A Tool to Support the CRC Design Method

AUTHORS:

Steve Roach, Department of Computer Science, University of Texas at El Paso, sroach@cs.utep.edu
Javier C Vásquez, Department of Computer Science, University of Texas at El Paso, jvasquez@cs.utep.edu

Abstract — *Software design is an essential yet challenging concept for students in computer science and software engineering. Classes, Responsibilities, and Collaborations (CRC) is a design method focused on creating highly cohesive and modular systems. Classes describe real-world objects that exist in a system. These classes are assigned responsibilities, i.e., data and actions that the class is required to support. A class may fulfill a responsibility by itself, or it may collaborate with some other class to fulfill the responsibility. The interactions among classes must be described in detail and eventually translated into method signatures. Related responsibilities of each class are grouped into clusters called contracts. Contract responsibilities are those that perform a general service for other classes. Responsibilities that do not service outside classes are known as private responsibilities. A higher level of abstraction of this model is invoked through the use of subsystems. Subsystems can contain classes and other subsystems that combined perform a general function or set of related functions. In doing this, the design can hold several levels of abstraction.*

When using the CRC method as described in text books, a design team writes information on index cards. Each card represents a class. It shows the name, description, superclass, subclasses, responsibilities, collaborations, and subsystems of a class. The advantage is that the design team can easily move the cards around to visualize the design, and modifications to the design can be made quickly by simply replacing cards. Some difficulties with this approach is that design layouts are not easily transferred to team members when the spatial relations are eliminated, modification of complex classes becomes tedious, and since the data is contained only in hand-written form, it is necessary to transfer the content of cards into an electronic medium in order to use software development tools such as Rational Rose or to document the design.

The CRC Design Assistant is a software tool created to assist students during the design process. It stores a design in a database and assists designers in the creation and modification of designs using the CRC method. CRC cards are represented graphically and can be easily manipulated using the mouse and keyboard. The tool can generate design documents in MS Word and UML diagrams, which can be uploaded into other tools for processing. The automation allows students to focus on the actual design of a system rather than spend time revising documents. The tool is available for download from the University of Texas at El Paso web site.

Index Terms — *CRC, Design Environment, Object-Oriented Design, Software Engineering Education.*

INTRODUCTION

Object-oriented development has become a prevalent approach for producing flexible, maintainable software systems. Because of its potential for reusability and extensibility, object orientation is a result of the search for techniques to manage the complexity of modern software. An object-oriented system is organized around models of objects in the problem domain. These objects maintain their own state and interact with each other to achieve system behaviors. The importance of object-orientation is underscored by its prevalence in the ACM computing curriculum CC2001 [6] and CE2003 [7]. In the Computer Science Department at the University of Texas at El Paso (UTEP), the software engineering course is taught in the context of the senior capstone project. The course is presented over two semesters during which student teams construct a software system for a client. Our client list includes the Pan American Center for Earth and Environmental Studies, the US Army Research Laboratory, Sandia National Laboratories, the US Geological Survey, the UTEP Department of Geology, and the New Mexico State University Department of Agronomy. Our experience in this course has been that while students are familiar with programming in object-oriented languages such as Java or C++, they have difficulty designing object-oriented solutions to larger, real-world problems. Our students need a method for developing and analyzing object-oriented designs that is easy to learn, facilitates modification of designs, and encourages team interactions.

The *Classes, Responsibilities, and Collaborations* (CRC) method [10] [11] is a design method focused on creating highly-cohesive and modular software systems. The CRC method is a relatively simple way for students to investigate object-oriented design with minimal investment. The advantages of CRC include ease of learning for new team members, rapid design development, ease of change of designs, and effective integration of team members. The use of 3x5 index cards allows designers to view the design in different configurations and to modify a design by quickly modifying a card or by removing a card completely.

Some difficulties with this approach are it does not scale well to large groups of designers, design layouts are not easily transferred to team members when the spatial relations are eliminated, modification of complex classes becomes tedious, and since the data is contained only in hand-written form, it is necessary to transfer the content of cards into an electronic medium in order to use software development tools such as Rational Rose. Additionally, it is time-consuming, tedious, and error-prone for students to maintain documentation that accurately reflects the state of the design, particularly in later stages of design when the modifications are more subtle.

The *CRC Design Assistant* [4] is a freely available software tool created to assist students during the CRC design process. It stores a design in a database and assists designers in the creation and modification of designs using the CRC method. CRC cards are represented graphically and can be easily manipulated using the mouse and keyboard. The tool can generate design documents in MS Word and UML diagrams that can be uploaded into other tools such as Visio or Rational Rose for processing. The automation allows students to focus more time on the actual design of a system and improves the accuracy of the documentation.

This paper presents the CRC Design Assistant. Section 2 briefly describes object-oriented software development and the terms used later in the paper. Section 3 describes the CRC design method. Section 4 describes the CRC Design Assistant. The paper concludes with a description of future work.

OBJECT-ORIENTED SOFTWARE DEVELOPMENT

The goal of object-oriented design is to develop an object model of a system to implement the identified requirements. To design a solution to a problem, the programmer must identify the objects and classes, their capabilities, the knowledge they contain, and the interactions among the objects. There are many good references for object-oriented design [2][3][5][9]. This section briefly introduces the terminology used in the CRC Design Assistant.

Objects and Classes

Abstractly, the term *object* refers to an individual, identifiable entity with a well-defined role in the problem domain. An object can be a tangible entity such as an employee or a more abstract entity such as an interface. In software, an object can be viewed externally as a black box that encapsulates information and performs services. Internally, an object maintains state, a collection of values of attributes that describe the essential and distinctive characteristics of the object. For example, an object representing an employee might have a first name "Jane" and a last name "Smith".

