### Object-oriented design methodologies

An object-oriented methodology is defined as the system of principles and procedures applied to object-oriented software development.

Five years ago, there was no standard OOD methodology. Then, the proponents of three of the major methodologies—Grady Booch, James Rumbaugh, and Ivar Jacobson—joined forces to produce the Unified Modeling Language (UML).

### The UML method in overview

The UML method is an iterative approach.

In the past, many software-development methods “insisted” on a rigid series of steps: analysis first, then design.

The UML method allows for the reality that the development of a system is an iterative process.

- The method encourages early coding of pieces of the system to aid in the requirements-discovery process.

  The design should allow for change and discovery as the development process continues.

- At the same time, good project management requires focused deliverables.

  So the UML method presents a series of steps.

  Development of the system is accomplished by applying this series of steps iteratively to pieces of a system.

---

1. This discussion is adapted from *Using the Booch Method: A Rational Approach*, by Iseult White, © 1994, Rational.
• **Requirements analysis**, which provides the basic charter for the system’s functions.

• **Domain analysis**, which provides the key logical structure of the system.

• **System design**, which provides the key physical structure of the system.

<table>
<thead>
<tr>
<th>1. Requirements analysis</th>
<th>System charter</th>
<th>System function statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Domain analysis</td>
<td>Class diagrams</td>
<td>Inheritance diagrams</td>
</tr>
<tr>
<td></td>
<td>Object</td>
<td></td>
</tr>
<tr>
<td></td>
<td>specifications</td>
<td>specifications</td>
</tr>
</tbody>
</table>

| 3. System design         | Package diagrams | Component diagrams         |
|                          | Class            | Architecture description   |
|                          | specifications   | Executable release plans   |

After this comes a fourth step, which is not part of an OOA/OOD methodology.

• **Implementation** refers to the actual coding and iterative process involved in generating the actual application.

A *use case* is (according TO Jacobson)

• “a particular form or exemplar of usage,

• a scenario that begins with some user of the system initiating some transaction or a sequence of interrelated events.

In an airline reservation system, what kind of action would be a use case?

Every use case describes some way of using the system.
Together, all of the use cases specify the complete functionality of the system.

*Requirements analysis*

During **requirements analysis**, the analyst works with—

- end users, and
- domain experts

to enumerate the use cases that are key to the system’s operation.

*Objective:* Requirements analysis essentially forms a contract between the customer and the developers on what they system is to provide.

*Deliverables:*

- *System charter*—outlines the responsibilities of the system.
- *System function statement*—outlines the key use cases of the system.

*Domain analysis*

During **domain analysis**, an o-o model of the real-world system is developed.

This gives the developers enough knowledge to create a system carrying out the required functions.

*Objective:* Domain analysis identifies all major objects in the domain, including data and major operations.

The development of the model concentrates on resolving —

- the potentially bewildering vocabulary used to define the system,
- contradictory requirements,
- obscure policy, and
- varying styles of explanation and communication.
Good domain analysis also adds appropriate levels of abstraction. This allows structures to be built that will be reusable in many other domains.

**Deliverables:**

- *Class diagrams*—identify key classes of the domain.
- *Inheritance diagrams*—show which classes inherit from other classes. (Do not over-use inheritance!)
- *Use-case diagrams*—illustrate how objects will interact to carry out key functions.

**System design**

During **system design**, an effective and efficient implementation is defined.

**Objective:** Select appropriate computers, devices, software services, software units, and data storage strategies.

**Deliverables:**

- *Architectural descriptions*—capture major design decisions, such as choice of processors, language, etc.
- *Executable-release descriptions*—define the goals and content of successive executable release implementations.
- *Package diagrams*—partition the system into high-level groupings of classes and objects.
- *Component diagrams*—show how the classes relate to the actual code. Specify the libraries and subsystems that are integrated to make up the system.
- *Sequence diagrams*, which show the detailed operational logic to carry out functions. Tie together use-case diagrams and class diagrams, by showing how the classes collaborate to carry out a use-case.

To understand the process of object-oriented design and analysis, we will consider an example system.
Example: The PG system

Peer Grader system—analysis

We are going to consider a system for peer grading of homework problems, e.g., of design problems.

The basic function of the system is to allow students to submit homework problems and review other students’ problems.

Students—

- follow a link to the Web page for their assignment;
- submit their homework using a form on the Web;
- are presented with a set of homework problems submitted by other students;
- engage the other students in dialog about their problem over a shared Web page;
- submit a grade for each problem they review.

The instructor—

- Creates each homework assignment and posts a description;
- devises rules for mapping authors to reviewers; and
- obtains a set of grades suitable for input into a grading program.

Understanding the problem specification: Notice that the program is going to have to deal with different courses, and different assignments for each section.

Students will have to submit assignments and reviews over the Web.

- This is relatively easy to implement if the submission is text, typed into a text window in an HTML form.
• But if the submission is a pre-created set of Web pages, this becomes a difficult problem, due to security considerations.

To keep things simple, we will assume that a submission is a single text window, but we will try not to build this assumption in too deeply, so we can change it later.

We want to encourage communication between authors and reviewers, so authors should be permitted to see reviewers’ comments and revise their submissions in response to them.

What does that tell us about submissions, and the relationship between submissions and reviews?

