A GLIMPSE OF EIFFEL

Bertrand Meyer

WHAT IS EIFFEL?

Method

Language

- Analysis
- Design
- Implementation
- Maintenance and evolution

Environment (ISE Eiffel)

Libraries

IMPLEMENTATIONS

ISE (Interactive Software Engineering, Santa Barbara)

2 other commercial suppliers

OpenSource: SmallEiffel

SANTA BARBARA

MELBOURNE

PLATFORMS (ISE EIFFEL)

Windows 95/98/NT/2000/XP

Microsoft .NET

Unix: Solaris, HP, ...

Linux

BSD

VMS

Completely source-compatible, including graphics

WHAT IS EIFFEL GOOD AT?

What can one language do that no one else can?

WHO USES IT? (ISE EIFFEL)

Chicago Board of Trade: Price Reporting System

National Health Board, Sweden: Accident reporting

Boeing/Xontech, USA: Simulation of Ballistic Missile Defense

Lockheed-Martin / EPA: Climate and environment models

EMC: Disk design, multi-channel video

Axa Rosenberg: Equity management

Crédit Agricole Lazard (US/France/UK): Futures trading

AMP investments (Sydney, Australia): investment coverage analysis.

MIT Lincoln Lab

Many universities

EIFFEL SCOPE

Method

Language

Analysis
Design
Implementation
Maintenance and evolution

Environment (ISE Eiffel)

Libraries

GOALS

Software quality:

- **Correctness**
- **Robustness**
- **Extendibility**
- **Reusability**
- **Efficiency**

BOOKS

- ***Object-Oriented Software Construction*, B. Meyer, Prentice Hall, second edition, 1997.**
- ***Eiffel: The Language*, Prentice Hall, 1991. Complete language description.**
- **Many other books**

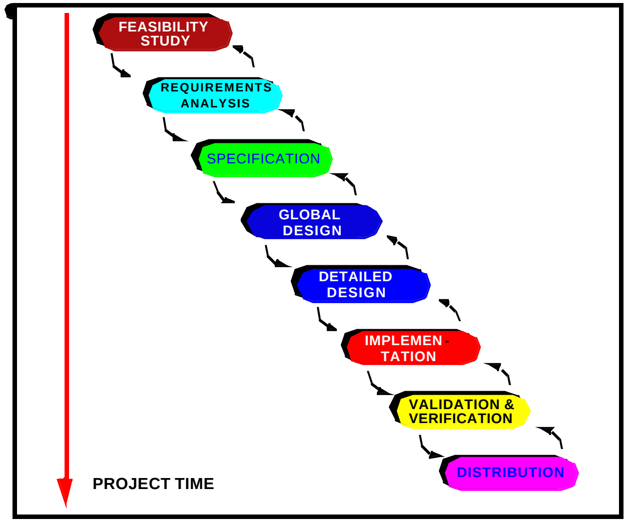
COMMENTS

"CALFP had considered Smalltalk and C++, but those programming languages were not as robust or user-friendly as Eiffel. We wanted people who were not professional programmers to develop code for the system, and Eiffel seemed to be the only [environment] powerful enough to do that."

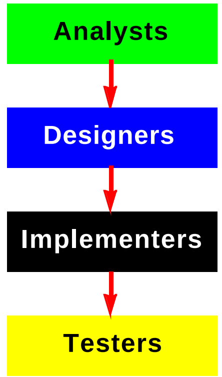
Richard Bielak, CALFP,

in *ComputerWorld,* August 1995

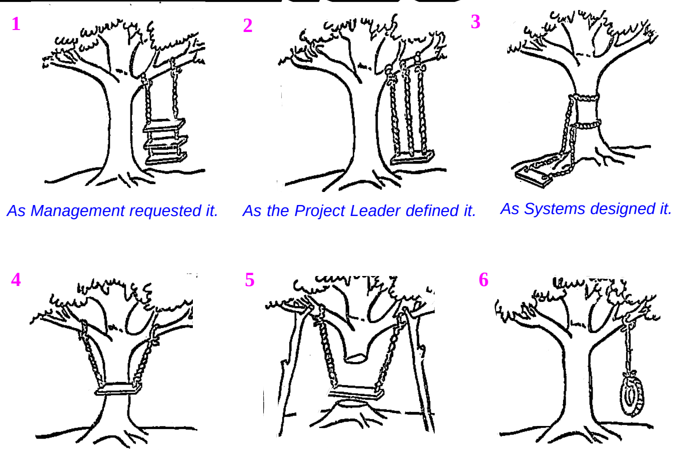
THE WATERFALL MODEL OF THE LIFECYCLE



TASKS



IMPEDANCE MISMATCHES



1 As Management requested it.

2 As the Project Leader defined it.

3 As Systems designed it.

4 As Programming developed it.

5 As Operations installed it.

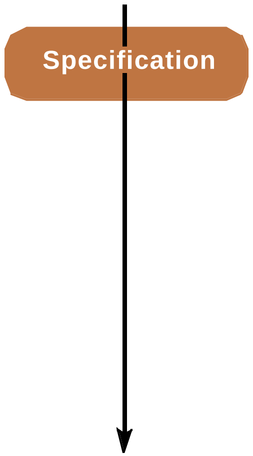
What the user wanted.