A class is a generic description of a set of objects that share the same behaviors. An object is an instance of a class. For example, every object representing an employee will have a first name and a last name. The object with first and last names "Jane" and "Smith" might be instances of the class *Employee*. A different instance of the same class might have names "John" and "Doe". Thus, a class is a template for an object.

Responsibilities and Collaborations

Objects exhibit behavior. A *responsibility* is an operation that an object can perform or knowledge that the object must maintain. For example, a *Cell* object in a business application might be responsible for displaying its value and knowing its location. A *SpreadSheet* object might be responsible for keeping track of a set of *Cell* objects and asking them to display themselves. An object can be viewed as a provider of services. Every object in a class can provide the same set of services. Therefore, responsibilities are associated with classes.

When an object requires the services of some other object in order to achieve a responsibility, a *collaboration* is formed. In the previous example, the class *SpreadSheet* collaborates with the class Cell in order to fulfill *SpreadSheet*'s responsibility to update the display. By separating the responsibilities for displaying the contents of a cell from the responsibility for displaying the entire spreadsheet, the design is more cohesive. It is possible to create different types of *Cell*s (e.g., for text, numbers, and formulas), each with its own formatting rules, without the need to modify the *SpreadSheet* class. One fundamental goal of the design process is to identify the objects and the collaborations between them. When a responsibility is only used within an object, the responsibility is a *private responsibility*. Responsibilities that are not private form the basis for class interfaces, and in a language such as Java, these interfaces are implemented using public methods.

Inheritance and Polymorphism

Inheritance is the ability to describe the behavior of objects of one class as a superset of the behavior of objects of another class. Programmers can create a class as a refinement of another class. This allows the common behavior to be abstracted and reused. For example, a simple graphics application might have classes for ellipses, circles, and rectangles. Each object is responsible for knowing its size and location. A programmer might declare a class *DrawingObject* that knows size and location. The classes *Ellipse*, *Circle*, and *Rectangle* could then be defined to inherit from *DrawingObject*. In this case, *DrawingObject* would be called a superclass or a parent class, and *Ellipse*, *Circle*, and *Rectangle* would be called subclasses. An object in a subclass inherits all of the capabilities and responsibilities of its superclass. Thus, anything that can be required of an object of type *DrawingObject* can also be required of a *Circle*. Only those characteristics common to every subclass can be placed in the superclass. The UML class diagram representation of this scheme is shown in Figure 1.

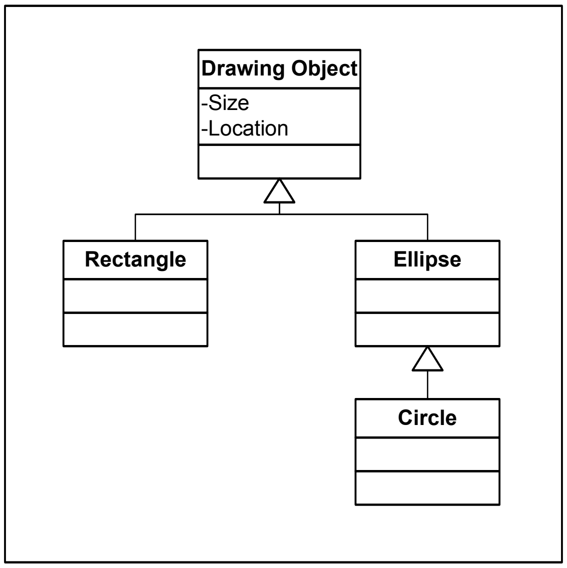


FIGURE 1
UML CLASS DIAGRAM OF GRAPHICS APPLICATION

One feature of object-oriented software is the ability for a variable to be *polymorphic*, i.e., to have more than one type. For example, if variable *c* is an object of type *Circle*, then *c* is also of type *DrawingObject*. Any method that takes a *DrawingObject* as an argument also accepts *c*.

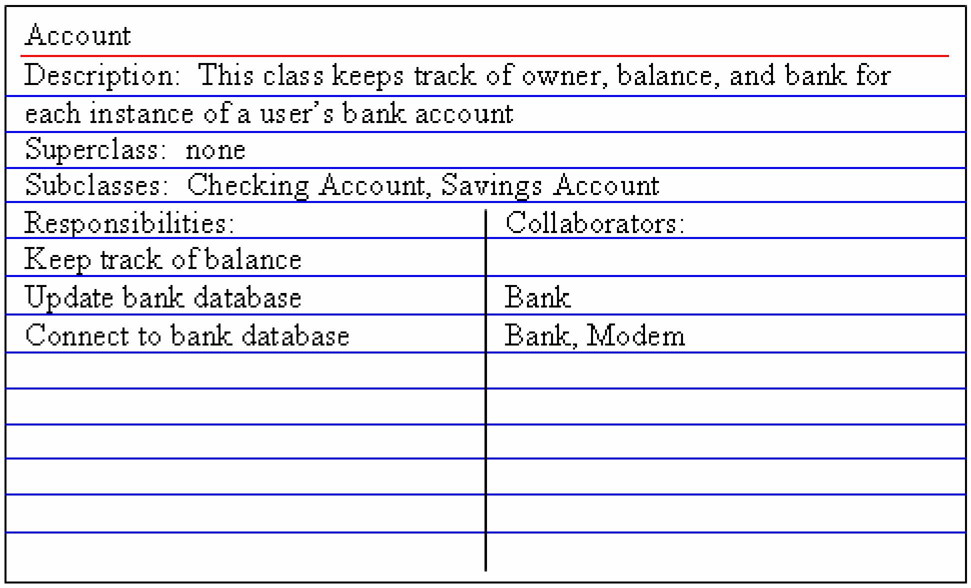
CRC

An object-oriented design method is a set or sequence of steps that ideally results in a robust, easily extended, and maintainable object-oriented design. While many design methods have been proposed and are in industrial use [2][3][5], these methods generally require that practitioners have some experience developing object models. Teaching students to analyze and construct designs for substantial software systems is challenging.

