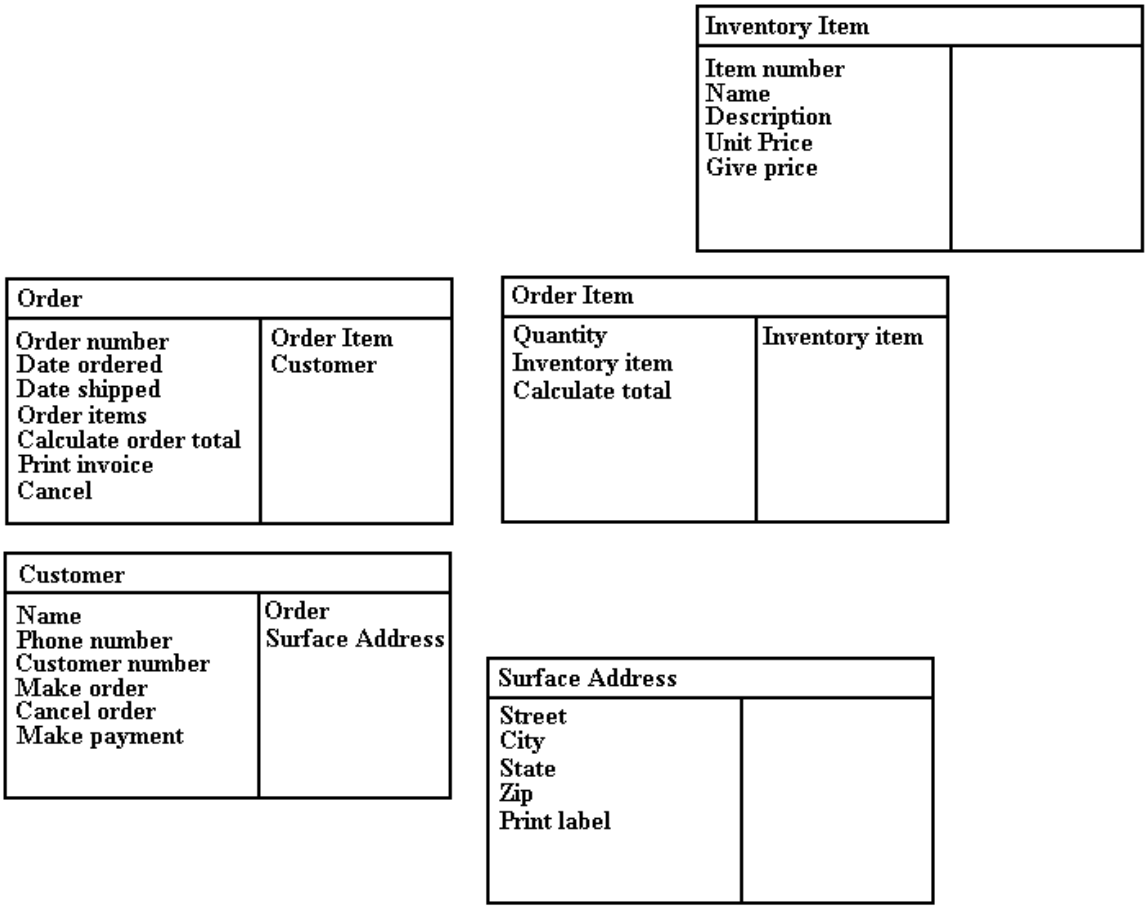

Figure 2. A CRC model for a simple shipping/inventory control system.

CRC models are created by groups of business domain experts, led by a CRC facilitator who is assisted by one or two scribes. The CRC facilitator is responsible for planning and running the CRC modeling session. In the next section we will discuss the CRC modeling process in detail.

[1] Other common uses for CRC cards are to explain OO concepts to novice OO developers (Beck & Cunningham, 1989) and to work through the design of a small portion of an OO application.

CRC Modeling in a Nutshell

So how do you create a CRC model? There are six steps to CRC modeling, which are:

1. Put together the CRC modeling team.
2. Organize the modeling room.
3. Do some brainstorming.
4. Explain the CRC modeling technique.
5. Iteratively perform the steps of CRC modeling.
6. Perform use-case scenario testing.

