Computational Thinking and Practice — A Generic Approach to Computing in Danish High Schools

Michael E. Caspersen and Palle Nowack
Centre for Science Education, Faculty of Science and Technology
Aarhus University
DK-8000 Aarhus, Denmark

{mec, nowack}@cse.au.dk

Abstract

Internationally, there is a growing awareness on the necessity of providing relevant computing education in schools, particularly high schools. We present a new and generic approach to Computing in Danish High Schools based on a conceptual framework derived from ideas related to computational thinking. We present two main theses on which the subject is based, and we present the included knowledge areas and didactical design principles. Finally we summarize the status and future plans for the subject and related development projects.

Keywords: curriculum structure, course content, high school, computational thinking, core competencies, application areas, knowledge areas, learning activities, didactical design principles.

1 Introduction

Computing, particularly in the specific form of computer science, has been a topic in high schools in many countries for more than three decades, but without achieving the break-through in terms of adoption that the topic deserves in the post-industrial society.

But things are changing, and they are changing at a global scale. Internationally, there is a growing awareness on the necessity of providing relevant computing education in schools, particularly high schools. Computing education in schools is considered increasingly important as expressed by e.g. Wing (2006) who argues for teaching fundamental computing principles for all: "Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability". In the book *Program or be Programmed*, Rushkoff (2010) puts it even more bluntly: "In the emerging, highly programmed landscape ahead, you will either create the software or you will be the software".

Half a century ago, Perlis (1962) said that everyone should learn to program as part of a liberal education. He argued that programming was an exploration of process, a topic that concerned everyone, and that the automated execution of process by machine was going to change everything (Guzdial 2008). It took fifty years to get here, but finally it seems that (a contemporary interpretation of) Perlis' vision has come to pass.

As mentioned by Cutts, Esper, and Simon (2011), several national initiatives are being taken to address this challenge. For example, the UK Royal Society has recently published the report *Computing in School* (Royal Society 2012), and the US National Science Foundation and the College Board are supporting development of an Advanced Placement course, CS Principles (Astrachan et al. 2012), aiming at broadening participation in computing and computer science by transforming high school computing (Astrachan et al. 2011). Similar initiatives are taken in other countries, e.g. Israel (Gal-Ezer and Harel 1998 and 1999, Bargury 2012), Germany (Steer and Hubwieser 2010), The Netherlands (Van Diepen et al. 2011), and Norway (Hadjerrouit 2009). Especially the effort in New Zealand seems to be similar with respect to motivation, and challenges, but perhaps not with respect to the content and form (Bell et al. 2010, Bell et al. 2012).

In this paper, we report on a recent Danish initiative to redefine and revitalise computing in Danish high schools. The Danish initiative is similar to many of the other initiatives in focusing on fundamental computing principles (including computational thinking) as a fundamental skill for all. However, the Danish initiative is different from most of the other initiatives in taking a broader and generic approach to computing rather than the traditional and narrower computer science or software engineering approach. This is a deliberate choice made primarily to embrace more fundamental aspects of computing (e.g. impact of information systems, the role of it in innovation, and interaction design for it-based systems), but also to accommodate the four different types of high schools in Denmark (general high schools, upper secondary shorter general education programme, technical high schools, and business high schools) with one generic computing subject.

In section two we briefly recap the history of computing in Danish High School curricula. Section 3 describes the two main theses that together define the perspective from which the new generic computing subject was designed. The subject is then fleshed out in the following two sections: Section 4 describes the knowledge areas of the subject, and Section 5 describes the didactical design principles behind the subject. Finally, Section 6 briefly summarizes the current status and plans for the subject.

2 Computing in Danish High School 1971-2011

Various flavours of computing has been a topic in Danish high school for more than forty years.

After some early individual initiatives in the late sixties with computing education in high schools, the *Johnsen Committee* was formed in 1971 to give recommendations regarding EDP education (Electronic Data Processing) in the Danish education system (Johnsen 1972). The recommendations of the Johnsen committee guided the computing curriculum decisions in general high school for more than ten years. However, the full set of recommendations —encompassing a mandatory computing subject for all high school students— were never implemented.

