CAMBRIDGE

Cambridge University Press
978-0-133-74679-2 - Using CRC Cards: An Informal Approach to Object-Oriented Development
Nancy M. Wilkinson
Excerpt
More information

Introduction

"Who cares for you?" said Alice . . .

"You're nothing but a pack of cards!"

— Lewis Carroll
Alice's Adventures in Wonderland

EVERYBODY IS TALKING about objects. Presently, there are at least 27 published methods for doing object-oriented analysis and design. There are probably many more unpublished methods used by software development groups today[1]. Methodologies are extremely diverse and will continue to change for some time to come. Now is a time of confusion for anyone hoping to find *the* method for doing object-oriented development.

With all the hype surrounding particular object-oriented methods and notations, it is possible to miss the fact that there is an object-oriented way of thinking that must be mastered to effectively use this new technology. Project members may spend time and effort learning a particular method and tool and be no closer to understanding the essence of the object paradigm.

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-133-74679-2 - Using CRC Cards: An Informal Approach to Object-Oriented Development
Nancy M. Wilkinson
Excerpt
More information

As the interest in object-oriented technology has grown in recent years, people have searched for ways to get their arms around the notion of objects and start object-oriented development while minimizing pain and risk. Class, Responsibility, Collaborator (CRC) cards provide a simple way for projects to investigate this new approach with minimal investment. Using simple index cards, CRC cards help convey, in an inexpensive, informal, and familiar way, an intuitive understanding of the object-oriented paradigm.

CRC cards have become popular not only as the initial step into the world of object-oriented technology, but also as support throughout the project life cycle. Projects that prefer informal approaches and those that are not yet ready to choose a formal method find CRC cards to be a convenient and natural way to develop a model of communicating objects.

This book is primarily aimed at small teams of people using CRC cards for all stages of the project life cycle. We have three main goals. The first is to convey the essence of the CRC card session: its informality, spontaneity, and energy, coupled with its power in creating an object-oriented decomposition of a problem. Readers interested only in an introduction to this simple technique may wish to read just the first 3 chapters. The second goal of the book is to describe the way CRC cards can support the analysis and design of a real object-oriented application. The third goal is to describe ways in which a design created with CRC cards can be effectively implemented in C++.

Roadmap

Here is what can be expected from the next 200 or so pages of this book. Chapter 1 is devoted to an overview of the object-oriented paradigm. It is not meant to be an exhaustive description of object-oriented technology, but it helps set the context for CRC cards. The aim is to promote a common understanding of the goal of the CRC card technique and to introduce terms that are used in the rest of the book. For a more complete description of objects and object modeling, the reader is referred to books by Booch[2], Wirfs-Brock[3], and Budd[4]. Readers experienced in object-oriented technology may wish to skim this chapter.

Chapter 2 introduces CRC cards and provides some history and motivation for the CRC card technique. We give credit to its inventors and early users and explore various

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-133-74679-2 - Using CRC Cards: An Informal Approach to Object-Oriented Development
Nancy M. Wilkinson
Excerpt
More information

areas where the cards have been used with success. This chapter also describes, at a high level, the role of CRC cards in a project life cycle. Finally, it presents the syntax of the card itself.

Chapter 3 describes the CRC card session in detail, including such topics as selecting a group, choosing a subset of the problem, brainstorming and filtering classes, choosing and walking through scenarios, handling potential pitfalls, and facilitating a CRC card session. This chapter attempts, through a simulated group session, to give the reader a feel for the interactions and issues in a real CRC card session. The example begun here continues through the next three chapters.

The remainder of the book is focused on the role of CRC cards in the project life cycle. Chapter 4 concentrates on their use for problem modeling. It describes the benefits and limitations of the process at this stage of the project. What distinguishes CRC cards for analysis in terms of classes, responsibilities, and scenarios is also explored. Finally, it describes the convergence to a stable problem model and what to do next. For groups that choose to use formal methodologies, CRC cards can help provide input to formal notations by bringing together the right people to brainstorm classes and behavior. The specific contribution of CRC card information to particular methods and notations is discussed.