Putting Together the CRC Modeling Team

There are four different roles taken in a CRC modeling session:

1. **Business Domain Experts (BDEs**). These are the real users of the system and occasionally a senior developer with years of experience in the problem domain. However, never forget that users do the work day in and day out – They have the business domain knowledge. There are typically four or five BDEs involved in CRC Modeling. If you have less than four you'll find that you won't get a wide enough range of knowledge and experience, but once you get beyond five or six people you'll find that there are too many people involved and that they're getting in each others way. Good BDEs know the business, think logically, can communicate well, and are willing to invest their time in the development process.
2. **Facilitator.** This is the person who runs the session. The main role of the facilitator is to communicate what CRC modeling is all about, make sure that the cards are filled out properly, ask pertinent questions during modeling, and to recognize when prototyping needs to be done and to lead the prototyping effort. Facilitators need good communication and technical skills. The facilitator is often a trained meeting facilitator.
3. **Scribe(s).** You should have one or two scribes in the room. Scribes take down the detailed business logic that isn't captured on the CRC cards. For example, on the **Order** card we might indicate that it knows how to apply a discount, but what we haven't recorded is the logic to do so. This is the type of information that scribes record. Scribes do not actively participate in the session, although may ask questions to confirm the business rules or processing logic that they are recording. Scribes must be able to listen well, have good oral and written communication skills, and should have a good ear for business logic.
4. **Observers.** For training purposes, you may wish to have one or more people sit in on the CRC modeling session as observers. These people sit at the back of the room and do not participate in the session.

Organize the Room

There are several things that need to be done to organize a CRC modeling room:
1. **Reserve a meeting room that has something write on.** You need a flip chart and/or a whiteboard to write on to do brainstorming and prototyping. Flip charts are great because you can tape your sketches on the wall. Whiteboards are great for drawing quick pictures that you might have to edit.
2. **Bring CRC modeling supplies.** You'll need a couple of packages of index cards and some whiteboard markers. If you're going to do use-case scenario testing you'll also need a soft, spongy ball.
3. **Have a modeling table.** There should be a large table for people to work on.
4. **Have chairs and desks for the scribe(s).** Put them to the back or sides of the room where they are out of the way but can still see what is going on.
5. **Have chairs for the BDEs.** People will want to sit down during the session, so make sure you have enough chairs.
6. **Have chairs for the observers.** If there are observers, put them in the back of the room. Observers aren't there to participate, so it is valid to put them towards the back of the room.

Brainstorm

Brainstorming is an idea-generation technique, not an idea-evaluation technique, which your modeling team will use to identify and understand the requirements for the application they are building. It's ok, and even encouraged, to build on the ideas of others. Potential questions that can be used to generate ideas during brainstorming include:

- Who is this system for?
- What will they do with the system?
- Why do we do this?
- Why do we do this the way that we do?
- What business needs does this system support?
- What do/will our customers want/demand from us?
- How is the business changing?
- What is our competition doing? Why? How can we do it better?
- Do we even need to do this?
- If we were starting from scratch, how would we do this?
- Just because we were successful in the past doing this, will we be successful in the future?
- Can we combine several jobs into one? Do we want to?
- How will people's jobs be affected? Are we empowering or disempowering them?
- What information will people need to do their jobs?
- Is work being performed where it makes the most sense?
- Are there any trivial tasks that we can automate?
- Are people performing only the complex tasks that the system can't handle?
- Will the system pay for itself?
- Does the system support teamwork, or does it hinder it?
- Do our users have the skills/education necessary to use this system? What training will they need?
- What are our organization's strategic goals and objectives? Does this system support them?
- How can we do this faster?
- How can we do this cheaper?
- How can we do this better?

Explain the CRC Modeling Technique

Once brainstorming is finished the facilitator should describe the CRC modeling process. This usually takes between ten and fifteen minutes and will often include the creation of several example CRC cards. Because people learn best by doing it is a very good idea for the facilitator to lead the BDEs through the creation of the example CRC cards.

Perform The Iterative Steps of CRC Modeling

The group of BDEs stand and/or sit around a large table and fill out the CRC cards. While they'll spend most of their time sitting and discussing the system, BDEs will often stand up to get a bird's eye view of their model, allowing them to get a better feel for the overall application. The steps of CRC modeling are:

- Find classes
- Find responsibilities
- Define collaborators
- Define use-cases
- Arrange the cards on the table

Finding Classes

- Look for anything that interacts with the system, or is part of the system
- Ask yourself "Is there a customer?"
- Follow the money
- Look for reports generated by the system
- Look for any screens used in the system
- Immediately prototype interface and report classes
- Look for the three to five main classes right away
- Create a new card for a class immediately
- Use one or two words to describe the class
- Class names are singular

Finding Responsibilities

- Ask yourself what the class knows
- Ask yourself what the class does
- If you've identified a responsibility, ask yourself what class it "belongs" to
- Sometimes get responsibilities that we won't implement, and that's OK
- Classes will collaborate to fulfil many of their responsibilities

