ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/247927531

Block Model: an educational model of program comprehension as a tool for a scholarly approach to teaching

Article · September 2008

DOI: 10.1145/1404520.1404535

CITATIONS
54

READS
1,725

1 author:

Carsten Schulte
Universität Paderborn
102 PUBLICATIONS **1,384** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Gaze in Programming View project

Future Teachers Education: Computational Thinking and STEAM (TeaEdu4CT) View project

All content following this page was uploaded by Carsten Schulte on 28 May 2015.

The user has requested enhancement of the downloaded file.

Block Model – an Educational Model of Program Comprehension as a Tool for a Scholarly Approach to Teaching

Carsten Schulte
Freie Universität Berlin
Takustr. 9
14195 Berlin
Germany
schulte@inf.fu-berlin.de

ABSTRACT

In this paper, the Block Model, an educational model of program comprehension, is introduced. Its use for planning and analyzing lessons on algorithms is evaluated in a qualitative study with prospective computer science teachers. In addition, the background of the model, its use in computer science education research and for developing competence models is discussed.

Categories and Subject Descriptors: K3.2 [**Computers & Education**]: Computer and Information Science Education – *computer science education, information systems education*.

General Terms: Experimentation, Human Factors.

Keywords: program comprehension, teaching algorithms, educational model, CS, CS Ed Research, Pedagogy, teaching teacher students

1 INTRODUCTION

How can we improve teaching? For example, by developing and using educational models within computer science education. The Block Model described in this paper aims to be such a model for planning and analyzing lessons, and functions as a tool for research on teaching computer science.

The Block Model is an educational model for program understanding, aimed at supporting research and practice of teaching programming. From the viewpoint of practice, the focus is on activities of planning and analyzing teaching in the area of introductory programming, and teaching algorithms like Bubblesort.

From these objectives follow three important characteristics:

1. *Simple*: The model has to be simple to be useful for practice. Therefore the study to be presented was undertaken with teacher students (novices with none or few experiences in designing courses or lessons) to see if the model is simple enough that even novices can use it.
2. *Constructive* orientation instead of prescriptive orientation. This feature focuses on the necessity to provide support for different specific needs at different institutions with different curricula, types of students, objectives, languages, etc. Instead of prescribing too many details, the model should support adaptive design of courses to fulfill specific needs.
3. *Communicative*: The model should support communication of ideas. It provides vocabulary and a framework for argumentation, as it outlines basic and essential features of how to learn programming.

From the viewpoint of teaching practice, the core objective is to support a scholarly approach to teaching and to use the model as an instrument (for educators) in order to increase a) the variety of solutions to teaching problems, b) the depth and focus of arguments and c) to provide a means to communicate about teaching programming.

From the viewpoint of research in computer science education, the core objective of the model is a better understanding of what it needs to learn programming in order to d) increase knowledge about: learning process, comprehension process, pedagogical analysis of program texts, and competencies (dimensions and levels) needed by learners, e) provide a comprehensive summary of research knowledge on learning programming and understanding programs for applicability in practice, f) to support accumulation of knowledge in pedagogy of learning programming, and g) support theory-driven processes of developing pedagogical interventions (e.g. structures of examples, learning sequences, and teaching methods) and diagnostic instruments.

In this paper results of a small experiment will be presented, where usability of the Block Model for practice, and in particular for teacher students learning to develop programming courses, was tested. The study focuses on the three characteristics simplicity, constructive orientation, and support for communication.

In the next section the Block Model will be presented, after that the background of the model discussed. In the second part of the paper follows a short empirical study of its use in teacher education. The paper ends with a short general discussion.

2 THE BLOCK MODEL

This section describes the Block Model. First, the motivation for describing program understanding (reading) instead of programming (writing) is explained. Then the model is introduced and explained.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ICER'08, September 6–7, 2008, Sydney, Australia.
Copyright 2008 ACM 978-1-60558-216-0/08/09...$5.00.

2.1 Reading instead of Writing

In this section we will discuss why to focus the Block Model on reading and program understanding when support for the more general area of programming education is the goal.

There are several reasons to restrict the model to program understanding: First, learning programming involves – besides constructive knowledge and skills – much predefined given knowledge, e.g. about the language, tools like compilers and development environments, good programming style, and of course about standardized solutions to programming problems, including knowledge about data structures and algorithms; therefore it involves reading given programs. Second, the process of writing programs involves reading, too. For example to find errors, to continue the work on the program text after a break, to understand compiler messages, etc. The third reason is that while reading is a vital part of programming and learning programming, it consists of fewer variations than possible programming activities and processes of novices. Restricting the model to reading allows easier access to empirical data (note: of a cognitive process), allows producing smaller and therefore more useful models, and in addition is focused on a part of learning programming where today almost no didactic approaches are known. That is, such a model highlights an aspect of learning programming which is often overlooked, and where teaching approaches are very rare.

In addition, a model about program text comprehension can be built upon a variety of research from different fields. For example, via text comprehension a link is built to text-related educational fields, where a rich tradition on learning reading and related teaching methods can be accessed, which might be useful to expand the repertoire of teaching methods in programming education.

2.2 The Block Model

The Block Model is defined as a table consisting out of three dimensions and four levels. Each cell (or block) of the table highlights one aspect of the understanding process – and hence one learning issue.

The general idea is to think of the blocks as movable. That is, in planning courses,

1. not all blocks always need to be taken into account, and
2. based on the model, different learning paths can be build. Therefore the blocks can be arranged in different orders.

These two important features of the model are based on conceptualizing the comprehension process as bottom-up and yet chaotic and flexible. First, comprehension at some point has to start with sensing (reading) the material (the program text) – which is a bottom-up process. Word by word, new information is constructed and added in the internal mental model of the reader. This process usually moves from bottom to top in the table: from words (Atoms), to blocks, to inferences about the relations between blocks to recognizing the overall structure. Within this process maybe all, maybe only one dimension is involved. Which types of information are extracted, and how the process of abstraction and inferences develops, depends on the already constructed mental model. That is, reading is influenced a) by the material already read, b) by knowledge, c) by additional information, and d) goals the reader has. All these factors are constrained or even constructed by the teaching approach. For example, by giving information about function and goals on the macro level, the bottom-up reading process turns into a process in which already constructed preliminary understandings (hypotheses) are refined, become more detailed, or are rejected.

