Braided Teaching in Secondary CS Education: Contexts, Continuity, and the Role of Programming

Arno Pasternak
Fritz-Steinhoff Gesamtschule Hagen and
Technische Universität Dortmund
44227 Dortmund, Germany
arno.pasternak@cs.tu-dortmund.de

Jan Vahrenhold
Technische Universität Dortmund
Faculty of Computer Science
44227 Dortmund, Germany
jan.vahrenhold@cs.tu-dortmund.de

ABSTRACT

In this paper, we propose a new approach to thinking about and implementing Computer Science curricula in secondary education. The characteristic feature is to organize the items to be taught into what we call "strands" which then can be interlaced during the course. This naturally leads to a spiral curriculum in secondary Computer Science education. In the view of our proposed approach, we also comment on the role of programming in secondary education.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education

General Terms

Theory

Keywords

Curriculum, Contexts, Continuity, Programming

1. INTRODUCTION

In response to the diminishing or simply non-existent curriculum credit for Computer Science in secondary education, educators and researchers have developed transdisciplinary or multidisciplinary approaches to introduce and teach Computer Science concepts in Science or Arts classes [14, 15, 18]. The subject matter of such a course, e.g. Bioinformatics, then provides a natural context in which Computer Science concepts can be embedded.

Recently, a multi-year effort of a large group of German educators and researchers resulted in "Educational Standards for Computer Science in Lower Secondary Education" (see [4] for an overview). These standards were modeled after the National Council of Teachers of Mathematics' standards for Mathematics [19] and describe the competence standards that should have been reached at the end of tenth grade (assuming that Computer Science is taught at least one session per week starting in fifth grade). An open research and implementation issue, however, is to define concrete teaching units that can be used to achieve these goals.

SIGCSE'10, March 10–13, 2010, Milwaukee, Wisconsin, USA.

Contexts.

The contributors to the above-mentioned German standards have suggested that teaching units should be context-based: students should be able to see how the subject matter taught in class relates to everyday life. The contributors suggest to follow the "Context-based Chemistry" approach proposed for secondary Chemistry education [21]; a closer look at these two fields shows, that Chemistry indeed is facing almost the same problems as Computer Science—see the synopsis by Gilbert [12] who discusses four different concepts of "context".

It has been reported that contextualized courses on college level lead to an increasing motivation of the students and effectiveness of the course—see, e.g., [1, 15, 25]. Unfortunately, Computer Science in secondary education can also be implemented as an Information and Communication Technology class (see [17] for a discussion of the implications thereof), and thus a purely application-based context such as word processing is not suited to foster the understanding of Computer Science in the spirit of *Computational Thinking* [29]. What is needed is a concept that subsumes the notion of "contexts" but prevents them (and thus Computer Science) from being misinterpreted as a collection of isolated applications of Information and Communication Technology.

Continuity.

One of the distinguishing features of Computer Science is the breadth of the field which encompasses subject matters close to Engineering as well as subject matters from (Discrete) Mathematics. This breadth provides a rich source of topics for secondary education and gives educators the potential of reaching out to students with widely varying interests and aptitudes. The lack of (sufficient) curriculum credits, however, has actually turned this breadth into a major impediment to implementing a curriculum that follows the "spiral" approach advocated by Bruner [5]. In his seminal work, Bruner elaborates on the hypothesis that "any subject can be taught effectively in some intellectually honest form to any child at any stage of development" [5, p. 33]. Consequently, this approach requires multiple iterations over the subject to reach an understanding at adult level. The philosophy behind Bruner's approach is widely accepted and has led to the identification of, e.g., *fundamental ideas* [24]

and - with some grain of salt - *great principles* [8] in Computer Science—see [30] for a empirical determination of such concepts present in Computer Science courses on university level. If Computer Science is taught with little curriculum credit or as an elective, there is little room for an educator to visit subject matters more than once while at the same time covering the variety of subject matters deemed "necessary" by administrators, parents, and students.

The Role of Programming.