Defining Collaborators

- Collaboration occurs when a class needs information that it doesn't have
- Collaboration occurs when a class needs to modify information that it doesn't have
- There will always be at least one initiator of any given collaboration
- Sometimes the collaborator does the bulk of the work
- Don't pass the buck
- New responsibilities may be created to fulfill the collaboration

Defining Use-Cases

- The BDEs will identify them as responsibilities of actor classes
- Do some brainstorming
- Transcribe the scenarios onto cards

Moving the Cards Around

- Cards that collaborate with each other should be close to one another on the desk
- The more that two cards collaborate, the closer that they should be on the desk
- Expect to be moving the cards around a lot at the beginning
- Put "busy" cards towards the center of the table
- Actually move them around
- People will identify relationships/associations between classes as they move them around

Perform Use-Case Scenario Testing

Use-case scenario testing (Ambler, 1995; Ambler, 1998a; Ambler, 1998b) is a task process pattern, depicted in Figure 3, in which users are actively involved with ensuring that user requirements are accurate. The basic idea is that a group of business domain experts (BDEs), with the aid of a facilitator, step through a series of defined use cases to verify that CRC model accurately reflects their requirements. The facilitator distributes the cards to the BDEs, trying to ensure that two cards that collaborate are given to two different people (sometimes this is easier said than done). The facilitator then leads the group through acting out each scenario one at a time. There are six steps to acting out scenarios:

1. **Call out a new scenario.** Both the description of the scenario and the action(s) to be taken are called out by the facilitator. Once this is complete, the group must decide if this scenario is reasonable (remember, the system can’t handle some scenarios) and if it is which card is initially responsible for handling the scenario. The facilitator starts out with the ball, and throws it to the person holding that card. When the scenario is completely acted out the ball will be thrown back to the facilitator.
2. **Determine which card should handle the responsibility.** When a scenario has been described, or when the need for collaboration has been identified, the group should decide what CRC card should handle the responsibility. Very often there will already be a card that has the responsibility identified. If this isn’t the case, update the cards. Once the update is complete (if need be), whoever has the ball should throw it to the person with the card that has the responsibility.
3. **Update the cards whenever necessary.** If a card needs to be updated, one of two situations has arisen: The responsibility needs to be added to an existing card, or a new card needs to be created with that responsibility. If a responsibility needs to be added to an existing card, then ask yourself which card should logically have it, and then have the BDE with that card update it. If a new card needs to be created, the facilitator should hand a blank CRC card to one of the BDEs and ask them to fill it out. At the same time, you may also find that you need to update your prototype drawings as well (if an interface or report class changes, then the prototype may need to change as well).
4. **Describe the processing logic.** When the ball is thrown to someone, they should describe the business logic for the responsibility step-by-step. Think of it like this: The BDEs are effectively describing pseudo-code (high level program code) for the responsibility. This is often the most difficult part of use-case scenario testing, as some of your BDEs might not be used to documenting processes step-by-step. If this is the case with some of your BDEs, then the facilitator needs to help them through the logic. You’ll find that after running through the first few scenarios the BDEs will quickly get the hang of describing processing logic. As the BDE describes the processing logic, the scribe should be writing it down (remember, the job of the scribe is to record the business logic/rules for the system, which is exactly what the BDE is currently describing!)

5. **Collaborate if necessary.** As the BDE describes the business logic of the responsibility, they will often describe a step where they need to collaborate with another card to complete. That's great -- That's what use-case scenario testing is all about! If this is the case, go back to Step 2.
6. **Pass the ball back when done.** Eventually the BDE will finish describing the responsibility. When this happens, they should throw the ball back to whoever it was that originally threw it to them. This will be another BDE (remember, every time you need to collaborate you throw the ball to the person holding the card that you need to collaborate with) or to the facilitator (remember, the facilitator starts with the ball and throws it to the person holding the card that initially handles the scenario).

