Computer Science Education Based on Fundamental Ideas

A. Schwill
Fachbereich Informatik - Universität Oldenburg
D-26111 Oldenburg, Germany
Phone: +49-441-798-2412, Fax: +49-441-798-2155
e-mail: Andreas.Schwill@informatik.uni-oldenburg.de

Abstract

We sketch a pedagogical ‚theory' based on Bruner's educational psychology that might set computer science education for students and teachers on a well-founded basis and integrate different approaches to teach it, stress its long-lasting fundamentals and give a feeling of its essence. For it we (1) define the notion of fundamental ideas more precisely by several criteria, (2) present a collection of fundamental ideas of computer science, and (3) show how to develop a curriculum that centers around these ideas.

Keywords

fundamental ideas, curriculum research, Bruner's educational psychology

1 MOTIVATION

Although computer science is a regular subject in school for a long time, there is still more and more discussion on how to teach computer science and what to teach: Should computer science education be oriented more towards its applications or more towards its fundamentals or more towards its social effects? How can we improve computer science education with teachers who have not passed a full university education in that field? How can we cope with the rapid developments of computer science, both with respect to curriculum development and continuing education?

Just to explain the last point we see that after more than 40 years computer science is still developing dynamically. Paradigm changes are constantly announced. This leads to a complicated dichotomy: On the one hand, people seem to agree that this progress cannot be carried over equally fast to school education because of delays in change of curricula, pedagogical reflection, teacher education etc. On the other hand, since each student will probably face several paradigm changes in future life - with much of the respective knowledge becoming obsolete each time - the skills acquired earlier must be robust enough to meet the challenges of the latest fashion, and also enable the student to cope with changes.

The same argument holds with respect to teacher education. Teachers also have to be (re-)trained in a way that enables them, with mild support by continuing education programs, to integrate new results into their knowledge structure and to reflect and assess recent developments with respect to future relevance, school adequacy and other pedagogical issues.

This competency is possibly achieved best if students and teachers are given a sketch of the *fundamental ideas, principles, methods and ways of thinking* of computer science, instead of trying to teach them an incoherent set of recent technology-driven developments as is often done by inservice teacher education programs. Only the fundamentals seem to remain valid in the long term and

enable students and teachers to acquire new concepts successfully during their professional career, since these new concepts will then often appear to be just further developments or variants of ideas already familiar and so are comprehensible to students and teachers more easily.

2 FUNDAMENTAL IDEAS AS AN EDUCATIONAL PRINCIPLE

Whitehead (1929) proposed to deal in school with „few general ideas of far-reaching importance“, since the students are „bewildered by a multiplicity of detail, without apparent relevance either to great ideas or to ordinary thoughts“. The same situation we often have - so are the author's experiences - if teachers have passed a quick inservice education program in different „modern“ subjects of computer science that enables them neither to establish a stable cognitive structure integrating all these subjects nor to teach computer science to their students in a suitable manner.
Bruner (1960) formulated the teaching principle that lessons should be based on the **structure** of science which is defined by so-called **fundamental ideas**. He justified his approach as follows: Learning is mainly for preparing us to master our future life more successfully. In order to cope with changes occuring later in private life, economy and society students must be able to **transfer** knowledge acquired earlier to new situations. Of particular interest here is the **non-specific transfer** which relates to long-term (often life-long) effects: We should teach on a meta-level fundamental notions, principles and ways of thinking (so-called **fundamental ideas**) and may hope that students are able to use these abstract solution schemas in modified („transfered“) form for any (even absolutely new) problems they may face in their later lives.
Nonspecific transfer should dominate the entire educational process of schools providing general education: Permanent creation, extension and consolidation of knowledge in form of fundamental ideas. Therefore - and now we return to Bruner's request - all curricula and teaching methods in computer science classes as well as inservice or preservice education programs for teachers should stress the fundamental ideas of each topic. Teachers should be enabled to analyze each new subject to be included into computer science lessons what ideas it is based on and to present it according to the idea-oriented approach proposed in the rest of the paper.

