A Pattern for an Effective Class Responsibility Collaborator (CRC) Cards

Mohamed Fayad[1], Haitham Hamza[2], and Huáscar Sánchez[1]

[1]Computer Engineering Dept., College of Engineering, San Jose State University
One Washington Square, San Jose, CA 95192-0180
m.fayad@sjsu.edu, huascar_sanchez@yahoo.com

[2]Computer Science and Engineering Dept., University of Nebraska-Lincoln
Lincoln, NE 68588, USA
Ph: +1 402 4729492
hhamza@cse.unl.edu

Abstract

Class Responsibility Collaborator (CRC) Cards can be used in developing Object-Oriented models. They provide a simple and an easy to use way to explore objects in the development of a system. However, different problems might arise while adopting current CRC-Cards in identifying the system objects. In this paper, we discuss the main problems with current CRC-Cards. As a solution, we propose a new look at CRC-Cards that try to avoid most of the problems presented in traditional cards.

Context

In 1989, Kent Beck and Ward Cunningham introduced the notion of CRC cards at the annual OOPSLA conference [2]. The CRC-Cards technique can be used in either practical software development or in Object Oriented (OO) education. In development, CRC-Cards can offer valuable insights during the early stages of development. Another benefit of CRC Cards is in teaching Object Concepts in programming language courses. Several reported case studies have demonstrated the effectiveness of CRC Card as a tool for introducing OO programming concepts and for improving the understanding of objects/classes [1].

Problem

Current CRC-Cards lack some essential qualities that might affect the effectiveness of the developed system that uses them. The main problems in current CRC-Cards can be summarized in the following points:

1. Possibility of Low Cohesion and High Coupling: Because there are no limitations on the number of responsibilities allowed for a given class, it is possible to overload a class with too many responsibilities resulting in low cohesion within the model. A cohesive class should ideally have just one responsibility [3, 5]. Excessive responsibilities could also lead to a large number of collaborators required to support them. Too many collaborations between classes will produce high coupling and needlessly increase the complexity of the system. In software design, we strive for just the opposite –

high cohesion and the lowest possible coupling.

2. Macho Classes: Multiple responsibilities can also result in the creation of macho, classes. A macho class instantiates an object that performs most of the work, leaving all of the minor operations to a set of essentially useless classes [3]. Ideally, the system intelligence should be distributed as evenly as possible across the application and the work shared uniformly. When all of the intelligence is concentrated in one or two classes, it also increases the difficulty of making changes.

3. Exclusion of Services: Another problem with these CRC cards is the exclusion of the services provided by the class [5]. By including the services on the CRC card, the classes can be checked for duplicate functionality [3]. By identifying duplicate functionality, it may be possible to combine or consolidate classes that perform similar functions. In addition, because the responsibility of a class is merely a summary of its operations, explicitly providing the services performed by a class may help verify the responsibility is properly defined.

4. No Clear Role is Defined: The absence of a class role may lead to assigning wrong, useless, or even missing responsibilities. Although the role seems fairly insignificant, it serves a very important purpose. If a class performs more than one role, it is possible that a generalization exists where each role is actually a subclass of some superclass [3]. Humans provide a good example of performing multiple roles; a woman could be a mother, a wife, a daughter, etc. By defining the role, generalizations and specializations can be explored early in the process.

5. Difficulty in Defining Responsibilities: Coming up with class responsibilities can be a difficult task, especially with the absence of a clearly defined role [5]. It is easy to get off track and assign responsibilities that are either ambiguous or irrelevant. It isn't until the developer begins to actually map the CRC cards to various use case scenarios or the class diagram that these extraneous responsibilities are realized.

6. Difficulty in Mapping: Multiple responsibilities can make it difficult to map the classes identified by the CRC Cards to the actual class diagram and use case scenarios. When numerous responsibilities are assigned to one class, the interactions can become complex [3]. This complexity carries over when the CRC cards are used to map use cases. The use cases are provided to determine if the class model provides the necessary functionally to support all possible scenarios. Because the responsibilities of a class are actually a summary of its functionality, mapping can be greatly complicated when the class's operations are tied to multiple responsibilities.

Forces

For CRC-Cards to enhance the development of systems, main essential quality factors should be satisfied [4, 5]. Yet, satisfying these qualities is not easy. The following summarizes the main points that shows writing an effective CRC-Card is not straightforward:

- A major advantage of the CRC-Cards tool resides in its simplicity to understand [4]. Current CRC-Cards consists of three elements: name of the class, its responsibilities, and its collaborations. However, such simplicity may not convey all the required information needed in the following steps in the development. For example, collaboration section in current CRC-Cards does not provide any information about what kind of collaboration there is. In other words, the card does not specify the services that its class offers to the other classes that collaborate with it. Such information is important in developing the class diagram and in verifying its accuracy. On the other hand, having too much information in a CRC-Card might preclude them from being widely applied. They might become difficult to understand, or to use. This may scarify the simplicity of the technique. Therefore, compromising between completeness and simplicity should be considered when writing CRC-Cards [4].