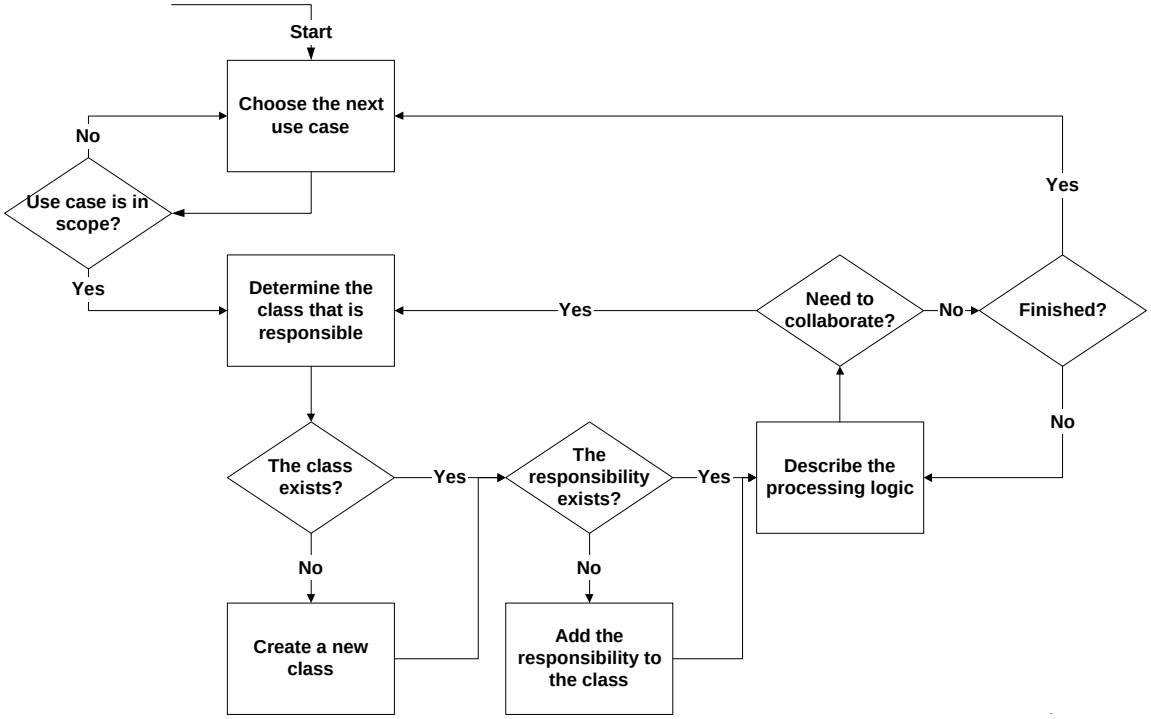

Figure 3. The Use-Case Scenario Testing process pattern (Ambler, 1998b).

The main point that needs to be made is that you need to verify that your model accurately reflects the problem domain. If it doesn't, then your project is in serious jeopardy. I would prefer to find this out early on in the project when I'm still in a position to do something about it rather than later when I probably can't. Wouldn't you?

How CRC Modeling Fits In

Although I argue in my second book, *Building Object Applications That Work* (Ambler, 1998a) that the four techniques shown in Figure 4 are completely interconnected, in general the definition of use cases and prototypes generally come before a CRC model, which in turn generally comes before a class diagram. The reason for this is simple: Use cases and user interface prototypes are less detailed than CRC models, which in turn are less detailed than class diagrams. Yes, while you are CRC modeling you will draw sketches, or prototypes of screens. Yes, while you are CRC modeling you will identify new use cases and/or update existing ones. In general, however, you will find that it is better to start out "simple" with use cases and prototypes, get a little more complicated with CRC cards, and then get even more complicated with class diagrams.

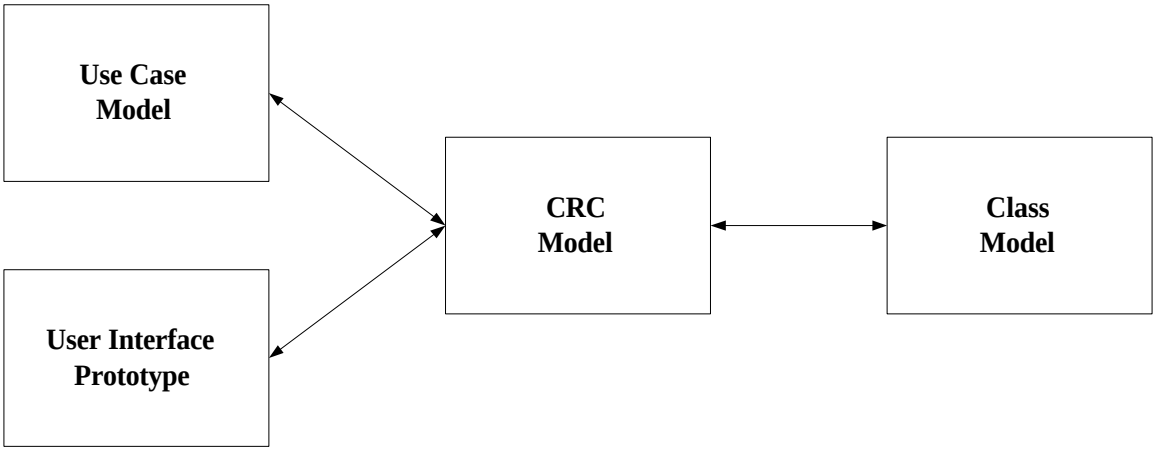

Figure 4. How CRC modeling fits in.

