Creating and Testing Textbooks for Secondary Schools

An Example: Programming in LOGO

Karin Freiermuth, Juraj Hromkovič, and Björn Steffen

Information Technology and Education
Department of Computer Science
ETH Zürich, Switzerland

Abstract. The main goal of this paper is to present our approach for writing textbooks that are self-contained and available for individual learning. These texts are written in the language of the corresponding pupils and are not restricted by any length limitations. This allows us to write as clearly and thoroughly as possible. Minimizing the time for mastering the subject instead of minimizing the presentation length is the main goal. The detailed lecture notes provide a safety net for the teacher and the pupils alike. They do not restrict teachers in their interaction with the class or in the freedom of choosing alternative ways in approaching the subject. On the contrary, the freedom of designing the content of the lesson increases because pupils have the certainty to be able to learn from the lecture notes if something was not fully understood.

Here some general rules for writing learning aids are presented and then applied for an introductory course about programming in LOGO. Finally, we summarize our teaching experience in different classes with the produced learning aids.

1 General Concepts and Basic Rules

We are missing textbooks for teaching fundamental concepts of informatics in german that would at least partially satisfy the following requirements:

- The main focus is on presenting the programming skills and fundamental concepts and ideas of information processing instead of reducing the computer science education to computer driving license and product knowledge.
- The texts are suitable for learning autonomously for pupils in the corresponding age.
- They systematically build the foundation of concepts following the historical roots of informatics in a similar way as other natural sciences do.

We do not discuss the misleading concepts of computer science education that were broadcasted in many countries as the consequences of the emphasis created by the fast development of information technologies. First, we present some guidelines we use when creating learning aids.

R.T. Mittermeir and M.M. Sysło (Eds.): ISSEP 2008, LNCS 5090, pp. 216–228, 2008.

1. **Mastering the Topic to be Teached.** At the beginning it is important not to think about the didactical methods to be used, but to check your own knowledge. Is my understanding of the subject deep enough? Do I see the topic in the correct context of my scientific discipline? Do I know the history of the development of the basic concepts of the subject I want to teach? Do I understand why the concepts were developed in the way they did and not in another one? There is always something that still has to be discovered. Study the appropriate literature and discuss the open questions with colleagues.
2. **Which Notions and Concepts are Known?** A careful formation of concepts is crucial for the success in teaching. A clear picture about the previous experience of the class has to be established and written down. The main point is not to think about skills and methods only, but mainly about concept formation in the sense of building new notions (terminology). Which notions and concepts are already known and how deeply are they understood? Which terms are unknown or only partially understood? One fixes the current state of the knowledge and thinks about questions and tasks suitable for checking the previous experience. This is not a black and white game. For instance, one can ask to which extent are the pupils able to correctly and transparently describe their course of action in a natural language (without programming).
3. **Learning Objectives.** What do I want to achieve? Again, thinking in the black-and-white-manner and stating that the pupils learn something that was unknown before, has to be avoided. Deepen the understanding of basic concepts, extend the applicability of different methods and develop various skills. All this has to be explicitly formulated.
4. **Recurrent Theme.** The chosen learning objectives need to be arranged in a linear sequence. The order of introducing new notions and concepts has to be defined. Usually several suitable orderings exist. This part of your work is more or less based on your experience. A good idea is to discuss your concept with colleagues and test it in classes.
5. **High Willingness to Revise.** The next steps go into the details. Here, you might recognize that some of your educational strategies are not optimal or even do not work. Be prepared and willing for frequent revising and change the work you have done in 1. – 4. if necessary.
6. **How to Explain?** Do not yet think about the choice of your didactical method. Formulate all explanations in detail. Use the language which is already understood by the class. Do not use terms or words that have not been carefully introduced before. Check and verify your explanations.
7. **Interaction between the Teacher and the Class.** For each learning step, think about the possibility to communicate with the class. Which questions and tasks are suitable for checking and deepening the understanding of the class for the just introduced subject? What kind of misinterpretations can occur and how to deal with them? The interaction with the class has to be planed and written down.

