Software Design Using CRC Cards

Harold Halbleib, Excel Software

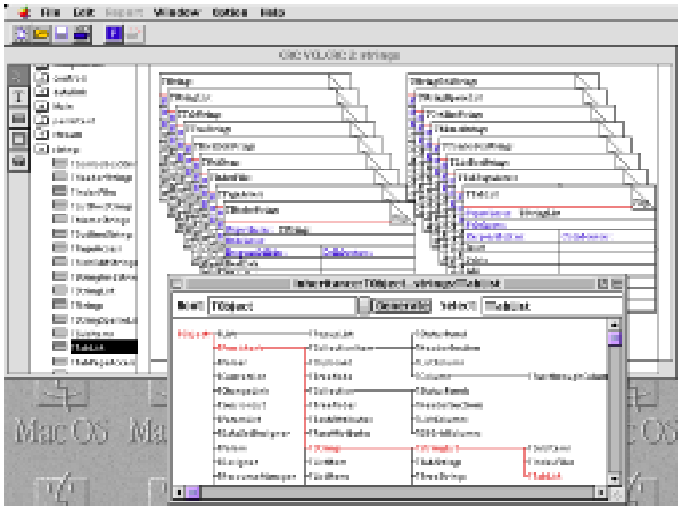

Figure 1: *CRC Cards Created With QuickCRC*

Introduction

Many notations, methods and books have been published in recent years regarding analysis and design of object-oriented software. While popular object-oriented notations and methods like Booch, OMT, Shlaer/Mellor and UML are becoming more prevalent in large organizations, the learning curve and overhead of formal methods can be excessive for small projects.

CRC cards (class, responsibility, collaboration) provide a simple design alternative that can be quickly applied to any object-oriented project. The short learning curve makes this responsibility driven design approach a natural choice for small projects. CRC cards are also highly effective as a front end to other design methods.

The software development process involves requirements gathering, software design, implementation and testing. The typical project begins with an analysis of the customer problem and the formulation of a list of requirements or specifications to solve the problem. Customer and developer understanding and acceptance of the requirements is key to the project's success. The customer's solution usually consists of one or more programs to be designed and implemented. The design phase of the development effort can be largely centered around the creation and refinement of CRC cards. When the design is complete, information can be listed to text to serve as a coding specification for the implementation phase of the project. The testing phase ensures that the solution fullfills the customer requirements.

It is not uncommon for larger, object-oriented projects to take an iterative approach, particularly during the design and implementation phase. To reduce risk, it is usually best to identify the more important CRC cards for the overall project and divide the project into smaller functional areas. When refining the design and writing the code, start with those functional areas having the biggest impact on the project success or presenting the greatest risks.

In its simplest form, CRC cards can be applied to a design project using a stack of index cards and a pencil. Each card identi-

fies a class, its properties and relationships with other cards. As a design grows, an automated tool to support CRC cards saves time and can reduce design errors. As information is entered or modified, references between cards can be instantly updated. Verification checks ensure consistency and completeness. Design scenarios can be identified and simulated. An automated tool can graphically illustrate relationships between cards, making it easier to manage large projects.

General Concepts

The CRC card shown below represents an object class and its properties. The words card, class and object are often used interchangeably although technically they are not the same thing. A class is a type from which an object is instantiated and a card is a collection of information about a class.

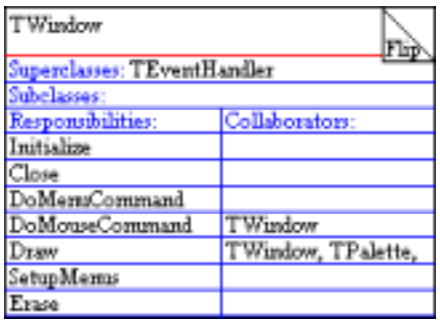

Figure 2: *Front Side of CRC Card*

The card, named TWindow for the class it represents, currently has one superclass named TEventHandler and no subclasses. This class has several responsibilities. A responsibility represents a function the object must perform for itself or to service the requests of other objects. This card must collaborate with other cards to accomplish some of its responsibilities, for example, TWindow's Draw responsibility collaborates with TWindow, TPalette and etc. The back side of a CRC card includes its description and a list of its attributes representing data the class must keep track of.