As you know, there is more to OO modeling than just the models shown in Figure 4. Figure 5 depicts the Detailed Modeling process pattern (Ambler, 1998b) in which the boxes represent the main techniques/diagrams of OO modeling and the arrows show the relationships between them, with the arrow heads indicating an "input into" relationship. For example, we see that an activity diagram is an input into a class diagram. In the bottom right-hand corner of each box are letters which indicate who is typically involved in working on that technique/diagram. The key is straightforward: U=User, A=Analyst, D=Designer, and P=Programmer. The letter that is underlined indicates the group that performs the majority of the work for that diagram. For example, we see that users form the majority of the people involved in developing a CRC model and designers form the majority of those creating statechart diagrams.

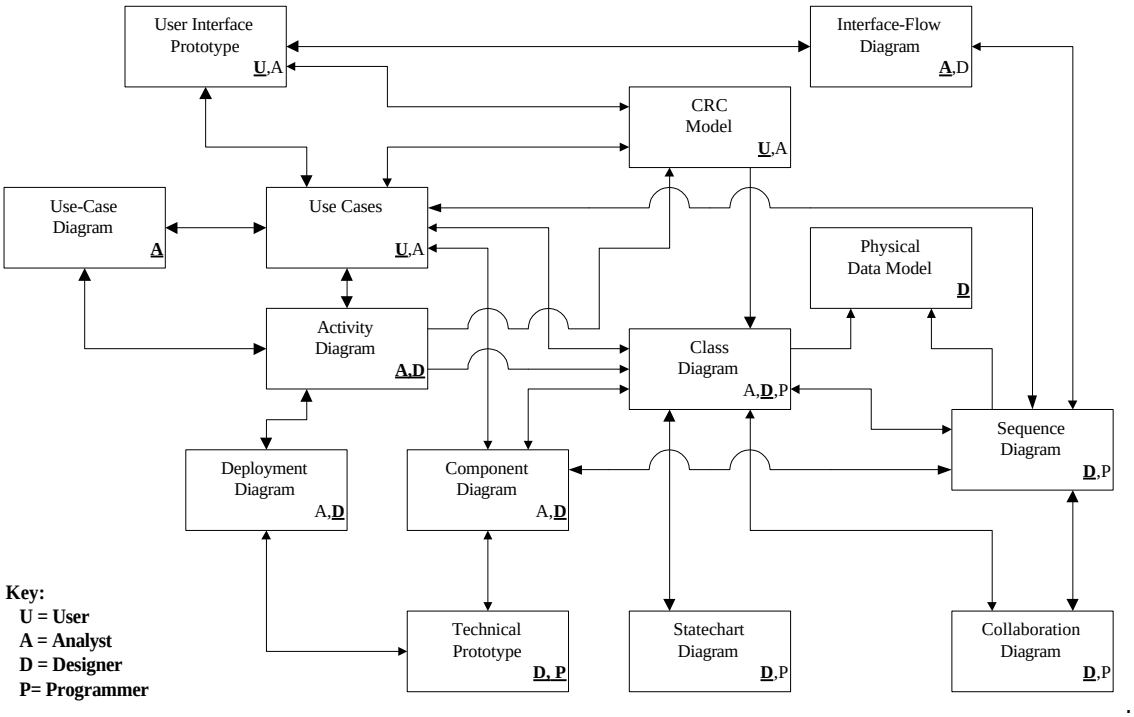

Figure 5. The Detailed Modeling process pattern.

An interesting feature of Figure 5 is that it illustrates that the object-oriented modeling process is both serial in the large and iterative in the small. The serial nature is exemplified when you look from the top-left corner to the bottom right corner: the techniques move from requirements gathering to analysis to design. You see the iterative nature of OO modeling from the fact that each technique drives, and is driven by, other techniques. In other words you iterate back and forth between models.

From a serial perspective, Figure 6 depicts the Deliverables Drive Deliverables approach process pattern (Ambler, 1998b), indicating the general order in which you will work on deliverables during the Construct Phase. It is important to point out that the views in Figure 5 and Figure 6 are complementary, not contradictory. In Figure 5 we see that we generally start modeling with techniques, such as use cases and CRC models, that focus on user requirements, moving into analysis-oriented techniques such as sequence and component diagrams, then into design techniques and finally to code. The arrows in Figure 6 represent a documents relationship. For example a use-case diagram is documented by use cases, which in turn are documented by sequence diagrams. Component diagrams are interesting in that a component within a component diagram is often documented by either another component diagram, a class diagram, and/or a use-case diagram.