(Pre-1970 cartoon; origin unknown)

SPIRAL

M.C Escher:
Waterval

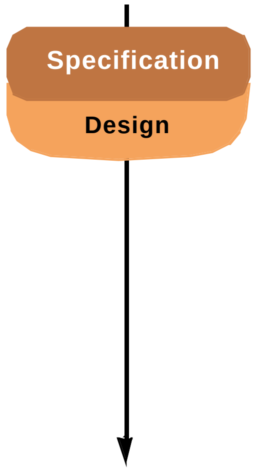
SEAMLESS DEVELOPMENT



- TRANSACTION, PLANE, CUSTOMER, ENGINE...

Example classes

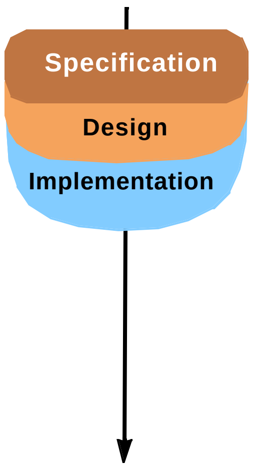
SEAMLESS DEVELOPMENT



- TRANSACTION, PLANE, CUSTOMER, ENGINE...
- STATE, USER_COMMAND..

Example classes

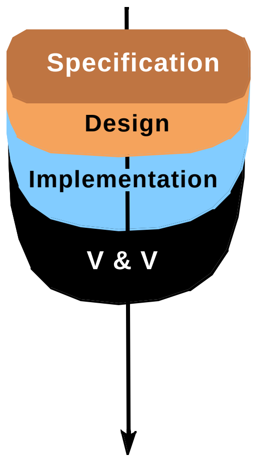
SEAMLESS DEVELOPMENT



- TRANSACTION, PLANE, CUSTOMER, ENGINE...
- STATE, USER_COMMAND..
- HASH_TABLE, LINKED_LIST...

Example classes

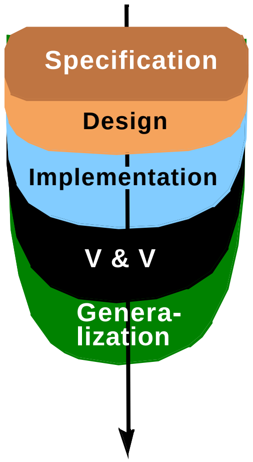
SEAMLESS DEVELOPMENT



- TRANSACTION, PLANE, CUSTOMER, ENGINE...
- STATE, USER_COMMAND..
- HASH_TABLE, LINKED_LIST...
- TEST_DRIVER, ...

Example classes

SEAMLESS DEVELOPMENT



- TRANSACTION, PLANE, CUSTOMER, ENGINE...
- STATE, USER_COMMAND..
- HASH_TABLE, LINKED_LIST...
- TEST_DRIVER, ...

Example classes

EIFFEL FOR ANALYSIS

```
deferred class VAT inherit
        TANK
feature
        in_valve, out_valve: VALVE

        fill is
            -- Fill the vat.
        require
            in_valve.open; out_valve.closed
        deferred
        ensure
            in_valve.closed; out_valve.closed; is_full
        end
        empty, is_full, is_empty,
        gauge, maximum, ... [Other features] ...
invariant
    is_full = (gauge >= 0.97 * maximum)and (gauge <= 1.03 * maximum)
end
```

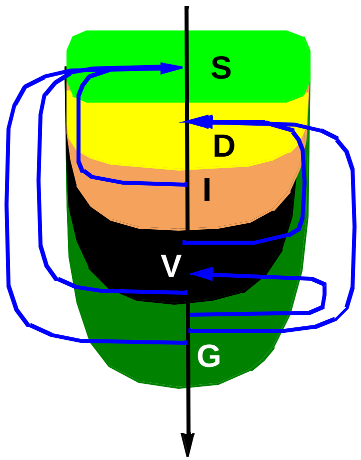
Precondition

-- i.e. specified only,
-- not implemented.

Postcondition

Class invariant

REVERSIBILITY



SEAMLESS DEVELOPMENT

Use consistent notation from analysis to design, implementation and maintenance.

Advantages:

- Smooth process. Avoids gaps (improves productivity, reliability).
- Direct mapping from problem to solution, i.e. from software system to external model.
- Better responsiveness to customer requests.
- Consistency, ease of communication.
- Better interaction between users, managers and developers.

SINGLE MODEL PRINCIPLE

Use a single base for everything: analysis, design, implementation, documentation...

Use tools (those of EiffelStudio) to extract the appropriate views.

EIFFEL TECHNIQUES

- Object-Oriented to the core
- Clean inheritance model
- Design by Contract
- Static typing
- Generic classes
- Agents (ML and Haskell, here we come!)
- Garbage collection
- Rich development environment
- Lots of libraries

WHAT'S NOT THERE

- Global variables.
- Pointer arithmetic.
- Goto, Exit, return etc.
- For loops, repeat loops etc.
- Main program.
- Routines as arguments.
- Union types (or records with variants etc.).
- Function overloading within a class
- Operators with side effects.