8. **Exercises and Solution Proposals.** Think about exercises for individual work. Solving these exercises has to guarantee to master the subject introduced to a reasonable extent. The number of exercises has to be larger than needed in average. For those exercises that essentially check or deepen the understanding of introduced notions and concepts, provide a detailed description of the ways how to solve them. Successful work with the formulated exercises must give the pupil a guarantee of being successful in the examination. A variety of tasks has to assure the ability of the class to apply the acquired knowledge in different situations. The tasks for the exam are formulated in harmony with the exercises and the learning objectives.
9. **The choice of the Didactic Methods.** Now decide about the didactical methods for each part of teaching. Think about those parts of the education process which require a strong control and which parts can be developed by the class on their own. As soon as you have taken this decision you can adopt the texts in an appropriate way.
10. **Summaries.** At the end of each teaching unit write a summary. The summary repeats the learning objectives and the knowledge of the unit in the acquired language of the class.

There are many principles and small hints that should be considered during the whole work on a textbook. Here we list some of them:

- It is worthwhile to ask the class to keep a dictionary of the notions and concepts introduced until now. If a notion or a concept is too complex record the expected depth of its understanding.
- Use pictures where they can be helpful. The meaning of these figures has to be carefully explained. Ask the class to develop similar graphical representations.
- Never use words whose meaning is not completely clear to everyone in the class.
- All written tasks need to have a clear and unambiguous interpretation. Omit long sentences whenever possible. The tasks should be clearly structured.
- No long sequences of explanations without intermediate questions and tasks are allowed.
- The most efficient way to teach is to connect the new knowledge with the previous experience. Try to use analogies whenever possible.
- Essential things have to be clearly highlighted. Posing appropriate questions and tasks can be helpful for this purpose.
- A good motivation must always be given. It is not necessary to always search for applications. The aim to discover something essential or to learn to understand something complex can be even more exciting.
- For each small part of your work with the class, plan the interaction (communication) with the pupils.

2 The Concept of Our Textbooks

To support and encourage teaching of computer science in german speaking countries, we decided to develop several modules, each dedicated to another topic. These modules have to be as independent and self-contained as possible. There is no restriction given on the number of pages and so we use as much space as we need to explain everything carefully, to train and to verify the acquired knowledge.

The style of these modules is close to "ETH-Leitprogramme" [1], which is an improvement of the Personalized System of Instruction (PSI), also known as Keller Plan. The main idea, however, is not to create texts for learning autonomously. The texts may be used in this way, but the main purpose is to provide complete information about the topic, and so to assure as many high-quality iterations of the subject as the individuals of the class need or wish. In addition to detailed explanations, the text contains numerous questions, tasks and exercises which are placed exactly where we recommend to apply them. At the end of every lesson the module contains also questions and exercises to check the understanding of the lesson. Furthermore, the textbooks contain hints for teaching persons. These hints are based on our experience and call attention to possible troubles and misinterpretations usually occurring when teaching some more complex matters. Proposals to overcome these difficulties are given too.

Each module is divided into lessons. One lesson is usually for 2–4 hours of teaching and is devoted to one or more new concepts or methods.

3 Introduction to Programming with LOGO

Here we present the concept of the module for an introductory course on programming. The first six of the fourteen lessons can be used in primary schools and the last four lessons are a challenge for the final classes at secondary schools.

3.1 Basic Idea

The idea of this module is not to completely replace a programming course in a high-level language. It is left up to the teacher to decide after which lesson she or he switches to a higher programming language.

The reasons for choosing LOGO for an introductory programming course are the following:

1. One can start with this language already in the third class of the basic school. Drawing pictures is exciting for pupils. They immediately see the actions of the turtle and can easily revise their instructions if something has gone wrong.
2. One can learn programming by starting with five instructions only and working totally with about fifteen instructions that are sufficient for programming any complex behavior of the turtle. Our philosophy is to follow the history of programming, and so to derive all complex instructions as programs consisting of a very small set of basic instructions.

