Software Engineering

Prof. Dr. Bertrand Meyer

Dr. Manuel Oriol

Dr. Bernd Schoeller

From Requirements to Design
Lecture objectives

- Understand which problems may arise when trying to bridge the semantic gap between requirements and the concrete design of the software system in object oriented terms

- Provide some caveats...

- ...and some suggestions

- Show an example
Why O-O analysis?

Same benefits as O-O programming, in particular reliability, extendibility, reusability

Direct modeling of the problem domain

Seamlessness and reversibility with the continuation of the project (design, implementation, maintenance)
What is object-oriented analysis?

- **Classes** around object types (not just physical objects but also important concepts of the application domain)
- **Abstract Data Types** approach
- **Deferred** classes and features
- Inter-component relations: “**client**” and inheritance
- Distinction between **reference** and **expanded** clients
- **Inheritance** — single, multiple and repeated for classification.
- **Contracts** to capture the *semantics* of systems: properties other than structural
- **Libraries** of reusable classes: reuse is for analysis too!
Expanded classes

You may declare a class as

expanded class $C$

... The rest as usual ...
Reminder: expanded & reference types

Reference types; value of an entity is a reference

If \textit{COMPANY} is a reference type:

\[ b : \textit{COMPANY} \]

Expanded types; value of an entity is an object.

If \textit{COMPANY} is an expanded class:

\[ d : \textit{COMPANY} \]
Expanded classes and entities support the notion of subobject

```pascal
class RECTANGLE_R feature
  corner1, corner2, corner3, corner4: POINT
  ...
end

class RECTANGLE_E feature
  corner1, corner2, corner3, corner4: POINT_E
  ...
end

expanded class POINT_E inherit POINT
end
```
More than implementation notion: a system modeling tool.

Two forms of client relation:

- Simple client
- Expanded client

What is the difference between these two statements?

- A car has an engine
- A car has an originating factory
The main problem

- A requirements specification
  - IEEE Std 830-1998, UML use cases,…

⚠️ Semantic gap!

- A description that possibly encompasses all the relevant abstractions and the related semantic properties (i.e. constraints)
Two common questions

- How much should we trust a requirements specification?
- How can we deal with important, semantic properties of the system right from the start?
Suggested answers

How much should we trust a requirements specification?

- **Not too much:**
  - Natural language is imprecise
  - Crucial abstractions may not be directly deducible from the requirements
  - Tends to suggest a specific, non-reusable solution

- How can we deal with important, semantic properties of the system right from the start?

- Using a method that enforces seamless development, e.g. enabling the insertion of class invariants when sketching the first design of the system
A well known grammatical game

- How can we assess the famous “Underline the nouns” grammatical criterion?

- Use this with caution! Nouns can
  - Suggest notions that do not yield classes
  - Fail to suggest notions that should yield classes
Trying to bridge the gap

- How to find the classes then?
  - What to consider?
  - What to reject?
What to reject (advisory negatives)

- The grand mistake: not semantically rich enough ADTs
  - Know the business domain and what is relevant to it
- Mixed abstractions
  - Should be split into several classes, one per abstraction
- Premature classification
  - Beware of the PH factor (Polymorphistic Hysteria)
  - May be a case of “taxomania”
- Known examples of bad/poor class choices for the specific domain
Other danger signals

- Class with verbal name (infinitive or imperative)
  - May be a routine
- Class described as “performing something”
  - May not be a proper data abstraction
- Fully effective, single-exported-routine class
  - May be a routine
- Class with no routines
  - May be some routines are missing, or may be it is not an ADT
What to consider: practical heuristics

- Analysis classes
  - Belong to the problem space
  - Describe a data abstraction directly drawn from the domain model
- Design classes
  - Belong to the solution space
  - Describe an architectural choice
- Implementation classes
  - Belong to the solution space
  - Describe data abstractions introduced for the internal needs of the software algorithms
Problem Domain vs Solution Domain

Requirements Specifications
- Understood by customer
- Natural Language
- Communication with clients and users
- Functional decomposition