Unfortunately, Bruner as well as subsequent researchers do not provide an explicit definition of fundamental ideas. Instead they give some examples from several subjects and leave it to the reader to develop an intuitive idea of what the term might mean in general. As applied to computer science we define the notion as follows:

A fun**damental idea** of computer science is a schema for thinking, acting, describing or explaining which satisfies four criteria:

The Horizontal Criterion. A fundamental idea is applicable or observable in multiple ways and in different areas of computer science and organizes and integrates a wealth of phenomena.
We call this property the Horizontal Criterion, since the idea may be considered as a horizontal line intersecting a large number of fields where it applies.

The Vertical Criterion. A fundamental idea may be taught on every intellectual level.
Bruner (1960) said that „any subject can be taught effectively in some intellectually honest form to any child at any stage of development“. This suggests that a fundamental idea organizes the topics of a field also in a vertical dimension: An idea can be taught on the primary school level as well as on the university level. Presentations differ only by level of detail and formalization. Thus, an idea can serve as a guideline for lessons on every level of the entire educational process and ideas can be revisited periodically in greater depth and complexity (so-called spiral p**rinciple**, see also below).

The Criterion of Time. A fundamental idea can be clearly observed in the historical development of computer science and will be relevant in the long run.
This aspect is important for two reasons. First, it gives a clue as to how to find fundamental ideas: Scientific notions, concepts or structures of computer science that have a definite historical background are more likely fundamental ideas than are recent developments. Second, lessons based on fundamental ideas will not become antiquated as quickly as conventional lessons - a major advantage in teaching computer science given its dynamic evolution.

The Criterion of Sense. A fundamental idea also has meaning in everyday life and is related to ordinary language and thinking - its context being pretheoretical and unscientific.
Only a precise definition turns an idea „with sense“ into an exact notion „without sense“. For example, consider „reversibility“ as an idea „with sense“ and „inverse function“ as a purely mathematical formalization of it „without sense“. While we can see examples of reversibility in many everyday situations - do vs. undo - the term „inverse function“ has no everyday meaning. From the pedagogical point of view this criterion is closely linked to the Vertical Criterion. Whenever we have to teach a fundamental idea on a low intellectual level, i.e. we have to give students a first vague impression of the idea, we may begin with those situations in everyday life where a fundamental idea becomes apparent. For a more specific explanation see Section 4.

3 FUNDAMENTAL IDEAS OF COMPUTER SCIENCE

By now fundamental ideas have been proposed mainly for mathematics (e.g. Halmos (1981)) and some of its branches (e.g. Heitele (1975)), but there are very few comments on how these ideas have been worked out. We have tried to determine fundamental ideas by abstracting from the contents of computer science to its ideas in three steps: (1) Analysis of the concrete activities of computer science and their relationships and analogies. Since a central purpose of computer science is to investigate the software development process in its broadest sense and to provide methods for it, it seemed reasonable to first analyze for fundamental ideas the concrete activities during this process and then to establish relationships and analogies to the field of computer science in general; (2) Revision and improvement of the results obtained in step 1 by checking whether each idea satisfies the four criteria for fundamental ideas; (3) Structuring the collection of ideas according to their relevance in computer science.
Due to space limitations we only present the final results. There are three fundamental ideas that dominate all stages of software development as well as all activities in computer science - algorithmization, s*tructured dissection* and *language*, each of which gives - on further analysis - a wealth of other fundamental ideas (below written in italics). These three are explained in turn in more detail now.

Algorithmization

By algorithmization we denote the entire process of designing, implementing and running an algorithm. The strong relevance of algorithmization within computer science, as required by the Horizontal Criterion, seems obvious.
A careful analysis of the activities when developing algorihms gives a wealth of other fundamental ideas which may be assigned to four large domains - design, programming, execution, evaluation: During design of an algorihm one often uses powerful paradigms such as di*vide and conquer*, *backtracking* etc. Afterwards this design is carried over into a program using several basic ideas to describe data and control structures, e.g. *concatenation*, *repetition*, *recursion* of commands and data. The finished program is executed on one or more processors. A fundamental idea here is the

notion of *process*, i.e. the separation of description and execution of an algorithm. The last group of ideas deals with assessing the quality of algorithms. The two main criteria are *correctness* and *complexity* each related to several corresponding ideas.

