Teaching Introductory Programming, the outside-in approach

Principles, application and a first report

Bertrand Meyer
Ways to teach intro prog

- Structured programming / Pascal etc.
- Teach a programming language: Java, C++
- “Structure & interpretation of comp progr”, Scheme
- Completely formal, don’t touch a computer
Background and scope

ETH Zurich: birthplace of
- Pascal
- Modula-2
- Oberon

My background: mostly industry
(most recently Eiffel Software in California)

At ETH since end 2001
Teaching introductory programming using object technology and Eiffel
Principles

- Fully object-oriented from the start
- Uses Eiffel as the language and method
- Component-based
- Outside-in
- “Inverted Curriculum”
- Requires supporting component library (TRAFFIC)

- Textbook under development: Touch of Class
Concrete context

About 250 students (now 160)  
(Competition from Electrical Engineering)

About 10 assistants

First programming course

2 x 2 lecture hours, 3 exercise sessions

Grading at end of year, through exam

Can use exercise and project to provide “Testat” required for exam, but cannot base grade on them

Support from Rectorat (new teaching techniques initiative, 1 position / two years); also used own establishment funds
Teaching programming today

- Ups and downs of high-tech economy
- Widely diverse student motivations, skills
- Short-term pressures (e.g. families), IT industry fads
- In Europe: “Confetti Bolognese”

- A revolution: software offshoring!
The objectives

Educate students so that they will:

- Understand today’s software engineering.
- Become competent professionals.
- Find work and have a successful career.
Training or education?

Seek balance between:

- Current needs as stated by industry
- Future needs of industry, not always stated
- Principles of the discipline
- Principles of a university education
- The students’ own interest
Student backgrounds, skills

The “Nintendo generation”

- Grew up with video games, advanced animation
- Some have extensive programming experience
- Others have barely touched a computer!
Giving exercises

“Google and Paste” programming!
The Software Engineering problem

Developing software systems that are

- On time and within budget
- Of high immediate quality
- Possibly large and complex
- Extendible
Fundamental paradigms

- Structure
- Reuse
- Change
- Abstraction, information hiding
- Specification vs implementation
- Complexity
- Scaling up

Typing
- Classification
- Notation
- Static vs dynamic
- Recursive reasoning
- Algorithmic reasoning
- Invariants

Scientific and pedagogical value has not been widely recognized.
The key skill that we should convey: abstraction

Teach, don’t preach.

Basic ideas:

- Start from libraries
- Progressive opening of the black boxes
- From programmer to producer
- Not bottom-up or top-down; outside-in

Students are able, right from the start, to “program” impressive and significant applications.
Using components

We give students a large amount of software, right from the beginning

TRAFFIC library, Flat Hunt

- They start out as consumers
- They end up as producers!

TRAFFIC is graphical, multimedia, extendible, and fun!
Application domain criteria

Immediately understandable, doesn’t have to be taught

Rich reservoir of algorithms, data structures, examples, exercises, extensions...

Allows graphics, multimedia

Interesting

Non-violent
Transportation in a city

Visual
Graphical
Source of data structures and algorithm examples
class TOUR inherit TRANSPORT

feature explore is

-- Prepare
-- and animate
-- route

do

Paris.display
Louvre.spotlight
Metro.highlight
Route1.animate

end
Principles of our course

- Reuse software: inspiration, imitation, abstraction
- See lots of software
- Learn to reuse through interfaces
- Design by Contract essential
- Gentle introduction to formal methods
- Interesting examples from day one
- Combination of principles and practices

We of course do not neglect traditional techniques
Object technology

Classes and objects right from the start
Modeling power

“To Program is to Understand”
Kristen Nygaard

Inheritance, both for reuse and for subtyping
Formal reasoning

Cannot be fully formal at start

But: use class and loop invariants, preconditions, postconditions, focus on logic