Analysis Classes
- Problem Domain
- User’s Point of View
- Understood by developers
- Formal
- OO

Design Documents
- Solution Domain
- Internal Structure
Finding the analysis classes

Any software system is based on an operational model of some aspect of the external world.
Different Kinds of Objects

Entity Objects
- Represent the persistent information tracked by the system
- Application domain objects, “Business Objects

Boundary Objects
- Represent the interaction between the user and the system

Control Objects
- Represent the control tasks performed by the system

Having three kinds of objects makes models more resilient to change
- Interface of system changes more likely than control
- Control of system changes more likely than application domain
Identifying Entity Objects

- Terms the developers or users must clarify to understand the meaning (e.g. account)
- Recurring nouns in the requirements
- Real-world entities that the system must track
- Real-world processes that the system must track
- Data sources or sinks
Identifying Boundary Objects

- Boundary objects collect information from user.
- Boundary objects translate information into format for entity and control objects.
- User interface controls to initiate actions.
- Forms to enter data.
- Messages the system uses to respond.
Identifying Control Objects

• Control objects coordinate boundary and entity objects
• Control objects do not have a concrete counterpart in the real world
• Identify one control object per business task
Finding the design classes

- Understand the importance of reuse: beware of the NIH (Not Invented Here) syndrome
- Reuse previous known designs in the same domain
- Know standard design patterns
- Consider describing abstractions as “machines” rather than “objects”
Finding the implementation classes

- Reuse
- Know and use the classical data structures & algorithms
- Know and use existing libraries of your language
- Consider the use of deferred classes
Other sources of classes

- Previous developments
- Evaluating candidate decompositions
- Adaptation through inheritance relationship
- Adaptation through client relationship
Additional ways of discovering classes

- Discussions with customers and future users
- Terms that are taken for granted in a certain domain
- Discussions with experienced designers
- CRC (Class Responsibility Collaboration) cards
- Use cases (see next)
Use Cases (scenarios)

One of the UML diagram types
A use case describes how to achieve a single business goal or task through the interactions between external actors and the system

A good use case must:
- Describe a business task
- Not be implementation-specific
- Provide appropriate level of detail
- Be short enough to implement by one developer in one release
Use case example

Place an order:
- Browse catalog & select items
- Call sales representative
- Supply shipping information
- Supply payment information
- Receive confirmation number from salesperson

May have precondition, postcondition, invariant
Use cases for requirements?

- Use cases are not a good tool for finding classes, because they:
  - Emphasize ordering
  - Focus on how users see the (non-existing) system’s operation
  - Favor a functional, top-down approach

They are, however, good for validation (e.g. for QA team)
class SCHEDULE feature
  segments : LIST [SEGMENT]
end
Schedules

note
description: “24-hour TV schedules”
deferred class SCHEDULE
feature

segments : LIST [SEGMENT ]
  -- Successive segments
deferred
end

air_time : DATE
  -- 24-hour period
  -- for this schedule
deferred
end

set_air_time (t : DATE)
  -- Assign schedule to
  -- be broadcast at time t.
require
  t.in_future
deferred
ensure
  air_time = t
end
print
  -- Produce paper version.
  deferred
end
end
Contracts

Feature precondition: condition imposed on the rest of the world

Feature postcondition: condition guaranteed to the rest of the world

Class invariant: Consistency constraint maintained throughout on all instances of the class
Why contracts

Specify semantics, but abstractly!

(Remember basic dilemma of requirements:
  ➢ Committing too early to an implementation
    Overspecification!
  ➢ Missing parts of the problem
    Underspecification!
)

)
Segment

**note**

description :
"Individual parts of a schedule"

defered class SEGMENT feature

schedule : SCHEDULE deferred end
  -- Schedule to which segment belongs
index: INTEGER deferred end
  -- Position of segment in its schedule
starting_time, ending_time :
  INTEGER deferred end
  -- Beginning and end of scheduled air time
next: SEGMENT deferred end
  -- Segment to be played next, if any

**sponsor** : COMPANY deferred end
  -- Segment’s principal sponsor
rating : INTEGER deferred end
  -- Segment’s rating (for children’s viewing etc.)