3. The most fundamental concepts such as
   - modular design (programs, subprograms)
   - loops
   - parameters and variable
   - branching of programs, conditional loops
   - recursion
   - descriptional and computational complexity

   can be successfully taught with LOGO.
4. LOGO can be used to build bridges between teaching mathematics and computer science. Teaching elementary geometry in LOGO is a well known example [2,3], but one can support teaching of trigonometry, function analysis and vector geometry as well.
5. Proper teaching of LOGO circumvents the gender problem in programming courses. Programming is a systematic work. Too many ad-hoc decisions and too much improvisation may be dangerous. If programming is correctly taught in a systematic way, girls and young women like it and are often more successful than their male classmates.

Note that LOGO is a programming language developed solely for educational purposes [4] and is used for introductory courses in programming for pupils in more than 40 languages.

3.2 Organization of the Module

To understand the concept of our module properly, we present it lesson by lesson and explain the goals of each one.

Lesson 1 – Programs as Sequences of Instructions
Following the strategy to work with as few simple computer instructions as possible, only four instructions `fd`, `bk`, `rt` and `lt` for the movement of the turtle are introduced. The enclosed exercises are devoted to pupils of basic school and focus mainly on viewing the movement direction from the point of view of the turtle. Another aim is to learn to see programs as sequences of simple instructions that are unambiguously interpretable by the computer.

Lesson 2 – Repeat-Loops
The aim is to learn to work with loops with a constant number of repetitions, and to put one loop into another. An essential point is that one does not need the concept of variables for this purpose. The exercises focus on recognizing how to partition the picture into repetitions of the same figure and what needs to be done between drawing recurring figures.

Lesson 3 – Modularity
Modular design is one of the fundamental concepts of engineering. Here, one has to learn to use it by giving names to programs and then using the given names as new instructions. The importance of this design method for the

transparent and systematic development of computer programs and their verification is explained. The notions of the main program and subprograms are introduced and the pupils learn to represent the program structure in a graphical way.

Lesson 4 – Drawing Circles and Regular Polygons

There is no new programming concept introduced in this lesson. A bridge is built to elementary geometry and working with repeat-loops and subprograms is trained. Using colors is introduced and time for creating own fantasy pictures is provided.

Lesson 5 – The Concept of Parameters

Parameters are introduced as variables whose values do not change during the execution of a program. The general concept of variables is still unknown. Our experience shows that children in the third and fourth classes can master the work with parameters, but only few are able to understand the concept of a variable. The values are assigned to the parameters exclusively as program inputs. In this way the pupils design programs for a class of pictures instead of having one specific program for each potential figure. Here we teach that the computer assigns a register (memory unit) to each parameter and saves its actual value in the assigned register. This enables to parametrize the size of figures drawn, as well as the number of repetitions.

Lesson 6 – Parameters and Subprograms

In this lesson we teach how to pass the values of the parameters of the main program to the subprograms. The pupils learn how to use a subprogram several times with different values of its parameters in one execution of the main program. With Lesson 6 the part of this course for the basic school finishes.

Lesson 7 – Optimizing Program Length and Computational Complexity

This lesson does not introduce any new programming concepts. It is about measuring the "quality" of programs by quantitative measures as descriptional complexity (the number of instructions) and as computational complexity (the number of instructions executed). While the length of a program is a constant, the computational complexity is a function of the program parameters. One learns to measure both. An interesting attraction is to organize competitions in writing the shortest program or the most efficient program for a given task.

Lesson 8 – The Concept of Variables

A good recommendation is not to start with the concept of variables to early. We do it by introducing the instruction `make` and taking care that we explain the execution of the instruction `make` on the level of computer registers. The pupils are motivated to work with variables by writing programs for drawing classes of figures and really complex figures.

Lesson 9 – Local and Global Variables
The understanding of variables is deepened by dealing with information transfer between (global) variables of the main program and the (local) variables of its subprograms. Everything is again explained and trained on the level of computer registers.

Lesson 10 – Branching of Programs and While-Loops
This lesson is devoted to the introduction of the instructions `if` and `while`. The pupils learn to use conditions not only for drawing pictures, but also for programming methods for solving mathematical problems.

