Overall impression on COOLIE-1 2002

The COOLIE-1 (Comprehensive Object-Oriented Programming Learning International Exchange) first workshop took place in Boston, at the *NORTHEASTERN UNIVERSITY*, invited by David H. Lorenz. This location was great. Indeed, *NORTHEASTERN UNIVERSITY* is a real american university and Boston a real university city, especially compared with our standards in Nantes! What I heard is that there are 75 universities in Boston, and tens of thousands of students. The *COLLEGE OF COMPUTER SCIENCE*, where the sessions tool place (not mentioning the restaurants…) is a very nice place, conveniently located closed to Boston "downtown" and to the rest of the campus.

The COOL project itself is an initiative of Kristen Nygaard. The idea is to produce material (textbooks, examples, DVDs…) to teach an introductory course in object-oriented programming. This material must follow the pedagogical approach advocated by Kristen and others, namely, to use, from the beginning, a sufficiently complex example, and to use several programming languages to illustrate the different concepts.

Here are what should come next, after this first workshop:

- Send to Kristen points to consider and notes + discussion with Pierre Cointe and others.
- Each participant should send slides for a 1-hour class on a subject following the pedagogical approach of the COOL project, as if the 10 participants were to give a common tutorial, contact the others (in particular David H. Lorenz) to get more information.
- Start exchanging material for the lecture among participants.
- Try to write questions you expect the students to answer on the concept of perspective.

Some questions remain unanswered:

- What about adaptation?
- What do you think you have that is better than the others?
- What are the post-conditions of the course?
- What it is to be a test site? Rights, duties, what do they get?
- Time schedule?
- Different modules?
- Why make a commitment before knowing what is in?
- Should more than one language be used up-front? Is this not too complex?

Workshop

Overview of the COOL project

Kristen Nygaard
(*UNIVERSITY OF OSLO, NORWEGIAN COMPUTING CENTER, AND SIMULA RESEARCH LABORATORY*)

Kristen gives an overview of the COOL project. There are still some unanswered questions but this gives a good general idea of what the COOL project is.
(For more information, look at Kristen's web site)

Notes

The *NATIONAL RESEARCH COUNCIL* of Norway (reluctantly) accepted to support the COOL project thanks to heavy political help.
Structure of the project, organization of the project:

- Central team: 5–7 people (plus PhD students).
- Main location: At the *INTERMEDIA CENTER* in Norway.
- Goal: Introductory courses all over the world in **programming and system comprehension**.
  - At the end of the course, the students must be as competent as any other students.
  - Not about algorithms but about understanding.
  - Make one course as a basis for other people to extend.
  - The restaurant example shall be adapted to different cultures.
  - A conceptual unified platform.
  - Process-based concepts are more general than object-oriented concepts.
- The 3-year project may become a 5-year project with double funding but beware of bureaucrats!

The project shall incorporate universities with interests in undergraduate teaching (unlike *STANFORD UNIVERSITY* or the *M.I.T.*) and (maybe) some colleges (*WILLIAMS COLLEGE*, *DUKE COLLEGE*…).
The project shall be carried on in Scandinavia, South America, North America (English), China, India (Bangalore?), United Kingdom (Leeds and New Castle, but without New Castle the U.K. might be dropped), and France (Pierre Cointe, Jean Bézivin, *SOPHIA ANTIPOLIS*?)
The project cannot be too opened ⇒ It requires commitment ⇒ It requires an agreement on the project: The participants must be willing to try out the modules.
The test sites can become active to create a community of active test sites.
The members of the COOL project make the modules and the people link them with their own material.
The modules must focus on the concepts with a mapping to languages; there must be a main language plus another to capture how things are different and not to be stuck with a specific language.
A module is about a week of class.

Open lecture: "How the basic concepts in object-oriented programming were developed"

Kristen Nygaard
(*UNIVERSITY OF OSLO, NORWEGIAN COMPUTING CENTER, AND SIMULA RESEARCH LABORATORY*)

(From the 20Mb file `OO-History and Basic Concepts` and the 16Mb file `OOPLSA-Lecture 2001` on the COOLIE-1 CD.)

In this lecture, Kristen presents how the major concepts in object-oriented programming came to life: Classes, objects, inheritance, and polymorphism… Starting with his job as "calculator" in the first project of civil nuclear reactor in Europe, Kristen shows how they came to **Simula I** and to **Simula 67** and the various reasons (scientific, political…) that led to the creation of object-oriented programming.

