Object-Oriented Software Analysis and Design

School of Computer Science
University of Windsor

Designing Objects with Responsibilities

- *Understanding responsibilities is key to good object-oriented design*—Martin Fowler

Object Design: Example Inputs, Activities, and Outputs

- What's been done? Prior activities (e.g., workshop) and artifacts.
- How do things relate? Influence of prior artifacts (e.g., use cases) on OO design.
- How much design modeling to do, and how?
- What's the output?

Artifact relationships emphasizing influence on OO design

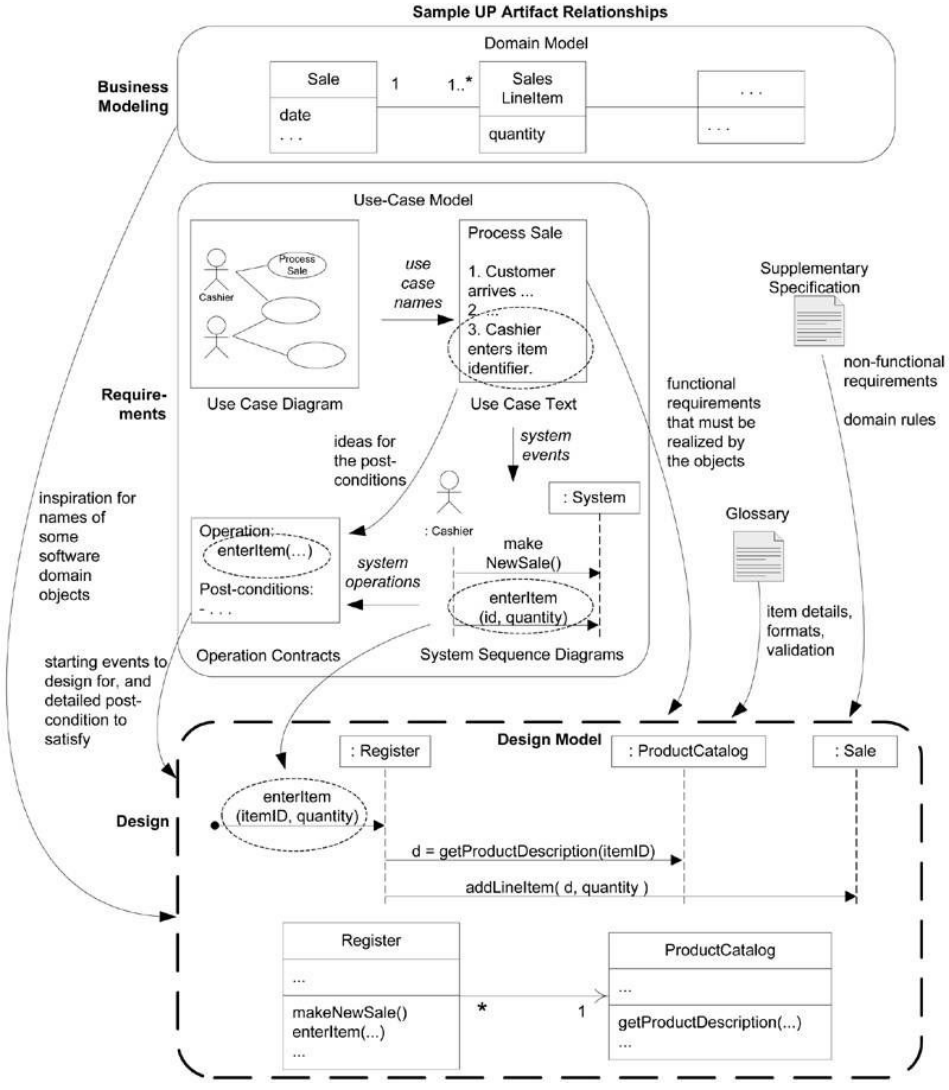

Figure: *Artifact relationships emphasizing influence on OO design.*

What are inputs to OOD?