In 1980, the Ministry of Education published the so-called *Obel/Fisher Circular* recommending computing in general high school to be integrated in other subjects and phased-out as an independent subject. In the 1980s, computing remained an independent subject only in one branch (out of four) of Danish high schools. In 1987, computing became again an independent subject, but still only as an elective, and it has remained as such until today.

In business high schools, computing has been a subject since the mid-1980s —always with a special flavour of business, management, and administration.

From the mid-1990s and onward, other computing subjects saw the light of day in the different types of high schools, e.g. Information Technology, Programming, and Multimedia.

A major high school reform in 2005 dramatically reduced the conditions for elective subjects such as computing, and the same pattern emerged in all types of high schools: hardly any pupils chose computing and the subject almost completely vanished from the schools.

In late 2008, the Ministry of Education established a task force to conduct an analysis of computing in high schools and provide recommendations for a revitalisation of the subject (Agesen and Nørgaard 2009). The major recommendations of the task force were:

- To distinguish between computer literacy (emphasizing it-usage, e.g. the use of spreadsheets, word processing, and other applications) and computational thinking and practice (emphasizing creational and constructional competencies).
- To develop a single, coherent, and uniform computational thinking and practice subject, which then can be offered in several flavours.
- To design the course such that it may inspire pupils to continue with computing studies after high school.

The recommendations gained political support at all levels, and a new generic computing subject has been developed and is offered by volunteering schools for a three-year test period (2011-2014).

3 Foundational Theses

In general, young people do not consider computing a proper subject, and they certainly do not realise the importance and potential of computing in modern society. The main purpose of the new computing subject for high school is to convey the message condensed in the first of two foundational theses:

Thesis 1: *Through computing, people can create, share, and handle thoughts, processes, products and services that create new, effective, and boarder-crossing opportunities -impossible without the digital technology.*

The wording is a bit heavy, but the essence is quite similar to Wing's notion of computational thinking. Thesis 1 is the keynote of the new computing subject; as such, it must permeate all concrete learning activities that will be developed.

The second thesis relates to our ambition of embracing more fundamental aspects of computing but also to accommodate the four different types of high schools in Denmark with one generic computing subject. The thesis also reflects the diversity and various flavours of computing in academia, education, and industry.

Thesis 2: *There exists a common and shared foundational set of computational concepts, principles and practices, which can be applied purposefully within science & technology, business and social science, arts and humanities, and health and life sciences.*

Both theses were formulated before we commenced concrete development of the new computing subject. Throughout development, the theses served as guiding principles for our efforts of refinement and concrete design of the subject. In particular, thesis 2 provided guidelines for identification of seven core knowledge areas that has come to define the new computing subject. The seven knowledge areas are presented in the following section.

4 Knowledge Areas

We use the term "Knowledge Areas" in the same sense as in the curriculum recommendations from ACM[1]: the areas are not to be thought of as *teachable* modules by themselves, but as appropriate categories for *describing* subject content. Hence, the categories are for description, and not didactical design of practical learning activities. We expand on these issues in Section 5.

In the following, we motivate and describe the seven knowledge areas that have been chosen for characterising the new computing subject and for formulating learning goals. The areas have been chosen after a short and intensive dialogue with selected colleagues from Danish universities. In retrospect, some of the areas are related to the computing practices suggested by (Denning 2003).

The knowledge areas are:

- Importance and Impact
- Application Architecture
- Digitisation
- Programming and Programmability
- Abstraction and Modelling
- Interaction Design
- Innovation

For each area, we provide a brief description and present the associated learning goals, as they appear in the formal curriculum. It should be noted, that the learning goals may appear overly ambitious, but they must of course be interpreted in the context of level, preconceptions, and allocated time for the actual course delivery.

[1]http://www.acm.org/education/curricula-

4.1 Importance and Impact

To truly understand and appreciate the importance of computing in modern society, the pupils must be presented to a portfolio of important and for the pupils relevant systems and innovations (e.g. Facebook, iTunes, GPS-based navigation systems, email, health care systems, etc.) — systems that the pupils know and can relate to. The design of an IT system has strong consequences for the people, organisations, and social systems that use it. Designers do not only design the system but also use patterns and workflows that unfold through the use of the system. The purpose is to make the pupils aware of the interplay between design of a system and the use patterns which the system intentionally or unintentionally generates.