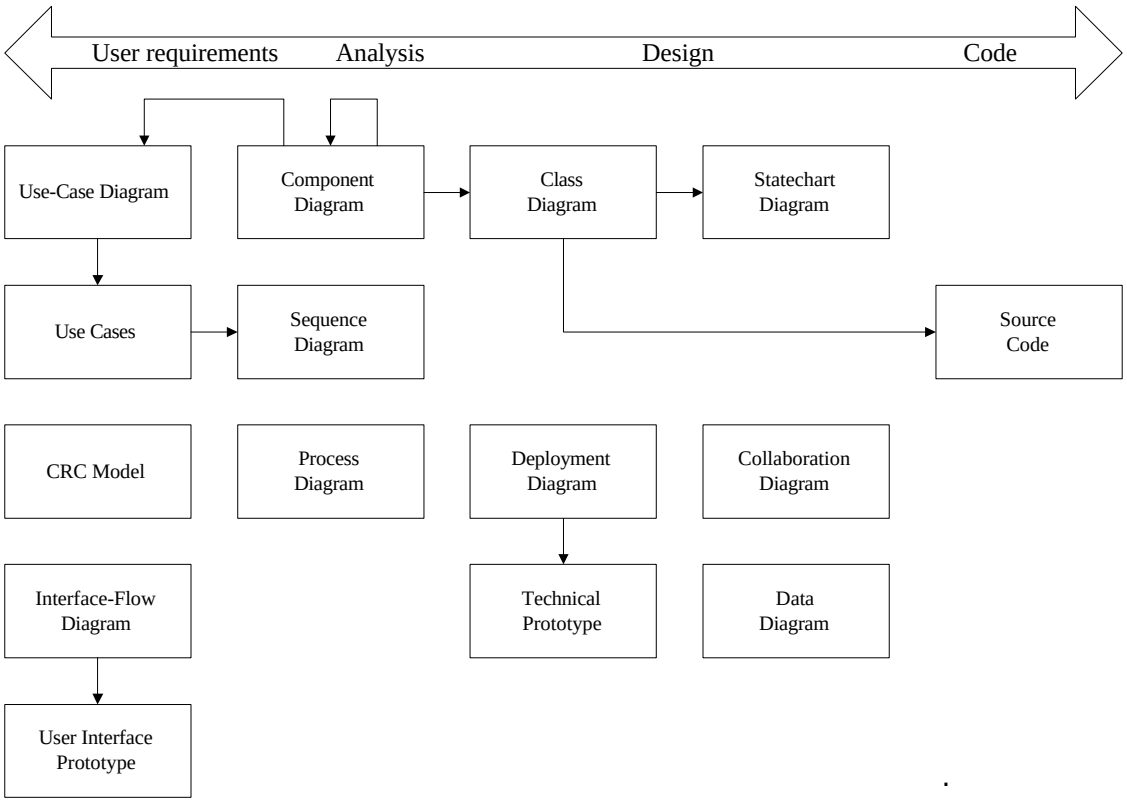

Figure 6. The Deliverables Document Deliverables process pattern.

As an aside, in the UML (Rational, 1997) the *traces* stereotype is used to connect related pieces of information in separate models to maintain traceability throughout your work. Traceability is an important concept for testing.

In Conclusion…

CRC Modeling Tips and Techniques

The following tips and techniques (Ambler, 1995) will help you to improve the effectiveness of your CRC modeling efforts:

1. **Send ahead an agenda a few days before the modeling session.** The agenda should indicate when and where the modeling session is to be held, who will attend and their roles, the purpose of the session, and how to prepare for it. Agendas help people to prepare themselves for a modeling session and are an easy way to increase the productivity of the session.
2. **Prominently display the CRC definitions.** I like to tape a large picture of a CRC card along with descriptions of what classes, responsibilities, and collaborators are, at the front of the room where everyone can see it. This gives BDEs something to refer to while modeling and acts as a reminder as to how to layout a card.
3. **Use their terminology.** You should always try to use the terminology of the problem domain wherever possible, avoiding the introduction of unfamiliar or artificial terminology. This makes it easier for the BDEs to understand what it is that's being modeled.
4. **Keep it low tech.** In the early 1990s there was a push to automate the CRC modeling process, however, I just don't see how that makes sense. CRC cards are very effective just the way that they are. Use CASE (computer aided system engineering) tools for class diagrams where you need the extra modeling capabilities, not for CRC cards where you don't. Use the right tool for the job.
5. **Expect to prototype.** You will always find that you need to sketch a few screens or reports while CRC modeling. This helps people to visualize what they're talking about and provides valuable input into the development process.
6. **Expect to take a few days.** For large systems you will likely need to hold several CRC modeling sessions. Don't worry, this is normal. The larger the system, the more effort you will need to take to understand and identify the requirements for it.
7. **Get management support.** Your organization must recognize the benefits of understanding the problem domain and of modeling a solution for that problem domain before writing code. Many organizations give lip service to "doing it right" but when push comes to shove they'll often forsake requirements gathering and modeling in favor of writing code.
8. **Include CRC modeling in your system development life cycle.** CRC modeling is an effective technique for identifying and validating user requirements. Do it!
9. **Do CRC modeling with front-line staff only.** My experience has been that CRC modeling is very effective with front-line staff because it allows you to get down into the details easily. It doesn't work as well with executives who are geared towards the "big picture." Use cases are a more effective mechanism for these types of people. Once again, use the right tool for the job.