***M*acro structure**	(Understanding the) overall structure of the program text	Understanding the „algorithm“ of the program	Understanding the goal/ the purpose of the program (in its context)
***R*elations**	References between blocks, e.g.: method calls, object creation, accessing data...	sequence of method calls „object sequence diagrams“	Understanding how subgoals are related to goals, how function is achieved by sub-functions
***B*locks**	'Regions of Interests' (ROI) that syntactically or semantically build a unit	Operation of a block, a method, or a ROI (as sequence of statements)	Function of a block, maybe seen as sub-goal
***A*toms**	Language elements	Operation of a statement	Function of a statement. Goal only understandable in context
	***T*ext surface**	***P*rogram execution (data flow and control flow)**	***F*unctions (as means or as purpose), goals of the program**
Duality	***“Structure”***		***“Function”***

Table 1: The Block Model

To summarize: understanding automatically strives to build or refine an abstract and general mental model. And because this process is automatic, teaching should focus on supporting this process, instead of focusing on providing all necessary steps in a predefined order, with predefined bits of information. That is, teaching has not to find ways to start this process, but to support learners to focus on core issues, in order to enable learners to build their own strategies how to process these core issues.

Another crucial factor is that learning means to be able to process these core issues (or at least some of them) more and more unconsciously and automatically, so that cognitive resources are freed to focus on other issues.

The Block Model aims to describe only the minimum of these core issues.

3 Theoretical Background and Related Work

In this section concrete sources of the model will be discussed. The aim is to elaborate the model, and to present evidence for the chosen structure and features. The Block Model draws on the following sources:

- psychological research about (text) comprehension
- research on program comprehension in computer science

- computing education research on learning programming and difficulties of novices and beginners
- philosophical and psychological research on characteristics of technical artifacts

As can be seen from this list of 'trading zones' for building an educational model of program reading, it is vital to keep in mind simplicity, while maintaining accuracy and internal consistency.

One consequence is the need for condensing and abstracting the knowledge of other areas. At the same time the model should interlock the different trading zones, and (thereby) also provide a means for further educational research.

3.1 Program Comprehension

For an overview and introduction to research in program comprehension see e.g. [2], [15] and [16].

Program comprehension is usually conceptualized as a process in which an individual constructs her own mental representation of the program (text). Therefore the fluent and intertwined cognitive processes of understanding are described as a *sequence of steps*. In general, three different sequences are discussed [16]: 1) a "bottom-up" process in which understanding is achieved from reading program statements, or 2) a "top-down" process in which the reading and understanding process is guided by initial hypotheses about the goal and functionality of the program (text), or 3) an "as-needed" process in which the process is either more top-down or more bottom-up, depending on the task and other issues.

However, program comprehension models tend to become quite complex and so far an accurate description of comprehension sequences does not exist. Instead it is more likely that the process depends on situational features. In other words, the comprehension process can be influenced, e.g. by the learning environment.

In order to support teachers to influence this process properly, the model should not aim to describe the best process, but help to construct learning paths for the specific situational needs. Accordingly we initially talked about constructive instead of prescriptive orientation.

Therefore the Block Model describes a set of steps (the different blocks), which can be arranged in different order. The Block Model provides a kind of map of a city, showing the blocks between locations A and B, and the teacher can choose how to go from A to B: either three blocks north before three blocks east, or vice versa, or zigzagging north-east-north…

A second characteristic of models in the area of program comprehension refers to the *types of information* needed or used in these individual mental models. For example, Pennington distinguishes between 1) function (and goals), 2) data flow, 3) control flow, and 4) conditionalized action ([13], pp. 298). She also gives an example for each of these abstractions – and each one has more elements than the original program text. In addition, other information types are discussed, like e.g. beacons, concepts, or roles of variables.

From the educational viewpoint and with respect to the first core characteristic of the model (simplicity), it seems necessary to reduce this variety of knowledge types to the essential ones. These will appear as dimensions in the model.

A third characteristic is: for understanding, the appropriate information has to be extracted, and based on the so extracted information, *inferences* are made. Usually these inferences are abstractions. Pennington argues: "The goal hierarchy can be at least partly recovered from the data flow abstraction by working up from the bottom, although it requires the application of knowledge to infer the grouping of subgoals with their goals." ([13], pp. 300).

A consequence for education is that understanding involves inferences based on the extracted information from the program text and on prior knowledge.

A fourth characteristic is that different kinds of *knowledge types* are needed for understanding. A typical distinction is between program knowledge and domain knowledge. Program knowledge focuses on knowledge needed for extracting and understanding the appropriate information, whereas domain knowledge is knowledge needed for contextualizing, and thereby understanding the goals of a program.

3.2 Text Comprehension

Research in text comprehension is also relevant, because programs are given as text, and of course comprehension models for programs discussed in the previous section are influenced from research in text comprehension. For example, the model from Pennington (see [16]) is based on the text comprehension model of van Dijk and Kintch from 1978. In 1998 Kintsch [7] published a generalized and expanded version of earlier work on text comprehension.

The focus is on describing the sequence of steps in the comprehension process. It is seen as flexible, context bound and strictly bottom-up. It is also considered as a cyclic process ([7], p. 101f). Textual information is perceived on a word-by-word basis and immediately incorporated in the mental representation. At the end of a sentence – in our context we should say: of a perceived program block – the capacity of the short-term-memory is reached, information needs to be transferred and integrated in working-memory so that short-term-memory is freed for the next cycle. In this integration process only some information (not all) of the former cycle is transferred, the mental representation is stepwise abstracted from the perceived material. This process is conceptualized as a hierarchic succession of mental representations. Each hierarchy level is more abstract and independent from the perceived information ([7], p.10-16). On higher levels it becomes more conscious and organized ([7], p. 93f). During the process, the extracted information is connected or integrated with prior knowledge to build a coherent (Kintsch says 'gestalt-like) whole ([7], p. 93).

From an educational viewpoint, and in connection with the sequence of steps discussed in the previous section, the educational model can be built on a general bottom-up basis. Within this general bottom-up sequence, as described above, different sequences of blocks can be chosen (north-east, east-north, zigzag …). While in general comprehension moves from parts to the whole, along this way different 'aspects' (=different blocks) come into focus.

Another aspect of text comprehension is the discussion of different information types extracted from the text, in alignment with different types of knowledge. While the overall mental model is seen as a 'gestalt-like' whole, Kintsch proposes to distinguish between text base for information extracted from the text, and a situation model for the activated prior knowledge from long-term memory which is included in the mental representation. This is done for analytical reasons. For example, the situation model can differ from previously extracted information (= the text base, which might be incorrect or incomplete) due to inferences based on prior knowledge of the reader ([7], p. 50).

The differentiation of text base and situation model is also useful in program comprehension models ([13], pp. 335ff). For example, "the functional hierarchy [goals] can be developed with reference to a situation model expressed in terms of real world objects" ([13], pp. 335).

Noteworthy for an educational model is that understanding of function and goals of a program relies on *inferences*, and on *knowledge type* referred to as domain knowledge. This aspect will be reflected in the construction of dimensions (the knowledge types necessary from an educational viewpoint) in the Block Model.