•
•

Another way to understand what the system is supposed to do is to take a look at some of its outputs:

<table>
<thead>
<tr>
<th>PG: Peer Grader System</th>
</tr>
</thead>
<tbody>
<tr>
<td>◁ Review your own page.</td>
</tr>
<tr>
<td>◁ Review #1</td>
</tr>
<tr>
<td>◁ Review #2</td>
</tr>
<tr>
<td>◁ Review #3</td>
</tr>
</tbody>
</table>

| Reset | Submit |

After clicking on “Submit,” a student is presented with a screen that looks something like this:
Student submission:

Student submission appears as text in this box.

Comments:

Reviewer 1, Wed. Dec. 10, 12:02 EST.
I really liked this paper!
Grade: 100

Reviewer 2, Thu. Dec 11, 20:19 EST.
This paper is no good. I wonder what Reviewer 1 could possibly have liked about it.
Grade: 60

Reviewer 1, Thu. Dec 11, 22:04 EST
The author has done a good job of succinctly writing up his results.

New comment:

New grade:  

Reset  Submit
In addition to this screen seen by students, there will also be a screen seen by administrators (instructors). Ongoing PG projects are focusing on—

- **Servlet interface**: In the past, PG ran under CGI. It is changing to use a Java servlet, removing the need to run under CGI, and permitting it to use multithreading to service requests concurrently.

- **User interface**: The interface you see in this lecture is pretty rudimentary. We are changing it to do a better job of presenting information to the author and reviewers.

- **Reviewer mapping**: The mapping of authors to reviewers can be fairly involved, especially if done dynamically.

Associating each review with a version of a submission.

*Domain analysis*

Domain analysis seeks to identify *key abstractions*—the classes and objects that are common to applications within a given domain. Not just for this program. The first step is *discovering key classes*. Start by underlining the nouns.

- Don’t underline nouns that refer to implementation characteristics.

- Be aware of ambiguity, aliasing, and overloading.

- Distinguish between classes and objects.

As candidate abstractions are discovered, they are added to the *data dictionary*.

Some of the abstractions will turn out to be—

- classes,
- relations, and
- attributes of other abstractions.
Choosing class names. A good class name should be—

- M
- A singular noun.
- Characteristic of the abstraction.

The inability to name a class simply and clearly is a symptom of an ill-defined abstraction. It can be a warning sign of—

- incompleteness,
- weak cohesion, or
- strong coupling.

*Cohesion* refers to the strength of association of member data and functions of a class.

*Coupling* refers to the amount of dependency this class will have on other classes.

Identifying key abstractions in the PG system:

- A submission, each of which is found on a Web page, or a set of Web pages.
- A *review* of a submission, which is also on a Web page.
- The *mapping* between authors and reviewers.
Are most of the key abstractions identified above classes?
Are all of them classes?

*Events*: What events are relevant to the system?

- An instructor sets up a new course to use the system.
- An instructor creates a new assignment.
- A student submits a submission.
- A student reviews someone else’s submission.
- A student submits a revision of a submission.
- A student assigns a grade for a submission.

*Use-cases*

A “use-case” is a narrative that describes the sequence of events when an actor (external to the system) uses the system to complete a process.

They are “stories” of how the system is used.

Based upon the events we have just listed, we can develop use-cases.

**Use case:** Set up course

**Actors:** Instructor
Description: An instructor creates a new course. The system fetches the class roll from the registrar’s computer system, and creates a login screen for students in this course to use to gain access to the system.

Let us try one or two more use-cases.

Use case:

Actors:

Description:

Use case:

Actors:

Description:

To gain a deeper understanding of the processes and requirements for a system, an expanded use-case can be developed.

They are often done in a “conversational” style between the actors and the system.

Let us consider the use-case of reviewing a submission.

Use case: **Review a submission**

Actors: Student

Description: Student selects a submission to review, types in a review, and optionally a grade, then submits the review.

Typical course of events:

<table>
<thead>
<tr>
<th>Actor action</th>
<th>System response</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. A student logs into the system.</td>
<td>2. The system presents a screen listing problems to submit or review.</td>
</tr>
</tbody>
</table>
5. Student enters a review of the problem and, optionally, a grade, and submits it.

6. The system updates its database of reviews and displays the new review to the student.

The use-cases in a system can be illustrated in a use-case diagram:

![Use-case Diagram]

Note that the actors are represented by stick figures, and the use cases by ovals.
Later we will see that it is possible for one use case to refer to another, by “using” it or “extending” it.
Individual use-cases are diagramed by *sequence diagrams*. These can be used to show any pattern of interactions between objects.

In a sequence diagram,

- the vertical dimension represents time, and
- the horizontal dimension represents different objects.

Here is a sample sequence diagram.

![Sequence Diagram](image)

Of course, this shows the system at only the highest level of detail. In the course of design we would refine the diagram to include individual objects within the PG system.

When we consider the video-store example in coming lectures, we will see more detailed sequence diagrams.

Let us summarize our analysis by listing the classes that we have found in the system.
The next step is to flesh out the analysis by deciding on the data attributes and operations that each class will have.

What attributes does course need to have?

What attributes does assignment need?

What attributes does student need?

How about submission?

When we proceed to design, we will also decide on methods for each