Object-Oriented Software Analysis and Design

School of Computer Science
University of Windsor

Requirements To Design - Iteratively

Iteratively Do the Right Thing, Do the Thing Right

- The requirements and object-oriented analysis covered so far has focused on learning to **do the right thing**; that is, understanding some of the outstanding goals for the case studies, and related rules and constraints.

- By contrast, the following design work will stress **do the thing right**; that is, skillfully designing a solution to satisfy the requirements for this iteration

Iteratively Do the Right Thing, Do the Thing Right (contd.)

- In iterative development, a transition from primarily a requirements or analysis focus to primarily a design and implementation focus will occur in each iteration.
- Early iterations will spend relatively more time on analysis activities.
- In later iterations it is common that analysis lessens; there's more focus on just building the solution.

Provoking Early Change

- It is natural and healthy to discover and change some requirements during the design and implementation work, especially in the early iterations.
- Iterative and evolutionary methods "embrace change" although we try to provoke that inevitable change in early iterations, so that we have a more stable goal (and estimate and schedule) for the later iterations.
- Early programming, tests, and demos help provoke the inevitable changes early on.
- This sounds a simple idea, yes, it lies at the heart of why iterative development works.

Didn't All That Analysis and Modeling Take Weeks To Do?

- When one is comfortable with the skills of use case writing, domain modeling, and so forth, the duration to do all the actual modeling that has been explored so far is realistically just a few hours or days.
- However, that does not mean that only a few days have passed since the start of the project.
- Many other activities, such as proof-of-concept programming, finding resources (people, software,. . .), planning, setting up the environment, and so on, could consume a few weeks of preparation.

On to Object Design

How do you design objects?

1. ***Code***
   - Design-while-coding (Java, C#,...). From mental model to code.

2. ***Draw, then code***
   - Drawing some UML on a whiteboard or UML CASE tool, then switching to #1 with a text-strong IDE (e.g., Eclipse or Visual Studio).

3. ***Only draw***
   - Somehow, the tool generates everything from diagrams. Many tool vendor has washed onto the shores of this steep island. "Only draw" is a misnomer, as this still involves a text programming language attached to UML graphic elements.

If we use **Draw, then code** (the most popular approach with UML), the drawing overhead should be worth the effort.

Agile Modeling and Lightweight UML Drawing

- Reduce drawing overhead and model to understand and communicate, rather than to document.
- Modeling with others.
- Creating several models in parallel.

  For example,

  five minutes on a wall of interaction diagrams, then five minutes on a wall of related class diagrams.

UML CASE Tools: Guidelines

- Choose a UML CASE tool that integrates with popular text-strong IDEs, such as Eclipse or Visual Studio.
- Choose a UML tool that can reverse-engineer (generate diagrams from code) not only class diagrams (common), but also interaction diagrams (more rare, but very useful to learn call-flow structure of a program).
- Many developers find it useful to code awhile in their favorite IDE, then press a button, reverse-engineer the code, and see a UML big-picture graphical view of their design.

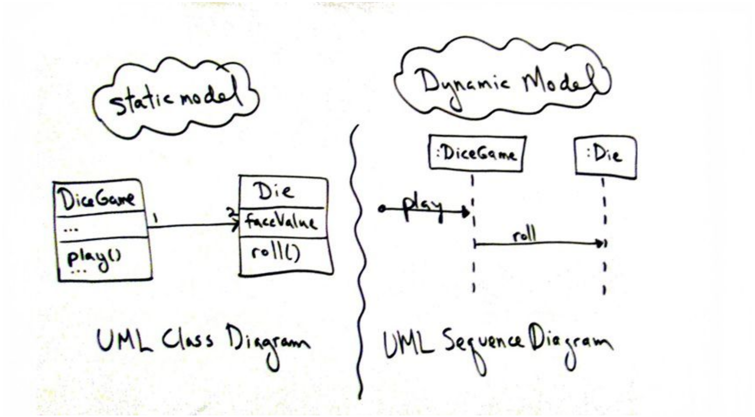
How Much Time Spent Drawing UML Before Coding? Guideline

- For a three-week timeboxed iteration, spend a few hours or at most one day (with partners) near the start of the iteration "at the walls" (or with a UML CASE tool) drawing UML for the hard, creative parts of the detailed object design.
- Then stop - and if sketching-perhaps take digital photos, print the pictures, and transition to coding for the remainder of the iteration, using the UML drawings for inspiration as a starting point, but recognizing that the final design in code will diverge and improve.
- Shorter drawing/sketching sessions may occur throughout the iteration.

Designing Objects

- Two kinds of object models: dynamic and static.
- **Dynamic models**, such as UML interaction diagrams (sequence diagrams or communication diagrams), help design the logic, the behavior of the code or the method bodies.
- They tend to be the more interesting, difficult, important diagrams to create.
- **Static models**, such as UML class diagrams, help design the definition of packages, class names, attributes, and method signatures (but not method bodies).

