Lecture on Object-Oriented Modeling

Thierry Géraud

EPITA Research and Development Laboratory (LRDE)

2006

Outline

Outline

Outline

Objectives

Keywords:

- concepts of the object-oriented (OO) paradigm
- OO modeling
- overview of OO languages
- introduction to the UML
- introduction to software engineering
- design patterns

How to work

The how-to:

- concentrate and listen during class
- read the slides every week
- read them again!
- think about and argue the pros and cons
- then start looking at recommended readings

Outline

There's soft in it!

Software means:

- neither rigid

  what are the artifacts?

- nor too loose / lax / flaccid

  what are the artifacts?

- so something that should evolve under your control

see also:
`http://en.wikipedia.org/wiki/Software`

Good Software

Getting good software is

- hard
  v. getting unimpressive / second-rate / poor software is *very* easy
- complex
  think about how many klines are involved in some mainstream software...
- definitely **not** spontaneous

see also:
`http://en.wikipedia.org/wiki/Software_engineering`

Exercise

Imagine that you want to design your own house.
What do you do first? Why?

Difficulties

Difficulties in software construction are:

- technical ones
  e.g., with the language, the compiler, the debugger, the libraries
  - exercise: name precisely the related difficulties...
  - question: are they really the main difficulties?
- design issues
  - what are the difficulties here?
- project management
  - what does that means to you?

Think (1/2)

1. draw the curve “number of functionalities w.r.t. time” corresponding to your development process
2. refine the shape of this curve to take into account “special events” in software life
3. justify yourself...
4. what’s wrong? what’s the ideal curve? why?

Think (2/2)

1. have you ever think about the way you work?
2. what have you done to work in a better way?
3. what can you do
   - to manage complexity?
   - to enhance software quality?

Outline

Textbook case

- Consider this simple piece of C software:

```
#define FALSE 0
#define TRUE 1

int negate(int b)
{
  return b == TRUE ? FALSE : TRUE;
}
```

- is it valid? (justify yourself)
- is it trusty? (criticize yourself)

Quality criterions (1/3)

- validity = follows specifications
- robustness = runs well even when outside of specification
- reliability = validity + robustness

- how can you ensure validity? (justify yourself)
- how can you enforce robustness? (criticize yourself)

Quality criterions (2/3)

Exercise: try to define

- usability...
- efficiency...
- integrity...
- portability...

warning: those concepts are general so do not restrict them!

Quality criterions (3/3)

Some very important ones:

- extensibility
- compatibility
- reusability

“Adaptability” is not listed here; think about why.

Object-Orientation and Quality

On one side:

people claim that object-orientation is a good solution to ease reusability

on the other side:

people claim that, from a practical point of view, object-orientation fails to ease reusability

Outline

Module: Definition

A module is a piece of software which is

- coherent
- reasonably sized
- well decoupled from the other modules

exercise: describe your last project in terms of modules...

Module: Definition

- granularity of modules goes from fine to coarse
- modules are organized in a hierarchical way
- a module
  - exposes its interface
  - encloses its contents
- modules interact each other through their interface

Achieving Quality thru Modularity

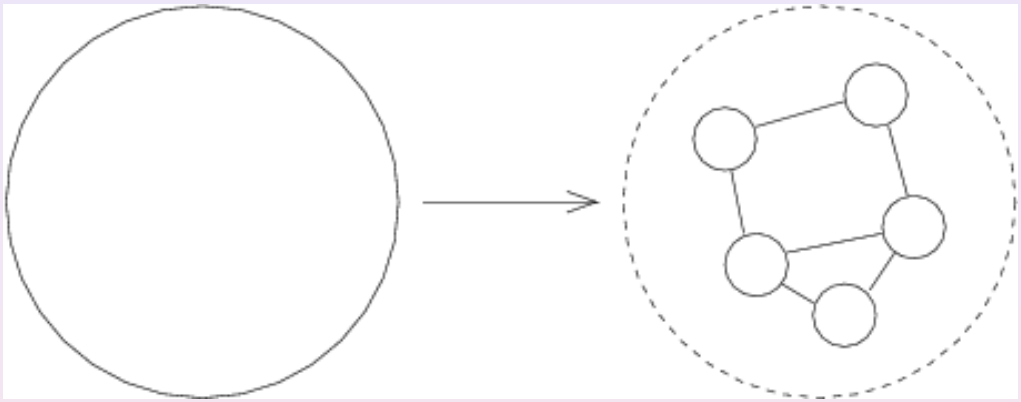
the basic idea:

- good modularity $\Rightarrow$ good software
- good modularity can be evaluated with some criteria
- you should think in those terms to achieve modularity