In the CRC approach, designers identify classes associated with the problem domain and responsibilities associated with the system requirements. Responsibilities are assigned to classes. Common behaviors of classes are combined, and superclasses and subclasses are formed. Collaborations between classes are identified, and these collaborations become the basis of the public interface defined for classes. A *contract* is a set of cohesive, externally visible responsibilities for a class. A contract embodies a service that the class performs on behalf of some other class and is the basis for a collaboration. The extent of coupling in a design can be visualized by drawing a line between each pair of collaborating classes, i.e., from a user of a contract to the contract owner. The design is iteratively refined and analyzed until a stable set of responsibilities is defined for each class.

Traditionally, classes, responsibilities, and collaborations are documented on 3x5 cards, and the design is developed in highly interactive group settings. When using the traditional CRC method, a design team writes information on index cards. Each card represents a class. It shows the name, description, superclass, subclasses, responsibilities, collaborations, and subsystems of a class. The advantage is that the design team can easily move the cards around to visualize the design, and modifications to the design can be made quickly by simply replacing cards. An example CRC card is shown in Figure 2.

In large designs, the number of classes and class interactions quickly becomes difficult to manage. One approach is to collect sets of classes into subsystems. Abstractly, a subsystem is a set of classes that collectively provide a set of services to classes outside the subsystem. It is possible, then, to think of a subsystem as a black box that supports contracts in much the same way that a class supports contracts. Internally, the subsystem may contain classes or other subsystems. This allows designers to view the system at varying levels of abstraction.



Account

Description: This class keeps track of owner, balance, and bank for each instance of a user's bank account

Superclass: none

Subclasses: Checking Account, Savings Account

Responsibilities:	Collaborators:
Keep track of balance	
Update bank database	Bank
Connect to bank database	Bank, Modem

FIGURE 2
EXAMPLE CRC CARD

THE CRC DESIGN ASSISTANT

In order to discuss the design of the CRC Design Assistant we will differentiate between the *CRC system* and *application systems.* The CRC system is the software that implements the CRC Design Assistant. An application system is a system that a user develops using the CRC Design Assistant environment. We will similarly refer to a *CRC design* and an *application design*.

The CRC Design Assistant was first conceived to assist students in creating real-world software applications in the context of the software engineering course at UTEP. Few of these students have the experience needed to construct an efficient modular design. Specifically, students have difficulty visualizing the design, tracking design changes, maintaining the list of system responsibilities when classes are removed, revising documentation when designs change, and presenting a design to other team members and observers. The purpose of this tool is to help alleviate these problems. The key requirements of the system are:

- The system must support the CRC design method and assist students in creating, modifying, and documenting object-oriented designs.
- The system must support a team environment and be accessible over the internet.
- The system must be able to track changes made by each team member.
- The system must automatically generate design documentation in the form of RTF files and diagrams.
- The entire design must be stored in a single database.
- The system must be easy to maintain.

The following sections describe the CRC Design Assistant.

CRC Design

The CRC system is composed of a simple Access database and a set of classes. The database schema is shown in Figure 3. The database stores all the required design information for classes, responsibilities, collaborations, contracts, subsystems, users, and documentation. The system provides tools that allow team members to document their work to facilitate communications with other team members.