The Advantages of CRC Modeling

There are many advantages to CRC modeling (Ambler, 1995), which include:

1. **The experts do the analysis.** The people who understand the problem domain, the business domain experts (BDEs) are the people who create the model. What better way exists to ensure that you get the right information?
2. **User participation increased.** Because users are actively involved in defining the model their satisfaction with the work will be much greater. CRC modeling increases user buy-in to the project.
3. **Breaks down communication barriers.** Users and developers work togther side-by-side to create the CRC model.
4. **It's simple and straightforward.** You get a group of people together in a room and fill out a bunch of index cards. Because of its simplicity you can explain CRC modeling to a group of people in 10 or 15 minutes.
5. **It's non-threatening to users.** Many people are afraid, and rightly so, of losing their jobs to automation. Few people, if any, are afraid of losing their jobs to a stack of index cards.
6. **It's inexpensive and portable.** You can buy 100 index cards for two or three dollars and throw them into your briefcase.
7. **It goes hand-in-hand with prototyping.** CRC modeling and prototyping are both iterative processes in which users are greatly involved. It is very common to draw rough sketches of screens and reports during CRC modeling sessions.
8. **It leads directly into class diagramming.** CRC models and class diagrams show many of the same concepts. In many ways class diagrams are simply supersets of CRC models.

The Disadvantages of CRC Modeling

There are several, albeit minor, disadvantages to CRC modeling:

1. **It's threatening to some developers.** Too many developers do not recognize the need for working closely with users, thinking that because they know the technology they also understand the business domain. This belief is obviously false, the users who do the job day in and day out almost always know more about it than the developers.
2. **It's hard to get users together.** Scheduling people for meetings is always hard to do. Try to keep the number of people in the room to just a few key people and try to plan at least a week in advance.
3. **CRC cards are limited.** CRC models are just part of the definition of user requirements for an OO application, you should also consider use cases, prototypes, and formal requirements documents. Furthermore, in most organizations it isn't acceptable to simply submit a collection of index cards as your analysis deliverable.

And a Few Closing Words

CRC modeling is a very effective technique for identifying and validating user requirements. It works hand in hand with use cases and prototypes, and leads directly into class modeling. The goal of application development is to solve business problems, not to satisfy the intellectual curiosity of developers who only want to play with new toys. Work with your users, not against them.

I welcome any comments or questions that you may have about the material presented in this white paper, so please feel free to email me at scott@ambysoft.com. Let's learn together.

References and Recommended Reading

Ambler, S. (1995). *The Object Primer: The Application Developer's Guide to Object Orientation*. New York: SIGS Books/Cambridge University Press.

Ambler, S.W. (1998a). *Building Object Applications That Work: A Step-By-Step Handbook for Developing Systems Using Object Technology*. New York: SIGS Books/Cambridge University Press.

Ambler, S.W. (1998b). *Process Patterns: Building Large-Scale Systems Using Object Technology*. New York: SIGS Books/Cambridge University Press.

Ambler, S.W. (1999). *More Process Patterns: Delivering Large-Scale Systems Using Object Technology*. New York: SIGS Books/Cambridge University Press.

Beck, K. & Cunningham, W. (1989). *A Laboratory for Teaching Object-Oriented Thinking*. OOPSLA'89 Conference Proceedings, pp. 1-6.

Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G. (1992). *Object-Oriented Software Engineering – A Use Case Driven Approach*. ACM Press.

Rational (1997). *The Unified Modeling Language v1.1 Documentation Set*. Rational Software Corporation, Monterey California.

Glossary of Terms

Business Domain Expert (BDE) – People with intimate knowledge of the problem domain, typically users who do the job day in and day out.

Class – A class represents a collection of similar objects. An object is a person, place, thing, event, or concept that is relevant to the system at hand.