Another interesting aspect of Kintsch's comprehension model is that it was placed in a pedagogical context by assuming that with increasing competence of the reader a) lesser errors occur in extracting information, b) integrating, and c) activating the relevant prior knowledge. In addition, with more expertise more processes are done automatically or unconsciously so that more cognitive resources are free for more complex and intentional processing. The sequence of steps described by this model was used to construct the competence dimensions and interpreting the empirically derived competence levels in the PISA reading literacy test. These results can be used as a blueprint or How-to manual supporting the construction of competence models in computing education, and to construct the Block Model so that it can be used to derive and interpret competence levels. Note that therefore in the Block Model the concept of 'dimensions of understanding' is used instead of different *types of knowledge* that have to be extracted from programming texts.

In summary, from research in text comprehension the following can be applied to the Block Model: a refined understanding of sequence of understanding steps, the need for external knowledge for understanding goals and functions, and hints for deriving competence levels and dimensions.

3.3 Learning Programming

For a general overview of the discussion about learning programming see the workshop series on novice programmers (e.g. [4]), or more recently the summarizing article of Robins et.al. [1]. While the given model is in line with this work, we focus in this section on some results in detail.

One issue is the debate on learning difficulties, and hence the types of knowledge and dimensions of learning programming. In this debate several issues were found:

du Boulay describes five dimensions of learning programming [3]. Soloway and Spohrer [4] conclude that despite folk wisdom, mistakes are not all construct-based, but often are due to other reasons. These can be qualified as problems of integrating different kinds of information types. They propose goals (intentions or meanings of the program) and plans (ways of reaching goals) as important *knowledge types*. Plans need to be implemented in language syntax; during program reading certain properties are hints (so-called beacons) for the existence of a plan.

Results of a study [9], asking for the perceptions of students and teachers of difficulties in learning programming, were as follows: "The respondents perceived as the most difficult issues […those,] where the student needs to understand larger entities of the program instead of just some details about it" ([9], p. 16). From a similar study [5], the conclusion is drawn that students have the most problems „to comprehend what is happening to their program in memory, as they are incapable of creating a clear mental model of its execution." ([5], p. 55). Results of a think-aloud study with program-reading tasks [11] showed that students were focusing on the text-basis, without developing a situation model: "Few students in any quartile reasoned explicitly at a higher level than the walkthrough. That is, few students articulated the intent of the code, to count the number of common elements in portions of the two arrays" ([11], p.136). In a more recent interpretation of this result [10], which used the SOLO-taxonomy, the authors claimed that students are not able to see "the forest for the trees" ([10], p. 118).

These results indicate that complexity of understanding is somehow connected to the 'size of the entity' to be understood; in accordance with these findings and the above discussed *sequence of steps*, the Block Model lists the following levels: atoms, blocks, relations, and macrostructure.

A second issue is the importance of understanding program execution. Program execution is one of the *dimensions* of the Block Model. Another dimension refers to goals and functions (the situation model), whereas the last dimension refers to language constructs, the text surface from which information is extracted and transferred in the text base.

3.4 Digital Artifacts

A fourth background stems from a more general, philosophical debate about the nature of technical artifacts. This debate is illuminating the difference between what we so far called text base vs. situation model, and the empirical results stating that understanding of function and goals often appears in later stages, or not at all (see sections on learning programming and program comprehension).

According to Kroes [8], technical artifacts have a dual nature - and thus also have digital artifacts like program texts [14]. This duality is called structure vs. function. Structure draws on the empirically observable and objectively measurable properties of the program: The way the program text is structured, and accordingly the resulting execution of the program. In contrast, function draws on the intended goals, purposes and desired effects of the so-to-speak 'physical' structure. The crucial point in the philosophical debate is that function cannot directly be measured or inferred from structure, but is a socially grounded interpretation.

This has important educational implications. The above mentioned three dimensions belong to different sides of the duality: text surface and program execution is structural, whereas functions are functional. The former are 'technical' in a narrow sense, whereas the latter are 'social' in a narrow sense. Therefore the educational difficulty is to build bridges between these two sides of the duality. Especially, we have to avoid understanding of function that is detached from understanding the underlying structure – and we also have to avoid teaching structure without bridging to function.

3.5 Summarizing Conclusion

In summary, the above discussed sources lead to following conclusion.

In Figure 1 influences of the different theoretical backgrounds on the overall structure and on specific parts of the Block Model are shown. For example, from text comprehension (3) we can derive two different dimensions: the domain model, which captures goals (dimension C in the Block Model), and the text base, which includes text surface (dimension B) and as hidden information also program execution (dimension B).

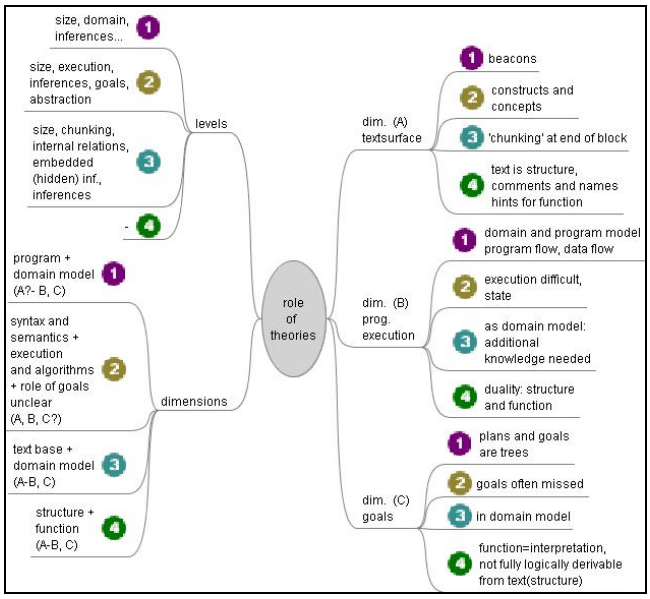

Figure 1: Contributions of different sources to the model (1 = prog. comp., 2 = learn. prog., 3 =text comp., 4 = duality)

The process of understanding can be conceptualized as a bottom-up sequence of hierarchic steps (cognitive processes). Comprehension necessarily involves inferences to more abstract or chunked mental representation of information.

In each of these steps, different types of information are extracted from the text surface, maybe in parallel, maybe sequential. Hence, a program text conveys a variety of different information types in a very condensed way. In order to reduce complexity, the educational model should reduce the needed information types to a minimum. These knowledge types are included as three dimensions into the model.

The discussed sources mention different kinds of dimensions, e.g. du Boulay lists five different areas to be mastered. By comparing the different theoretical backgrounds – in my opinion – three dimensions seem to be the minimal and hence appropriate choice. These are: text surface, program execution and understanding purposes (or function, goals).