Programming has always been an integral part of Computer Science up to the point that well-designed visual programming languages and environments such as Alice, Logo, Scratch, or Greenfoot are used as entry points to Computer Science in primary and (lower) secondary education. Also, open source software projects have been used to increase interest and improve retention at college level [16].

In the general public's view, however, Computer Science often is misidentified with programming exclusively: In a recent issue of the *Communications of the ACM*, Felleisen and Krishnamurthi observed that a "large part of the [enrollment and acceptance] problem is due to how computing is portrayed to schools, parents, the people who allocate the education budgets, and the students" [9]. Freeman's immediate rebuke [10] of their viewpoint that "programming [...] is our field's single most valuable skill" [9] (instead bringing up the ability to abstract) clearly shows the differentiation of programming and algorithmics and, in this respect, is exemplary of several non-programming based approaches to outreach and curriculum (re-)design—see, e.g., [3, 29]. In passing, we note that a study undertaken in 1999 reported on a significant decrease in the percentage of introductory Computer Science courses for *non-majors* on college level that actually taught programming [27]. Unfortunately, no data more recently acquired is available.

In recent years, the main question related to programming was not whether or not to teach programming but *when* to teach programming and when to introduce object-orientation; as an illustrative example, see [2]. Independent of which answer to the above question a particular educator prefers, he or she must judge any new suggestion for how to implement a given Computer Science curriculum also by how programming can be embedded in the resulting course.

2. STRANDS AND BRAIDED TEACHING

Definition.

The central concept of our approach is to organize the subject matters to be taught in secondary education not by topic but along what we call *strands*. This organizational concept is general enough to subsume the concept of a *context* and, as we will demonstrate, allows for a smoother linkage of different subject matters in class.

DEFINITION 1. *A* strand *is a sequence of items addressed in class that satisfies the following criteria:*

1. *The items can be assigned to a well-defined subject matter (by their structure or their content).*
2. *The subject matter is identifiable and recognizable to the students throughout the sequence.*
3. *The subject matter is being presented from more than one point of view or embedded in more than one context.*
4. *The sequence of items is addressed in more than one teaching unit.*

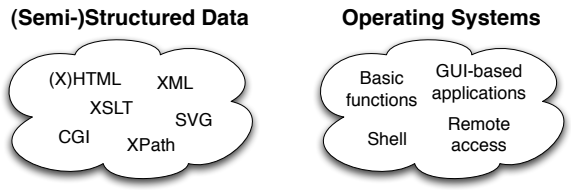

Figure 1: Strands: Sequences of items assigned to a subject matter by structure (left) or content (right).

While the first criterion is of an objective nature, the remaining three criteria depend on structural and pedagogical decisions made by the educator responsible for the course. Most notably, the above definition does not impose any particular order on the items in the sequence. As the examples given below will illustrate, the particular order of teaching the items in a strand will be subject to external factors such as curriculum design or personal preferences of the educator.

Another important aspect is that a strand is defined to appear in more than one teaching unit. While, as we will sketch below, one can construct a strand for a clearly delineated subject matter such as *operating systems*, our approach forbids this subject matter to be taught *en bloc* and thus isolated from other topics and concepts in Computer Science. At first, this may seem overly restrictive; in practice, however, it provides the educator with more leeway to present concepts from different viewpoints (or, in the terminology of Denning's *Great Principles* [8], through different windows of computing mechanics) and thus to better reach out to students with varying interests and aptitudes.

Braided teaching then aims at covering the contents in a given curriculum by a collection of strands that are interlaced wherever appropriate. Interlacing the strands and presenting the items embedded in more than one context has to be planned carefully, since using too many context switches will present a ragged, incoherent, and confusing image of Computer Science. On the other hand, findings from the "Chemistry in Context" project indicate that teaching units that stay too long within a single context lead to a rapid decline in the students' interest [23]. Furthermore, staying too long within a context may result in misidentifying the concept taught with the context and thus counteract the spiral curriculum's intended effect of enabling transfers of principles across contexts.

Examples.