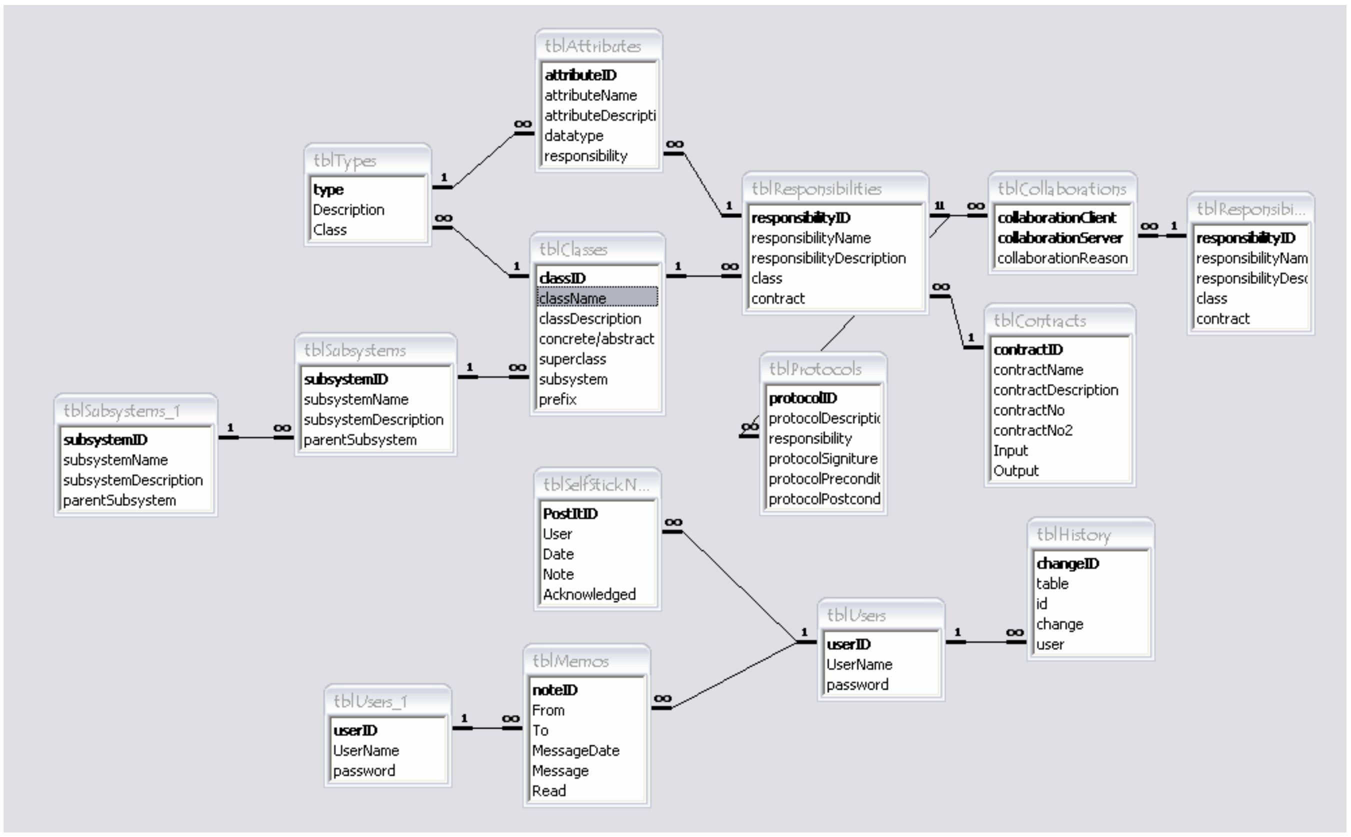


FIGURE 3
DATABASE SCHEMA FOR CRC DESIGN ASSISTANT

As shown in Figure 3, subsystems contain other subsystems. They also contain classes. Classes contain responsibilities. Responsibilities can be fulfilled through the use of attributes and/or protocols (methods). Responsibilities can be grouped into related clusters called contracts. It is not shown in the database, but responsibilities in the same contract must all belong to the same class. This is enforced in the object design. Finally, a collaboration is defined as the relationship between two responsibilities.

The object design is somewhat, but has selected differences in order to avoid lengthy runtime when searching for relationships. Figure 4 shows the UML class diagram for the portion of the CRC system that represents an application design. Subsystems and classes have an abstract superclass called a *component*. While not apparent from the diagram, components have responsibilities, contracts, and collaborations. A subsystem's responsibilities are obtained from the union of the responsibilities of the classes contained in the subsystem. Responsibilities can belong to either a class or a contract, never both. If it belongs to a contract, then is also belongs to the class that contains the contract. If it does not belong to a contract, then it is a private responsibility.

Features

Some of the most important features of this tool have to do with the way it displays an object design. One of the purposes of this system was to incorporate CRC and UML, and to maintain all of the information in one place. One advantage of the traditional CRC method is that it is easy to move the cards around to visualize the system. The CRC view shows a desktop with index cards scattered around the workspace.

The CRC view was designed to be comparable to using real index cards when going through the CRC process. This view has some advantages over using real index cards. One feature is the use of drag-and-drop to make changes to a design. By dragging one card onto another, changes are automatically made to both classes (Class A is assigned as the subclass of Class B, and Class B is declared the superclass of Class A). This assures that changes made to the design are consistent with each other. Any relationship between two classes is known by both classes, never just one. The Design Assistant also provides help with details often overlooked when implementing an application system. It becomes easier to identify classes that have too many responsibilities because of the CRC card view. Also, the diagrams generated make it easier to notice classes that are too interconnected with the rest of the system.