... Commands such as change_next, set_sponsor, set_rating omitted ...

Minimum_duration : INTEGER = 30
  -- Minimum length of segments, in seconds

Maximum_interval : INTEGER = 2
  -- Maximum time between two successive segments, in seconds
Segment (continued)

**invariant**

in_list: \(1 \leq \text{index}\) and \((\text{index} \leq \text{schedule.segments.count})\)

\[\text{in\_schedule: } \text{schedule.segments.item} \left(\text{index}\right) = \text{Current}\]

next_in_list: \((\text{next} \neq \text{Void})\) implies

\[(\text{schedule.segments.item} \left(\text{index} + 1\right) = \text{next})\]

no_next_iff_last: \((\text{next} = \text{Void}) = (\text{index} = \text{schedule.segments.count})\)

non_negative_rating: \(\text{rating} \geq 0\)

positive_times: \((\text{starting\_time} > 0)\text{ and } (\text{ending\_time} > 0)\)

sufficient_duration:

\(\text{ending\_time} - \text{starting\_time} \geq \text{Minimum\_duration}\)

decent_interval:

\((\text{next.starting\_time}) - \text{ending\_time} \leq \text{Maximum\_interval}\)

end
Commercial

**note**
- description: "Advertizing segment"

**deferred class COMMERCIAL inherit SEGMENT**

**rename** sponsor as advertizer

**end**

**feature**
- primary: PROGRAM deferred
  - Program to which this
  - commercial is attached
- primary_index: INTEGER deferred
  - Index of primary

**set_primary (p: PROGRAM)**
- Attach commercial to p.

**require**
- program_exists: p /= Void
- same_schedule: p, schedule = schedule
  - before:
    - p.starting_time <= starting_time

**deferred**

**ensure**
- index_updated:
  - primary_index = p.index

**primary_updated:**
- primary = p

**end**

**invariant**
- meaningful_primary_index: primary_index = primary.index
- primary_before: primary.starting_time <= starting_time
- acceptable_sponsor: advertizer.compatible (primary.sponsor)
- acceptable_rating: rating <= primary.rating

**end**
## Obligations & benefits in a contract

<table>
<thead>
<tr>
<th>deliver</th>
<th>OBLIGATIONS</th>
<th>BENEFITS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Client</strong></td>
<td><em>(Satisfy precondition:)</em> Bring package before 4 p.m.; pay fee.</td>
<td><em>(From postcondition:)</em> Get package delivered by 10 a.m. next day.</td>
</tr>
<tr>
<td><strong>Supplier</strong></td>
<td><em>(Satisfy postcondition:)</em> Deliver package by 10 a.m. next day.</td>
<td><em>(From precondition:)</em> Not required to do anything if package delivered after 4 p.m., or fee not paid.</td>
</tr>
</tbody>
</table>
O-O analysis process

Identify abstractions
  ➢ New
  ➢ Reused

Describe abstractions through interfaces, with contracts

Look for more specific cases: use inheritance
Look for more general cases: use inheritance, simplify

Iterate on suppliers

At all stages keep structure simple and look for applicable contracts
Practical advice

Take advantage of O-O techniques from the requirements stage on

Use contracts to express semantic properties
Practical advice

Write ADT specifications for delicate parts of the system requirements
An example specification

**Identified Stakeholders**

- Company Management
- System Administrator
- Project Team Leader
- Project Team Member
We develop custom software solutions for different customers. Having a project-driven management model, we have organized our developers into project teams. Each team is led by a team leader and typically includes 2 to 20 developers. Increasingly, our development is distributed: team members may be in different locations, often several time zones apart.