As a first illustrative example, we present a strand that contains items related to *(semi-)structured data* (Figure 1, left). The structural concept behind all of these items, the hierarchical or semi-structural organization of data is easily identifiable and recognizable; in fact, it represents the fundamental idea of *structured dissection* [24]. Also, the subject matter can be visited more than once and be taught in increasing complexity from static and dynamic HTML to queries on data represented in XML. Since the strand con-

tains *scalable vector graphics* and may be extended to also address automata for validating XML documents according to a given schema, it is not restricted to web data.

The occurrence of SVG in the strand on (semi-)structured data illustrates another consequence of our approach: due to the requirement that strands must not be taught *en bloc*, the subject matters will be presented interlaced. Also, items may be assigned to more than one strand. In the above example, SVG may be a point where the strand on (semi-) structured data can be linked with a strand on multimedia.

Another (content-based) strand which has been used by one of the authors in class is organizing topics related to *operating systems* (Figure 1, right). The details of this course will be presented in Section 3.

Related Concepts.

It remains to discuss how the concept of strands relates to other organizational concepts in Computer Science education. Besides the canonical topic-by-topic sequence which can be trivially distinguished from a strand, the two most relevant concepts are *contexts* and *fundamental ideas*.

In the terminology of the Gilbert's discussion [12], *contexts* as they are used (not only) in Computer Science on college level and (higher) secondary education are mainly a "reciprocity between concepts and applications" [12, p. 967] whereas contexts in (lower) secondary education mainly relate to "topics and people's activities that are considered of importance to the lives of communities within the society" [12, p. 969].

In the terminology of our approach, a context (or a much larger project) can be defined as a sequence of items groups by content. Since, in general, the presentation of these items from multiple viewpoints and their distribution over more than one teaching unit will introduce unpleasant continuity breaks, we prefer to think of a context (in the traditional sense) as a temporally coherent frame in which items from one or more strands can be interleaved and related to applications or social circumstances.

A *fundamental idea* [5, 24] is a concept that is general enough that its mastery allows for transfers of principles (as opposed to transfers of instructions to activities). Fundamental ideas (such as *modularity* or *verification*) thus represent concepts of the underlying science and are exemplified by well-chosen items; these items, just as the items in a strand, can be arranged according to a spiral curriculum, and in this respect, the strand on (semi-)structured data has a close resemblance to exemplifications of a fundamental idea. On the other hand, a strand (for instance the strand on operating systems) may also be comprised of items that belong to the same content, and thus not every strand resembles a fundamental idea. The major difference, however, between a strand and a fundamental idea is that by design a fundamental idea organizes items exclusively from the perspective of an educator. To qualify as a strand in the sense of Definition 1, the sequence of items must be arranged and taught in a way that is identifiable and recognizable by the students as well.

The concept of strands appears to be closely related to the concept of *threads* that has been implemented, e.g., at Georgia Tech [11]. In this system, students are given a choice from a collection of sequences of courses to fulfill the curriculum requirement and to prepare for their chosen career path. The concepts are related in the sense that each strand and thread has to fulfill the requirement that the items taught have to be coherent and not restricted to a single subject matter. Furthermore, the union of threads as well as the union of strands is required to cover the respective curriculum. In contrast to the concept presented in [11], our approach does not provides means for assembling a career-specific set of courses but is a concept for educators that wish to (re-)organize items to be taught in class according to a spiral curriculum, embedded into different contexts and possibly interlaced. Due to the fact that secondary education and education on university level are organized quite differently, the final distinguishing feature is that in a thread-based setting the choice which thread to follow is made by the students, whereas the strands to be used in teaching are selected by the educators.

3. PROGRAMMING AS A STRAND?

It may seem trivial to present any current *programming* course as a strand encompassing, e.g., "Sequence", "Control Structures", "Variables", and "Object/Class". A closer look, however, at Definition 1 reveals several shortcomings of this straightforward approach: the requirement that the common subject matter be identifiable and recognizable by the students can only be fulfilled if both students and educators are willing to accept programming as a higher-ranking concept on its own. It will be hard if not impossible, to present, say, control structures from more than one point of view. Finally, most current courses known to the authors that include programming do so in a blocked fashion, either as an entry point (see Section 1), as a programming-only course, or as a hands-on teaching unit in which the (algorithmic) concepts previously taught in class are being realized in a programming language.