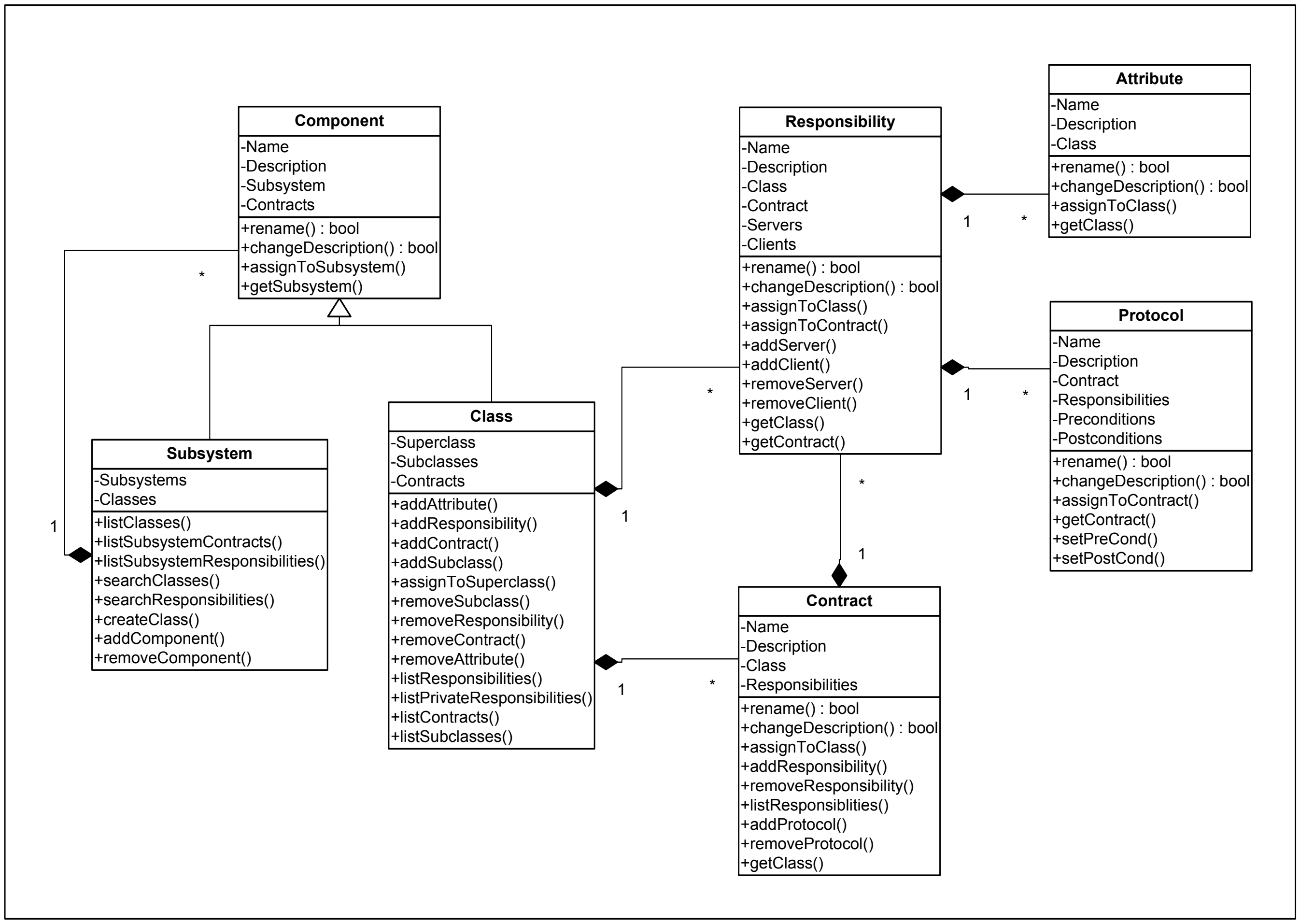


FIGURE 4
OBJECT DESIGN FOR CRC DESIGN ASSISTANT

The tool allows design changes to be made more easily. For example, when designing a system, if a class is discarded, then the card is simply thrown away. However, the responsibilities assigned to that class do not disappear. A designer would have to make sure that those responsibilities are rewritten on another index card. With this system, the responsibilities are not deleted, they are simply declared as unassigned. They can later be assigned to others classes.

The tool also provides automatic documentation for the application design. The CRC system generates the protocols document for the application system. The document is in rich text format (RTF) which can be opened with Microsoft Word or WordPerfect. It also generates comma separated variable (CSV) files that can be opened with Microsoft Visio to create CRC diagrams and object diagrams.

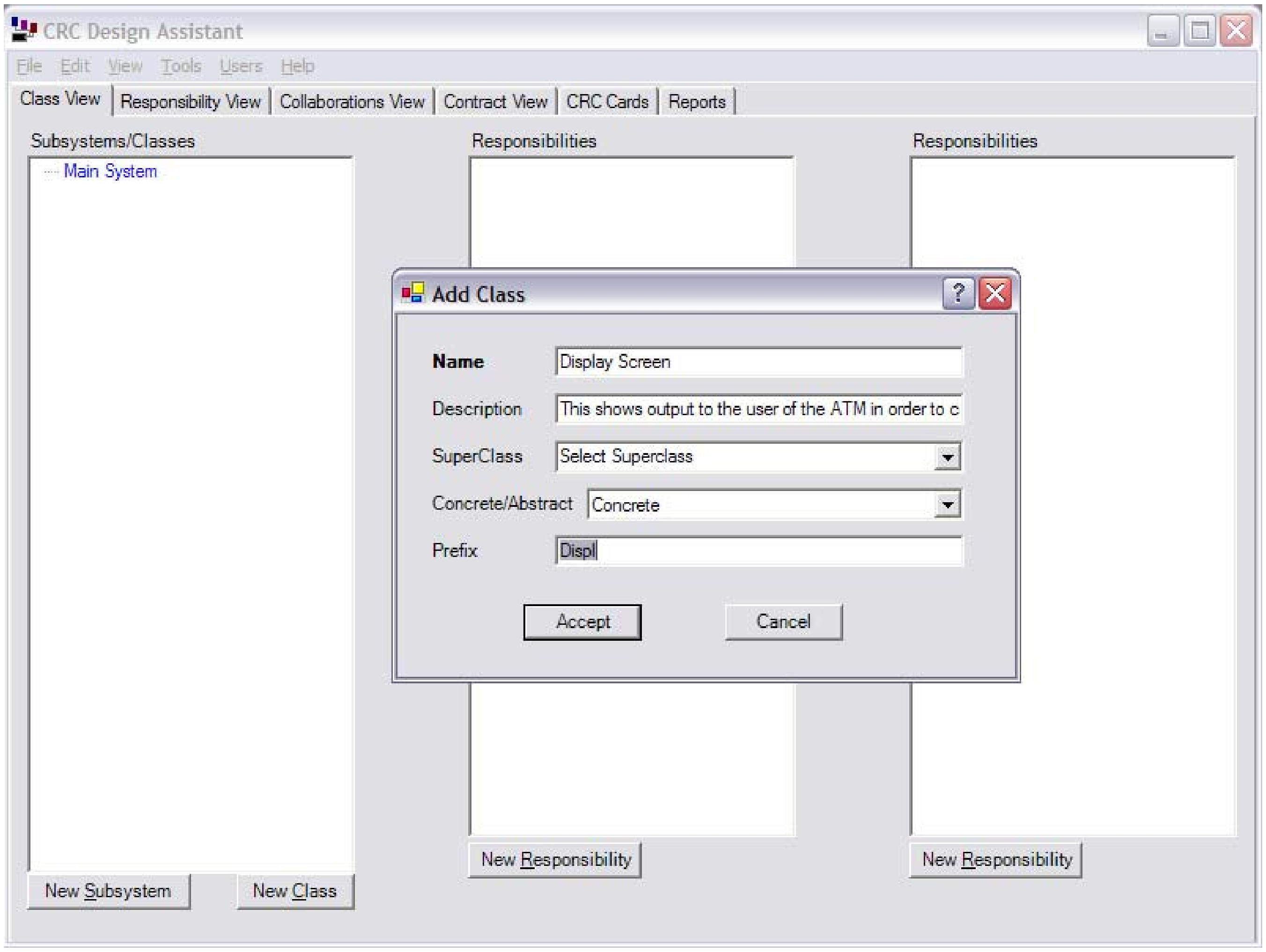
When creating a design using real index cards, the collaborations are not easy to visualize because there is no visual representation of collaborations. With the Design Assistant, lines are drawn to represent collaborations between classes. This makes it easier to visualize the extent of coupling in the system.