Pupils must be able to

- Give examples of the impact of IT systems on human behaviour.
- Analyse and assess the importance and implications of IT systems and how they impact human behaviour.
- Apply user-oriented techniques for construction or modification of IT systems.

4.2 Application Architecture

The majority of IT systems are structured according to the so-called three-tier model consisting of a presentation tier, a logic tier, and a data tier. The model is relevant partly because it provides a general framework for understanding a very large class of IT systems, their components, and the interplay between these, and partly because the model is useful for qualified use of concrete systems, e.g. the Office package, Photoshop, iTunes, Facebook and general types of systems, e.g. simulation tools, accounting systems, content management systems, mobile technology, and computer games.

Pupils must be able to

- Describe principles for the architecture of IT systems.
- Apply specific architectures for construction of simple IT products and adjustment of existing IT systems.

4.3 Digitisation

In order to understand the basic characteristics of the computer, the pupils must understand and work with representation and manipulation of data. The main point is that data need to be digitised to allow representation in a computer and manipulation by programs. The purpose with this topic is that the pupils gain concrete experience with (and hence understanding of) representation and manipulation of data including the fact that digitising often results in loss of information. The other side of the coin is that digitisation and manipulation makes it possible to create new data. IT security is another important issue that may be addressed.

Pupils must be able to

- Describe the representation of selected types of data (e.g. images, sound, text, etc.) and construct IT products (programs) that make simple manipulations of data.
- Integrate various types of data in simple IT products and extend functionality of existing IT systems by adding new types of data.

4.4 Programming and Programmability

Computers are indeed very simple machines that gain their power through scale. The defining characteristic of the computer is its programmability and universality. Programming comes in many forms, but common to these is the principle of defining and hence automating computations that can be executed again and again with arbitrary data and data sets.

Pupils must be able to

- Identify basic structures in programming languages, construct IT products (simple programs) and adjust existing programs.
- Apply programming technologies for development of IT products and adjustment of existing IT systems.

4.5 Abstraction and Modelling

The purpose of this topic is to provide insight into modelling where data, processes and systems are described at an abstract level where design alternatives and properties can be evaluated and choices and decisions can be made.

Pupils must be able to

- Give examples of models of data, processes and systems and describe the relation between a concrete model and the relevant associated parts of an IT system.
- Implement selected models in a concrete IT product and adjust existing models and implement these adjustments in existing IT systems.

4.6 Interaction Design

The previous topic is primarily about models for elements of the presentation and logic tiers of the three-tier model. This topic is about models and design principles for the presentation tier — the interface where users and other systems meet an IT system. It's the purpose that the pupils understand the premises for as well as the consequences and importance of interaction design.

Pupils must be able to

- Describe and analyse selected elements of a user interface design, construct simple user interface designs and adjust existing designs.
- Implement selected interaction design in a concrete IT product and adjust existing designs and implement these adjustments in existing IT systems.

4.7 Innovation

The subject treats innovation from a product as well as process perspective. The subject takes an innovative approach to IT product development and provides a background for understanding aspects of IT product development and the interplay between IT and users/society.

Pupils must be able to:

- Characterise innovative development processes and sketch ideas for innovative IT products.

5 Didactical Design Principles

A number of didactical design principles and guidelines have been enforced, or at least highly recommended, for the development of learning materials for the new computing subject. In this section, we present the five major didactical principles:

- A learning activity is not (necessarily) the same as a knowledge area.
- Learning activities should be application-oriented.
- Learning activities should facilitate and guide a consume-before-produce progression through the materials.
- Learning activities should include several substantial worked examples.
- Learning activities should illustrate stepwise improvement as a general approach to incremental development of artefacts.

Other principles have been used such as game-based learning and narrative media-approaches (e.g. Andersen et al. 2003).

For a more general discussion of didactical approaches to computing, see Bennedsen et al. (2008) and Hazzan et al. (2011).

5.1 Knowledge Areas vs. Learning Activities

The knowledge areas introduced in Section 4 helps to structure the entire curriculum, but it is not a feasible structure for teaching the subject, as it would imply a sequential depth-first approach to the subject as a whole.