In these shortcomings, the following assessment of programming (in contrast to algorithmics) made by the K–12 Computer Science curriculum task force is manifested:

> "While programming is a central activity in computer science, it is only a tool that provides a window into a much richer academic and professional field." [26, p. 6]

Interlacing Programming and Strands.

Especially in (lower) secondary education, students are able experience the power of computers and software without having to program themselves. Furthermore, the extent to which the average student can realize programs is unlikely to result in assignments that can compete (regarding their attractiveness) with easy-to-access free software available on the internet. In this respect, Astrachan's Law ("Do not give an assignment that computes something that is more easily figured out without a computer." [22]) may have to be phrased even more restrictive when applied to lower secondary education.

We propose to break with the traditional habit of teaching programming *en bloc* and to treat programming as if it were just another strand where the items are embedded in more than one context. Admittedly, this still does not provide us with multiple perspectives on `if`-statements; it tremendously helps, however, with presenting programming as a tool in the sense of the above assessment.

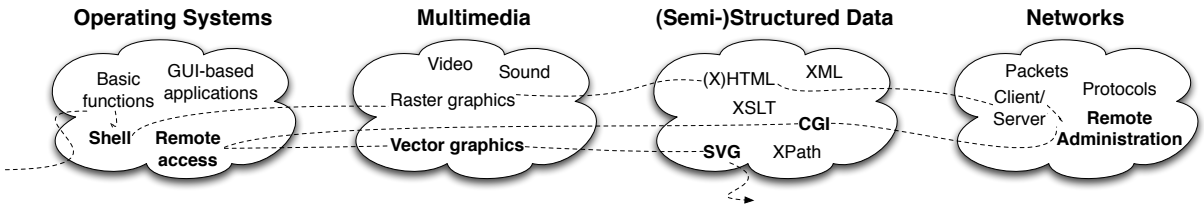

Figure 2: Braided teaching: In the (abbreviated) example depicted here, the educator has decided to visit various strands along the dashed line. Algorithmic concepts and items from "programming as a tool" are interlaced with the items in bold face.

Example.

One of the authors has designed classes for grades 9 and 10 organized according to four major strands: *operating systems*, *multimedia*, *(semi-)structured data*, and *networks* (see Figure 2; note, however, that the collection of items depicted is reduced for ease of presentation and thus not comprehensive. Most notable, the interlaced strand on *algorithmics* is left out to unclutter the picture). In an introductory phase, students learn how to use shell commands for working with files and directories as well as for an exemplary installation of a small software package. Elementary programming concepts are introduced as means of automating certain tasks in shell scripts. After a short switch to the *multimedia* strand, a first visit is paid to *(semi-)structured data* strand where students experience (X)HTML not only as means for encoding web pages but also for representing structured data. The following teaching units obviously are located in the *networks* strand.

In the next phase of the course, the students revisit the *(semi-)structured* strand and learn how to use CGI for structured documents automatically and with dynamically varying content. These scripts then must be uploaded to servers. Coming full circle to one of the initial parts of the course, vector graphics and their structured representation using the SVG format are discussed and used for again interlacing with "programming as a tool".

4. IMPLICATIONS

An intended side-effect of interlacing programming with strands is that programming is put on the same level of importance as all other strands and thus is no more (but also no less) important than any specific strand. Consequently, the question of which programming language to use (or whether and when to introduce objects) is no more important than which other strands to teach. If an educator wishes to do so, he or she can actually choose a programming language such that it supports the contexts in which items are embedded and not vice versa.

Braided Teaching Using Tcl/Tk.