Collaborator – A class will often have a responsibility to fulfill, but will not have enough information to do it. Therefore it needs to work with, collaborate with, other classes that do have the information that it needs. Collaboration will take on one of two forms: A request for information or a request to do something.

CRC card – A standard index card representing one of the classes that make up an application. CRC cards are divided into three sections: The name of the class, the responsibilities of the class, and the collaborators (if any) of the class.

CRC model – A collection of CRC cards which represent whole or part of an application.

Facilitator – The person who plans, with the aid of the project manager, and who runs the modeling session. The main role of the facilitator is to communicate what CRC modeling is all about, make sure that the cards are filled out properly, ask pertinent questions during modeling, and to recognize when prototyping needs to be done and to lead the prototyping effort.

Observer – A person who attends a CRC modeling session for training or informational purposes. Observers typically sit at the back of the room and do not participate in the session at all.

Pattern – The description of a general solution to a common problem or issue from which a detailed solution to a specific problem may be determined. Software development patterns come in many flavors, including but not limited to analysis patterns, design patterns, and process patterns.

Process pattern – A pattern which describes a proven, successful approach and/or series of actions for developing software.

Responsibility – Anything that a class knows or does, or in other words its attributes and its behaviors.

Scribe – The person(s) who record the detailed business logic that isn't captured on the cards. Scribes do not actively participate in the session, although may ask questions to confirm the business rules or processing logic that they are recording.

Task process pattern – A process pattern that depicts the detailed steps to perform a specific task, such as detailed modeling or performing a technical review.

About the Author

Scott W. Ambler is an object development consultant living in Newmarket, Ontario, which is 45 km north of Toronto, Canada. Scott is the author of *The Object Primer* (1995), *Building Object Applications That Work* (1998), *Process Patterns* (1998) and *More Process Patterns* (1999) all published by SIGS Books/Cambridge University Press. He has worked with OO technology since 1990 in various roles: Process Mentor, Business Architect, System Analyst, System Designer, Project Manager, Smalltalk Programmer, Java Programmer, and C++ Programmer. He has also been active in education and training as both a formal trainer and as an object mentor. Scott is a contributing editor with *Software Development* (http://www.sdmagazine.com) and writes feature articles for *Component Strategies* (http://www.sigs.com) and *Computing Canada* (http://www.plesman.com). He can be reached via e-mail at **scott@ambysoft.com** and you can visit his personal web site **http://www.ambysoft.com.**

About *The Object Primer*

The Object Primer is a straightforward, easy to understand introduction to object-oriented analysis and design techniques. Object-orientation is the most important change to system development since the advent of structured methods. While OO is often used to develop complex systems, OO itself does not need to be complicated. This book is different than any other book ever written about object-orientation (OO) – It's written from the point of view of a real-world developer, somebody who has lived through the difficulty of learning this exciting new approach. Readers of *The Object Primer* have found it to be one of the easiest introductory books in OO development on the market today, many of whom have shared their comments and kudos with me. Topics include CRC modeling, use cases, use-case scenario testing, and class diagramming. Visit http://www.ambysoft.com/theObjectPrimer.html for more details.

About *Building Object Applications That Work*

Building Object Applications That Work is about: **architecting** your applications so that they're maintainable and extensible; analysis and design techniques using the Unified Modeling Language (UML); creating applications for stand-alone, client/server, and distributed environments; using both relational and object-oriented (OO) databases for persistence; OO metrics; applying OO patterns to improve the quality of your applications; OO testing (it's harder, not easier); user interface design so your users can actually work with the systems that you build; and coding applications in a way that makes them maintainable and extensible. Visit http://www.ambysoft.com/buildingObjectApplications.html for more details.

Uses the

Unified Modeling Language

About *Process Patterns* and *More Process Patterns*

Process Patterns and *More Process Patterns* are ground-breaking texts, describing proven, reusable techniques for developing large-scale, mission-critical object-oriented software that is robust and extensible. The focus of the book is The Object-Oriented Software Process (OOSP), presented as a collection of process patterns that are geared toward medium to large-size organizations that need to develop software that support their main line of business. Process patterns are the reusable building blocks from which your organization will develop a tailored software process that meets its exact needs, and have been shown to be ideal for enhancing the industry-standard Unified Process. Visit http://www.ambysoft.com/processPatterns.html and http://www.ambysoft.com/moreProcessPatterns.html for more details.

Uses the

Unified Modeling Language

About *The AmbySoft Inc. Coding Standards for Java*

The AmbySoft Inc. Coding Standards for Java summarizes in one place the common coding standards for Java, as well as presents several guidelines for improving the quality of your code. It is in Adobe PDF format and can be downloaded from http://www.ambysoft.com/javaCodingStandards.html.

Index