With an automated tool, CRC card data is entered into a property dialog, much like writing it on a paper index card. One big advantage in using a tool is that new cards can be automatically created for you for undefined superclass, subclass or collaborating class references. Information can easily be inserted, changed or deleted and the card can be resized. When renaming a card, all references on other cards are instantly updated.

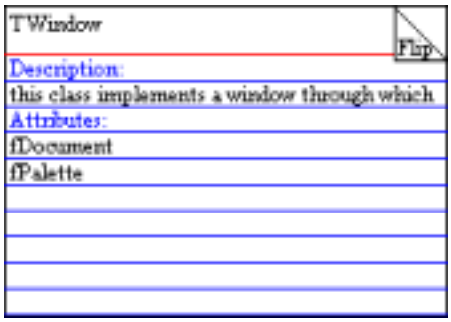

Figure 3: *Back Side of CRC Card*

To discover the data each class knows about and the responsibilities it must perform, scenarios are created. A scenario is a list of steps outlining the interaction between a group of classes to implement a mechanism in the design. Below the scenario name and description is three columns titled Client, Server and Responsibility. For each scenario step, a client class uses a responsibility of a server class. A scenario can be created with a paper index card or by typing information into a tool's property dialog.

As an example, the Read Document scenario is shown below. TDocument is a class that holds data about a list of shapes read from disk into memory. This scenario has three steps starting with any client class calling the Read responsibility of TDocument. TDocument then uses the Initialize and Read responsibilities of TShape to create shape objects and read in their data from disk.

Read Document		
read document's data from disk into memory		
Client:	Server:	Responsibility:
AnyClient	TDocument	Read
TDocument	TShape	Initialize
TDocument	TShape	Read

Figure 4: *Scenario*

During the early design process, developers usually focus on the normal interactions between objects when creating scenarios. Special situations often arise due to error conditions that must be handled, but usually these can be deferred until detailed design. If an error condition is significant to the overall design, it can be dealt with as a separate scenario.

Scenarios can also refer to other scenarios. For example, the Open Document scenario shown below uses two other scenarios, Initialize Document and Read Document (illustrated above).

Open Document		
open an existing document by creating it and reading it		
Client:	Server:	Responsibility:
TApplication	TDocument	Open
TApplication	TDocument	Initialize Document
TApplication	TDocument	Read Document

Figure 5: *Scenario Using Other Scenarios*

Designing an object-oriented system using cards and scenarios is an iterative refinement process. An automated tool allows a designer to exercise part of a design expressed by a group of cards and scenarios by using high level simulation. The designer can single step forwards or backwards through each design mechanism to locate errors and make changes before writing any code.

A Design Example

The following example will illustrate the process of designing a program using CRC cards with an automated tool. The program to be designed allows the user to edit a diagram using a tool palette for drawing box and circle shapes. To model this program, the designer creates a card for each class, establishes relationships, assigns responsibilities and attributes, defines and simulates scenarios, checks the model for errors and illustrates interesting relationships between objects.

Create CRC Cards

A CRC model is usually created by an individual or small group of designers during the early phase of an object-oriented development project. Each CRC card can be added to a diagram workspace by clicking with a palette tool and typing information into a property dialog.

The Card Property dialog shown here indicates that TShape has a superclass called TObject and two subclasses, TBox and TCircle. Each new superclass or subclass card is added by typing its name or selecting from a popup list of existing cards. Each referenced class that doesn't already exist and corresponding superclass and subclass relationships get added when the dialog is dismissed.

Figure 6: *Card Property Dialog*

The CRC diagram below shows two more classes TList and TWindow each derived from the superclass TObject. Obvious relationships can be added at the beginning of a CRC card session. However, if designers are uncertain as to whether or not a relationship exists, it is better to keep classes separate, and see if a superclass or collaboration relationship arises out of the design scenarios. Assuming a relationship too early may force a particular decision

and bias the distribution of responsibilities. The emphasis and strength of the CRC card technique and responsibility driven design in general lies in deriving the behavior of the class and not the structure.