The tool also provides support for subsystem abstraction. Groups of classes that, together perform a set of related tasks, can be clustered together into a subsystem. This helps the designer see the design at different levels of complexity.

There are some advantages, however, that using real index cards offers over using the Design Assistant. When using index cards, the workspace can be as large as an entire table. On a computer screen, the workspace can only hold so much, and zooming out makes the cards unreadable to the user. One approach to this problem is to print CRC cards from the database and use the traditional approach, then update the database after a work session. A second approach is to use a projector or an electronic whiteboard.

ATM Example

The Automated Teller Machine (ATM) example is developed in detail using CRC in Wirffs-Brock [11]. ATM provides an example for demonstrating some of the features of the CRC Design Assistant. The ATM is a machine that allows a bank customer with an identification card to access his or her account and obtain cash. To begin designing the system, classes must be identified to represent real world objects. In this example, we will focus on a few components of the ATM, namely keypads, a bank card reader, a printer, a display screen, and a cash dispenser. The first step in using the Design Assistant is to create these classes. Figure 5 shows the initial CRC Design Assistant screen with the Create Class Dialog after selecting the "New Class" button. The dialog allows the user to enter the class name, a brief description, and, if applicable, superclass and subclasses.



Name: Display Screen
Description: This shows output to the user of the ATM in order to c
SuperClass: Select Superclass
Concrete/Abstract: Concrete
Prefix: Displ

FIGURE 5
CRC DESIGN ASSISTANT MAIN SCREEN AND CREATE CLASS DIALOG WINDOW

After the classes have been created, the class hierarchy can be refined. For example, a card reader and a keypad are both input devices. To create the hierarchy, a new class, InputDevice, is added, and then the subclasses are dragged over the superclass. The result of the operation is shown in the Class View in the Subsystem/Classes window, as shown in Figure 6.

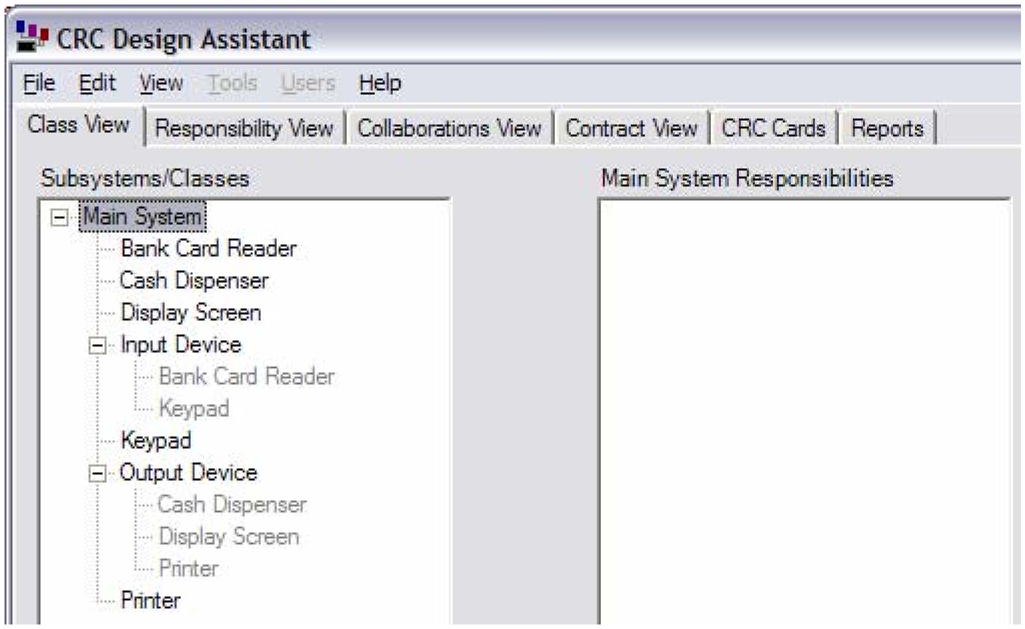


FIGURE 6
HIERARCHY OF ATM EXAMPLE

Here is a partial list of the responsibilities mentioned by Wirfs-Brock:

- display the greeting message
- read a bank card
- inform the user of unreadable cards
- prompt the user for a PIN
- transfer funds

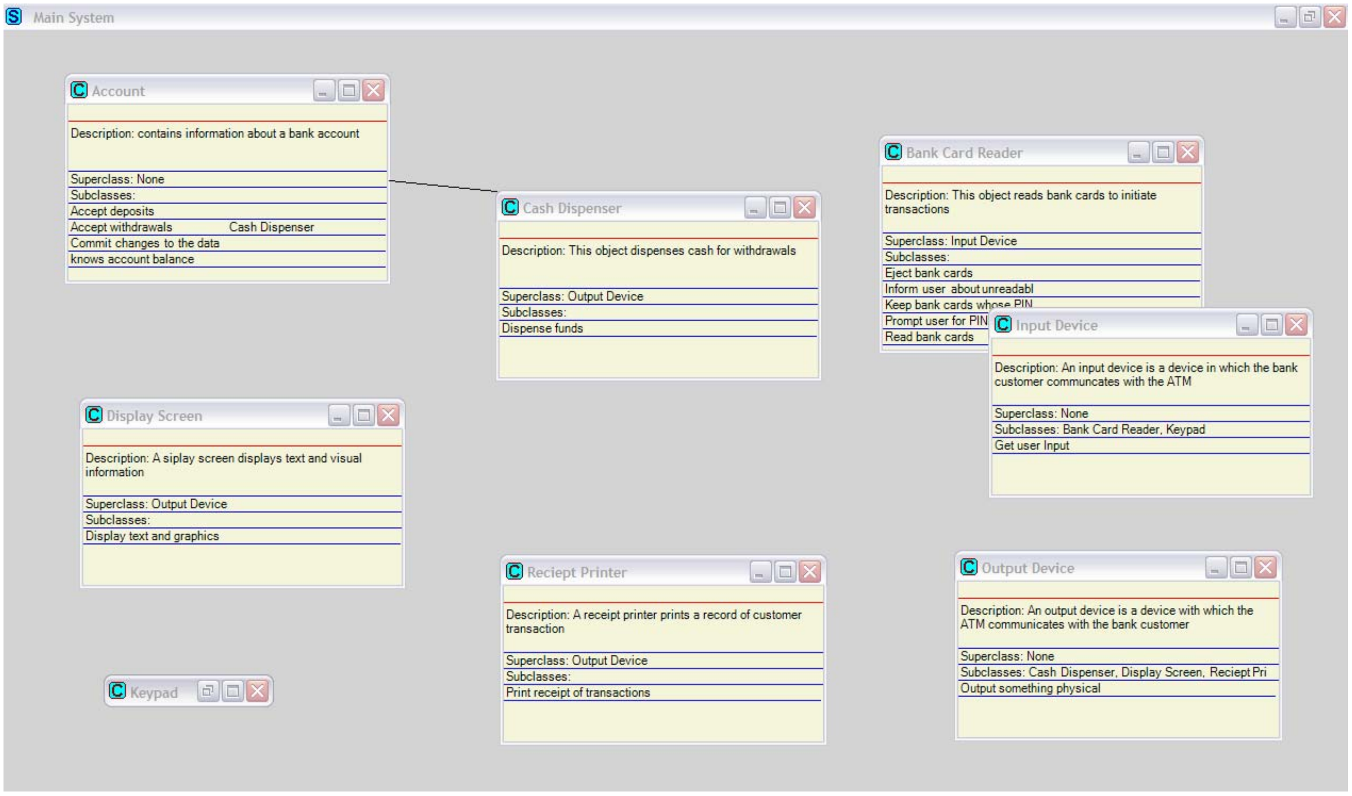
Responsibilities are added through the Create Responsibilities Dialog. They can be added directly to a class or added to the list of unassigned responsibilities. This is the list of responsibilities that are not currently assigned to classes, but must eventually be assigned before the design of the system is complete. A responsibility can be reassigned by first selecting the class (or unassigned responsibilities) in the subsystem/class list. This causes the associated responsibilities to appear in the responsibilities window to the right. A responsibility can be selected and dragged to its new class in the subsystem/class list. Classes can also be created in the "CRC Card View" shown in Figure 7.

Collaborations can be created using the "Collaborations View". For example, in order for account to accept a withdrawal, it must collaborate with cash dispenser to give money to the user. To establish this collaboration, the Account class is selected from the class list in the collaboration view. "Accept Withdrawals" is selected from the responsibilities list. A list will be shown of potential servers (responsibilities that help fulfill another responsibility). From the list "Cash Dispenser – Dispense funds" is selected (Figure 8).

Implementation

The system was implemented using the Microsoft .NET framework and C#. The.NET framework was chosen because the common language runtime (CLR) supports multiple languages and handles tasks such as memory management, security management, and error handling; ASP.NET assists building Web applications and Web services; and ADO.NET assists connecting applications to databases. Several other languages could be used with Microsoft .NET. C# was chosen because of its advantages in terms of translating object-oriented designs into code and as an opportunity to experiment with the language.

During the development of the CRC system, the CRC design was maintained on the same database used by the system to store other software systems. This means that, confusingly, the first application system to be built using this system was the CRC system. This caused several problems when using terminology to describe the CRC system. There is a class called *class* that has responsibilities, and there is a class called *responsibility* that has responsibilities.