Basically, necessary information has to be extracted from the program text, and then inferences are made to comprehend the execution and the goals/function of the program. From educational discussion in learning programming we know that it is difficult to understand the execution of a program. And from program comprehension, learning programming and dual nature approach it becomes evident that understanding goals and function is difficult, too. These aspects are not easily extractable information from the text surface, but instead based on inferences, including real world or domain knowledge. It is necessary to conceptualize this step literally as interpretation of a (program) text.

4 THE EMPIRICAL STUDY

The research question of the empirical study is, whether the Block Model in its current version incorporates the three prime characteristics mentioned in section 1 for the planning of lessons. These three characteristics are simple, constructive, and communicative.

In particular, the question is whether the Block Model a) is simple enough to be used by teacher-novices, b) supports their attempts in constructing ideas how to teach, and c) supports communication and reasoning about these ideas.

In the following sections the design of the study, participants, implementation of the study, the resulting data, and the interpretation of the results is presented.

4.1 Design of the Study

The study is designed as a comparative, observational study of two groups planning a lesson. The obtained data therefore is qualitative. The author of the article was not one of the observers.

Each group consisted of 4–6 members, and was encouraged to develop a concept for teaching Bubblesort – guided by a worksheet. The worksheet had four pages, see Table 2.

Page	Content
1	General information about how to plan a lesson.
	"Worksheet: Planning a Lesson About Bubblesort Some General Information: 1. Typical outline of a lesson is: Introductory phase, elaboration phase (main part), and wrap up phase (evaluation, expansion) 2. The main part is the core of the lesson. Usually, this main part is divided into different steps: E.g.: A) the teacher explains the core of Bubblesort, B) the pupils do some exercises, C) followed by a discussion in class, and then D) the algorithm is to be implemented by everybody. The main part has to be divided into steps; planning has to break down the topic of the lesson in its smaller parts. 3. All of these smaller parts are important. Think about their enactment: What information is provided by the teacher? What can be discovered by the pupils? 4. Therefore planning a lesson can be done in the following steps: a. Analyze Bubblesort in your group b. Collect ideas about the structure (steps) of the lesson (A,B,C,D, or: G,C,E,A…) c. Choose one of these ideas and elaborate the idea! 5. Important: The group's result for each of the three steps has to be written down on an overhead slide for later presentation in class!"
2	First task: Analyze the program text
	The source code as given in Table 5 was printed on this page, and some guiding questions: "Analyze Bubblesort: What is important? What is difficult? What has to be understood by everybody? What can perhaps be omitted? Has some additional information to be given?"
3	Second task: Develop ideas on how to give a lesson on Bubblesort
	"Your task: Develop ideas, how the lesson can be structured: Different steps, different sequences (A,B,C,D, or: G,C,E,A, or…)"
4	Third task: Develop a lesson plan
	"In your group: Refine and precise one idea. Describe and give reasons for the chosen steps."

Table 2: Worksheet, given to both groups

The observers remained passive, but took notes of the discussion. They were instructed to take notes on three issues:

1. Teaching ideas that were produced
2. If and how these ideas were explained and substantiated
3. If there were obstacles in communication.

In addition, overall impression and additional observations were written down afterwards. And as a backup medium to the notes, the group-work was videotaped, in case the observers noted interesting issues, but were not able to write them down.

A second data source is the results produced by the two groups.

The experiment was planned to consist of the following steps:

1. Brief introduction of both groups in a) how to plan lessons, and b) the worksheet that divides this task in three phases.
2. Handing out the worksheet, discussion of open questions
3. Control group starts planning (therefore moves to a second room). Experimental group is introduced into the Block Model before starting to work. (Note: only information of section 2 without sub-sections 5.1 and 5.2)
4. While both groups were working, the teacher (author of the paper) was available for questions concerning the tasks, and to remind the groups to focus on their task (in case needed)

The worksheet was the same for each group. In addition the experimental group received another paper with the table of the Block Model (see Table 1, left column).

In addition, both groups were instructed to design a lesson (or teaching unit of one to three hours, so that time restrictions did not matter) for a high school class which had learned all the basic programming languages skills beforehand and now should be introduced to sorting algorithms. The aim of the lesson was to teach Bubblesort in its simplest version.

After giving this information to the students, the control group went to another room, whereas the experimental group was introduced to the Block Model. Therefore the control group had about 10-15 minutes more time for discussing the lesson. Rationale for giving the control group more time was to provide a small advantage by giving them some extra minutes, because the Block Model contains helpful information on issues to look at, even when it is not directly used later on by the students.

4.2 Participants of the Study

Both groups consisted of students, mostly in their fifth semester. All are aiming at a Bachelors degree for teaching in secondary education with computer science as one of two subjects. Besides courses in computer science they have to take courses in a second subject (in this group mostly math), in general education, and in computer science education. The study takes place in the first lesson of the second course in computer science education. Topics of this course are planning, analyzing, and enacting computer science courses in secondary education.

Usually the students already have some teaching experiences, for example from a required internship at local high schools in their second subject (e.g. math), but it is their first time to actually plan a lesson in computer science on their own.

The groups were divided according to the experiences from the first course in the previous semester. The division aimed at building equal groups concerning teaching experiences, engagement in discussions, and quality of didactic thinking; that is, a general approach towards teaching including the ability to recognize important planning dimensions while also being able to concentrate on important issues.

4.3 Questions

The three general questions are refined for the empirical study in the following way:

1. Simple: Do students in their planning process use the model?
2. Constructive: A) Does the model support development of a variety of ideas? B) Does the model support reasoning about ideas?
3. Communicative: Is the vocabulary provided by the model used by the students?

4.4 Results of the Study

The results of the group work will be presented in the following order: 1. General data (time taken, group members, etc.), 2. Phase 1: Analyzing Bubblesort, 3. Phase 2: Developing ideas, 4. Phase 3: Refinement of one idea and 5. Additional observations

4.4.1 General data

The experimental group consisted of six students, the control group of four. One students of the experimental group had to leave early, another came late, so he arrived only after the introduction of the Block Model.

Before the discussions and results of the two groups are presented in the next sections, some information on the presentation:

The lists are generated primarily from the notes of the observers. The observers wrote down which ideas were produced, and (in case) argumentation. They did that shorthand, and wrote down key words. These key words closely resemble the words used by the students. In the experimental group especially key words from the Block Model are interesting; if these occur they are written in “quotation marks”. In some cases the wording was checked by looking in the video.

If some member of the group raises a question, or the group thinks about one issue, this is indicated by a question mark.

If the group exchanges arguments, and / or does not agree on an issue, this is indicated by the word ‘discussion’.

