Teaching Introductory Programming with the Inverted Curriculum Approach

Diploma Project

05-02-2003 – 09-01-2003

Michela Pedroni

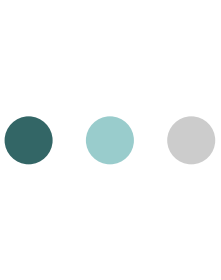
Scope of the diploma thesis

- First part: Survey on teaching introductory programming
  - The difficulty of teaching programming
  - Learning theory and Bloom's taxonomy
  - Selected teaching strategies
  - Special emphasis on the Inverted Curriculum, OO and Eiffel
- Second part: Assisting in the design of the introductory programming course at ETH
  - Designing the exercises and solutions
  - Reviewing the text book

Two quotations

- “Learning to program is a key objective in most introductory computing courses, yet many computing educators have voiced concern over whether their students are learning the necessary programming skills in those courses [1].”
- “[…] recommending a strategy for the introductory year of a computer science curriculum all too often takes on the character of a religious war [2].”

Learning to program is difficult.
Teaching it is difficult, as well.

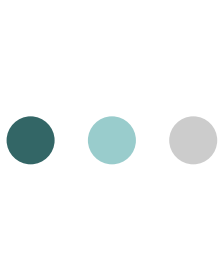


Overview

- The Difficulty of Teaching Introductory Programming
- What can we do about it?
- The Approach taken at ETH

The difficulty

- Teaching introductory programming is a challenge.
- What are the reasons? Three questions:
  - What are we teaching?
  - How are we teaching it?
  - Who are we teaching it to?



The ‘what’

- Computer science is a very young and very rapidly evolving academic discipline.
- Technical advances in the field affect the content of introductory programming courses (e.g. multi-media, graphics, object-orientation).
- There is no consensus on whether new technologies should be included in the courses.

⇨There is no consensus on what to teach in introductory programming courses.

The ‘how’

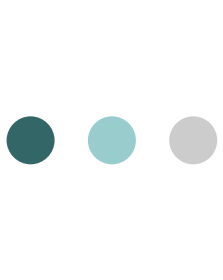
- “New” technologies (like OO) open up the way for new teaching strategies.
- “Many strategies have been proposed over the years, most of which have strong proponents and equally strong detractors [2].”
- There is no consensus on which approach is the best.

⇨There is no consensus on how to teach introductory programming.

The ‘who’

- Computers have become a mainstream equipment → increased familiarity with computing.
  - Some students will already have had years of programming experience.
  - Some students that are not familiar at all with programming will be attracted by a programming course.

⇨Students attending an introductory programming course start with very diverse backgrounds and prerequisites.

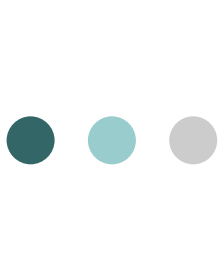


Summary

- ⇨There is no consensus on what to teach in introductory programming courses.
- ⇨There is no consensus on how to teach introductory programming.
- ⇨Students attending an introductory programming course start with very diverse backgrounds and prerequisites.

What can we do about it?

- Carefully look at the approach that is chosen:
  - Analyze the “how”:
    - What teaching strategy?
  - Analyze the “what”:
    - What programming paradigm?
    - What programming language?
  - Analyze how it all fits together:
    - Does the teaching strategy support what we want to be learned?



The approach taken at ETH

- The teaching strategy: The Inverted Curriculum
- The programming paradigm: Object-Orientation
- The programming language: Eiffel

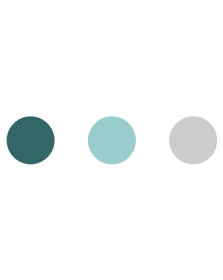
Analyzing the approach taken at ETH

- Carefully look at the approach that is chosen:
  - Analyze the “how”:
    - What teaching strategy?

      ⇨ **The Inverted Curriculum**
  - Analyze the “what”:
    - What programming paradigm?

      ⇨ Object-orientation
    - What programming language?

      ⇨ Eiffel
  - Analyze how it all fits together:
    - Does the teaching strategy support what we want to be learned?