Lesson 11 – First Bridge between Programming and Mathematics: Geometry and Equations
In this lesson no new programming paradigm is presented. The aim is to apply the acquired knowledge and skills to develop graphical solutions to several tasks of elementary geometry (constructions of triangles, intersection of geometric objects). Furthermore methods for solving simple equations are implemented. Using `while`-loops one can also find approximate roots of polynomials.

Lesson 12 – Recursion
Recursion is the most complex programming concept of this course and so we devote it a lot of space. Again we take care of showing all important details about the execution of recursive calls on the level of the computer registers. Starting with infinite recursion depth we continue with programs using only one recursive call. By now we defined the recursion depth and present the pushdown principle of executing a recursive program. Rewriting recursive programs to `while`-loops and vice versa is also trained. After that recursive programs with a few recursive calls are introduced and analyzed. The working of the copies of the variables is explained into detail. Graphical representations of the execution of recursive programs are developed and applied.

Lesson 13 – Second Bridge between Programming and Mathematics: Trigonometric Functions
The pupils learn to implement different methods for solving trigonometric tasks. After the introduction of the instructions `sin` and `cos`, they also learn to develop programs that approximately compute the functions `arcsin` and `arccos`.

Lesson 14 – Third Bridge between Programming and Mathematics: Vector Geometry
One develops programs for working with vectors in a graphical way without using the powerful operations working with the values of coordinates. Hence, by drawing a line between two points it is required to compute the corresponding angle and its length first. This essentially supports the understanding of methods for solving different tasks in the two-dimensional (and partially in the three-dimensional) space.

3.3 How We Introduce New Concepts

To demonstrate our approach to introducing new programming concepts, we show how we stepwise motivate, illustrate and teach the concept of the variable.

1. **Basic Commands.** First the LOGO module introduces a few basic commands. Some of the most important commands of the programming language LOGO are shortly explained here:

   `fd 100`**.** The command `fd` moves the turtle a certain amount of steps forward, e. g. `fd 100` moves the turtle 100 steps forward.

   `rt 90`**.** There are two ways of turning the turtle: Right turn (`rt`) and left turn (`lt`). The command `rt 90` executes a 90 degree right turn.

   `repeat 4 [fd 100 rt 90]`**.** The command repeat is followed by an integer which indicates how many times in a row the commands in the brackets are executed. The repeat command above draws a square of length 100.

2. **Named Programs.** After doing some exercises, where the pupils always use the same programs, it is easy to motivate the need to give the programs unique names to later call them. It also shows how to structure programs into subprograms.

   Instead of writing `repeat 4 [fd 100 rt 90]` the pupils can now give this program the name `Square100` such that later on, they can just write `Square100` to draw a square of size 100×100. For drawing a square of size 50×50, they need to write another program called `Square50` for example.

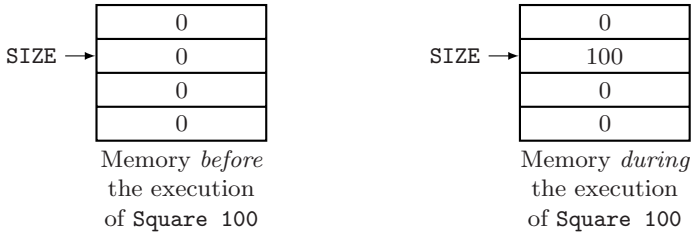
3. **Programs with Parameters.** With named programs the pupils are able to write more complex programs without rewriting commands over and over again. We introduce a further concept to make programming more convenient, namely parameters. The need of parameters is depicted by a concrete example:

```
to Square50
repeat 4 [fd 50 rt 90]
end

to Square100
repeat 4 [fd 100 rt 90]
end

to Square200
repeat 4 [fd 200 rt 90]
end
```

   With the knowledge the pupils have gained up to that point they are forced to rewrite a new program for every size of the square to be drawn. By doing so, the idea of introducing parameters, which lets us call the same programs with different values, becomes obvious. We introduce parameters for the size of the square and write the following program by replacing the concrete size of the square in the program above by the parameter `SIZE`.

```
to Square :SIZE
repeat 4 [fd :SIZE rt 90]
end
```

Not only the concept of the parameter and its usage is explained, but also the relationship with the memory is illustrated. The computer memory can be considered as composed of a set of registers. In each of these registers exactly one number can be placed. Each unassigned register contains the value 0. Furthermore, every register can be named.