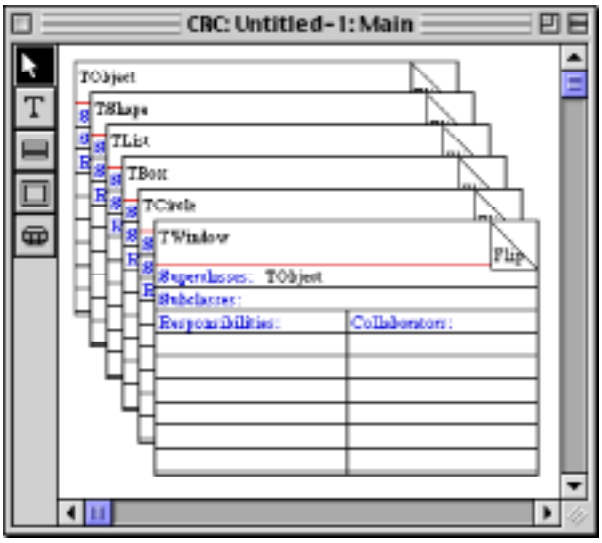

Figure 7: *CRC Window*

Assign Responsibilities

Once a set of classes are defined, behaviors can be assigned that will provide the functions of the application. Responsibilities that are derived from the requirements or that are obvious from the name of the class can be listed before any scenario execution commences. The responsibilities of a class are also referred to as its functions, operations or methods.

For example, the TShape card has responsibilities Initialize to create it, Free to dispose of its memory, Read to load its data from disk into memory, Write to save it to disk and Draw to illustrate it on the diagram. Give each responsibility a short description that indicates what it accomplishes. Other responsibilities may be discovered later after creating scenarios to work through mechanisms in the design.

Add Attributes

Attributes of classes may also be identified early in a CRC session. Often, nouns that are not classes but rather characteristics of classes are best represented as attributes. Attributes can be assigned to classes as they are discovered, but should be done in moderation and only when it becomes apparent that the class must know that information.

The TShape class has attributes fPosition, fType and fSelected. Each attribute can be given a short description. This information is added to the back of the CRC card.

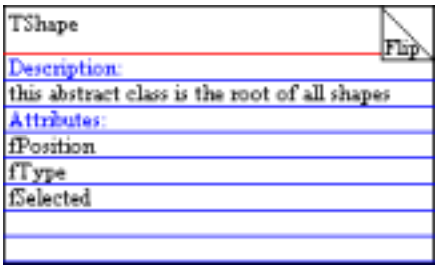

Figure 8: *Back Side of TShape Card*

Define a Scenario

A scenario describes a sequence of steps in the design using the responsibilities of a group of collaborating classes. Collaboration between classes refers to a client object that uses a responsibility performed by a server object. Often a class must call upon several collaborating classes to implement one of its responsibilities. Assign responsibilities to classes by exploring how the system responds to external events.

Scenarios are detailed examples of the functions of the system, where each function refers to visible, testable behavior. A scenario describes what happens in the system from a high-level, user point of view. For example, the Open Document scenario involves only those classes and the subset of their responsibilities involved in opening a disk document. The goal of walking through scenarios is to discover where additional classes, responsibilities

and collaborations are required, or where existing items have become redundant or inconsistent.

Consider the scenario named Draw Window. The TWindow class draws itself by looping through a list of shapes in the document drawing each one and then drawing the tool palette. Each step in a scenario has a Client, Server and Responsibility field. For each step, a client class uses a responsibility of a server class.

A designer adds each step in a scenario to a property dialog by selecting class names and responsibilities from a popup list of what is already defined in the model. A new class or responsibility can be added to the model by typing its name.

Draw Window		
draw document's window		
Client:	Server:	Responsibility:
AnyClient	TWindow	Draw
TWindow	TWindow	Erase
TWindow	TList	EachItemDo
TWindow	TShape	Draw
TWindow	TPalette	Draw

Figure 9: *Scenario Object on Diagram*

Simulate Scenarios

Scenarios can use other scenarios to get their work done. Initialize Document and Read Document are both subscenarios related to the server class TDocument. For example, the Open Document scenario references the Initialize Document subscenario by specifying its server class TDocument in the server field and its scenario name in the Responsibility field. The first step in the Initialize Document subscenario uses TDocument as the server class name.