The Inverted Curriculum

- Also called “progressive opening of black boxes”.
- Library: core around which the whole course is built.
- Object-orientation: the main pillar of the Inverted Curriculum.
- Students gradually grow from reuse customers to producers of reusable components.
- First, using interfaces, then gradually opening the black boxes by accessing the internals.
- Topics are covered outside-in: Classes, objects before variables and control structures.

Bottom-up teaching strategies

- Traditionally topics are introduced in a bottom-up way:
  - Start with building blocks of programming: variables, assignment, control and data structures.
  - Usually focus on “programming-in-the-small”
  - small, introductory examples
  - no code bases

The Inverted Curriculum in comparison (1)

- Reading of code:
  - **Bottom-up**: No reading of code required.
  - **The Inverted Curriculum**: Uses an OO library: hides unnecessary details from students.
- Well-designed examples:
  - **Bottom-up**: Well-designed examples are only found outside of exercises.
  - **Inverted Curriculum**: Exercises are based on well-designed code. Encourages imitation within exercises.

The Inverted Curriculum in comparison (2)

- Attention to software engineering principles:
  - **Bottom-up**: First emphasizes attention to detail. Design principles are covered later. Students might perceive attention to detail as the first step and most important part to problem-solving.
  - **Inverted Curriculum**: Emphasizes design principles. Special focus is on code reuse, information hiding, well-documented code, well-designed program structure, as the Inverted Curriculum heavily relies and is based on these. Attention to detail is stressed later.

The Inverted Curriculum in comparison (2)

- Interesting examples:
  - **Bottom-up**: Only small, introductory examples and exercises can be used.
  - **Inverted Curriculum**: Flashy, exciting applications can be used right from the start.

⇨ The Inverted Curriculum seems to have major advantages over the traditional bottom-up strategies.

Analyzing the approach taken at ETH

- Carefully look at the approach that is chosen:
  - Analyze the “how”:
    - What teaching strategy?
    - ⇨ The Inverted Curriculum
  - Analyze the “what”:
    - What programming paradigm?
    - **⇨ Object-orientation**
    - What programming language?
    - ⇨ Eiffel
  - Analyze how it all fits together:
    - Does the teaching strategy support what we want to be learned?

Object-orientation vs. Procedural programming (1)

- The paradigm shift:
  - **Procedural programming**: The shift from procedural to object-orientation seems to be a major difficulty. Might even be harmful to learn OO after it.
  - **Object-orientation**: The paradigm shift from object-orientation to other paradigms is not seen as posing as many problems.
- Real-world modeling ability:
  - **Procedural programming**: The structure of a program is not immediately implied by the counterparts is real-world.
  - **Object-orientation**: Objects impose an organization on a problem that reflects the aspects of the real world being modeled. It also serves as a first step towards breaking down a problem into more manageable subproblems.

Object-orientation vs. Procedural programming (2)

- Support for software engineering principles:
  - **Procedural programming**: Supports SE concepts, but they are not as clearly reinforced as in OO.
  - **Object-orientation**: Supports elegantly most of the software engineering principles like code reuse, encapsulation, incremental development, testing, and program design.

⇨ Object-orientation seems to be adequate as a first programming paradigm.

Analyzing the approach taken at ETH

- Carefully look at the approach that is chosen:
  - Analyze the “how”:
    - What teaching strategy?
    - ⇨ The Inverted Curriculum
  - Analyze the “what”:
    - What programming paradigm?
    - ⇨ Object-orientation
    - What programming language?
    - ⇨ **Eiffel**
  - Analyze how it all fits together:
    - Does the teaching strategy support what we want to be learned?

Eiffel for teaching (1)

- Programming languages that are used for teaching face other requirements than programming languages needed for other purposes.
- Kölling lists in [3] the requirements on teaching languages:
  - **Easy transition**: Learned concepts must be easily transferable to other languages.
    - Eiffel: Skills should be easily transferable to many other languages.