Currently every team manager is using a specific software product, or no software at all, for maintaining the project schedule and to organize the tasks of the project. This situation causes numerous problems:
Statements: management (2)

- It is difficult for the company management to get an overview of the project status.
- It is difficult to move project leaders from one project to another, as there is no common standard for the important project information.
- Data is stored on the individual laptops of project leaders, with all the well-known implications on safety and security.
- Management has trouble understanding the effect of changed circumstances such as a delay in one part of the project (what is the effect on the rest of the project and the final delivery date?), the temporary unavailability of a team member.
- Marketing has trouble obtaining realistic costs for change requests by customers or initial offers to new customers, as comparable tasks are not available.

The goal of the project is to develop a global solution that will remedy these problems. MM
The system must support the specific needs of the management of software projects. Applicability to other kinds of projects is not required.

The functionalities must include:

- Defining tasks and subtasks
- Defining dependencies between tasks
- Assigning time estimates to tasks
- Assigning people to tasks (one person may be assigned to multiple tasks, and one task may be assigned to multiple people)
- Assigning availability levels to people (e.g. number of hours per week)
Statements: team leader A (2)

- Changing any previous assignment
- Reporting completion of task
- Estimating the completion time of a task, on the basis of timing estimates for subtasks, dependencies between tasks, project members’ assignments and availability, completion data.
- Providing output in various forms including individual project member schedules, overall project schedules, PERT, Gantt
- “What-if?” scenarios: assessing the results of various hypothetical changes.
- User login with various privileges, including at least “manager” and “project member”
As I am traveling a lot and do many of the project management tasks during these travels. I am currently using a spreadsheet to manage projects. The most important ability of a software project management system for me is the ability to experiment with the scheduling of tasks. That way, I can interactively develop the project schedule together with the other developers and the customer.

Report generation should offer many different views. Specifically, the possibility to plot GANTT diagrams is important to me. Other functionality should include the definition of tasks, the assignments of developers to tasks, describing the dependency of tasks, and to connect risks with tasks.

It would be great if the project management tool could also interface with a bug tracking tool. Currently we seem to converge on Mozilla but it doesn’t really matter what the tool is as long as we can interface with it.

The tool should also interface with a configuration management systems such as CVS or SourceSafe.

**Team Members**

A questionnaire with six questions was issued among all team developers. A number of statements were rated. The scale is between 1 (strong disagree) to 5 (strong agree).

**Results:**

- ☐ “I always want to know the exact time frame of the project.” 2.8
- ☐ “My vacations are important to me and I need to book them half a year in advance.” 4.2
- ☐ “Rescheduling of tasks must be avoided.” 1.7
- ☐ “I am not interested in the exact schedule of the project. That is the job of the team leader.” 4.1
- ☐ “It would be great if I could easily switch tasks within a project.” 3.5
- ☐ “I do not want to be moved away from a project until it is finished.” 2.2
- ☐ “There needs to be better ways to collaborate with other developers.” 3.2.
As I am traveling a lot and do many of the project management tasks during these travels. I am currently using a spreadsheet to manage projects. The most important ability of a software project management system for me is the ability to experiment with the scheduling of tasks. That way, I can interactively develop the project schedule together with the other developers and the customer.

Report generation should offer many different views. Specifically, the possibility to plot GANTT diagrams is important to me. Other functionality should include the definition of tasks, the assignments of developers to tasks, describing the dependency of tasks, and to connect risks with tasks.

It would be great if the project management tool could also interface with a bug tracking tool. Currently we seem to converge on Mozilla but it doesn’t really matter what the tool is as long as we can interface with it.

The tool should also interface with a configuration management systems such as CVS or SourceSafe.
We are supporting a very heterogeneous landscape of operating systems. The application should run on several operating systems, including at least Windows (XP and perhaps Vista), Linux and MacOS X. It is very important that there is clean procedure on how to do updates and fix security bugs.

I believe that the application should be implemented as a web-based application that runs on our web server. That way, it is located within our DMZ, accessible from the outside as well as from within the company. This is important, as the project teams often work at customers and not within the company building. The web server is running a standard LAMP (Linux, Apache, MySQL, Perl) environment.