To illustrate the flexibility stated above, we point out that an approach to braided teaching could be based, e.g., on the Tcl/Tk programming language [20]. This particular choice is not due to features of this language *per se* (but see Warren [28] advocating the use of scripting languages) but motivated by the fact that Tcl/Tk seamlessly integrates with a shell. In the context of braided teaching, this is a major advantage over other languages or microworld-based environments, since the student do not regard programming as switching to a different environment. Furthermore, the Tcl/Tk shell allows for an interpreter-based approach to Logo-like graphics, thus providing the agreed-upon advantages [6] of having direct interaction and visual feedback—see also [13] for comments on how to reach out to students with Tcl/Tk. We note in passing that even the answer to whether and when to introduce object-orientation is not preempted by our selection of the programming language.

As mentioned above, the collection of strands reflects a pedagogical and sometimes personalized decision of the educator (for example, to start from an operating systems strand), and this is especially true for the choice of the programming language. One could just as well imagine a collection of strands that naturally leads to a completely different selection of a programming language. To some extent, the development of Alice [7] can also be seen as the selection of a programming language based upon the requirements of a *virtual worlds* context.

5. CONCLUSIONS

In this paper, we have advocated what we call *braided teaching*: a new way of thinking about the organization of items when implementing a Computer Science curriculum in secondary education. This approach relies on grouping the items along strands, and its main conceptual advantage is that it allows for an easier implementation of a spiral curriculum while at the same time giving the educator more leeway with didactical decisions. We have exemplified our approach and observed that it allows for a more seamless integration of programming giving it the role delineated in the ACM K–12 Computer Science curriculum.

Our approach as formulated in this paper pertains to (lower) secondary education, and thus two directions for future research present themselves: The obvious task is to demonstrate both in theory and in practice that braided teaching can be instantiated with a reasonable collection of strands and that the effects expected by following the spiral curriculum can be observed as well. The first author's experience with teaching along single strands leads us to believe so. Also, it remains to investigate how this concept can be extended to propaedeutic pre-college or freshman courses and how it might be combined with existing organization concepts on college level.

6. REFERENCES

[1] C. Alt, O. Astrachan, J. Forbes, R. Lucic, and S. Rodger. Social networks generate interest in computer science. In D. Baldwin, P. Tymann, S. Haller, and I. Russel, editors, *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, pages 438–442, 2006.

[2] O. Astrachan, K. Bruce, E. Koffman, M. Kölling, and S. Reges. Resolved: Objects Early has failed. In W. Dann, T. Naps, P. Tymann, D. Baldwin, and J. D. Dougherty, editors, *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, pages 451–452, 2005.

[3] T. Bell, I. H. Witten, and M. Fellows. Computer Science Unplugged: An enrichment and extension programme for primary-aged children. `http://www.google.com/educators/activities/unpluggedTeachersDec2006.pdf`, 2002.

[4] T. Brinda, H. Puhlmann, and C. Schulte. Bridging ICT and CS: Educational standards for computer science in lower secondary education. In *ITiCSE '09: Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*, pages 288–292, 2009.

[5] J. S. Bruner. *The Process of Education*. Harvard University Press, Cambridge, MA, 1960.

[6] D. H. Clements and J. S. Meredith. Research on Logo: Effects and efficacy. *Journal of Computing in Childhood Education*, 4:263–290, 1993.

[7] W. P. Dann, S. Cooper, and R. Pausch. *Learning to Program with Alice*. Pearson Education, London, UK, second edition, 2009.

[8] P. J. Denning. Great principles of computing. *Communications of the ACM*, 46(11):15–20, Nov. 2003.

[9] M. Felleisen and S. Krishnamurthi. Viewpoint: Why computer science doesn't matter. *Communications of the ACM*, 52(7):37–40, July 2009.

[10] P. A. Freeman. Letter to the editor: Computer science *Does* matter. *Communications of the ACM*, 52(9):8, Sept. 2009.

[11] M. Furst, C. Isbell, and M. Guzdial. Threads^{TM}: How to restructure a computer science curriculum for a flat world. In I. Russell, S. Haller, J. D. Dougherty, S. Rodger, and G. Lewandowski, editors, *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, pages 420–424, 2007.

[12] J. K. Gilbert. On the nature of "context" in chemical education. *International Journal of Science Education*, 28(9):957–976, July 2006.