[C/C++]	v = x++	[5 keystrokes]
[Eiffel]	y := x; x := x+1	[11 keystrokes]

THE COMPONENT COMBINATOR

Syntax for calling out C functions.

Syntax for calling out C++ methods, data members, constructors, destructors etc.

Legacy++: C++ class encapsulator.

Cecil (C-Eiffel Call-In Library): calling Eiffel from the outside — creating objects, calling features etc.

EiffelCOM: OLE/COM library.

DAIS-Eiffel: CORBA interface.

WEL (Windows Eiffel Library).

Java run-time encapsulation.

Complete language interoperabiity on .NET.

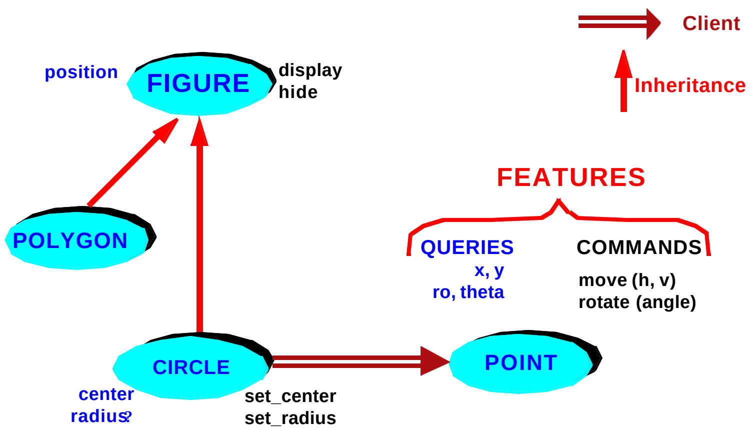
OBJECT-ORIENTED DESIGN

Object technology is about ...?

OBJECT-ORIENTED DESIGN

Object technology is about **abstraction**.

O-O STRUCTURE



THE KEY CONCEPT: CLASS

A class is the description (in the software text) of a set of potential runtime objects, accessible to the rest of the software exclusively through a set of specified operations, or “features”.

ABSTRACTION

From the outside, a POINT object is accessible ONLY through its features:

What is your abscissa?	(x)
What is your ordinate?	(y)
What is your distance to the center?	(ro)
What is your angle to the horizontal?	(theta)
Move yourself by a certain displacement!	(move)
Rotate around the origin by a certain angle!	(rotate)

NO DIRECT FIELD MANIPULATION!

~~your_point.x := 23~~

THE POINT CLASS

```
class POINT feature

    x, y: REAL

    move (a, b: REAL) is
            -- Move by a horizontally, b vertically.
        do
            x := x + a;   y := y + b
        end

    scale (factor: REAL) is
            -- Change the distance to the origin by factor.
        do
            x := factor * x;   y := factor * y
        end
```


CLASS POINT (CONTINUED)

```
  distance (p: POINT): REAL is
          -- Distance to p
      do
          Result := sqrt ((x – p.x) ^ 2 + (y – p.y) ^ 2)
      end

  ro: REAL is
          -- Distance to origin (0, 0)
      do
          Result := sqrt (x ^ 2 + y ^ 2)
      end

  theta: REAL is
          -- Angle to horizontal axis
      do
          ...
      end

end -- class POINT
```

AN ANALYSIS CLASS

```
deferred class VAT inherit
        TANK
feature
        in_valve, out_valve: VALVE

        fill is
            -- Fill the vat.
        require
            in_valve.open; out_valve.closed
        deferred
        ensure
            in_valve.closed; out_valve.closed; is_full
        end
        empty, is_full, is_empty,
        gauge, maximum, ... [Other features] ...
invariant
    is_full = (gauge >= 0.97 * maximum)and  (gauge <= 1.03 * maximum)
end
```

Precondition

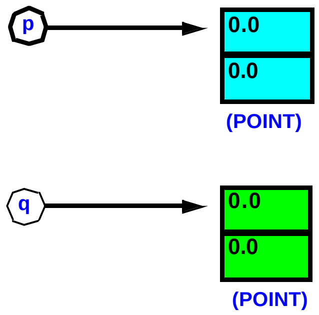
-- i.e. specified only,
-- not implemented.

Postcondition

Class invariant

USE OF THE CLASS (IN A CLIENT)

```
class GRAPHICS feature
  p, q: POINT
  ...
  some_routine is
  local
     u, v: REAL
  do
        -- Creation instructions:
     create p;  create q
     p.move (4.0, –2.0)
        -- Compare with Pascal, C, Ada:
        -- move (p, 4.0, –2.0)
     p.scale (0.5)
     u := p.distance (q)
     v := p.x
     p := q
     p.scale (–3.0)
  end
end -- class GRAPHICS
```



AVOID “OBJECTSPEAK”

The run-time structures, some of them corresponding to “objects” of the modeled system, are objects.

The software modules, each built around a *type* of objects, are classes.

A system does not contain any “objects” (although its execution will create objects).

MORE TO COME...