In addition, sometimes a short description is given, e.g. “After silent reading of the algorithm, one explains…” (see Appendix section 8)

The following table shows how much time was spent on the different phases of the study:

Phase	Experimental Group	Control Group
General Intro	(16 Min)	
Intro. Block Model	13 Min	-
Analyze Bubblesort	28 Min	17 Min
Generate Ideas	9 Min	34 Min
Refine one idea	20 Min	19 Min
Overall time	13 + 57 Min	70 Min

Table 3: Time needed by the two groups

4.4.2 Analyzing Bubblesort

Both group results mention the 'double for-loop' and constantly avoided the term nested loop. The experimental group – after an initial phase of about 10 minutes in which broad topics related to teaching Bubblesort were discussed – focused strictly on the provided questions on the second page of the work sheet (see Table 2), and presented their results in the same order.

The control group discussed more openly, and discussed two issues, which are not in the focus of the Block Model: The name of the algorithm and the missing comments of the source code. Both ideas survived throughout the group work and were included in the final results of phase 3. Some members in the control group failed to see that the program sorts from big to small numbers, but instead assumed the opposite (and relied on this wrong assumption in their final lesson plan).

In summary, the analysis of the experimental group is a little more thoroughly done. Sometimes vocabulary from the Block Model was used, but the group was obviously more guided by the provided analysis questions on the worksheet.

As a conclusion, the chosen short introduction in Block Model by presenting it as a table plus some minutes of explaining seem to provide not enough support for analyzing a program text. Therefore afterwards the concept of Block Analysis (see section 5.2) was developed as a lightweight help to use the Block Model for the didactical analysis of program texts.

4.4.3 Generating Ideas for teaching Bubblesort

The experimental group implicitly decided early to use an inductive (or: discovery based) approach, were the pupils discover the idea of the algorithm by themselves (and with support from teacher and the task, the teacher gives). The first six ideas (of 10 overall) discussed refer to this approach (see section 8.2).

In the second part of the discussion, a kind of Block Analysis of the algorithm was done. The discussion focused on possible sequences of steps to introduce the program text (with which Block to start, when and how to relate Blocks).

Overall, the experimental group generated some ideas. Two different issues or possible steps were regarded. Results presented mention three alternatives for these two steps, so overall only six ideas.

The control group discussed similar issues: First, how to introduce the algorithmic idea (see appendix: 1-6), and then the discussion switched to possible steps to introduce the source code (7 – 12); then some broader issues were discussed (13-15), like how to check understanding, how to ensure that working at the computer is effective for learning and to make the learning topic meaningful for learners (15). The discussion on introduction (1-6) involved inductive (1, 3) as well as deductive (4) ideas. Whether 2 (showing source code) was seen as inductive or deductive, is not clear. It was discussed in step 5 and ended in a fruitless search for asking open-ended questions (6), that is the group became unsure how to come up with an inductive teaching approach.

Accordingly, in the list of results written on the slide ideas on the introduction of the 'idea of the algorithm' (for clarification: cell MP in the Block Model) are merely variations of how the teacher conveys this information: explain the idea, explain the name, or work with model. A second range of ideas then focuses on how to introduce the source code (results labeled with E1 to E4). Remaining results focus on details.

In addition and in contrast to the experiment group even different sequences were presented: The first two are deductive, the third one inductive.

Overall the control group developed more ideas, and they developed different sequences. In both groups, the core of ideas focused on two issues: teaching the idea of the algorithm (deductive or inductive), and the possibilities to introduce the program text. Both groups agreed not to start with the source code, but to introduce it in a second step. In these discussions different possibilities were mentioned. In comparison it seems that the Block Model supported a deeper discussion with exchange of reasons and pros and cons of a solution, whereas in the control group merely variations were mentioned, but not judged. However, this might be due to a general approach to discussion, and in a brainstorming a group should refrain from judging or even sorting out ideas.

This was to be done in phase three instead.

4.4.4 Refining one idea

The experimental group literally refined the idea generated in the phase before (that is the idea of the overall framework of the lesson). They focused on refining each step in itself, and discussed reasons for the steps. This was done completely within the framework provided by the Block Model, and with extensive and fluent use of the vocabulary provided by the Block Model.

Initially, the group discussed a deductive approach (1), but then switched to an inductive approach (2 - 4). The second part of the discussion focused on introducing the source code (6 and 7). Here, also the Block Model was discussed (7). The results of the discussion were written on slides, and some reasons for the chosen steps were given, too.

This is in contrast to the control group. Here several ideas how to introduce the source code were discussed in phase two (see results E1 to E2 in result sheet (section 8.2), and then in phase three one of these option was chosen by quickly deciding on one option. Neither in phase 2 nor in phase 3 reasons for the steps were discussed.

The idea to provide pieces of two lines of code refers to the first phase; there was discussed which blocks were most difficult, and one of the group members thought of the two loops as a block – and therefore each piece of code should have two lines.

Overall, the control group did produce several ideas, but did not reason about implications of these ideas. The general outline of the lesson is very clear though: First, the program execution is presented visually, secondly, the program text as a description of this execution is given. In this second step two methods are used to force pupils to work with the source code: a) by ordering the given pieces, and b) by commenting the code. The third step reinforces the program execution, but now on the source code level and as individual work of each pupil who has to execute the code manually.

While the control group developed a sound lesson plan, the core objective of the task (analysis of the provided program, and developing a lesson based on this analysis) was merely touched upon. Throughout the discussions in the groups, there were no reasons given, and therefore choices made by chance, or by a general pedagogical understanding – but not based on the con-

crete topic to teach. In addition – with exception to some members of the group – the group failed to see that the example program sorts from big to small and not vice versa. Therefore the suggested first phase of their lesson would not match the later provided source code. The lesson plan remained vague in issues concerning the concrete example: "Pupils should comment the source code", in "comparison" "key issues (for-loop, tmp)" should be taken care of, but there is no explanation how to do this. It is also not clear, why the provided pieces of source code should have two lines each.

4.4.5 Additional observations

The experimental group initially does not really work on the three exercise (phases) given, but discusses planning ideas for the lesson in a somewhat freestyle brainstorming way. In contrast the experimental group immediately focuses on the first exercise, and throughout the working phase, students stick closely to the provided read thread by the three phases (they sometimes notice this by themselves, when they switch from one phase to the next). However, there is seldom any argumentation or rationale for ideas of group members. It seems that all have the same opinion, and they agree very quickly on how the lesson should be. In the third phase, against the instruction, the group does not provide a rationale for their solution. Until the end, they do not realize that the given source code sorts from big to small.

The experimental group initially does not use the model, although some students use vocabulary provided by the model and e.g. refer to the loops as for-blocks instead of for-loops. In the last phase the group makes extensive use of the model. In this phase each group member places the paper with the Block Model in front of his desk.

4.5 Summarizing Interpretation

In summary, the Block Model indeed leads to a deeper discussion of the concrete example to be taught. It supports reasoning about the example, it supports communication. It is simple enough to be used by novices after a short introduction. It is also constructive, as it enables to discuss variants, and in some cases to detect variants.