Notes

We wanted **Simula** to be an *existing* language, i.e., with a user community.
Object-oriented is the dominant programming style.

Son: “How object-oriented were invented?”
Dad: “From God my son.”
Son: “So why are the parameter rules so difficult?”
Dad: “Don’t bother with such difficult question. God had a deeper meaning.”

Simula 67 had to be compatible with **Algol**.
Mathematics is about relations: $x = x + a \Rightarrow a = 0$.
Informatics is about processes: $x = x + a \Rightarrow$ x is given a new value.
Process = Description of sequences of events (state transition).
Algol sees the world as a stack of instances.
Calculating the radius of a uranium rod in Europe’s first civilian atomic reactor, Garwick developed a numerical solution and Kristen made the calculation.
From mathematics to “Monte Carlo” simulation.
“To describe is to understand” ⇒ “To program is to understand” ⇒ Multiple inheritance is very bad!
Two types of objects: Active and passive.
May 1962: UNIVAC ⇒ Jim Nickitas ⇒ Bob Beamer, director of System Programming.
Richard L. Wexelblat ; *History of Programming Languages* ; Academic Press, 1981.
Duality: Active objects can be passive and vice-versa, and then everything can be active or passive ⇒ processes ⇒ objects.
Simula 67 = Action sequences (multi-threads) organized as a stack.
The life spans of the objects are nested in one another but indirect referencing, solution = qualified references (Tory Hoare) but not flexible enough, the solution is inheritance: 6 of January 1967, at 2am.
With a new idea, do not protect it! Go with a club or an ax! If it survives…
Then polymorphism, then procedure parameter, and then late binding.
Read sci-fi because it shows “what will happen next, if this becomes true?”.

Lecture in David Lorenz’s OOP undergrad class: “The restaurant example”

Kristen Nygaard
(*University of Oslo, Norwegian Computing Center, and Simula Research Laboratory*)
David H. Lorenz (*College of Computer Science, Northeastern University*)

(From the 4Mb file `OO-Pedagogical Approach` on the COOLIE-1 CD.)

In this lecture, in front of undergrad students, Kristen shows the restaurant example. First, he asks the students to act “as if” they do not know anything about object-oriented programming and then he asks the students to describe different pictures and interact with them to show them how the perspective is important when identifying the objects (and their classes) important to the design. Among other things, he also shows them that it may be appropriate to model a table as being something with four legs, a stool… but this model might be useless in the perspective of modeling the restaurant. He also shows the difference between active and passive objects and the concepts of attributes through the example of James Smith and Venus Jones. Then, he goes on with the four important qualities of processes:

- Substance.
- State.
- Transition.
- Structure.

He introduces a notation for the objects representing the guests James Morgan IV and James Smith.
From those two objects, he abstracts, with the students’ help, the class `Guest`.
Finally, he concludes on the different categories of structures for classes in **Beta** (?):

- Reference.
- Type.
- Pattern.

As far as I can see, the restaurant example seems to trigger much interest from the students as well as from the professors! It seems well suited as introductory example for the various concepts of perspective, abstraction, class, and action… However, I believe it should be made clear right from the beginning what is the aim of the restaurant example (is it to model the restaurant from the manager’s *perspective*, or from the cashier’s *perspective*…) to make clear for the students the reasons of the various choices that will be taken along the course.

Notes

"Simula 67 is considered as an improvement over the programming languages that came *after* it."
"They are lots of things in programming languages that are stupid", about name parameters…
Understand, describe, and communicate.
Object-orientation is easy to understand but seldom taught, see the 14Mb file `OOPLSA-Lecture 2001` on the COOLIE-1 CD.
System = A whole consisting of components, each component has properties and may interact with other components.
Different perspective/points of view (technical, money, social).