Structured dissection

Dissection covers the process of subdividing an object into several parts in a structured way. That includes a detailed description of the parts, their relations to the whole as well as interactions between the parts. Typical ‚objects' in computer science which are dissected in this manner are problems (into subproblems), algorithms (into procedures or modules), modules (into smaller modules, called hierarchical modularization), the software life cycle (into different phases), languages and machines (into different complexity classes), etc.

We can distinguish two aspects of dissection, a vertical aspect called *hierarchization* (Figure 1(a)), i.e. the construction of certain levels of abstraction often distinguished by different language levels, and a horizontal aspect called *modularization* (Figure 1(b)) where an object is subdivided into different parts all of the *same* level of abstraction. The well-known *hierarchical modularization* is obtained by merging these two aspects (Figure 1(c)). The idea of hierarchization can be observed in many different contexts, e.g. level-oriented models of computer architecture, language hierarchies (main example is the Chomsky hierarchy), machine models, complexity and computability classes, virtual machines, ISO-OSI reference model. With both hierarchization and modularization many other „small" fundamental ideas are connected.

Obviously every dissection procedure comes to an end some time, at the latest if an atomic level is achieved. This observation leads us to the third fundamental idea within structured dissection - orthogonalization -, which is, roughly speaking, the definition of a small set of constituents that spans a certain domain. Due to space limitations we cannot explain this idea in more detail.

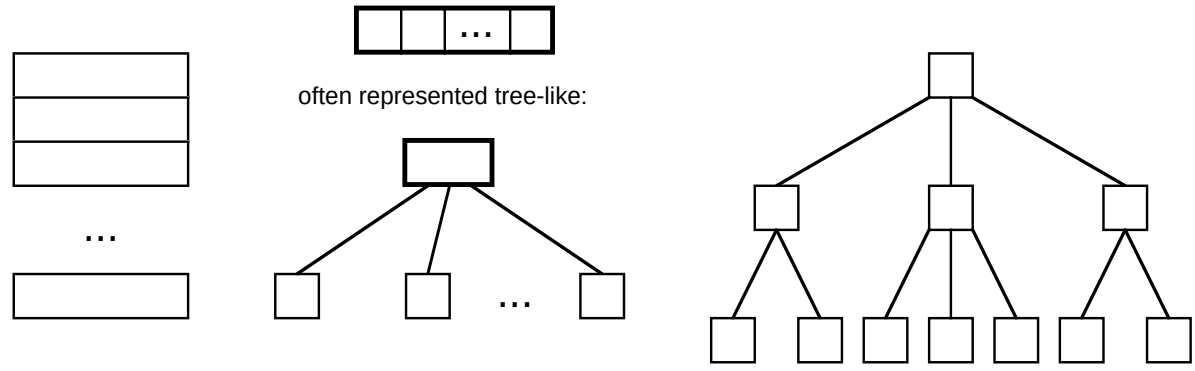

Figure 1 (a) Hierarchization; (b) Modularization; (c) Hierarchical modularization.

Language

Language plays an important role, not only for programming (programming languages), for specification (specification languages), for verification (logic calculi), in data bases (query languages), in operating systems (command languages), but there seems to be a general trend in computer science to formulate any facts by a language. This approach has the following advantage: First, it simplifies the view of facts, since every problem can be considered as a problem upon words now; second, manipulation of languages has been successfully investigated in the past: There are powerful theoretical results and efficient algorithms for translation. In close relation to languages

there are two fundamental ideas -*syntax and semantics.*

We conclude these brief considerations by presenting part of the catalog of fundamental ideas of computer science (Figure 2). The complete catalog will appear elsewhere. Note that names written in italics have been added for systematization only and denote groups of ideas but are not ideas themselves.