Instead we have adopted a well-known teaching strategy from Danish high schools, in which subject matter from various different knowledge areas are extracted and combined to piecemeal construct and deliver smaller packages of contextualised and interdependent subject matter components. These learning activities form the toolbox, from which the teacher select, combine, design, and implement his/her particular version of the subject which should be adapted and adjusted to the relevant context (education, level, and individual pupils). A learning activity may include subject matters from one, multiple, or all of the seven knowledge areas as illustrated in Figure 1. A learning activity is comprised by a description for pupils and teachers, materials and resources, and a process (cookbook) for using the materials in the learning activity.

The latter also illustrates a characteristic difference between knowledge areas and learning activities: the former are more static and are expected to change at a much slower pace than the learning activities, which are expected to change rapidly over the years, as technology and trends changes. Put another way: when the knowledge areas change, the whole identity of the subject changes (ranging from minor adjustments to radical changes in conceptual frameworks). Furthermore, changes in learning activities could be made for purely pedagogical reasons.

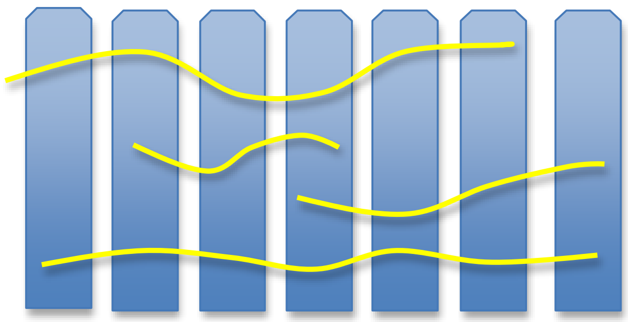
Figure 1: Content Structure Framework: Knowledge Areas (blue columns) versus Learning Activities (yellow lines)

5.2 Application-oriented (outside-in)

Traditionally, introductory computer science courses apply a bottom-up approach, in the sense that pupils are introduced to basic and foundational concepts and expected to master these before more advanced concepts and principles are introduced. Hence, in a traditional programming course, pupils are often trained in constructing a "Hello World" program as the very first activity, and then later on are trained in adding more layers of complexity to a system in terms of user interfaces, databases, etc. For the technically inclined pupils this may be a feasible approach, but in our case, this could pose severe motivational problems, as we are dealing with a wider range of pupils with much more diverse interests and backgrounds.

There is an even more important reason why a traditional bottom-up approach is fallible. We are not aiming at developing detailed and specific competences in the seven knowledge areas. Overall, we are aiming at developing interest, critical thinking, and broader skills in computational thinking and practice. Therefore we have decided on an application-oriented top-down approach. This means, that we start the various teaching activities by introducing well-known or familiar applications, which we then split apart for conceptual and/or technical examination, evaluation, and modification. For motivational reasons, we choose applications based on the criteria, that they must by themselves be naturally appealing to pupils in our age range. Applications, which they find interesting to use and hopefully to examine and improve. Examples could include pedagogical lightweight versions of Facebook, iTunes/Spotify, YouTube, Twitter, Blogs, Photoshop, and similar applications.

5.3 From Consumer to Producer

When designing learning activities, we aim at organising the material in such a way that the pupils experience a *consume-before-produce* progression through the material. Initially, the pupils act as consumers of an artefact by using and studying it; then, they go on to make first simple and then gradually more complex modifications to the artefact. Eventually, the pupils may be requested to build similar artefacts from scratch.

The *consume-before-produce* principle —sometimes alternatively characterised as a *use-modify-create* progression— can be applied in many areas. In programming, pupils can use programs or program modules before they start making modifications and eventually create modules or complete programs on their own. The approach applies equally well to other areas, e.g. modelling and interaction design.

The origin of (a specialisation of) this principle can be traced back at least to 1990 where Pattis introduced the *call-before-write* approach to teaching introductory programming (Pattis 1990). In Christensen and Caspersen (2002), the authors apply the principle to provide an alternative and incremental way of teaching about software frameworks and event-driven programming in CS1. In Schmolitzky (2005), the author briefly mentions the notion of consuming before producing by providing three specific examples of using the principle in the context of learning object-oriented programming using the BlueJ system (Kölling 2003).