Static and Dynamic Modeling



Dynamic Object Modeling

- There's a relationship between static and dynamic modeling and the agile modeling practice of create models in parallel:
  - Spend a short period of time on interaction diagrams (dynamics), then switch to a wall of related class diagrams (statics).
- Guidelines
  - Spend significant time doing interaction diagrams (sequence or communication diagrams), not just class diagrams.
  - Ignoring this guideline is a very common worst-practice with UML.
- Note that it's especially during dynamic modeling that we apply responsibility-driven design and the GRASP principles.
- There are other dynamic tools in the UML kit, including **state machine diagrams** and **activity diagrams**.

Static Object Modeling

- The most common static object modeling is with UML class diagrams.
- Note, though, that if the developers are applying the agile modeling practice of create several models in parallel, they will be drawing both interaction and class diagrams concurrently.
- Other support in the UML for static modeling includes **package diagrams** and **deployment diagrams**.

The Importance of Object Design Skill over UML Notation Skill

- It's been said before, but is important to stress:
  - *What's important is knowing how to think and design in objects, and apply object design best-practice patterns, which is a very different and much more valuable skill than knowing UML notation.*
- While drawing a UML object diagram, we need to answer key questions:
  - What are the responsibilities of the object?
  - Who does it collaborate with?
  - What design patterns should be applied?

Object Design Skill vs. UML Notation Skill

- Drawing UML is a reflection of making decisions about the design.
- The object design skills are what matter, not knowing how to draw UML.
- Fundamental object design requires knowledge of:
  - principles of responsibility assignment
  - design patterns

Class Responsibility Collaboration (CRC) Cards

CRC Cards

- A popular text-oriented modeling technique is Class Responsibility Collaboration (CRC) cards, created by the agile, influential minds of Kent Beck and Ward Cunningham (also founders of the ideas of XP and design patterns).
- CRC cards are paper index cards on which one writes the **responsibilities** and **collaborators** of classes.
- Each card represents one class.
- A CRC modeling session involves a group sitting around a table, discussing and writing on the cards as they play "what if" scenarios with the objects, considering what they must do and what other objects they must collaborate with.

CRC Cards: An Example

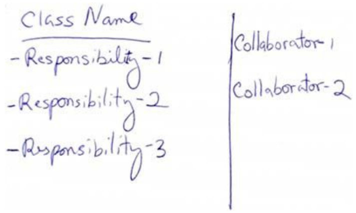


Figure: *Template for a CRC card.*

CRC Cards: Class

- A Class represents a collection of similar objects.
- Objects are things of interest in the system being modeled.
- They can be a person, place, thing, or any other concept important to the system at hand.
- The Class name appears across the top of the CRC card.

CRC Cards: Responsibilities

- Responsibilities of a class can be broken into two separate types: knowing and doing.

CRC Cards: Responsibilities: Knowing

- Knowing responsibilities are those things that an instance of a class must be capable of knowing.
- An instance of a class typically knows the values of its attributes and its relationships.

CRC Cards: Responsibilities: Doing

- Doing responsibilities are those things that an instance of a class must be capable of doing.

- In this case, an instance of a class can execute its operations or it can request a second instance, which it knows about, to execute one of its operations on behalf of the first instance.

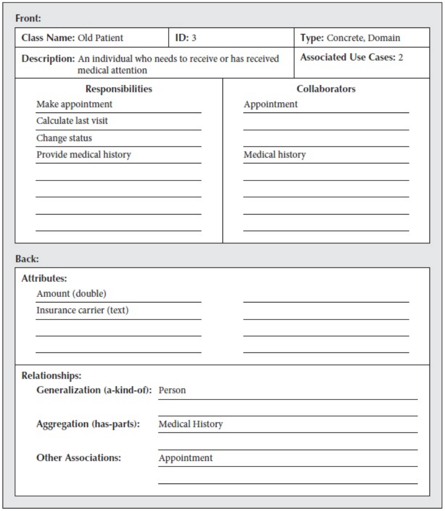
CRC Cards: Collaborator

- A Collaborator is another class that is used to get information for, or perform actions for the class at hand.
- It often works with a particular class to complete a step (or steps) in a scenario.
- The Collaborators of a class appear along the right side of the CRC card.

Developing CRC Cards

- The answers to the following questions are used to add detail to the evolving CRC cards.
  - Who or what are you?
  - What do you know?
  - What can you do?

Elements of a CRC Card



Front:

Class Name: Old Patient	**ID:** 3	**Type:** Concrete, Domain
Description: An individual who needs to receive or has received medical attention		**Associated Use Cases:** 2

Responsibilities	**Collaborators**
Make appointment	Appointment
Calculate last visit	
Change status	
Provide medical history	Medical history

Back:

Attributes:

Amount (double)

Insurance carrier (text)

Relationships:

Generalization (a-kind-of):	Person
Aggregation (has-parts):	Medical History
Other Associations:	Appointment

Figure: *Sample CRC Card*

It's Quiz Time

1. The design work will stress do the thing right. (True or False)

2. Agile modeling does not support "modeling with others". (True or False)

3. Dynamic models help design the definition of packages, class names, attributes, and method signatures. (True or False)