Informatics should be defined as the study by scientific methods of a domain of phenomena and a perspective selecting a set of characteristics of those phenomena

Kristen Nygaard
(*UNIVERSITY OF OSLO, NORWEGIAN COMPUTING CENTER, AND SIMULA RESEARCH LABORATORY*)

The following highlights some of the pedagogical choices made in the COOL project. The main things are (in my opinion) that the COOL project is about:

- Teaching object-oriented programming to undergrad students, this implies:
  - The teaching is **not** about **Java**, **Beta**, or software design, or… The teaching is about the concepts in object-oriented programming languages and the methodological approach to object-oriented programming: Perspective, maneuvering space, abstraction, understanding, and communication.
  - The students **do not** need any knowledge in computer science or programming (not even in abstract data types or algorithms). The students need the basic knowledge (mathematics, scientific reasoning) expected for a student done with his **high-school diploma** (*baccalauréat* in France?).
- Designing an introductory course and the associating material, this implies:
  - The course **introduces** the concepts of object-oriented programming; it is not about **everything related to** object-oriented programming. The different test sites may adapt, expend, and link this introductory course with their own course the way they think fit.
  - The course **shall contain** lecture notes, questions, answers, practical questions, exams, examples, application, and source code… and is given as a textbook. Eventually, DVDs or on-line courses may be derived from the core material.
  - The pedagogical landscape will expand and this requires hypertext navigation.
- Pedagogical choices, this implies:
  - The examples must be about things all the students (around the world) may relate about in some ways.
  - The examples must be **sufficiently complex** to arouse the students' curiosity and to provide sufficient case studies, but without being overly complicated.
  - The **restaurant example** is used all along the course, from the notion of perspective to the implementation in **Java** …
  - The different algorithmic notions (type, binary tree, recursion…) are introduced along the course when required.

Notes

"I want this project done before I die."
"I want people to understand object-oriented programming."

The restaurant example is good because it is sufficiently complex and everybody can relate to it. (Another is the traffic example.)
There exist different paths in the landscape ⇒ Hypertext navigation.
No obligations to cover the course, there will be pointers to say that if you want to go in that particular direction, here is what you need to learn.
To present the difference between a programming language and a concept, just show a slide with a single concepts and words in different natural languages describing this concept.
All is a question of level of understanding; along the course, the level of understanding shall both raise (higher-level concepts) and set (lower-level concepts).
There still exists the problem of mapping between concepts and programming languages.

A **perspective** is what allows you to say that "this" "is-a" "that". (Remember the example of the wooden chair and the firewood.)
Programming is about understanding and communicating about reality.
There is much about abstraction, but how **to obtain good abstractions**? This is much like art…
Education is about providing people with models of thinking.
Design patterns are good and are a sign of maturity.
Employability is important and it is important to think about the context.
We must teach student to go beyond brute force ⇒ We must teach student to build tools first!
How to integrate the tools together to solve the problem on the long term ⇒ Programming with some intelligence.
We must present examples with some choices and limitations ⇒ To show the **maneuvering space**.
See so-called **Object Logo** from Bratislava, subject of a book in *ADDISON-WESLEY*.
See the *LEGO* **IDE**.
See the book "Great ideas in computer science" at *M.I.T. PRESS*.

The visualization language might not be done by the Norwegian staff, so what do we do?
Is there a list of what should be included? (⇒ Metrics.)
What example is good at what?

Presentations from the participants

Other participants were:

- Yann-Gaël Guéhéneuc
- David H. Lorenz
- Kristen Nygaard
- Richard A. Rasala
- Joel Spiegel[1]

Viera K. Proulx

(*COLLEGE OF COMPUTER SCIENCE, NORTHEASTERN UNIVERSITY*)

Viera presents a set of **Java** tools that she uses with her students. Among other things, she developed with others a graphic framework, the **Java Power Tools**, which allows the construction of powerful GUIs in a simple way. She uses this graphic framework with the students to develop their exercises. The first advantage of this approach is the ease of development for the students and the complexity in user interaction their applications can reach without much effort. The second advantage of this approach is that it keeps the students' interest high: They are not bored by limited console interaction, they can build "real"-like tools and examples. I have not tested the **Java Power Tools** yet. However, this seems much promising as a unified Java-platform for the examples and exercises (?).

Notes

Lab-like application for the students.
Having objects and methods to modify the state of the objects.
Turtle to show the methods and the things being executed
+ Console displays exactly the methods and the objects
+ In case of typing errors, a window pops up and only goes away when the parameter is fixed.
"Say what you want, and then do it."
Visual graph algorithm.
Animated automata

```
toStringData()
fromStringData(String)
```