5.4 Worked Examples

A Worked Example (WE), consisting of a problem statement and a procedure for solving the problem, is an instructional device that provides a problem solution for a learner to study (Atkinson et al. 2000, Chi et al. 1989, LeFevre and Dixon 1986). WEs are meant to illustrate how similar problems might be solved, and WEs are effective instructional tools in many programs, including computing.

Bennedsen and Caspersen (2004) illustrate implicitly how WEs can be used to teach object-oriented programming using a systematic, model-based programming process. Caspersen & Bennedsen (2007) present an instructional design for an introductory programming course based on thorough use of WE. Caspersen (2007) provides an overview of WE literature related to programming education as well as a survey of the related cognitive load theory.

Through didactical training of teachers and systematic enforcement, WE have come to play a key role in the didactical design of most learning activities developed for the new computing subject. A multitude of examples are available from a website maintained by the Danish Association of High School Teachers in Computing[2]. Unfortunately, the material is only available in Danish.

5.5 Stepwise Improvement

The Danish Ministry of Education's official guidelines for the new computing subject recommend that all constructional activities be designed according to *Stepwise Improvement*. In its original form, stepwise improvement (not to be mixed with stepwise refinement although the two are somewhat related) is presented in the context of program development (Caspersen 2007, Caspersen and Kölling 2009), but the methodology is applicable for the construction of any concrete or abstract artefact.

Stepwise improvement is a framework for incremental development of an artefact. According to stepwise improvement, development takes place in three dimensions: from abstract to concrete, from partial to complete, and from unstructured to structured. Thus, development of an artefact can be characterised as a mixed sequence of refinements, extensions, and restructurings of the artefact.

[2] http://www.iftek.dk

For the new computing subject, the recommendation from the Ministry of Education is that stepwise improvement is used systematically in all constructive learning activities. A number of concrete examples as well as more general guidelines are provided in eight reports published by the Danish Ministry of Education (2011).

6 Summary, Status & Plans

In this paper we have described the international context and the national history, which together form the background for a radically new and integral computing subject in Danish high schools. The new subject has been described in terms of two foundational theses, seven knowledge areas, and five didactical design principles.

6.1 Status

The status of the subject is that after the first year of the test period (2011-2012), 18% of the high schools taught the new subject. In the second (2012-2013, current) year of the test period, at least 26% of the high schools are teaching the new subject. Although no formal quantitative evaluation has yet been conducted, the informal feedback from teachers, examiners and pupils has been very positive.

As mentioned, the Danish Association of High School Teachers in Computing offers a number of learning activity packages on their website. Teachers are encouraged to develop and share their own learning activity packages. This bottom-up approach to material development of course encourage diversity and multiplicity, which challenges the content structure framework, and the conceptual framework, understanding and application of knowledge areas.

To reinforce the common understanding of the knowledge areas, a number of short reports have been developed by academics from Danish universities. Furthermore teacher training has been initiated in an ad-hoc fashion, offering 3 days of seminars during the winter of 2012, and again in the fall of 2012, where teachers are instructed in the use of the learning activity packages. Teachers from roughly 20% of all high schools attend these courses. While these ad-hoc seminars are necessary means in the process of developing the new subject, they are far from sufficient for fulfilling the requirements for in-service training of teachers to qualify them for teaching the new subject.

6.2 Plans

The plans for the continued development of the subject are fourfold:

- To further develop materials and resources
- To develop formal teacher training
- To establish professional learning communities
- To initiate relevant research
- To gain political interest and momentum

With respect to materials, we want to further iterate, increment and refine the content structure framework and the associated learning materials (both the knowledge area reports and the learning activity packages). A possible next step could be to develop free online materials supporting inversion of the classroom.

With respect to teacher training, we need to replace the current ad-hoc approach with a regular 120 ECTS education in computational thinking and practice to be offered to high school teachers – both pre-service and in-service.

We would like to support the former initiatives by further evolving the current formal and informal networks among high school computing teachers into professional learning communities based on action learning.