- It is important to understand the role of each class in the system in order to identify its responsibilities [3]. A class within the system might have several roles; however, current CRC-Cards do not provide a way to differentiate between the different roles of the same class. How can we handle multiple roles for the same class in a simple way?
- Defining the class responsibility is crucial for developing an accurate class diagram and latter an effective system. A class might have several responsibilities within the system; however, identifying all these responsibilities in one CRC-Card might create great confusion for the developers [5]. This is because different responsibilities for a class can result in different collaborations between this class and other classes in the system. By listing all the responsibilities and collaborations of a class in one CRC-Card, it becomes confusing to match a responsibility of a class to another collaborating class in the system. This complicates the deriving of the system class diagram from the written CRC-Cards.

Solution

In this section we present an enhanced representation for CRC cards as a solution to some of the problems in current CRC-Cards. The new version will include a clear role for each class which will aid in the discovery of superclasses and their respective subclasses. This class role will also be useful when defining the class responsibility. Each class will be allowed to have only one unique responsibility. If more than one responsibility is identified, additional classes should be formed. Limiting responsibilities will help prevent low cohesion and high coupling as well as reduce the possibility of macho classes. Finally, the revised CRC card will include the services offered by each class. This will help verify the validity of the class responsibility as well as ensure that overlapping functionality is avoided. The proposed CRC-Card format is shown in Figure 1 below.

Creating CRC-cards with the proposed format requires three main steps (See Figure 2):

(1) Class Identification.

(2) Assigning Roles and Responsibilities.
(3) Discovering Collaborators.

Each step is concern with filling one element of the proposed CRC Card at a time. In summary, relevant classes of the system will be identified. Then, proper roles would be assigned to them based on a number of responsibilities found in particular design scenarios. Collaborators or Clients of each class would be discovered to account for the certain inabilities of some classes to fulfill its assigned responsibility [7]. This process will be aided by the explicit declaration of the external services offered by each relevant class.

Applicability: An Example

In this section we show an example of applying the proposed CRC-Card on a simple problem.

Problem statement

The services that are linked with the word *Genealogy* have been growing tremendously across the Internet landscape. This idea has been touched by several online businesses, such as Ancestry.com, Msn, etc., but it is still in its infancy. We propose a simple Family Tree design. This system will offer a central storage device, where all the information of current members will be stored. Each member will have full control of his/her information portrayed in the system.

The system will provide consistent updating, searching and tracking mechanisms to facilitate an easy interaction between the system and the members throughout the entire Family Tree. An efficient user-friendly navigation mechanism will be presented as well. This will guarantee full access to all the features of the system. Members of the system will be able to share this experience by inviting new users to either start up their own family tree, and/or enroll in a member's family tree. The latter would happen in the case that these potential members are relatives of an already enrolled member. Regarding guests of the system, there will be a section exclusively for them, allowing them to visit current Family trees with certain limitations. They can also create their own family tree if they desire.

Step 1: Identifying Classes

The Class identification process does not vary in both approaches. Both approaches use similar techniques [6, 7] along with the overall knowledge on the subject from analysts, and designers. Similar naming conventions are applied, except they vary when dealing with compound nouns. Compound nouns are treated as one word, and no spaces are allowed between these two words. This is sometimes called "camel-casing". For method declarations the accepted notation is just as it is for classes *except* that the first letter is in lowercase. The following is an example of a possible class in the presented problem (See Figure 3):

Step 2: Assigning Roles and Responsibilities

The proposed CRC Card format offers much more to the process of discovering responsibilities than the current one. The presence of a well-defined role makes things easier for the analyst because each role is tightly bound to a unique responsibility [3, 5]. Therefore, the analyst can map the distinct responsibilities per class based on particular scenarios. Other techniques may be used to assist this process [7].

A class that contains multiple responsibilities will be partitioned into several classes [5]. No naming rules are required in this step. However, for understanding purpose, analysts need to define clear and cohesive responsibilities. The following is an example of assigning Responsibilities and Roles (See Figure 4):

Step 3: Discovering Collaborators.
The goal of this step is the definition of a set of methods that will help to achieve a responsibility. The most common techniques used for services identification is the application of a grammatical parse over the problem statement, looking for verbs and/or verbs phrases [3, 7]. These verbs or verb phrases usually corresponds to the methods used to fulfill certain functionality or unique responsibility of a particular class. They will be explicitly nested on the right compartment of the Collaboration section of the proposed CRC Card. This compartment is name *Server*. They obey certain naming rules to assure a proper definition. The Identifying Classes section refers to these naming rules.