[13] S. Graham and C. Latulipe. CS girls rock: Sparking interest in computer science and debunking the stereotypes. In S. Grissom, D. Knox, and D. Joyce, editors, *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education*, pages 322–326, 2003.

[14] S. Hambrusch, C. Hoffmann, J. T. Korb, M. Haugan, and A. L. Hosking. A multidisciplinary approach towards computational thinking for science majors. In S. Fitzgerald, M. Guzdial, G. Lewandowski, S. Wolfman, and T. J. Cortina, editors, *Proceedings of the 40th SIGCSE Technical Symposium on Computer Science Education*, pages 183–187, 2009.

[15] M. Hart, J. P. Early, and D. Brylow. A novel approach to K-12 CS education: Linking mathematics and computer science. In S. Rodger, J. Dougherty, M. Guzdial, S. Fitzgerald, and E. Walker, editors, *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, pages 286–290, 2008.

[16] G. W. Hislop, H. J. C. Ellis, A. B. Tucker, and S. Dexter. Panel: Using open source software to engage students in computer science education. In S. Fitzgerald, M. Guzdial, G. Lewandowski, S. Wolfman, and T. J. Cortina, editors, *Proceedings of the 40th SIGCSE Technical Symposium on Computer Science Education*, pages 134–135, 2009.

[17] J. Hromkovič. Contributing to general education by teaching informatics. In R. Mittermeir, editor, *Proceedings of the Second International Conference in Informatics in Secondary Schools - Evolution and Perspectives*, volume 4226 of *Lecture Notes in Computer Science*, pages 25–37. Springer, 2006.

[18] C.-C. Lin, M. Zhang, B. Beck, and G. Olsen. Embedding computer science concepts in K-12 curricula. In S. Fitzgerald, M. Guzdial, G. Lewandowski, S. Wolfman, and T. J. Cortina, editors, *Proceedings of the 40th SIGCSE Technical Symposium on Computer Science Education*, pages 539–543, 2009.

[19] National Council of Teachers of Mathematics. *Principles and Standards for School Mathematics*. National Council of Teachers of Mathematics, 2000.

[20] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, MA, 1994.

[21] I. Parchmann, C. Gräsel, A. Baer, P. Nentwig, R. Demuth, B. Ralle, and the ChiK Project Group. "Chemie im Kontext": A symbiotic implementation of a context-based teaching and learning approach. *International Journal of Science Education*, 28(9):1041–1062, July 2006.

[22] N. Parlante. Astrachan's law. *Inroads – the SIGCSE Bulletin*, 35(4):26–27, Dec. 2003.

[23] B. Ralle, D.-S. Di Fuccia, and W. Schwarz. Lehrer planen gemeinsam - Ein Einblick in die Arbeit des Projektes Chemie im Kontext. *Der mathematische und naturwissenschaftliche Unterricht*, 58(7):388–393, 2005. In German.

[24] A. Schwill. Fundamental ideas – rethinking computer science education. *Learning and Leading with Technology*, 25(1):28–31, Sept. 1997.

[25] A. E. Tew, B. Dorn, W. D. Leahy, Jr., and M. Guzdial. Context as support for learning computer organization. *ACM Journal on Educational Resources in Computing*, 8(3), Oct. 2008. Article 8, 18 Pages.

[26] A. Tucker, F. Deek, J. Jones, D. McCowan, C. Stephenson, and A. Verno. A model curriculum for K-12 computer science: Final report of the ACM K-12 task force curriculum committee, 2004.

[27] M. Urban-Lurain and D. J. Weinshank. Is there a role for programming in non-major computer science courses? In *Proceedings of the 30th Annual ASEE/IEEE Conference on Frontiers in Education*, pages T2B–7–T2B–11, 2000.

[28] P. Warren. Teaching programming using scripting languages. *Journal of Computing Sciences in Colleges*, 17(2):205–216, 2001.

[29] J. M. Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, Mar. 2006.

[30] A. Zendler and C. Spannagel. Empirical foundations of central concepts for computer science education. *ACM Journal on Educational Resources in Computing*, 8(2), June 2008. Article 6, 15 Pages.