Eiffel for teaching (2)

- **Clean concepts**: The language should represent the concepts that will be taught in a clean, consistent and easy-to-understand way.
  - Eiffel: Most concepts are represented in a clean way. Only point of criticism: Two storage models for objects (expanded and reference mode).
- **Safety**: Errors that can be detected during compile time should be detected. Error messages should be clear.
  - Eiffel: Statically typed.
- **High Level**: Programmers should not have to deal with machine internals like memory management.
  - Eiffel: Automatic garbage collection.

Eiffel for teaching (3)

- **Simple execution model**: “The model of execution should be simple and easy to understand [4].”
  - Eiffel: Two storage models (expanded and reference mode).
- **Readable and consistent syntax**: The syntax should be easily readable and consistent.
  - Eiffel: Uses readable and meaningful keywords.

Eiffel for teaching (4)

- **Small and orthogonal set of features**: “The language should be as small as possible while including all important features that we want to discuss in the first year programming course [4].”
  - Eiffel: Avoids multiple constructs for the same concept. But: supports a large amount of concepts.

Eiffel for teaching (5)

- **Suitable programming environment**: “A suitable programming environment is crucial for the success of an introductory course. Of all the problems reported by educators connected to teaching object-orientation, problems with the environment used were the most frequent and the most severe [5].”
  - Eiffel: On the next slides: EiffelStudio as a teaching environment.

EiffelStudio for teaching (1)

- Programming environments must fulfill other requirements than professional environments [5]:
  - **Ease of use**: It must be easy enough to use by an inexperienced student.
    - EiffelStudio: Too complex as a teaching environment: many different compilation modes, many different viewing modes. Metrics are unneeded for teaching (professional edition only).
  - **Integrated tools**: All in one. The benefit is a smaller, unified interface.
    - EiffelStudio: All needed tools are integrated. The diagram tool is missing in the Free edition.

EiffelStudio for teaching (2)

- **Object support**: The concept of objects must be directly represented in the environment. The idea of object-orientation: objects exist independently from each other → interactive object creation tool (see BlueJ).
  - EiffelStudio: No interactive object creation possible.
- **Support for code reuse**: It must provide features to find reusable components, e.g. a class browser.
  - EiffelStudio: Class browser exists. Reusing code is encouraged.

EiffelStudio for teaching (3)

- **Learning support**: It should support active learning, by offering tools for interaction with objects, for visualization of class relationships, and for stepping through the code.
  - EiffelStudio: The diagram tool visualizes class relationships (only professional edition). The debugger allows stepping through code. No interactive object tool is available.

Eiffel for teaching - Conclusions

⇨ The language Eiffel seems to be predestined as a teaching language. EiffelStudio, however, seems to be too complicated for novice programmers.

The approach taken at ETH

- Carefully look at the approach that is chosen:
  - Analyze the “how”:
    - What teaching strategy?
    - ⇨ The Inverted Curriculum
  - Analyze the “what”:
    - What programming paradigm?
    - ⇨ Object-orientation
    - What programming language?
    - ⇨ Eiffel
  - **Analyze how it all fits together:**
    - Does the teaching strategy support what we want to be learned?

How it all fits together

- The Inverted Curriculum supports and is based on object-oriented concepts that will be taught.
- Eiffel represents object-oriented concepts in a clean way.
- Only point of concern: EiffelStudio might be quite difficult to handle for beginners.

Bibliography

[1] ITICSE 2001 Working Group on Assessment of Programming Skills of First-Year CS Students. *A multi-national, multi-institutional study of assessment of programming skills of first-year CS students.* ACM SIGCSE Bulletin, 33(4):125-140, 2001.

[2] ACM The Joint Task Force on Computing Curricula, IEE Computer Society. *Computing curricula 2001, computer science, final report.* ACM Journal of Educational Resources in Computing, 1(3), Fall 2001.

[3] Michael Kölling. *The problem of teaching object-oriented programming, part 1: Languages.* Journal of Object-Oriented Programming, 11(8):8-15, January 1999.

[4] Michael Kölling. The design of an object-oriented environment and language for teaching, 1999.

[5] Michael Kölling. *The problem of teaching object-oriented programming, part 2: Environments.* Journal of Object-Oriented Programming, 11(9):6-12, February 1999.