Main System

Account
Description: contains information about a bank account
Superclass: None
Subclasses:
Accept deposits
Accept withdrawals Cash Dispenser
Commit changes to the data
knows account balance

Cash Dispenser
Description: This object dispenses cash for withdrawals
Superclass: Output Device
Subclasses:
Dispense funds

Bank Card Reader
Description: This object reads bank cards to initiate transactions
Superclass: Input Device
Subclasses:
Eject bank cards
Inform user about unreadabl
Keep bank cards whose PIN
Prompt user for PIN
Read bank cards

Input Device
Description: An input device is a device in which the bank customer communcates with the ATM
Superclass: None
Subclasses: Bank Card Reader, Keypad
Get user Input

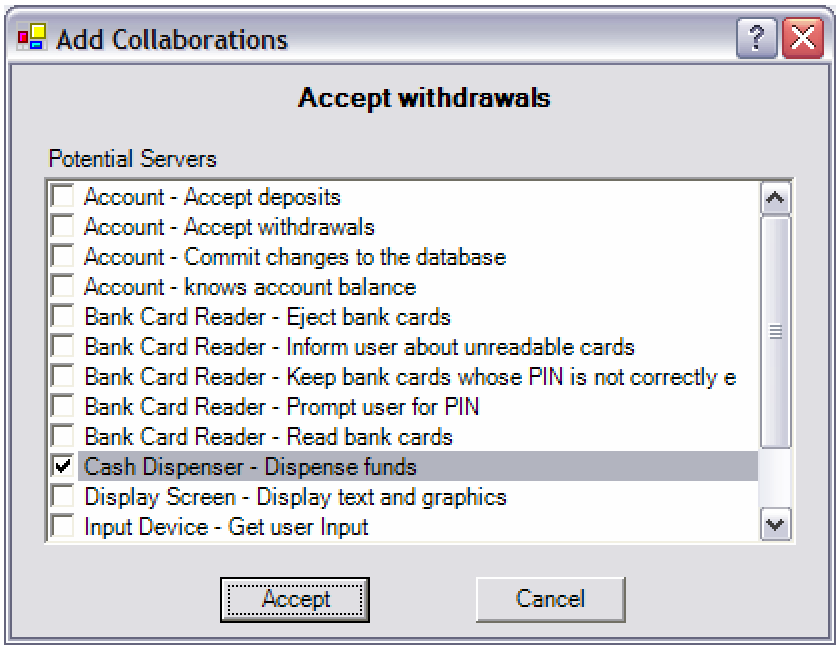
Display Screen
Description: A siplay screen displays text and visual information
Superclass: Output Device
Subclasses:
Display text and graphics

Reciept Printer
Description: A receipt printer prints a record of customer transaction
Superclass: Output Device
Subclasses:
Print receipt of transactions

Output Device
Description: An output device is a device with which the ATM communicates with the bank customer
Superclass: None
Subclasses: Cash Dispenser, Display Screen, Reciept Pri
Output something physical

Keypad

FIGURE 7
CRC CARD VIEW


Add Collaborations

Accept withdrawals

Potential Servers

- [ ] Account - Accept deposits
- [ ] Account - Accept withdrawals
- [ ] Account - Commit changes to the database
- [ ] Account - knows account balance
- [ ] Bank Card Reader - Eject bank cards
- [ ] Bank Card Reader - Inform user about unreadable cards
- [ ] Bank Card Reader - Keep bank cards whose PIN is not correctly e
- [ ] Bank Card Reader - Prompt user for PIN
- [ ] Bank Card Reader - Read bank cards
- [x] Cash Dispenser - Dispense funds
- [ ] Display Screen - Display text and graphics
- [ ] Input Device - Get user Input

FIGURE 8
COLLABORATIONS WINDOW

FUTURE WORK

There are several features that are planned to be included in the CRC system as the project evolves. One is the integration with Rational Rose. This an application used to apply UML to software development. It helps with the visualization and documentation of a software system.

Another future addition to the Design Assistant is the incorporation of web services. As mentioned earlier, this is one of the reasons that the system was implemented using Microsoft .NET. Through web services, group interaction becomes much easier. One of the problems of group work with student is the inability for students to meet all at once because of conflicting schedules. Web services may make it easier to upload progress to an application system to a server until teammates have the opportunity to review changes.

REFERENCES

[1] Albahari, B., “A Comparative Overview of C#”, http://genamics.com/developer/csharp_comparative.htm, August 2000

[2] Booch, G., *Object-Oriented Analysis and Design with Applications, 2nd* Ed., Benjamin Cummings, 1994.

[3] Coad, P., and E. Yourdon, *Object-Oriented Analysis, 2nd ed*., Prentice-Hall, 1992.

[4] CRC Design Assistant, http://www.cs.utep.edu/sroach/crcda.html

[5] Jacobson, I., G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999.

[6] Joint IEEE Computer Society/ACM Task Force on the "Model Curricula for Computing" (CC) http://www.computer.org/education/cc2001/cc2001.pdf

[7] Joint IEEE Computer Society/ACM Task Force on the "Model Curricula for Computing" (CE) http://sites.computer.org/ccse/

[8] Microsoft Corporation, “What is the Microsoft .NET Framework?” http://www.microsoft.com/net/basics/framework.asp

[9] Pressman, R., *Software Engineering A Practitioner’s Approach, 6th Edition*, McGraw Hill Higher Education, Boston, 2005.

[10] Wilkinson, N., *Using CRC Cards, An Informal Approach to Object-Oriented Design*, Cambridge University Press, 1995.

[11] Wirfs-Brock, R., Wilkerson, B., Wiener L., *Designing Object-Oriented Software*, P T R Prentice Hall, New Jersey, 1990