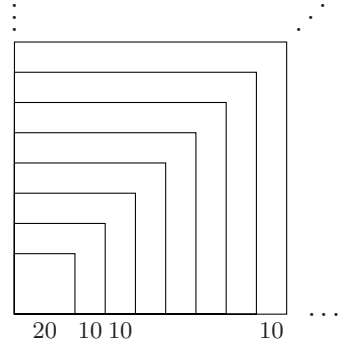


Memory *before* the execution of `Square 100`

Memory *during* the execution of `Square 100`

There exists a register with the name `SIZE` representing the parameter of the program above. Since no value has been placed up to that point, the register contains the value 0. As soon as the program has been called with `Square 100`, the number 100 is placed into the register with the name `SIZE`. During program execution the register is accessed every time the value of the parameter `SIZE` is needed.

4. **Variables.** Finally, to motivate the introduction of variables, we discuss with the pupils the following problem: We would like to draw an arbitrary number of squares with growing sizes as shown in the following picture. The number of squares should be configurable by the parameter `N`.



Without the concept of the variable this problem is not solvable.

We explain now the variable as a generalization of the parameter. This means that in contrast to the parameter, the value of a variable may be changed during program execution. What we want to achieve is something like the following:

```
to ManySquares :SIZE :N
Square :SIZE
Increase the value of :SIZE by 10
Square :SIZE
Increase the value of :SIZE by 10
⋮
Square :SIZE
Increase the value of :SIZE by 10
end
```

N-times

The pupils see that the following lines

```
Square :SIZE
Increase the value of :SIZE by 10
```

are repeated `N` times.
Using the newly introduced `make` command, the pupils learn to rewrite the program to:

```
to ManySquares :SIZE :N
repeat :N [Square :SIZE make "SIZE :SIZE + 10]
end
```

As with parameters the concept of the variable is explained on the level of registers. To train this, there are also several exercises where the pupils need to specify the contents of variables at different stages of the program execution.

4 Observations and Considerations

In this section we present how we teach LOGO with our module. The following suggestions have been developed during numerous lessons at a variety of secondary schools in Switzerland.

4.1 Using the Module in Classes

As already mentioned, our module about programming in LOGO discussed in section 3 does contain all information, theory and exercises to learn individually. This is very important because it allows the pupils to iterate through the material as often as needed to understand the introduced programming concept.

None the less, teaching of programming can be even more improved by the following guidelines:

- The more difficult topics of programming, like variables and recursion, are best introduced with a short presentation by the teacher. By carefully motivating and introducing the new concepts, the class will much better understand them. Furthermore an additional iteration of the topic is achieved.

– Even though the pupils work mostly on their own, a certain control is often helpful. By letting the pupils present some of their solutions to the teacher, their understanding of the new concepts can be controlled and additionally it shows emerging problems as early as possible. Also hints and further explanations can be given to the students to guide them or to even improve their programs.
– If a pupil wrote a good program, let him or her present that solution to the entire class. By discussing the structure of the program and the approach taken, the students learn by good examples. Also this brings more interaction and variation to the class.
– The previous experience in programming of the individual pupils is often very diverged. Since all the details of each lesson are clearly written down, its possible that every student works in his own speed. For the very experienced pupils additional exercises or puzzles can be provided.

4.2 Experiences Made with the Teaching Material

The pupils who worked with the module were between 15 and 19 years old. In some schools we acted as teachers, in others we did only support their ordinary teacher. We present the experience we made during the lessons at these schools.

At the schools where we did not teach ourselves, the concerning teacher did usually neither have any experiences with LOGO nor with this special way of teaching. We only provided the teaching material and an introduction including tips and advices we have collected during our own teaching with the module. Additionally we would have had provided support in case of problems or when further information was needed. However, this was actually never used. The fact that in most cases the teachers could use the teaching material without any additional support shows that the material is convenient for teachers and does not contain any gaps or obscurities.