The new Danish initiative is an excellent opportunity for (and it deserves) a thorough treatment in terms of a number of related research projects. For example, we would like to investigate the following research questions:

- Why is computational thinking and computing practice generally and universally important to society and the individual?
- What are the relevant didactical design principles for the new subject?
- What is the ideal selection of knowledge areas for the new subject, and how do they compare to similar efforts internationally?
- How can we develop methodological and technological support for developing motivating and efficient learning activities that properly exploits the chosen didactical design principles?
- How can we develop efficient teacher training for the new subject?

Finally we find it of utmost importance, that we obtain political awareness about the importance of the subject, as it should not be an elective, but an integral, mandatory part of any high school education. A possible next step in this direction could be to host a conference on the importance of the subject.

6.3 Acknowledgments

This work was supported in part by Central Denmark Region and was conducted as part of the project Create IT which includes the partners: it-vest – networking universities, the Danish High School Computing Teachers Association (IFTEK), Egaa Gymnasium, and Centre for Science Education at Aarhus University.

The authors would like to thank Elisabeth Husum, Jakob Stenløkke Bendtsen, Bartlomiej Rohard Warszawski, Henning Agesen, and Peter Nørgaard for contributions and inspiring discussions. We would also like to thank the anonymous reviewers for constructive and relevant feedback.

7 References

Agesen, H. and Nørgaard, P. (2009): *Investigation of Computing Subjects in High School* (in Danish: *Undersøgelse af IT fagudbuddet I de gymnasiale uddannelser*), Department for High Schools, Ministry of Education, Denmark.

Andersen, P.B., Bennedsen, J., Brandorff, S., Caspersen, M.E., and Mosegaard, J. (2003): Teaching Programming to Liberal Arts Students — A Narrative Media Approach. *Proc. of the Conference on Innovation and Technology in Computer Science Education*, Thessalonica, Greece, **8**:109-113, ACM Press.

Astrachan, O., Cuny, J., Stephenson, C., and Wilson, C. (2011): The CS10K Project: Mobilizing the Community to Transform High School Computing. *Proc. of the 42nd ACM Technical Symposium on Computer Science Education*, Dallas, TX, USA, **42**:85-86, ACM Press.

Astrachan, O., Briggs, A., Cuny, J., Diaz, L., and Stephenson, C. (2012): Update on the CS Principles Project. *Proc. of the 43rd ACM Technical Symposium on Computer Science Education*, Raleigh, NC, USA, **43**:477-478, ACM Press.

Atkinson, R.K., Derry, S.J., Renkl, A., and Wortham, D. (2000): Learning from Examples: Instructional Principles from the Worked Examples Research, *Review of Educational Research*, **70**(2):181-214.

Bargury, I.Z. (2012): A New Curriculum for Junior-High in Computer Science. *Proc. of the Conference on Innovation and Technology in Computer Science Education*, Haifa, Israel, **17**:204-208, ACM Press.

Bell, T., Andreae, P., & Lambert, L. (2010). Computer Science in New Zealand High Schools. presented at the meeting of the Twelfth Australasian Computing Education Conference (ACE 2010), Brisbane, Australia.

Bell, T., Andreae, P., & Robins, A. (2012). Computer science in NZ high schools: the first year of the new standards. presented at the meeting of the 43rd ACM technical symposium on Computer Science Education, Raleigh, North Carolina, USA.

Bennedsen, J. and Caspersen, M.E. (2004): Teaching Object-Oriented Programming – Towards Teaching a Systematic Programming Process. *Proc. of the Eighth Workshop on Pedagogies and Tools for the Teaching and Learning of Object-Oriented Concepts*, 18th European Conference on Object-Oriented Programming (ECOOP 2004), Oslo, Norway.

Bennedsen, J., Caspersen, M.E., and Kölling, M. (2008): *Reflections on the Teaching of Programming*, Lecture Notes in Computer Science, Vol. 4821, Springer-Verlag.

Caspersen, M.E. (2007): *Educating Novices in the Skills of Programming*, DAIMI PhD Dissertation PD-07-04, ISSN 1602-0448 (paper), 1602-0456 (online).