4 COMPUTER SCIENCE EDUCATION WITH FUNDAMENTAL IDEAS

Bruner said that lessons oriented towards fundamental ideas have to be organized according to the **spiral principle** which he describes as „The early teaching of science, mathematics, social studies, and literature should be designed to teach these subjects with scrupulous intellectual honesty, but with an emphasis upon the intuitive grasp of ideas and upon the use of these basic ideas. A curriculum as it develops should revisit these basic ideas repeatedly, building upon them until the student has grasped the full formal apparatus that goes with them.“ (Bruner, 1960)
Bruner (1966) also recommended three representations of concepts to be learned: Before the *symbolic* demonstration of notions or concepts (by formulas etc.) and their structural analysis students should obtain an intuitive idea of the notions by pictures (*iconic*) and actions (*enactive*).
The following examples illustrate the spiral principle and also demonstrate the Vertical Criterion for three fundamental ideas by sketching subjects for lessons in primary school (P), in grades 5-9 (S1) and in grades 10 and above (S2).

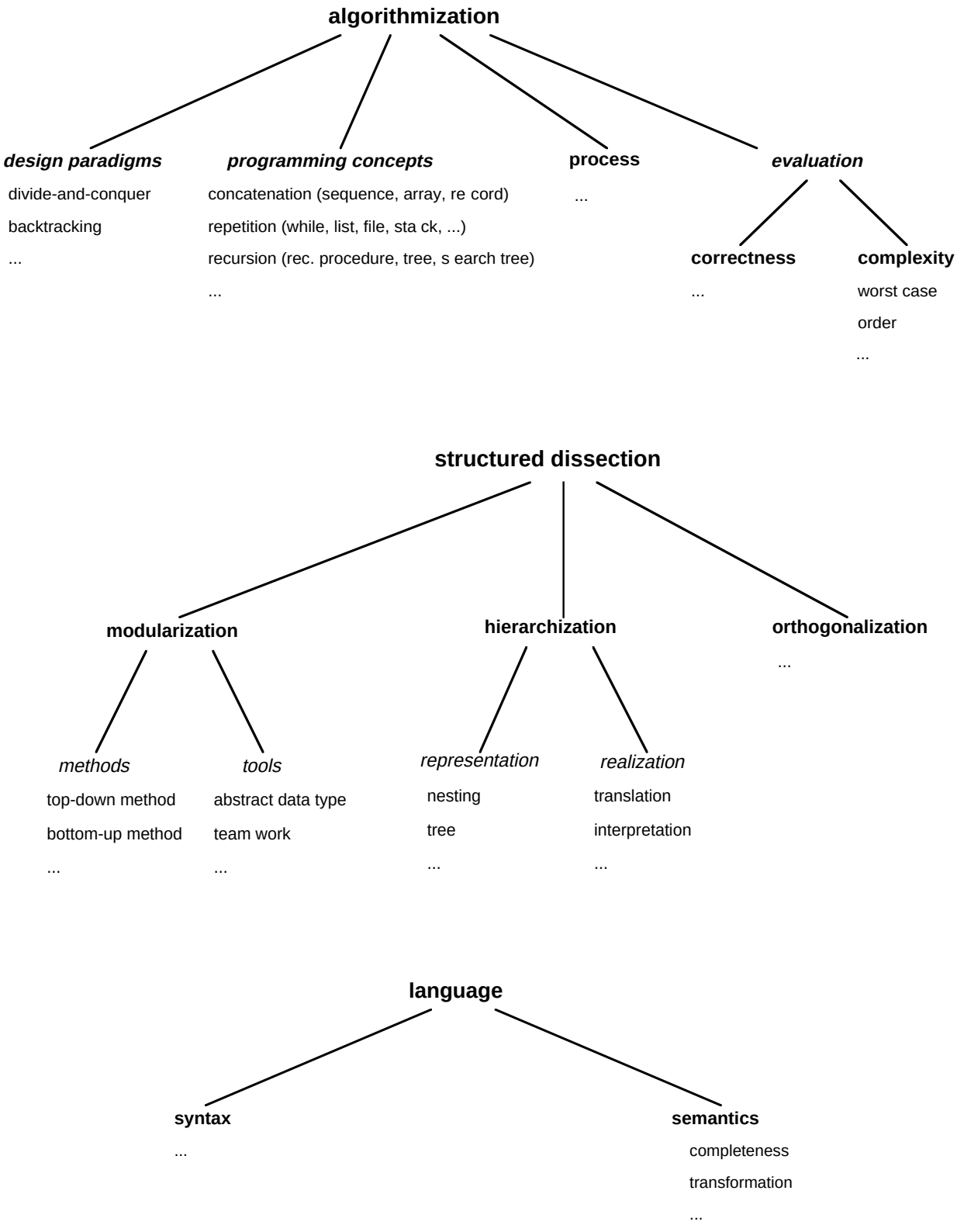

Figure 2 Fundamental ideas of computer science.