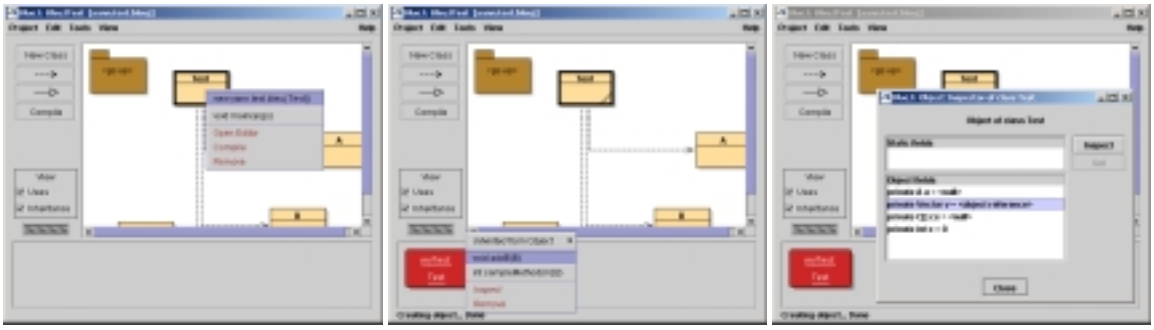
"What is JUnit?"!?!
Model – View – Action – Controller
Soda machine ⇒ To use different implementation of the stack and see if these implementations work

[1] Bio modified from http://www.adm.ufrgs.br/professores/hfreitas/vso/arquivos/amazon.doc: Joel Spiegel is now retired (at 41-year old). He joined Amazon.com in March 1997 as vice president of engineering. From March 1995 to March 1997, Joel held several positions with Microsoft Corporation, including Windows 95 Multimedia development manager, Windows Multimedia group manager and product unit manager for information retrieval. From June 1986 to March 1995 he held a variety of positions at Apple Computer, most recently as Senior Manager responsible for new product development in the Apple Business Systems Division. Prior to that Joel held software product development positions at a number of companies, including Hewlett-Packard and VisiCorp. Joel received his B.A. in biology with honours from Grinnell College.

Michael E. Caspersen – Introductory object-oriented programming

(*DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF AARHUS*)

Michael presents his pedagogical approach to teaching object-oriented programming. The main point being his systematic approach to software development and the idea of presenting concepts as they come (and without naming them up-front, but after the students have "discovered" them). He also presents a demo of **BlueJ**. **BlueJ** is a (light) development environment for Java that (partially) addresses the problem of mapping between the Java programming language and a visual representation *à la* **UML**. The latest version is nice and works very well on small-size examples. The major capability of **BlueJ** for teaching is the possibility to instantiate on-the-fly classes, to send them messages, and to inspect them and their attributes. This makes **BlueJ** a nice and easy-to-use tool to present the students the concepts of encapsulation, instantiation, and message sends…



Notes

Aim:

- To apply fundamental principles to the systematic construction of software system.
- To gain experience with development of solutions to simple problems.
- To become familiar with a modern object-oriented programming language.

Systematic:

Problem	→	Conceptual model (**UML**)
Association / Aggregation / Composition???		
Conceptual model	→	Design model (interfaces, pre-/post-conditions)
Design model	→	Partial implementation (structure)
Partial implementation	→	Implementation

Structure of textbooks is lame!

To draw a penta with a turtle ⇒ Recursion.

"A recursive solution is derived with the students without ever mentioning the notion of recursion."

GUIs are used as example of frameworks, not up-front.

But "What about instantiation?"

Kristen Nygaard:

"Substance, states, transitions":

Substance only	= Object-oriented programming
States only	= Logic programming
Transitions only	= Functional or imperative programming.

"Everything is coming from objects: No loose parts on the floor, no actions coming out of thin air."

"Object-Oriented programming languages should have been named system-oriented programming languages."

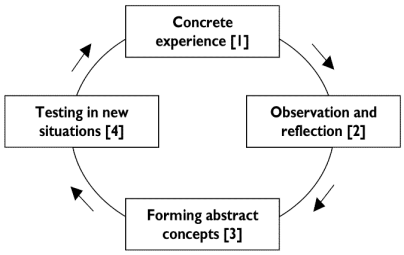
How to get started? "The big bang problem" by Richard Pattis, 1993

Pedagogical aspects:

- Hide details.
- Abstraction.
- Open/closed exercises.
- Good examples / model software.

Design is a very difficult task!