Caspersen, M.E. and Bennedsen, J. (2007): Instructional Design of a Programming Course: A Learning Theoretic Approach. *Proc. of the International Computing Education Research Workshop*, Atlanta, Georgia, USA, **3**:111-122, ACM Press.

Caspersen, M.E. and Kölling, M, (2009): STREAM: A First Programming Process, *ACM Transactions on Computing Education*, **9**(1):4.1-4.29.

Christensen, H.B. and Caspersen, M.E. (2002): Frameworks in CS1: a Different Way of Introducing Event-Driven Programming. *Proc. of the Conference on Innovation and Technology in Computer Science Education*. Aarhus, Denmark, **7**:75-79.

Chi, M.T.H., Bassok, M., Lewis, M.W., Reimann, P., and Glaser, R. (1989): Self-explanations: How students study and use examples in learning to solve problems, *Cognitive Science*, **13**(2):145-182.

Cutts, Q., Esper, S., and Simon, B. (2011): Computing as the 4th "R". *Proc. of the International Computing Education Research Workshop*, Providence, RI, USA, **7**:133-138, ACM Press.

Danish Ministry of Education (2011): Information Technology B and C, *Eight Reports With Guidelines for Information Technology B and C at stx, hf, htx, and hhx*, Department of High Schools, Ministry of Education. http://www.uvm.dk/Uddannelser-og-dagtilbud/Gymnasiale-uddannelser/Studieretninger-og-fag/Forsoegsfag-i-de-gymnasiale-uddannelser/Informationsteknologi-C-og-B (in Danish, accessed 24th August 2012).

Denning, P. J. (2003): Great principles of computing. *Communications of the ACM*, **46**(11):15-20.

Gal-Ezer, J. and Harel, D. (1998): What (Else) Should CS Educators Know?, *Communications of the ACM*, **41**(9):77-84.

Gal-Ezer, J. and Harel, D. (1999): Curriculum and Course Syllabi for a High-School Program in Computer Science, *Computer Science Education*, **9**(2):114-147.

Guzdial, M. (2008): Paving the Way for Computational Thinking, *Communications of the ACM*, **51**(8):25-27.

Hadjerrouit, S. (2009): Teaching and Learning School Informatics: A Concept-Based Pedagogical Approach, *Informatics in Education*, **8**(2):227-250.

Hazzan, O., Lapidot, T., and Ragonis, N. (2011): *Guide to Teaching Computer Science: An Activity-Based Approach*, Springer-Verlag.

Kölling, M., Quig, B., Patterson, A., and Rosenberg, J. (2003): The BlueJ system and its pedagogy, *Computer Science Education*, **13**(4):249-268.

LeFevre, J.-A. and Dixon, P. (1986): Do Written Instruction Need Examples?, *Cognition and Instruction*, **3**(1):1-30.

Pattis, R.E. (1990): A philosophy and example of CS-1 programming projects. *Proc. of the 21st ACM Technical Symposium on Computer Science Education*, Washington D.C., USA, **21**:34-39, ACM Press.

Perlis, A. (1962): The computer in the university. In *Computers and the World of the Future*, 180-219. Greenberger, M. (ed.). MIT Press.

Royal Society (2012): *Shut down or restart? The way forward for computing in UK schools*. The Royal Society, UK.

Rushkoff, D. (2010): *Program or Be Programmed – Ten Commands for a Digital Age*. New York, OR Books.

Schmolitzky, A. (2005): Towards Complexity Levels of Object Systems Used in Software Engineering Education. *Proc. of the Ninth Workshop on Pedagogies and Tools for the Teaching and Learning of Object-Oriented Concepts*, 19th European Conference on Object-Oriented Programming (ECOOP 2005). Glasgow, UK.

Steer, C. and Hubwieser, P. (2010): Comparing the Efficiency of Different Approaches to Teach Informatics at Secondary Schools, *Informatics in Education*, **9**(2):239-247.

Van Diepen, N., Perrenet, J., and Zwaneveld, B. (2011): Which Way with Informatics in High Schools in the Netherlands? The Dutch Dilemma, *Informatics in Education*, **10**(1):123-148.

Wing, J. (2006): Computational Thinking, *Communications of the ACM*, **49**(3):33-35.