Divide-and-conquer approach

(P) A child may sort a stack of paper cards by size by dividing the stack, giving the parts to his or her class-mates and merging the sorted stacks he or she is given back. Numbers may be guessed by binary searching.

(S1) Algorithms in computational geometry, e.g. for computing the convex hull, may be used to extend the knowledge.
(S2) Complexity considerations for general divide-and-conquer algorithms; establishing and solving a recurrence relation for the runtime.

Worst case-analysis

(P) Worst case considerations can start with questions like: How long does it take to get to school in the worst case, if the bus is late, if all traffic lights are red, if the roads are icy? Or: How many questions are necessary in order to guess a number using binary search?
(S1) A more formal approach may follow at this level, e.g. by relating the runtime to the length of the input and determining the worst case for each input length.
(S2) Formal definition of worst case runtime and proof of lower bounds.

Abstract data type

(P) The blocks world may be defined as an abstract data type: On a table there is a number of cubic blocks that may be piled up. The only two operations are (1) putting one block onto another one and (2) testing whether a block lies on another one or whether it lies immediately on the table. Is it possible to establish any situation in the blocks world by these two operations (also related are the ideas of completeness and orthogonalization)?
(S1) The example in (P) may be made more precise here. Problems concerning consistency and completeness of an abstract data type may follow. Which laws hold for the operations in the blocks world?
(S2) On this level a formal notation for abstract data types may be introduced using more complex examples (stack, queue, file). Considerations of implementation may follow.

5 CONCLUSIONS

We have applied J.S. Bruner's principle of orienting lessons towards fundamental ideas to computer science education of students and teachers. So far this approach seems to have the following advantages:

- A subject is more comprehensible if students and teachers grasp its fundamental principles.
- Fundamental ideas condense information by organizing uncoherent details into a linking structure which will be kept in mind for a longer time. Details can be reconstructed from this structure more easily.
- Fundamental ideas enable teachers (1) to evaluate all the current buzz-words and modern themes of computer science according to school relevance, (2) to find the „beef", if any, in these subjects, and (3) to teach these subjects to their students pedagogically sound.
- Fundamental ideas reduce the lag between current research findings and what is taught in schools (Vertical Criterion). This expresses the conviction of Bruner (1960) that „intellectual activity is the same anywhere, whether the person is a third grader or a research scientist."
- While fundamental ideas remain modern even in the long term (Criterion of Time), details become antiquated very early. So, computer science education based on fundamental ideas can free itself from the innovative pressure of science without the content getting out-of-date.

At the present time, computer science has no „philosophy of computer science" while established sciences have evolved such philosophies. A collection of fundamental ideas may serve as a first approach in this direction, may help to determine the essence of computer science and to dissociate it from other sciences, and may provide a useful way for thinking about teaching computer science

in the classroom.

6 REFERENCES

Bruner, J.S. (1960) *The process of education*. Harvard University Press, Cambridge MA.

Bruner, J.S. (1966) *Toward a theory of instruction*. Harvard University Press, Cambridge MA.

Halmos, P.R. (1981) Does mathematics have elements? *The Mathematical Intelligencer*, **3**, 147-153.

Heitele, D. (1975) An epistemological view on fundamental stochastic ideas. *Educational Studies in Mathematics*, **6**, 187-205.

Whitehead, A.N. (1929) *The Aims of Education*, MacMillan.

7 BIOGRAPHY

Dr. Andreas Schwill graduated in computer science from the University of Dortmund in 1983. Since then he held research assistantships at the University of Dortmund and at the University of Oldenburg. In 1990 he was awarded a doctoral degree from the University of Oldenburg. Between 1991 and 1996 he was guest professor at the University of Paderborn. Recently he has moved to the University of Potsdam to take up a professorship in didactics of computer science.