The Importance of Concepts when Teaching Programming

Peter Van Roy
Université catholique de Louvain
Louvain-la-Neuve, Belgium

Position statement
SIGCSE 2003 Panel
"The Role of Language Paradigms In Teaching Programming"
Feb. 21, 2003

The Elephant

Elephant photo © Wayne Matthews 2001

Six blind sages were shown an elephant and met to discuss their experience. “It's wonderful,” said the first, “an elephant is like a rope: slender and flexible.” “No, no, not at all,” said the second, “an elephant is like a tree, sturdily planted on the ground.” “Nonsense,” said the third, “an elephant is like a wall.” “Incredible,” said the fourth, “an elephant is a tube filled with water.” “What a strange and piecemeal beast this is,” said the fifth. “Strange indeed,” said the sixth, “but there must be some underlying harmony. Let us investigate the matter further.”

– Freely adapted from a traditional Hindu fable

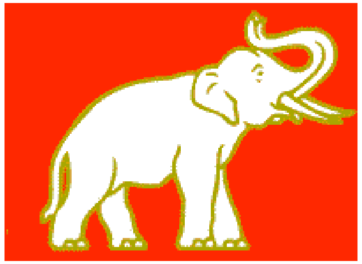
Côte d’Or Elephant

Elephant logo © Côte d’Or

Six blind sages were shown an elephant and met to discuss their experience. “It's wonderful,” said the first, “an elephant is like a rope: slender and flexible.” “No, no, not at all,” said the second, “an elephant is like a tree, sturdily planted on the ground.” “Nonsense,” said the third, “an elephant is like a wall.” “Incredible,” said the fourth, “an elephant is a tube filled with water.” “What a strange and piecemeal beast this is,” said the fifth. “Strange indeed,” said the sixth, “but there must be some underlying harmony. Let us investigate the matter further.”

– Freely adapted from a traditional Hindu fable

As a Belgian and a Côte d’Or chocolate lover, let me continue with their logo!

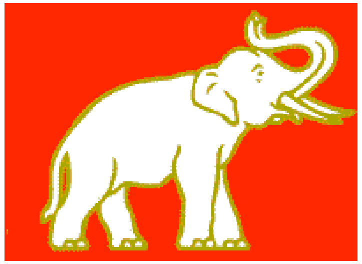


Programming paradigms

- Why are there so many programming paradigms?
  - Each is based on a different mathematical theory
  - Each is good for certain kinds of problems
  - Are all these paradigms really that different?
- Look closely and you will see that **paradigms have much in common**.
  Two examples among many:
  - Object-oriented programming is functional programming plus state (and different syntax)
  - Logic programming is functional programming with relations instead of functions
- Research shows that there is a fundamental set of concepts underlying all these paradigms, a **kernel language**
  - There are many possible such sets. Because we focus on practical programming, we consider a set of **programmer-significant concepts**, not a minimal set for theoreticians.
  - Each paradigm uses a different subset of the kernel language
  - Let's look at a couple of concepts to see how this can work

Example: closures

- The concept of a **procedure value with captured environment** (also known as a lexically-scoped closure) is the basis for many derived concepts, e.g., in object-oriented programming:
  - **Abstraction**: turn a piece of code into a procedure, method, or class
  - **Instantiation**: make instances of a class or a component
  - **Genericity**: parameterize a class (abstract class, inner class, template) or a component
  - **Components**: group related operations together
- With closures, these apparently different concepts are just programming techniques!
  - Popular languages give them syntactic support, to enforce the right invariants
  - It is easy to use them together
- Why not teach it this way instead of teaching these concepts as completely different?



Example: concurrency

- Concurrency can be added to other paradigms as a separate concept
- There are three main paradigms for practical concurrent programming
- **Declarative concurrency**: add concurrency to functional programming (no state)
  - Gives pipes, streams, dataflow, and much more (**no race conditions**!)
  - A little-known but very nice paradigm
- **Message-passing concurrency**: use concurrency together with asynchronous communication channels (a simple form of state)
  - Gives active objects (like in Erlang)
  - Great for applications with multiple agents (independent entities that cooperate)
- **Shared-state concurrency**: use concurrency together with mutable variables (state)
  - Gives locks and monitors (like in Java), and also transactions
  - Great for applications with a central data repository (like databases)
  - It's the best-known paradigm, but paradoxically also **the hardest to reason in**!
- These three paradigms seem very different but are actually closely related

Program design: the kitchen analogy

- Let’s say something about **program design**
  - So far, we have rather focused on concepts and paradigms
- Let’s compare programming to what a chef does in his/her kitchen
  - **Concepts** are like ingredients (closures and concurrency are like flour and eggs)
  - **Techniques** are like “tricks of the trade” (e.g., divide-and-conquer, how to make a sauce thicker)
  - **Algorithms** are like recipes (a set of instructions that gives a result in finite time)
  - **Paradigms** are like national styles (Indian, Chinese, Italian, Tex-Mex, etc.): each one favors certain ingredients and recipes
  - **Design** is the planning you need to prepare a three-course meal: carefully choosing dishes that go well together, finding the recipes, selecting the right ingredients, and timing the preparation so that all dishes are ready at the right time (nontrivial!)
- Concepts (ingredients) can’t be introduced in a vacuum; they must be introduced together with their design principles (how to cook with the ingredients)
  - Concepts and design principles must be taught together

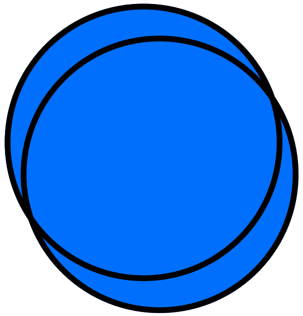
Teaching with “concepts first”

- Programming paradigms are not what really matters
  - **What matters is the concepts they are made of**
  - Concepts and design principles must be taught together
- Teaching programming with concepts is completely natural
  - Paradigms appear like styles
  - Complicated paradigms can be explained in a simple way
  - Traditional paradigm boundaries are seen as artificial
  - Student understanding transcends traditional paradigm boundaries
- We have been using this approach for almost three years
  - In courses at UCL and KTH, but also NMSU and Cairo University
  - We have teaching materials (textbook, software, slides, etc.)
- The approach is based on more than a decade of research in language design and implementation by many people
  - In the Mozart Consortium, which groups labs in Sweden, Germany, and Belgium (see **`http://www.mozart-oz.org`**)

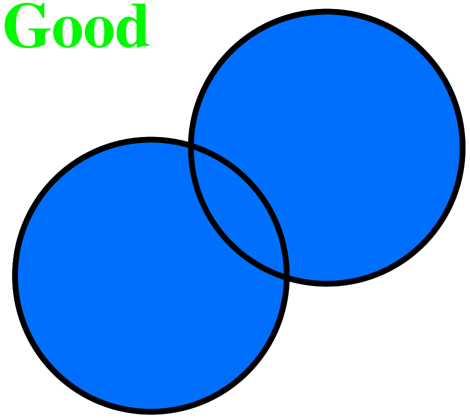
During the panel discussion: a comment from Joe Armstrong

Why we need people with different backgrounds

Bad



Good



- In a team project, you need people with **different backgrounds**
  - If they have the same backgrounds, their total knowledge is only as much as one person's
  - Good companies know this: they search for people with complementary skills
  - Knowledge must overlap a little, though, otherwise people can't talk with each other!
- This is why it's bad if a computer science curriculum is too homogeneous
  - Diversity is essential
  - It's good for students to learn more than one paradigm
  - It's good for schools to have different curricula