However, it does not support to produce more variants than the control group, on the contrary, the control group overall produced more ideas, and it produced ideas with regard to a broader range of issues to think of in planning (like provide a motivation, provide information of the practical application, use of media in learning, etc. That is, the Block Model indeed focuses planning activities on the core of steps to be used in the work out or main part of a lesson (see description of tasks in Table 2). Another explanation for the difference could be that the control group focused on easier things, and therefore produced a lot of general ideas and included other then the asked aspects in their planning discussions.

Interestingly, both groups implicitly discussed the duality-problem (Table 4).

	Experimental group	Control group
Algorithmic Idea (**function**)	Inductive / explorative	Deductive / observant
Source code (**structure**)	Deductive / observant	Inductive / explorative

Table 4: Comparison of the groups' overall lesson plans

A focus of their planning activities was on discussing how to switch from the general idea (function) to source code (structure).

5 CONCLUSIONS

In this section, some conclusions for the use of the model are drawn.

5.1 Understanding as Bridging Structure and Function

As can be seen in Table 4, the distinction between function and structure plays a crucial part in the overall structure of the lesson plans of the students. Presumably this distinction is of general importance. For example, anecdotically textbooks typically start with presenting the general idea of an algorithm, and then proceed with discussing code (structure). That is, structure and function are divided.

Based on the Block model, understanding a program means to be able to build a bridge from block (=cell in the table) ***AT*** (at the bottom left), to cell ***MF*** (top right).

Even for experts, this bridging is a complex process, because it involves crossing the border between function and structure (see section 3.4).

The duality might be seen like an ambiguous illusion (Figure 2).

Figure 2: A white vase, or two black faces? (http://commons.wikimedia.org/wiki/Image:Facevase.png)

In Figure 2 one might recognize two faces, or a vase – or both. While in these optical illusions we can focus on one side of the dual meaning of the picture only, in program text full understanding comprises both sides. Like in the figure, structure constrains function, and vice versa. From one side we can make inferences to the other side – and it is necessary to do so for comprehension.

The learning problem is crossing the border between structure and function. This border may be seen as a gap; and in order to overcome this gap, learners have to take a run-up and jump – but they might not get the necessary momentum, as they fiddle with finding the correct steps, or they are even aiming at jumping in a wrong direction.

This might be due to e.g. limited understanding of structure – but it is also due to the fact that from structure no direct, error free path leads to function. With experience and knowledge of typical structures and their assorted function, the transition from structure to function (and vice versa) becomes easier.

In order to think about this issue, and to educationally analyze a program text, a Block-Analysis can be done. This will be described in the following section.

In addition, functions and goals (the function-side of the duality of technical artifacts), are subjected to interpretation. My personal impression is that this side in teaching is usually taken for granted, like for example the interpretation of a mechanism (the

structure-side of the duality) of a sorting algorithm with the metaphor of bubbles – although for example bubbles in a glass of water just vanish instead of forming a line in which they are sorted by size.

The crucial issue is, function and goals can be interpreted differently, can be seen from different perspectives and communicated by different metaphors. If understanding is the objective of teaching, educational approaches should consider the possibility that differences in learners' understanding are natural, and not always a sign for misinterpretation.

However, the point is not criticism of the interpretations chosen by the computer science community (like the bubbles in Bubblesort). One aspect of learning can even be conceptualized – as is done by situated learning – as introduction into a community of experts, and into their ways of communicating and interpreting concepts of the field. Therefore teaching should help and encourage learners increasing participation in this community of experts by allowing them to converge their understanding, and their ways of communicating. This is often done by the way, quite natural. Examples might be programming styles students adopt on the way, or typical names for variables like 'i' – is it an integer?

Likewise, experts adopt typical figures of speech and strategies to describe and uncover functionalities of programs. (By the way: I think this is the reason why in research on program comprehension (see section 3.2) models aim at describing the one correct or best process of understanding; and why in teaching, we should allow different ways to understanding, although in the end these different ways are likely to converge)

In teaching, it is important to be aware of different backgrounds and thus different ways to express the same content of cell ***MF***, or of different understandings, that nevertheless are stable and viable in the current context (e.g. expertise level, or curriculum).

The Block-Analysis can be used to think about how novices (or readers unfamiliar with the program, and the domain) make inferences, and how this is different from experts, or those familiar with the program.

5.2 Block Analysis

This section shows how to use the Block Model for an (educational) analysis of a program text. As example, Bubblesort was chosen – the same example was used in the empirical study, described in section 4.

The Block-analysis aims at figuring out the relevant parts and aspects of the program text, which are important for teaching and learning. Of course, such an analysis is dependent on e.g. the concrete learning objectives and prior knowledge, motivation and skills of the learners.

Basis of the analysis is a division of the program text in Blocks. In the example (see program text in Table 5) four Blocks were identified (rows two to five) and named: outer loop, inner loop, comparison, swap-operation. Then, for each or some dimensions of the Block Model, columns are added. In Table 5 for example, goals of blocks are described.

The Program text	Name	Goal
1 public static void sort (int[] array) { 2 int tmp=0;		-
3 for (int i=0; i<array.length-1; i++) {	Outer loop	Repeat stepwise moving for each bubble /variable
4 for (int j=0; j<array.length-1; j++){	Inner loop	Move currently smallest Bubble to the end
5 if (array[j] < array[j+1]) {	Comparison	Check whether a move is needed
6 tmp = array[j]; 7 array[j] = array[j+1]; 8 array[j+1] = tmp;	Swap	Move Bubble one step (position in array)
9 } 10 } 11 } 12}		-

Table 5: Block analysis of Bubblesort

Such descriptions and short explanations could be added for text surface and program execution as well. For example, the text surface of the Comparison-Block uses a less-than sign instead of a greater-than – this could be noted in the 'text surface'-cell of the Comparison-Block.

In summary, the descriptions used in the cells should highlight relevant learning issues, and hence support planning a lesson: finding relevant issues to explain, thinking about the time necessary, thinking about proper sequences, and also e.g. about issues not important. In the example above, the outer frame (the header of the sort method) wasn't included in the analysis because it was seen as irrelevant.

In addition, some Blocks might be treated as Atoms – that is, pupils are not required to understand the 'internals' of the block, but only recognize it, and understand its effect – e.g. seeing the swap-block as a simple 'atomic' operation. It might then in addition be useful to hide the block, and use a method call instead in the program text for the pupils.

There is an additional aspect to be seen, which is connected to the duality of structure and function discussed in the prior section. This is the question, whether the chosen goal-descriptions in the example (see Table 5) are really valid or authentic. That is, such abstract descriptions rely on several inferences, and especially on knowledge which cannot be extracted from the block, but only from the macrostructure. Insofar the headings are not accessible from within the limited understanding gained by processing a block, but only afterwards from inferences based on the overall knowledge of the text. For example, from within a local understanding, the Swap-block is simply a swap, and not a 'movement of a bubble'. This difference can be seen as description of what a reader has to do in order to proceed from local understanding of a Block to a global understanding of the macrostructure. It might therefore be very useful to develop two different versions of Goal-descriptions of blocks: One version is focusing on the local understanding, whereas the other version focuses on the global understanding of a block.