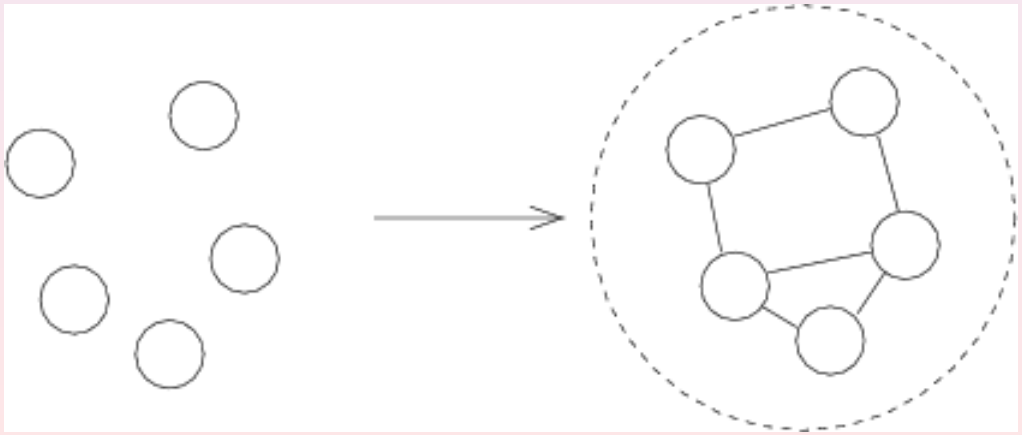
see: “Object-Oriented Software Construction,” 2nd ed., Bertrand Meyer, Prentice Hall Professional Technical Reference, 1987.

Composability / Decomposability

top-down approach:

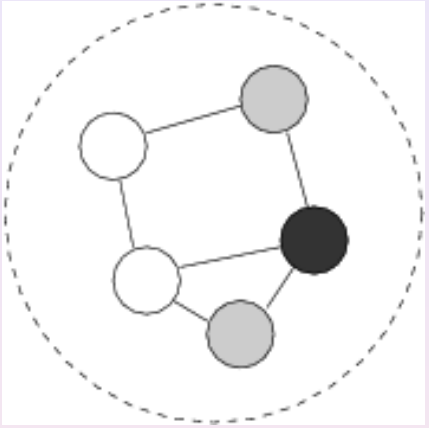


bottom-up approach:

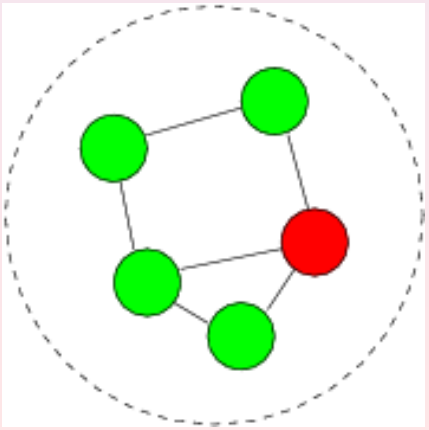


Continuity / Protection

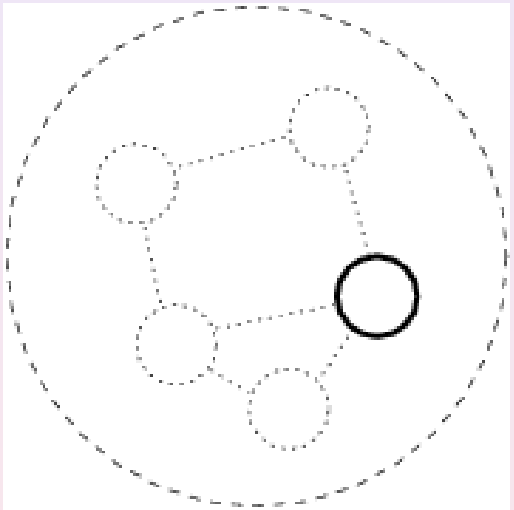
continuity is related to loose coupling and extensibility:



protection is related to integrity (and aspect):



Comprehensibility



Open-close principle

you should keep in mind that

- a module should be *open* enough
  but not too much opened
- a module should be *close* enough
  but not too much closed

Exercise: practice on an example of module and clearly draw a line to define the limits of the module...

Responsibility v. Collaboration

you should keep in mind that:

- a module has some clearly defined *responsibilities*
- a module expects some *collaborations* from few other modules

Exercise: practice!

Flavors

Many flavors of object-orientation exist:

- frame-based languages
- actor-based languages
- prototype-based languages (JavaScript)
- class-based languages (C++)

also read:
`http://en.wikipedia.org/wiki/Programming_paradigm`
`http://en.wikipedia.org/wiki/Object_orientation`

Two Main Families

	Static	Dynamic
from C	C++	Objective C
orientation	compile-time	run-time
principle	method calling	message passing

Java and C# are static languages...

Concepts

an object-based language:

- class / object
- encapsulation / information hiding
- abstraction / polymorphism

plus:

- inheritance

gives an object-oriented language.

Definitions of Object (1/2)

1. structural definition:

   object = state + behavior

2. definition as actor:

   object = autonomous active entity

3. other definition:

   object = prototype of a concept

Definitions of Object (2/2)

object	=	state	+	behavior
description	↓		↑	*instantiation*
class	=	attributes	+	methods

please consider:

objects as entities, just for what **they are**, like a whole

please forget the old scheme “program = data + algorithms”!