Chapter 5 describes the use of CRC cards in the design phase. This includes a discussion of the strength and limitations of CRC cards for the design process. What distinguishes CRC cards for design in terms of classes, responsibilities, and scenarios is also explored. Enhancements to the cards are suggested that can facilitate their use in recording more detailed design information. This chapter continues the simulated group session into design. It also gives guidelines for documenting the cards and scenarios and tips for when prototyping should begin. Available tools for capturing a CRC card design are also described.

Prototyping C++ classes from a CRC card design is not a straightforward transliteration. Low-level design decisions must be used at this stage to bridge the gap between the cards and the code. Chapter 6 describes how CRC design elements map to C++ features and then how to build C++ classes using the CRC card input. This includes how the state of the objects are represented with C++ data members and how responsibilities, collaborations, and superclass/subclass relationships determine member function declarations and implementation.

Finally, the Appendix provides a listing of the class declarations and implementations for some key classes in the example application.

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-133-74679-2 - Using CRC Cards: An Informal Approach to Object-Oriented Development
Nancy M. Wilkinson
Excerpt
More information

A Note on Programming Language

A number of languages support object-oriented programming, notably C++[5], Smalltalk[6], and Eiffel[7]. Although these languages have much in common, such as the direct support for object-oriented modeling concepts, they also differ in terms of philosophy and specific features. An application design will depend, to a great extent, on the choice of language. The target language that we have chosen and for which we can provide guidance is C++. The low-level design and implementation discussions in this book depend on C++ and do not pretend to apply to an implementation in any other language.

Common Terms

The following is a list of terms that are used throughout the book that may require some clarification or explanation.

Object-oriented model: We use this term to indicate either an object-oriented analysis or design, regardless of level of detail. It simply refers to a set of collaborating objects with accompanying scenarios.

Object-oriented development: This term will refer to all phases of software development: analysis, design and implementation, and testing, not just the implementation and testing stage.

Execution of scenarios: This term refers to the physical simulation, by a group of people in a CRC card session, of the steps and collaborations necessary to perform a system function. It does not refer to the execution of the program that implements the scenario.

Methodology: We define a methodology as a collection of disciplined processes used to generate models of a software system, using a well-defined notation. Many object-oriented methodologies exist today for projects that prefer formal methods. The CRC card technique is *not* a methodology.

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-133-74679-2 - Using CRC Cards: An Informal Approach to Object-Oriented Development
Nancy M. Wilkinson
Excerpt
More information

Summing Up

CRC cards are a simple technique, but scrutinized closely, they prove to have much to offer. They are very good at helping to intuitively convey the idea of objects, and proper use of them will help a project model their application and translate their model into C++. My hope is that all of the above will also be true of this book.

Coming Up

We divert from the topic of CRC cards in Chapter 1 to provide an overview of object-oriented technology in general. This will help set the context for the CRC card discussions we return to in Chapter 2.

References

1. Jacobson, Ivar. "Time for a Cease-Fire in the Methods War." *Journal of Object-Oriented Programming* (July/August 1993), p.6 .
2. Booch, Grady. *Object-Oriented Analysis and Design with Applications*, 2nd ed. Benjamin-Cummings, Redwood City, CA, 1993.
3. Wirfs-Brock, Rebecca, Brian Wilkerson, and Laura Wiener. *Designing Object-Oriented Software.* Prentice Hall, Englewood Cliffs, NJ, 1990.
4. Budd, Timothy. *Introduction to Object-Oriented Programming*, Addison-Wesley, Reading, MA, 1990.
5. Ellis, M. A., and Bjarne Stroustrup. *The Annotated C++ Reference Manual*, Addison-Wesley, Reading, MA, 1990.
6. Goldberg, A., and D. Robson. *Smalltalk-80: The Language and Its Implementation.* Addison-Wesley, Reading, MA, 1983.
7. Meyer, B. *Object-Oriented Software Construction.* Prentice Hall, Englewood Cliffs, NJ, 1988.