The following list summarizes knowledge we have gained while teaching programming with the module:

– As long as the students are motivated to work with the module they make progress. It has been shown that motivation is sustainable during at most 6 lessons. We therefore recommend having a break after the first six lessons. The best solution might even be to teach lessons 1 to 6 at the beginning of secondary school and the rest at the end of secondary school (age 16 to 19).
– We observed differences in the way female and male pupils work with the learning material. Female students basically read the text without skipping any sections. They usually solve all the prescribed exercises carefully. Male students, on the other hand, often use the trial-and-error method by skipping reading parts and exercises. As soon as they realize that something is missing they go back in the module. However, both ways of learning are possible and the material is suitable for both.

- Especially in lesson 4 (Drawing Circles and Regular Polygons) and lesson 12 (Recursion) the students become very creative. They individually start drawing new figures, which are not part of the module. We consider it as important not to prevent the students from creating invented figures but to give them the freedom to design fancy graphics. By getting the chance to include own ideas the students are more motivated during the learning process.
- During periods when students work individually they set their own work pace and work independent of their colleges. It is interesting that students who have realized that they work slower than others don't get affected by their faster colleges, as one might expect, and continue to work in their own pace.
- Generally we can say that the material provided a pleasant atmosphere in the classes. More and more we observed that students help each other, they create complex drawings together, discuss ideas and solutions and they consider the learning material as fun.

5 Future Work

We have already written other modules of the same style for different topics of informatics. Even more textbooks are planned or have already been started.

The first textbook *Lehrbuch Informatik* [5] contains additionally to the introductory programming course in LOGO presented here, two more modules: *History and Concept Formation* and *Methods for Designing Finite Automata*.

6 Conclusion

In this paper we formulated our strategy for creating textbooks and lecture notes and illustrated it by showing a few details of the teaching module *Programming in LOGO*. After that we presented our experience with its use. The main points can be summarized as follows:

1. The length of the text is not necessarily correlated to the time one needs to master the subject presented there. Therefore it is not a good idea to start writing textbooks with given size limits by the publisher. Take as much space as the ideas need to present them understandably.
2. Use the language of the pupils in all explanations and take care on developing it. Also check the success of this effort. Your text has to be suitable for individual learning (similarly as "ETH-Leitprogramme"). After each step forward check and deepen the acquired knowledge by posing questions or exercises.

   The reason of going into detail is not to reduce the interaction between the teacher and the class by using the texts for individual study or to fix the program of the lessons too much. The opposite is true. The detailed lecture notes make the teacher free to use alternative ways of explanations

and different didactical approaches. Nothing can go wrong, because if a pupil does not understand everything developed in the lesson, she or he can get full information by reading the lecture notes afterward. These further iteration can be performed by individual speed which is hard to assure when working with the whole class.

3. Test your materials in classes before publishing them. Your experience with possible troubles when teaching complex subjects has to be communicated to the teachers in your textbook. The textbook has to be a valuable source for pupils as well as for teachers.
4. You do not start to think about the choice of the didactic method for particular parts of your lessons before you found a very clear way of approaching your teaching goals from the subject point of view.

References

1. Frey, K., Frey-Eiling, A.: Allgemeine Didaktik (2004)
2. Kalaš, I., Blaho, A.: Exploring visible mathematics with IMAGINE: Building new mathematics calculus with a powerful computational system. In: Learning in School, Home and Community, pp. 53–64 (2002)
3. Kalaš, I., Blaho, A.: Young students and future teachers as passengers on the Logo engine. In: Secondary School Mathematics in the World of Communication Technology, pp. 41–52 (1977)
4. Papert, S.: Mindstorms: Children, Computers and Powerful Ideas, 2nd edn. Basic Books, New York (1993)
5. Hromkovič, J.: Lehrbuch Informatik: Vorkurs Programmieren, Geschichte und Begriffsbildung, Automatenentwurf. Teubner, Wiesbaden (2008)