From Domain Classes and Use Cases to Design Classes
CRC Cards Technique ("Responsibility-Driven Design")

- Informal, non-detailed
- Used for group brain-storming
- End result is a first cut at classes for an object-oriented model,
- Not intended to provide a complete design
Use-Case Input

- A good **starting point** for CRC analysis is a clear statement of all of the use-cases.

- Use-cases drive the introduction of CRC cards.

- Use-cases, or their accompanying scenarios, can be used as a kind of script for the **role-playing method** of checking the CRC cards.

- The role-playing could be replaced with sequence diagrams.
CRC

- Stands for:
  - Classes
  - Responsibilities
  - Collaborations

- (Not as in “CRC Handbook”: “Chemical Rubber Company” )
CRC

- Stands for:
  - Classes (of objects)
  - Responsibilities (of the objects in each class)
  - Collaborations (with objects in other classes)
    - In UML, these will be examples of "associations"

- Remember that an application may have "singleton" classes (classes instantiated only once).
Origin of CRC

- Kent Beck and Ward Cunningham, formerly of Tektronix in Oregon
- Rebecca Wirfs-Brock popularized with “Responsibility-Driven Design” (RDD)
References


The Basic Idea

- Develop set of cards or cards images.
- Each card represents one class.
- A card contains:
  - The name of the class.
  - The responsibilities of the class.
  - Collaborations: other classes with which this class inter-operates, in conjunction with the attendant responsibility.
CRC cards represent a *static* view of the system’s classes.

- Domain class diagrams similarly static.
- Must eventually be augmented by dynamic description, e.g. sequence diagrams.
- Informal dynamic description can be acted out with “role-playing”, similar to the creation of scenarios for use cases.
Use index cards, or single PowerPoint slides.

Limiting the size of a card is an attempt at preventing the class from becoming too complex.
Sample Application:
A graph-drawing program

Possible screen image

Typical Application
Use-Cases:
- Draw shape
- Move shape
- Resize shape
- Connect shapes
- Erase shape
- Erase connector
Example of CRC card for a graph-drawing program (1)
Example of CRC card for a graph-drawing program (2)

Shape

- Remember size
- Remember position
- Remember fill color
- Remember border
- Remember connectors
- Change size
- Change position

Responsibilities
Example of CRC card for a graph-drawing program (3)

Shape

Remember size
Remember position
Remember fill color
Remember border
Remember connectors
Change size
Change position

Line

Connector

Collaborations
Example of CRC card for a graph-drawing program (4)

- Shape
  - super class: Drawable
  - sub-classes: Rect, Oval, Group
  - Remember size
  - Remember position
  - Remember fill color
  - Remember border
  - Remember connectors
  - Change size
  - Change position

- Line
  - Connector

Super- and Sub-classes
Example of CRC card for a graph-drawing program (5)

Note: The Drawable doesn’t necessarily need to remember a Canvas, since the Canvas could be passed as an argument to the `draw` method.
Example of CRC card for a graph-drawing program (6)

Canvas
Remember
Drawables
contained in self.

super class:

sub-classes:

Drawable

Responsibilities

Collaborations
Note:

- Responsibilities are usually for *members* (objects) of the class rather than the class itself, although

- Class-wide responsibility is possible (corresponding to *static* method)
An object of a class typically has one or more attributes.

Attributes have values that specify or describe the object.

A value might or might not deserve the distinction of being an object itself; it depends on what we intend to do with the attribute.

A would-be attribute that is object-valued is actually a collaboration (association in UML).
Immutable Objects

- Some objects only “know”, but don’t “do” anything.
- They can’t be changed once created, and therefore are called immutable.
- Values of attributes are often either immutable objects or scalars (non-objects).
- Can immutable objects have collaborations?
CRC Team Structure

- Usually ≤ 6 person team is recommended
- The team can include clients as well as developers (even though we are partly in the design phase)
  - 1-2 domain experts
  - 1-2 analysts
  - experienced object-oriented designer
  - leader
Once the CRC cards are constructed ...

- Team can engage in **role-playing** to verify that use-case **scenarios** make sense for chosen CRC.

- Each person can role-play one or more class cards.

- If something doesn’t work, change the class accordingly.

- Revision of use-cases might also be indicated.
Use-Case to Class
Traceability Matrix Example
(from the graph-drawing example)

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Class: Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Draw shape</td>
<td>x</td>
</tr>
<tr>
<td>Move shape</td>
<td>x x x</td>
</tr>
<tr>
<td>Erase shape</td>
<td>x x x x</td>
</tr>
<tr>
<td>Resize shape</td>
<td>x x x x</td>
</tr>
<tr>
<td>Connect shapes</td>
<td>x x x x x</td>
</tr>
<tr>
<td>Erase connector</td>
<td>x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Draw shape remember components</th>
<th>Shape: draw</th>
<th>Shape: remember position</th>
<th>Shape: remember size</th>
<th>Shape: remember connectors</th>
<th>Connector: draw</th>
<th>Connector: remember start</th>
<th>Connector: remember end</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
Contracts

- Pre- and Post-Conditions of a Use Case could be considered as a contract:
  - Environment ensures pre-condition
  - Use-case ensures the post-condition
Use of Pre- and Post-Conditions

- Post-condition can be considered a precise statement of what is accomplished by a use-case.

- Pre-condition can be considered a precise statement of what can be assumed.
**Goal:** To enable a set definer to dissolve a set no longer used.

**Actors:** Set definer

**Initiator:** Set definer

**Description:** The set definer dissolves an existing list. Participants on are notified.

**Pre-conditions:** A set with the label exists.

**Post-conditions:** No set with the label exists.

**Exception:** No set with the label exists.

**Options**

**Scenario**

---

**Example Use Case from Meeting Scheduler:**

**DissolveSet:** Dissolve a Participant Set Having a Given Label

---

- **Set Definer**
- **Participant Set**
  - **Label**
- **Participant**
Strength of Conditions

- The stronger the pre-condition, the easier life is for the implementer.

- The stronger the post-condition, the harder it is.
Post-Conditions are Not Actions

- Post-conditions are properties of state, not actions.

- They do not address how the property is obtained, only that it is obtained.
“Design by Contract”

- This is a slogan being pushed by Bertrand Meyers (inventor of the Eiffel language).

- Meyers’ interpretation is quite strict, and relates to class implementation (more later).