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-133-74679-2 - Using CRC Cards: An Informal Approach to Object-Oriented Development
Nancy M. Wilkinson
Excerpt
More information

CHAPTER 1

An O-Overview

"I think you are locating them in the wrong places. The heart is on the left and the liver on the right."

"Yes, that was so in the old days. But we have changed all that."

—Molière, *le Médècin malgrè lui*, act 2, scene 4

THE COMPLEXITY OF software problems and the high cost of their solutions has led to the continuous search for new approaches to software development. Certain of these new approaches take hold and, in time, become the de facto standard for the production of software. More than 20 years ago, structured programming, pioneered by E. W. Dijkstra[1], became the established way to solve software problems. Over the past decade, object-oriented programming has emerged. The object-oriented approach, with its potential for reusability and extensibility, has gained rapid acceptance in the software production community. While structured programming emphasized a hierarchical organization of problems by functionality, object-oriented programming organizes software around objects in the problem domain.

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-133-74679-2 - Using CRC Cards: An Informal Approach to Object-Oriented Development
Nancy M. Wilkinson
Excerpt
More information

Developing object-oriented applications starts with finding these objects. But it also means understanding what knowledge and abilities are associated with each object and how they interact to get a job done. In this chapter we present a motorcycle tour of object-oriented development topics, such as the essential characteristics of objects and how they interact, the components of an object-oriented model, the object-oriented software development process, and object-oriented programming languages. The intent is not to teach these topics but to place the notion of CRC cards in the overall context of object-oriented development. We hope to motivate the use of CRC cards and illustrate how the notions of class, responsibility, and collaborator support object-oriented development.

The Object-Oriented Mind Shift

It has been said that conveying object-oriented techniques to a non–computer professional is easier than teaching them to someone who is experienced in procedural thinking, modeling, and programming. This is because the object-oriented way of decomposing a problem parallels the way non–computer professionals are used to looking at real-world situations. The decomposition is into entities that more closely resemble those in real-life situations. It makes sense that our software should model the problem we are solving as closely as possible. But most software developers have a view that has been formed by years of emphasizing the procedural aspects of the problem in order to mimic, and thus exploit, the linear processes of a computer.

In conventional software development, problems are decomposed by functionality, and the data structures that support the problem and solution are only loosely coupled with the behavior or procedures that operate upon them. In contrast, an object-oriented decomposition models a problem as the interaction of a set of entities whose data and behavior are inherent and inseparable. These entities, called objects, closely resemble those found in the problem domain. Typical examples of objects from various domains are instances of Menus, Employees, Chess Pieces, Lists, Books, and SpreadSheets.

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-133-74679-2 - Using CRC Cards: An Informal Approach to Object-Oriented Development
Nancy M. Wilkinson
Excerpt
More information

What Is an Object?

Since objects are at the heart of this new approach, it is important to say explicitly what an object is. The best definition we have seen is the one coined by Smith and Tockey[2] and borrowed by Booch[3], which asserts that "an *object* represents an individual, identifiable item, unit, or entity, either real or abstract, with a well-defined role in the problem domain." An object is more than just data and functions bound together. It is a complete entity, either physical, such as an Employee or Chess Piece object, or conceptual, such as a Menu or a List object.

Objects, in software as in real life, are viewed as black boxes. They *encapsulate* information and behavior, restricting access to their internal parts and interacting with other objects in only well-defined ways. Essentially, an object has state, behavior, and identity.

State

Each object has a set of static properties, or attributes, which are its essential and distinctive characteristics. For example, Menu objects might have a list of option strings and a chosen option. Employee objects have names and salaries. SpreadSheet objects have cells.

The *state* of an object is the value of these attributes at any point in time. These may be simple values or the values of other objects. For example, a Menu object, called fileMenu, may have the value ("Close," "Open," "Save," "Print," "Exit") for its list of option strings and "Save" as the value of its chosen option, which represents its state. We can say that fileMenu "knows" the values of these attributes. A SpreadSheet object, called myBudget, will have a particular set of cells, each of whose value will be Cell objects, each with their own state. And an Employee object, called emp091037, might have a name attribute with value ("Nancy Wilkinson") and a salary attribute whose value is some small real number.