Design by Contract

Not Scheme ("Structure & Interpretation of Computer Programs", Abelson-Sussman) or functional programming
Computing the max of an array

highest (sl: LIST[STRING]): STRING is
    -- Greatest element of sl
    require
        sl /= Void
        not sl.is_empty
    do
        from
            sl.start ; Result := ""
        until
            sl.after
        loop
            Result := greater (Result, sl.item)
        end
    end
end
Loop as approximation strategy

\[
\text{Result} = s_1
\]

\[
\text{Result} = \text{Max} (s_1, s_2)
\]

\[
\text{Result} = \text{Max} (s_1, s_2, ..., s_i)
\]

\[
\text{Result} = \text{Max} (s_1, s_2, ..., s_i, ..., s_n)
\]
Importance of programming language

Shapes method and thought

Both syntax and semantics important

First language of instruction is critical
"[Dijkstra said in 1965] he had the painful vision that in the future programming will be equated with learning PL/I, and computer science with mastering OS/360 JCL. Replace PL/I by C++ or Java, and JCL by Windows or Linux, and you are miraculously transposed into the present time.

[A colleague’s] son could not understand why $x = y$ should differ from $y = x$..."
Why Eiffel?

- Method, language, environment
- Object-oriented mechanisms
- Quality focused
- Design by Contract
- Emphasizes reuse throughout
- Portability: Windows, Linux, Unix, soon Mac...
- Easy to learn
- Used in small & large industrial systems
Expressiveness

Usenet posting by David Clark, U. Canberra, taught both Eiffel & Java:

My experience has been that students do not find Java easy to learn. Time and again the language gets in the way of what I want to teach....The first thing they see is

```java
public static void main (String [] args) throws IOException;
```

There are about six different concepts in that one line which students are not yet ready to learn...."
The first “program”

class TOUR inherit TRANSPORT

feature explore is
  -- Prepare
  -- and animate
  -- route
  do
    Paris.display
    Louvre.spotlight
    Metro.highlight
    Route1.animate
  end
Tools

Eiffel as method and language
EiffelStudio:
- Academic license on ETH machine
- Free version on students’ computers

Windows, .NET, Linux, Macintosh - total portability, including graphics (EiffelVision library)
The traditional model

Separate tools:
Programming environment
Analysis & design tools, e.g. UML

Consequences:
Hard to keep model, implementation, documentation consistent
Constantly reconciling views
Inflexible, hard to maintain systems
Hard to accommodate bouts of late wisdom
Wastes efforts
Damages quality
The Eiffel model

**Seamless development:**
Single notation, tools, concepts, principles throughout
Eiffel is as much for analysis & design as implementation & maintenance
Continuous, incremental development
Keep model, implementation and documentation consistent
Reversibility: go back and forth
Saves money: invest in single set of tools
Boosts quality

Example classes:
- PLANE, ACCOUNT, TRANSACTION...
- STATE, COMMAND...
- HASH_TABLE...
- TEST_DRIVER...
- TABLE...
EiffelStudio: text-graphics equivalence

```eiffel
indexing

    description: "Ein exotisches Tier, das andauernd spricht"

class

    PROFESSOR

feature -- Access

    inaugural: LECTURE

    lectures_held: LIST [LECTURE]
```
Teaching other languages

Students need to know C, C++, Java/C#, Perl, Prolog, Javascript...

Service courses are appropriate
Educate a rounded personality

Technical writing and documentation

Single product principle:
  The product is software
Everything else, e.g. documentation, is a view
Results

All student evaluations are on line on the course page!

Students have enjoyed the course; high marks on course evaluation forms

Unhappy with first TRAFFIC version, but it’s being fixed

Course has appealed to students with and without programming experience

I have had to include a special lecture on “doing it all from scratch”
Problems and challenges

Maintain momentum!

Improve and develop the TRAFFIC software

Finish *Touch of Class* textbook

Learn German!
Other courses

Informatik 2 (Data structures and algorithms)

Informatik 4 (Software architecture)
6 weeks

About 60 different video games, using Eiffel and the EiffelMedia animation library developed at ETH (Till Bay)

http://games.ethz.ch
What to expect from a university

Teach what is important. Emphasis on quality.
Teaching what is “right“:

- Sound scientific principles
- Free of fashion, commercial pressures
- No need to cover everything

Concepts first; long-term skills.
Satisfy downstream requirements through utility courses if necessary.
Example: introduction of Pascal ca. 1972.
To follow the project

http://se.inf.ethz.ch/touch