Open Document		
open an existing document by creating it and reading it		
Client:	Server:	Responsibility:
TApplication	TDocument	Open
TApplication	TDocument	Initialize Document
TApplication	TDocument	Read Document

Figure 10: *Scenario Using Other Subscenarios*

Figure 11: *Path Through Subscenarios*

Using an automated tool, a designer can simulate a selected scenario. By single stepping forwards, backwards or through each subscenario, bugs in the design can be identified and corrected early. An active scenario list shows your position within a hierarchical stack of scenarios. This list helps you keep your bearings in a complex simulation and allows you to change to a different spot in the simulation path.

Partition The Design

As the number of CRC cards in the design grows, they can be grouped by function. In an automated tool, separate diagrams are used to partition the model into subject areas. The Contents view is used to navigate or arrange cards between diagrams. As illustrated below, the Contents view has a folder icon that can expand or collapse to represent each diagram level.

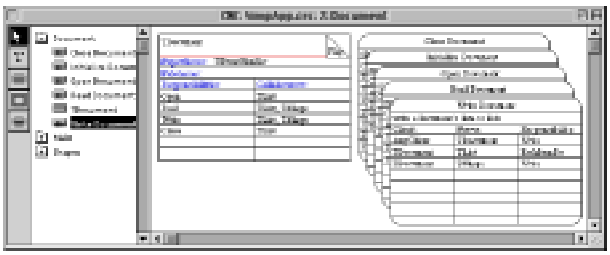

Figure 12: *Contents View Showing Three Diagram Levels*

Inheritance Graph

An automated tool can generate an inheritance graph from information on CRC cards. This diagram can concisely illustrate the big picture of a large project that might contain hundreds of classes and dozens of diagrams.

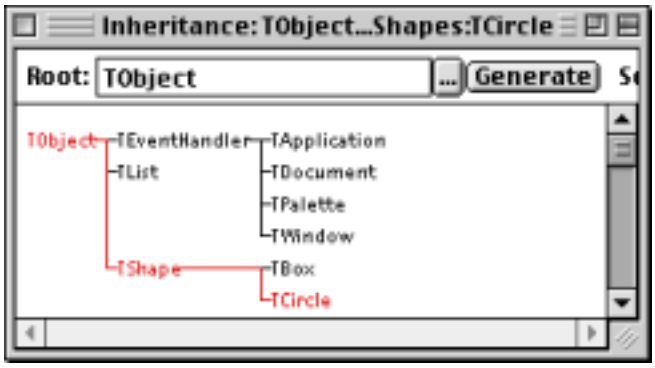

Figure 13: *Inheritance Graph Outlining Path to TCircle*

Verify Your Work

Creating and simulating scenarios will help verify that a design is correct and complete. An automated tool can perform other error checks to locate design problems. For example, responsibilities that are not used in any scenarios may indicate that the design is incomplete or perhaps the responsibility isn't needed. Likewise, a card that is not used by any collaboration may not be needed.

Conclusion

CRC cards provide a simple approach to identifying classes and related information for an object-oriented development project. Design scenarios identify how mechanisms in the design work. An automated tool supports the process by instantly updating references between cards and scenarios, providing scenario simulation, error checking, instant inheritance graphs and making it easier to locate or modify information.

An automated tool can generate text reports to be used as a coding specification. Information can be imported from or exported to other development tools. In today's world of instant communication and pressing deadlines, the advantages of electronic design documentation for peers, reviewers or customers cannot be ignored.

Bibliography

Designing Object-Oriented Software, by Rebecca Wirfs-Brock, Brian Wilkerson, and Laura Wiener. Prentice Hall, 1990.

Using CRC Cards: An Informal Approach to O-O Software Development, by Nancy M. Wilkinson. SIGS Books, New York, 1995.

The CRC Card Book, by David Bellin and Susan Suchman Simone, Addison Wesley Longman, 1997.

About the Author

This white paper was written by Harold Halbleib, product manager of software engineering tools at Excel Software. Excel Software produces tools for system analysis, requirement specification, software design, code generation and reengineering that run on Windows 95/NT and Macintosh computers. QuickCRC was used to produce the examples in this paper. It supports CRC cards, design scenarios with simulation and inheritance graphs.

For more information on software engineering tools for Macintosh or Windows 95/NT or to download a trial version of QuickCRC, see www.excelsoftware.com.

Excel Software
Ph. 515-752-5359
Fax 515-752-2435
Email: info@excelsoftware.com
Web: http://www.excelsoftware.com