The state is private to the object and not directly visible to other objects. This concept of denying access to the internals of an object is called *information hiding*. The techniques of encapsulation and information hiding are not unique to an object-oriented approach[4]. But objects, and the direct support for these concepts in object-oriented languages, provide a particularly nice medium for their use.

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-133-74679-2 - Using CRC Cards: An Informal Approach to Object-Oriented Development
Nancy M. Wilkinson
Excerpt
More information

Behavior

The behavior of an object is the set of operations, or *responsibilities*, it must fulfill for itself and for other objects. This includes the responsibility to provide and modify state information when asked by other objects as well as services to be performed at their request. For example, fileMenu may be responsible for displaying its options, collecting the chosen option from the user, and telling other objects the value of its chosen option when asked. Cell objects have the responsibility to display themselves or to update themselves when asked to do so by a SpreadSheet object.

Some of these responsibilities can also be hidden and used only by the object. The set of visible responsibilities form the *interface* of the object and completely define how it can act and react in relationship to other objects. The object has an integrity that cannot be violated; it is completely defined by its actions. It can only be modified, behave, or interact with other objects in very well-defined ways. An Employee object cannot display itself, a SpreadSheet object cannot provide salary information, and a Rook object simply cannot check out a library book or even move diagonally.

While an action may change an object's state, the values persist between invocations of its actions. A Queen's position, for instance, will be changed by its *move* operation, and the new value persists until it is modified by another operation. Therefore, the state of an object is dependent on the series of actions that the object has performed. For example, the value of a Cell object in a SpreadSheet depends on what update or edit operations have been done on it, and the position of a Pawn object on a Chess Board is the result of the set of moves it has made.

Identity

Each object is distinguished from all other objects by an *identity*, which is simply its inherent existence. A Person object with the name John will be different from all other objects of type Person with the same name. The Cell in row 4, column 5 of the SpreadSheet may have the same value as the Cell in row 6, column 2, but they are different objects with different identities.

Collaborators

An object cannot always carry out its responsibility on its own. It often needs the services

© Cambridge University Press www.cambridge.org

CAMBRIDGE

Cambridge University Press
978-0-133-74679-2 - Using CRC Cards: An Informal Approach to Object-Oriented Development
Nancy M. Wilkinson
Excerpt
More information

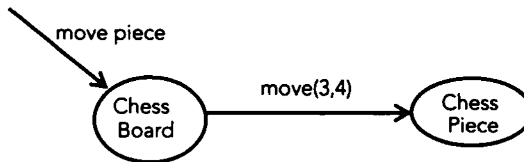

Figure 1.1 Sending a *move* message.

of another object. When a SpreadSheet objects asks a Cell object to display itself in order to fulfill its own *display* responsibility, it is collaborating with Cell, and Cell is said to be a *collaborator* of SpreadSheet. Asking a collaborator object to perform a service is done by *sending a message* to the object. The message consists of the responsibility name and any data that the collaborating object will need to perform the service.

For example, let's examine the operation of a chess program. Assume we have a Chess Board object to manage the interactions with the player and the Pieces on the board. In response to a message from the user to move a particular piece, the Chess Board object collaborates with the Chess Piece object via the *move* responsibility. This implies that Chess Board sends Chess Piece a message that consists of the responsibility name, *move*, and the data that represents the new position. This is illustrated in Figure 1.1. Ovals represent classes of objects, and collaborations are arrows labeled by the responsibility of the pointed-to class. The data (position) being passed is included in parentheses. Each object in an application has the set of attributes and behavior necessary to complete its portion of the activity of the system. Collaborations represent the way that objects work together to achieve the overall system behavior.

Class vs. Object

So far our examples have been talking about the "objects" that interact to solve problems. As we have said, objects represent the actual things in the problem domain that work together to accomplish a task. For example, fileMenu, myBudget, emp091037, and blackRook are all objects in various domains. Each object is an instance of a *class*, i.e., Menu, SpreadSheet, Employee, and Rook, which defines its essential characteristics and behavior. It is important to understand the distinction between class and object.

© Cambridge University Press www.cambridge.org