From Domain Classes and Use Cases to Design Classes

Contracts (Larman, ch 13)

- 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.

Strength of Conditions

- The stronger the pre-condition, the easier life is for the implementer.
- The stronger the post-condition, the harder it is.
- What is the strongest possible condition? the weakest?

Larman's Version of Post-Condition is Dubious

- Larman would consider the following to be a valid post-condition in the running POS (Point-of-Sale) example (p 181):

  "A SalesLineItem sli was created"

- What should it be?

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.

Post-Conditions are Not Actions

- An sli might have been created, but it could also have been subsequently destroyed in the same use-case.

Design by Contract

- This is a slogan being pushed by Bertrand Meyers (inventor of the Eiffel language).
- Larman seems to be trying to pull this in.
- Meyers' interpretation is quite strict, and relates to class implementation (more later).

Noun/Verb Heuristic for Transforming Use-Case Descriptions to Implementation Classes

- **Nouns** suggest classes or attributes, as already discussed
- **Verbs** suggest methods or "responsibilities"

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:
  - **C**lasses
  - **R**esponsibilities
  - **C**ollaborations
- (Not as in "CRC Handbook": "Chemical Rubber Company")

CRC

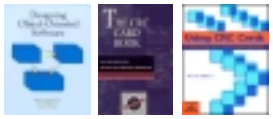
- Stands for:
  - **C**lasses (of objects)
  - **R**esponsibilities (of the objects in each class)
  - **C**ollaborations (with objects in other classes)
    - In UML, these will be examples of "associations"
- Remember that an application may have "singleton" classes (classes instantiated only once).

Origin of CRC

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

References

- Rebecca Wirfs-Brock, Brian Wilkerson, Lauren Wiener, *Designing Object-Oriented Software*, Prentice-Hall, 1990.
- David Bellin and Susan Simone, *The CRC Card Book*, Addison Wesley Longman, 1997.



The Basic Idea

- Develop set of index cards.
- 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.

Image of CRC cards

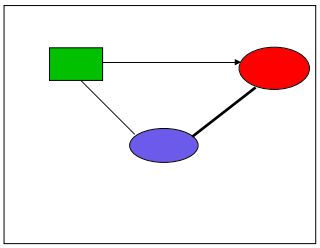
Class Name	super class sub-classes
Responsibilities	Collaborations
_____	_____
_____	_____
_____	_____

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)

Class → Shape

Example of CRC card
for a graph-drawing program (2)

Responsibilities →

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

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

Shape	
Remember size	
Remember position	
Remember fill color	
Remember border	→ Line
Remember connectors	→ Connector
Change size	
Change position	

← 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		Line
Remember connectors		Connector
Change size		
Change position		

← Super- and Sub-classes

Example of CRC card
for a graph-drawing program (5)

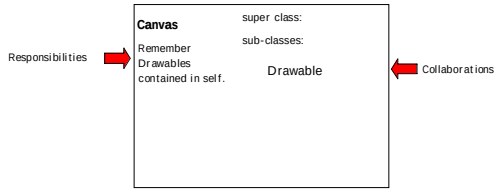
Responsibilities →

Drawable	super class:
	sub-classes: Shape
Draw self on canvas	Canvas

← Collaborations

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)



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)

Attribute Value vs. Object

- 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 $\leq$ 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)

Class: Responsibility

Use Case	Drawing: remember components	Shape: draw	Shape: remember position	Shape: remember size	Shape: remember connectors	Connector: draw	Connector: remember start	Connector: remember end
Draw shape	x	x	x	x				
Move shape		x	x		x			
Erase shape	x				x		x	x
Resize shape		x	x	x	x			
Connect shapes	x				x	x	x	x
Erase connector	x				x			