Kolb's learning circle:



Difference between boys and girls in the traffic example: Disorder and order…

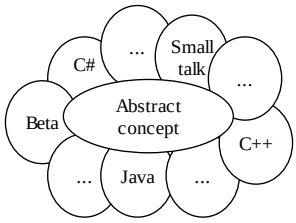
Arturo J. Sánchez-Ruíz – Teaching OOP: Patterns, visualization, cognitive issues

(*DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES, UNIVERSITY OF NORTH FLORIDA*)

Arturo bases his pedagogical approach on a concept centric approach. He calls his approach cognitive calisthenics[2]. The idea is to start from abstract concepts, to show that those concepts may be implemented in several different ways and then to visualize them in a given programming language. In particular, he uses a trace-generator (based on the **JDI** of the **JPDA**!) to generate the trace of a program's execution and then **Director v8.5** (**Flash**), to animate the trace. With this approach, he easily shows concepts such as message send, callbacks… to his students.

Notes

"Cognitive calisthenics."
Patterns!
Alexander's patterns came from the top ("he did not begin with a door knob!")
Semantic distance!!!!!!!!
Executable specifications (**Beta**)???
The approach is concept centric:



Then, visualization:

- Class–class.
- Object–object.

Then, animation of Java programs!!!! (Caffeine)
Java program – Analysis tools – Animation independent representation – Animation (**Director v8.5**).
Taxonomy of perspective:

- Social and ethical: …
- Pedagogical models: …
- Professional models: …
- Software-development models: …
- Delivery models: …

[2] Calisthenics (from the Merriam-Webster):
Pronunciation: `-niks`
Function: *noun plural but singular or plural in construction*
Etymology: Greek *kalos* beautiful + *sthenos* strength
Date: 1827
1: systematic rhythmic bodily exercises performed usually without apparatus
2 *usually singular in construction***:** the art or practice of calisthenics

Gunnar J. Carelius – SESE: Experiment Support Environment

(*SIMULA RESEARCH LABORATORY*)

Gunnar presents SESE. SESE is a web-based experiment environment. It consists in a web-based interface where the students can find their own "laboratories". A student logs into her laboratory and then answers questions and performs exercises at her own pace. The answers can be given directly using value fields, text fields… or by uploading source code files, depending on the questions. It is not possible to go back to a previously answered question. Each student can use whatever development environments, editors, tools she sees fit. The administrator is responsible for setting up the questions (the experiment) and can monitor the progress of the students, can provide help…
At the *SIMULA RESEARCH LABORATORY*, they used this experiment environment to assess the differences between a "good" and a "bad" design for a coffee machine. Given two implementations of coffee machine, they asked about 200 people to make some changes and thus assessed if the "good" design is good and if the "bad" design is bad.
This experiment environment could be used to share experiences, questions, and material among the different test sites and to set up worldwide experiments, exams, and practical courses!

Notes

SESE is a web-based experiment environment.
Coffee machine experiment:

- Good/bad designs
- Charge effort measurements
- Defects measurements
- 130 professionals, 60 students.

SESE on Windows Terminal Sever + Motivations + Statistics.

Farewell meeting

Kristen Nygaard

(*UNIVERSITY OF OSLO, NORWEGIAN COMPUTING CENTER, AND SIMULA RESEARCH LABORATORY*)

During the farewell meeting, Kristen presents the theater metaphor, which describes the model and the process generators, and which distinguishes actors and roles. The metaphor theater also is useful to describe program execution. However, I found the slide 11 on the actor/performance levels confusing, which describes the encapsulation of the different objects and the binding of their interfaces at the different stage.

Notes

Aspect of the font ⇒ Feelings.
Computer science ⇒ Informatics!
Multi-perspectives + Designing/generating/modifying phenomena (process).
Manifest phenomena (computer) vs. Cognitive phenomena (mind).
Intention – Extension – Designation / Designation – Substance – Action
Thomas Kuhn, see slides
See slide 72 of the 16Mb file `OOPLSA-Lecture 2001` on the COOLIE-1 CD on "Model" from Lindsjørn & Sjøberg, 1987 (modified by Kristen).
Beta and **Delta** programming languages.
Information system = System of processes + Components???
Should we consider operation in typed state attributes: Quantities.
Perception ≠ Recognition.
Theatre metaphor (see the 0.5Mb file `OO-Theatre Metaphor` on the COOLIE-1 CD) for multi-level open-systems (link with Andrés Farias' protocols???)
⇒ Courses in system development at the graduate level.
"Theory of cognitive dissonance" by Festinger.
Current courses are too abstract, too demanding ⇒ Better use a "learning by doing" approach.