Thereby the complexity of the program text, and maybe additional ideas for supporting the understanding process can be found;

because within this process from local to global knowledge also a bridge between function and structure has to be built. Local descriptions of a goal of a block are often more closely related to its structural aspects, whereas on the global level the function of a block (in terms of a sub-goal) has to be understood.

6 DISCUSSION

One result of the study is: The Block Model is simple, constructive and communicative when used by novices who discuss how to design a lesson about a given program.

Other objectives of the Block Model, its use for research or analyzing were not part of the study, and thus should be briefly discussed here, along with some open questions.

For analysis of lesson plans, a kind of Block Analysis of the core teaching steps could be used: Which blocks are in focus of a step, and which of its dimensions – or are other levels in focus, e.g. atoms, or macrostructure? Such an analysis could end in a kind of map, showing the focus of a lesson, and this map could be compared against the learning objectives and the discussed learning issues.

6.1 Research

In the introductory section some aims regarding research were mentioned (issues d) - g)). First, the Block Model can be seen as a comprehensive summary of parts of the knowledge related to learning programming (d and e). As the results of the study show, relying on the Block Model alone might lead to disregard issues like motivation and embedding the learning topic in a broader context – this is indicated by the fact that these issues are mentioned by the control group, but not by the experiment group. The conclusion is, the Block Model is restricted in its focus, which seems reasonable, and supports (by means of simplicity) applicability of the model. Using the model for planning a lesson is a scholarly or theory-driven approach to teaching (g).

Open issues are whether the Block Model is useful in research in order to gain new knowledge (f). Some small insights were produced by the presented study: The importance of a means or tool for analyzing (see section 5.2) and the importance of the gap between goals and program text, and between structure and function, respectively (see section 5.1). Another aspect would be the usefulness of the Block Model for research on competencies and diagnostic instruments. This will be discussed in the following section.

6.2 Describing Competencies

Another benefit of the model for research is its use for describing skills and competencies of learners. The model could be useful for defining competence levels and competence dimensions: With increasing skills and competencies of learners, the focus different levels are in focus, because the first levels are processed (more) automatically; but for novices the lower levels are more important. This is the major idea to use the model for describing competencies and competence dimensions. It was successfully done in the area of reading competencies, where text comprehension models based on Kintsch's work were used to develop and empirically refine levels of reading competencies in the PISA-tests.

In addition, maybe such levels resemble taxonomic descriptions. E.g. SOLO-taxonomy used for distinguishing skill levels of programmers, or the 'hierarchy of object-interaction'. However, taxonomic descriptions of competence levels should not be confounded or confused with the variety of possible learning sequences.

7 REFERENCES

[1] Robins, J. Rountree, and N. Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003.

[2] Détienne, Francoise: "Software Design – Cognitive Aspects", Springer Practitioner Series, 2001.

[3] du Boulay, B. (1989). Some difficulties of learning to program. In Soloway, E. and Spohrer, J. C. (Eds): *Studying the novice programmer,* 57-73

[4] E. Soloway and J. Spohrer. *Studying the Novice Programmer*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1989

[5] Milne & G. Rowe. Difficulties in Learning and Teaching Programming - Views of Students and Tutors. *Education and Information Technologies,* 7(1):55-66, 2002.

[6] J.-M. Burkhardt, F. Detienne, and S. Wiedenbeck, "Object-Oriented Program Comprehension: Effect of Expertice, Task and Phase," Empirical Software Eng., vol. 7, no. 2, pp. 115-156, 2002.

[7] Kintsch, Walter: Comprehension. A Paradigm for Cognition. Cambridge University Press, 1998.

[8] Kroes, P.: Technological Explanations: The relation between Structure and Function of Technological Objects in: Techné: Journal of the Society for Philosophy and Technology Vol 3, No.3, 1998

[9] Lahtinen, E., Ala-Mutka, K., and Järvinen, H. 2005. A study of the Difficulties of Novice Programmers. In ITiCSE '05

[10] Lister, R., Simon, B., Thompson, E., Whalley, J. L., and Prasad, C. 2006. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. In ITICSE '06. ACM Press, New York, NY, 118-122.

[11] Lister, Raymond et al. A multi-national study of reading and tracing skills in novice programmers. In *ITiCSE-WGR '04: Working group reports from ITiCSE 04.*

[12] Mayrhauser, Anneliese von; Vans, A. Marie. Program Comprehension During Software Maintenance and Evolution. *Computer* 1995, *28* (8) 44–55. http://doi.ieeecomputersociety.org/10.1109/2.402076.

[13] Pennington, Nancy: Stimulus Structures and Mental Representations in Expert Comprehension of Computer Programs. In: Cognitive Psychology 19, 295 -341 (1987)

[14] Schulte, Carsten: Duality Reconstruction – Teaching digital Artifacts from a socio-technical Perspective. In Mittermeir, Roland T.; Syslo, Maciej M. (eds.): Informatics Education – Supporting Computational thinking. Proceedings of the Third International Conference on Informatics in Secondary Schools, ISSEP 2008, pp. 110-121.

[15] Storey, Margaret-Anne: Theories, Methods and Tools in Program Comprehension: Past, Present and Future. In: (IWPC'05) (IEEE)

[16] Van Mayrhäuser, A.; Vans, A.M.: Program Understanding – A Survey. Technical Report CS-94-120. Colorado State University 1994

8 Appendix

8.1 Experimental Group

8.1.1 Phase 1: Analyzing Bubblesort

The experimental group started reading the general information written down on the first paper. Apparently they had to remember what the tasks were. However, after reading the general structure of a lesson, a discussion started on how to involve pupils in the lesson, how to engage them.

After about 10 minutes I reminded them to stick to the given exercises, and start with phase 1, and to produce a slide summarizing the results of the analysis of Bubblesort.

18 minutes were needed to analyze Bubblesort. Within this discussion the group kept referring back to other issues (how to involve pupils, sequence of teaching steps).