The inclusion of these services in the proposed CRC Card will enhance the ability for responsibility identification, making it a very straightforward process. Along with this inclusion of services, several classes that communicate with one particular class will be placed on the left compartment of the Collaboration section. This section is called *Clients*. These classes are called clients because they collaborate with a particular class, by requesting services from it. These requests are performed based on a message-wise action. These classes (Clients) are usually identified at the moment of establishing an interaction between classes and its surroundings in a particular design scenario (e.g. Use Case Scenario) [3]. This step not just a common step. Having exhibited two more sections of this CRC Card, we have granted the accomplishment of several quality factors at the same time (e.g. Self-descriptive Services, Explicit notion of Collaborators) [5]. This result will also increase the level of understandability for analysts and designers towards the proposed CRC Card structure.

Coming up with classes (Clients) is a repeatable process done by analysts/developers and it will be completed only when the analysts/designers feel that they have covered all the distinct design scenarios. The following is an example of filling out the Collaboration section of the proposed CRC Card (See Figure 5).

Conclusion

In this paper, we proposed an enhanced version of the CRC Card. The proposed version of the CRC Cards provides a solution to some of the problems that current CRC-Cards may embody. The high-level process of writing the propose card format is also presented. In addition, we demonstrated, through a simple example, the use of the new CRC- Card format.

References

[1] Jürgen Börstler, Thomas Johansson, and Marie Nordström , "Teaching OO Concepts—A Case Study Using CRC-CARDS and BLUEJ", 32nd ASEE/IEEE Frontiers in Education Conference, November 6 - 9, 2002, Boston, MA.

[2] Kent Beck, Ward Cunningham, "A Laboratory for Teaching Object Oriented Thinking", OOPSLA, Volume 24, Number 10, October 1989.

[3] M.E. Fayad. Object-Oriented Analysis and Design Lecture notes, Full 2000 to Spring 2003.

[4] M.E. Fayad, Huáscar Sánchez, and Hamza Hamza. Pattern Languages for CRC Cards, in progress.

[5] M.E. Fayad, Valeri Stanton, Huáscar Sánchez, Hamza Hamza, , A closer look at CRC Cards, in progress.

[6] Leszek A. Maciaszek, "Requirements Analysis and System Design – Developing Information System with UML," Addison-Wesley Publishing Company, 2001.

[7] Roger S. Pressman, "Software Engineering – A Practitioner's Approach," McGraw Hill Publishing Company, 2001.

Class (Role)		
Responsibility	**Collaboration**	
	Client	**Server**

Figure 1. Proposed CRC-Card Format

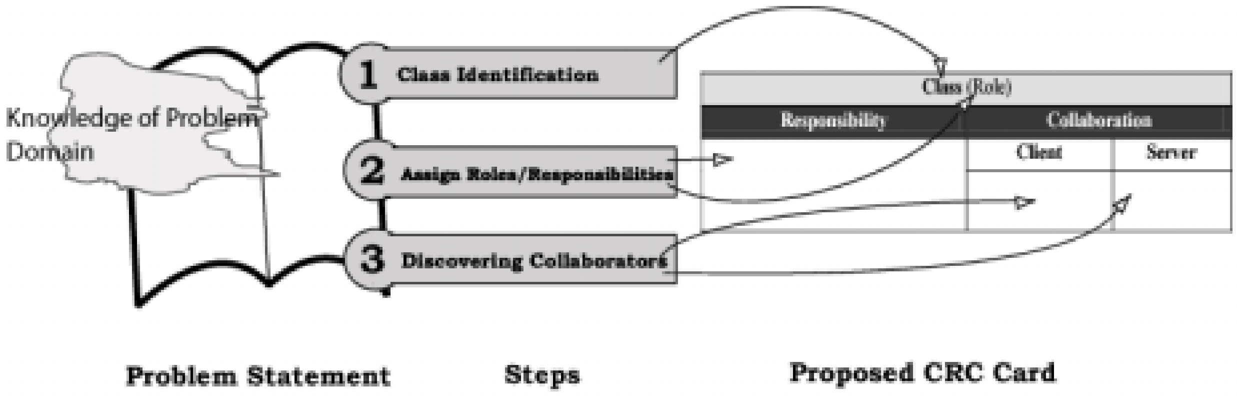

Figure 2: Applying the Proposed CRC-Card

FamilyTree (Role)		
Responsibility	**Collaboration**	
	Client	**Server**

Figure 3: Representation of a possible Class presented in the Problem

FamilyTree (FamilyTree)		
Responsibility	**Collaboration**	
Illustrate the bond of a group of people	**Client**	**Server**

Figure 4: Assigning Responsibility and Role.

FamilyTree (FamilyTree)		
Responsibility	**Collaboration**	
To show relationships between people	**Client**	**Server**

	Member
	Family
		initiateTree()
		connectFamily()
		joinToTree()
		search()

Figure 5: Filling Out the Collaboration Section.