The first ***two-day requirements workshop*** is finished.	The chief architect and business agree to implement and test some ***scenarios of Process Sale in the first three-week*** timeboxed iteration.
Three of the twenty use casesthose that are the most architecturally significant and of high business valuehave been analyzed in detail, including, of course, the *Process Sale* use case. (The UP recommends, as typical with iterative methods, analyzing only ***10%20% of the requirements*** in detail before starting to program.)	***Other artifacts*** have been started: Supplementary Specification, Glossary, and Domain Model.
Programming experiments have resolved the show-stopper technical questions, such as whether a Java Swing UI will work on a touch screen.	The chief architect has drawn some ideas for the ***large-scale logical architecture***, using UML package diagrams. This is part of the UP Design Model.

What are inputs to OOD?

The ***use case text*** defines the visible behavior that the software objects must ultimately supportobjects are designed to "realize" (implement) the use cases. In the UP, this OO design is called, not surprisingly, the **use case realization**.	The ***Supplementary Specification*** defines the non-functional goals, such as internalization, our objects must satisfy.
The ***system sequence diagrams*** identify the system operation messages, which are the starting messages on our interaction diagrams of collaborating objects.	The ***Glossary*** clarifies details of parameters or data coming in from the UI layer, data being passed to the database, and detailed item-specific logic or validation requirements, such as the legal formats and validation for product UPCs (universal product codes).
The ***operation contracts*** may complement the use case text to clarify what the software objects must achieve in a system operation. The post-conditions define detailed achievements.	The ***Domain Model*** suggests some names and attributes of software domain objects in the domain layer of the software architecture.

What are activities of Object Design?

- Given one or more of these inputs, developers

  1. start immediately coding (ideally with test-first development),
  2. start some UML modeling for the object design, or
  3. start with another modeling technique, such as CRC cards

What are the outputs?

- specifically for object design, UML interaction, class, and package diagrams for the difficult parts of the design that we wished to explore before coding.
- UI sketches and prototypes
- database models (with UML data modeling profile notation)
- report sketches and prototypes

Responsibilities and Responsibility-Driven Design

- A popular way of thinking about the design of software objects and also larger-scale components is in terms of
  - responsibilities,
  - roles, and
  - collaborations.
- This is part of a larger approach called **Responsibility-Driven Design or RDD**.

Responsibilities and Responsibility-Driven Design (contd.)

- In RDD, we think of software objects as having responsibilities—**an abstraction of what they do.**
- Responsibilities are **related to the obligations or behavior of an object** in terms of its role.
- Basically, these responsibilities are of the following **two** types:
  - Doing
  - Knowing.

Responsibilities and Responsibility-Driven Design (contd.)

- **Doing** responsibilities of an object include:
  - doing something itself, such as creating an object or doing a calculation
  - initiating action in other objects
  - controlling and coordinating activities in other objects

Responsibilities and Responsibility-Driven Design (contd.)

- **Knowing** responsibilities of an object include:
  - knowing about private encapsulated data
  - knowing about related objects
  - knowing about things it can derive or calculate

Responsibilities and Responsibility-Driven Design (contd.)

- Responsibilities are assigned to classes of objects during object design.

  For example,

  I may declare that "a `Sale` is responsible for creating `SalesLineItems`" (a doing), or "a `Sale` is responsible for knowing its total" (a knowing).

Responsibilities and Responsibility-Driven Design (contd.)

- **A responsibility is not the same thing as a method—it's an abstraction—but methods fulfill responsibilities.**

- RDD also includes the idea of **collaboration**. Responsibilities are implemented by means of methods that either act alone or collaborate with other methods and objects.

  For example,

  the *Sale* class might define one or more methods to know its total; say, a method named *getTotal*. To fulfill that responsibility, the *Sale* may collaborate with other objects, such as sending a *getSubtotal* message to each *SalesLineItem* object asking for its subtotal.

RDD is a Metaphor

- RDD is a **general metaphor** for thinking about OO software design. Think of software objects as similar to people with responsibilities who collaborate with other people to get work done.

- RDD leads to viewing an OO design as a **community of collaborating** responsible objects.

- **Key point:** GRASP names and describes some basic principles to assign responsibilities, so it's useful to know— to support RDD.

It's Quiz Time

1. A responsibility is not the same thing as a method—it's an abstraction—but methods fulfill responsibilities. (True or False)

2. Initiating action in other objects is one of the knowing responsibilities. (True or False)

3. In RDD, we think of software objects as having responsibilities—an abstraction of what they do. (True or False)