Referring to the analysis of Bubblesort, the following issues were discussed (as protocolled by the observer)

1. Start with "if-block" first?
2. Present only parts of the program ext to prevent confusion of pupils
3. Show pseudocode first (leave out syntax)
4. Important: Loop variable is counting up
5. Difficult: tmp-variable in swap operation
6. Difficult: Why two loops
7. Show "if-block" first; "text-surface" of "if-block"; leave out "if-Block"; explain "if-block"...show only effect, or is that too abstract?
8. Show inner loop at blackboard in front of the class
9. pupils might make suggestions to enhance the algorithm
10. What is important? Outer loop? Pupils have to understand both loops
11. Pupils might ask what happens when two identical values are in the array
12. Teacher should ask whether sorting is from small to big values or vice versa

Results written on the slides are:

- Important: Swapping of two array-elements; double for-loop; each loop repeats at the beginning
- Difficult: see important issues
- What is to be learned: loops; array sorted from big to small
- What can be omitted: Lines 1 and 12
- Necessary additions: end algorithm earlier; let algorithm sort from small to big

8.2 Phase 2: Developing ideas

The experimental group discussed the following:

1. Let pupil sort a set of cards, followed by a discussion of how a computer can sort
2. note on a blackboard, how the pupil sorted the cards; note the steps
3. one pupil instructs another how to sort the cards
4. Teacher gives hint so that pupils discover Bubblesort
5. let students work in groups; or let some sort cards in front of the class
6. use some balls or bubbles to sort
7. discussion where the difficulty of the algorithm is (only two elements at a time; "if-block" first)
8. "relate if-block to inner loop"
9. chose sequence of steps from inner (blocks) to outer blocks
10. sort real things, sort in groups?

Results written on the slides are:

- Pupil sorts e.g. cards as he likes; discussion: How can the computer sort cards?
- Note on blackboard, how the pupil sorted (exact phrasing, maybe ask other pupils to phrase)
- Introduce the rules of Bubblesort. Everybody should see how the algorithm works. Discussion in class: What is difficult?
- Go to the source code: start with if; move from inner to outer
- pupils develop code on their own
- In groups: try out Bubblesort

8.3 Phase 3: Refinement of one idea

1. Explain first what the algorithms does, or explain first why/how the algorithm works (with explicit reference to the Block Model)
2. Put numbers on blackboard, let pupil sort from big to small, so that it matches the given program
3. Provide motivation for the algorithm; real life examples or something (sort clothes in the closet according to their color). Pupils need to know what the problem is, need a vivid illustration
4. Teacher manually demonstrates and explains the algorithms. E.g. asking why this step is necessary. Explain function/goal (explicit reference to the Block Model)
5. Discussion: Let pupils develop the code, but not from scratch. How much time does that need? Let them figure out the algorithm in groups.
6. Discussion: Teacher presents the code in different Blocks, and explains how the Blocks are related. Or: use a sequence from Block Model from bottom to top, beginning with text surface, then execution, then goals. Or: Focus on the level macrostructure. Discussion of other possible sequences from the Block Model, e.g. 'zick-zack', 'spiral', 'inner to outer') (Students focus on Block Model, use the vocabulary provided by the model)
7. Discussion whether for-loops are appropriate Blocks

Results written on the slides are:

- One pupil should sort a sequence of given numbers from big to small.
- Another pupil should write down the steps
- -> Rationale: pupils should become aware of the problem; algorithm is vividly illustrated
- teacher introduces Bubblesort and illustrates the algorithm
- -> Rationale: Understanding goal and purpose of the algorithm
- Optional: Work in groups: Pupils develop Bubblesort on their own
- Present the source code and explain it block by block, relation block by block. Thereby repeatedly reinforce the algorithm

8.4 Control group

8.4.1 Phase 1: Analyzing Bubblesort

1. After silent reading of the algorithm, one explains the algorithm
2. In his explanation are mentioned: loop is executed several times, each time shorter, starts at the end
3. analysis of efficiency is too difficult
4. explain the algorithm visually/metaphorically
5. group the algorithms into "blocks", estimate the difficulty of the blocks
6. Find the most difficult part of the algorithm. Then a discussion starts whether the printed algorithm contains an error (at this point the group stopped working to ask the teacher whether the source code given to them was correct – it was correct)
7. leave out details of syntax
8. add comments to the code

Results written on the slides are:

- comments are missing
- lines 3 and 4 (for-loops) have to be explained in detail
- Functionality: executed several times; is filled from the end; each iteration until the end
- there are alternative possibilities to implement the algorithm
- explain name of the algorithm

8.4.2 Phase 2: Developing ideas

1. give pupils numbers to sort, give them five numbers to sort step by step

2. show the source code
3. ask students for ideas
4. explain name of the algorithm
5. discussion in the group: start by providing source code, or start with 'hand'-executing some examples
6. Can teacher start by asking some general questions? Groups finds no ideas for such general open questions
7. Group discussion on how to introduce implementation: present source code, let pupils implement the algorithm, let them implement arts, let them sort given pieces, use pseudocode
8. Problem with implementation is presumably the double loop. The loops should be taken extra care of
9. let the class collect ideas, so that pupils maybe can develop ideas about how the algorithm/implementation is 'structured'
10. teacher gives hints
11. pupils should sort easy sequence of numbers manually
12. discussion about the role of the implementation; and the role of working at the computer
13. discussion on how to check understanding of pupils
14. again discussion about how to let pupils work at the computer. e.g. how to prevent that pupils simply train their keyboarding skills
15. discussion of necessity and possibility to provide a framework / example in which the sorting-algorithm is useful

Results written on the slides are:

- A explain model/ principle
- B explain name
- C work with model / exercise principle
- D open questions
- E E1) give code, E2) give pieces of code, E3) pseudeocode. Should lead to F)
- F pupils comment the given code
- E4) give complete code including comments
- G For-loops in detail
- H pupils should develop ideas for the 'code principle'
- I Manually sorting an example
- J Exercises, work at the computer

- First sequence: A B C D (E2/E3) E1 F G I J
- second sequence: B A C G D (E2/E3) E1 F I D J
- third sequence: H A B C D E4 G I D J

8.4.3 Phase 3: Refinement of one idea

1. Short discussion about first steps. All agree on A; B, C, D. Discussion about which E to take. Discussion about different types of activity of pupils work, in groups, on their own, executing the algorithm manually, ... learning objective: everybody has to understand 'the principle' of Bubblesort
2. Discussion how to introduce the source code. Idea to present the source code in different pieces, which have to be arranged by the pupils. Idea: Each piece should consist of two lines because 'this fits nicely'
3. Then focus on loops
4. Introduce some relation to practice, or everyday experience of the pupils

Results written on the slides are:

- → A B C D E2 E1 F G I D J
- provide example: Show principle of moving up; therefore the numbers are written down vertically
- Explain the term Bubble using the example from A)
- In class: sorting game. Or each pupil on its own in order to practice the principle
- teacher tests understanding of the principle by observation
- Provide pupils with source code pieces (each two lines)
- for comparison / control of results give pupils the complete source code
- pupils should comment the source code
- Comparison of comments and focus on key issues (for-loop; tmp, etc.)
- short example sequence of numbers, and worksheets to ensure structured hand execution of the code
- check results, discuss individual problems of the pupils
- Relate the algorithm to an actual software project (to be developed in class)

View publication stats