Tapir, 1998, 195–203

Exploring didactic models for programming

Jens J. Kaasbøll
Department of Informatics, University of Oslo
P.O.Box 1080 Blindern, 0316 Oslo, Norway
Tel: +22 85 24 29
E-mail: jens.kaasboll@ifi.uio.no
1/2–1999

Abstract
Models for teaching programming have been suggested in the literature. Interviews of teachers indicate that they normally do not relate to such models. Since referring explicitly to didactic models in teaching may improve learning, using models seems preferable. Models concerning learning and teaching are suggested in the paper.

1. Introduction

High failure rates in many introductory programming courses may be symptoms of the limited research having been invested in the area. While pedagogical research has provided theories of learning and teaching that are useful in informatics education, each subject requires specific teaching practices (Stodolsky, 1988), and these have to be based on the subject itself.

Concerning learning of programming, Booth (1992), in a qualitative study of how university students, found that students' competence could be described in four advancing stages. Spohrer and Soloway's (1986) studied student-generated programs, and found that learners had more trouble putting the pieces of a program together than learning the programming language concepts. Pea (1986) observed that a substantial proportion of college students assumed that there is a hidden mind somewhere in computer programs.

On the teaching side, experiments with LOGO (Papert, 1980) provided knowledge of how kids inductively learn the imperative kind of programming which LOGO supported. This research is less relevant for adult students being introduced to programming. In tertiary education, developing software support for teaching, eg, specialized programming languages (Brusilovsky et al, 1997), has been a popular research area. Many teaching innovations have been suggested but not properly evaluated (Carbone and Kaasbøll, 1998). In a notable exception, Linn and Clancy (1992) demonstrated that when including expert programmer comments in teaching, the programming process was superior to using only expert code and to make students create programs on their own without the comments.

This study aims at developing a didactic model of how to teach introductory programming, based on exploratory interviews with teachers and the available literature. "Didactics" is here used in the broad sense of pedagogy of a subject matter, and not in the sense of a preprogrammed sequence of instructions. The didactic model is aimed at being

- a meta-model to be taught to students, so that they can verbalize more of their learning,
- a model for teaching to be taught to the tutors in a course, such that they can align their activities with the lecturers,
- a basis for formulating research questions for further studies of programming teaching.

While the model is intended for object-oriented introductory programming teaching, only details will depend on this programming paradigm.

Didactic models found in the literature will be described in the next section, prior to the interviews. Thereafter a discussion will lead to the didactic model.

2. Didactic approaches to programming

Three didactic models for introductory teaching of programming have been found in literature. A didactic model is based on the subject matter and may structure a whole course or be repeated within a larger framework.

2.1 Semiotic ladder

This approach to teaching is based on the language-like features of computer tools: programming languages, modelling languages, and formatting, formula, and search instructions that can be combined in general software products. The teaching and learning sequence starts out from syntax, and proceed to semantics and pragmatics of the language-like tools, see Figure 1. Its rationale is that syntactical knowledge is needed to express anything, and therefore it should precede the learning of meaning of the language constructs. When knowing the meaning, the students can start to learn how to use the language for specific purposes, which is the pragmatics.

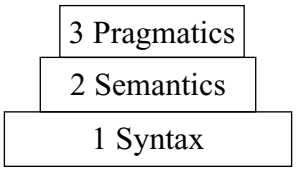

Figure 1. Knowledge of programming languages

Koffman's (1986) introductory book on programming is an example of the semiotic approach repeated for every language construct in Pascal. Liebenau and Backhouse (1990) use the semiotic ladder as their main structure of an introduction to information systems.

2.2 Cognitive objectives taxonomy

Others (Kirkerud, 1996; Reinfelds, 1995) have used a teaching strategy that resembles Bloom's taxonomy of cognitive objectives, illustrated in Figure 2. The sequence of instruction comprised using an application program, reading the program, an changing the program. Creating a program may also be added.

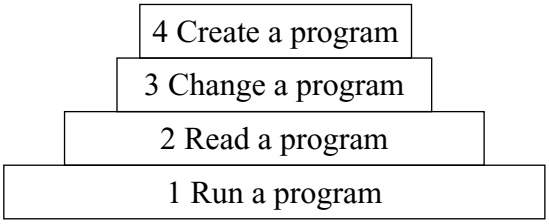

Figure 2. Cognitive objectives taxonomy

2.3 Problem solving

Rogalski and Samurçay (1990) argue for a problem-solving process as the way to the learn programming, see Figure 3. Through solving problems, the students should extend their experience and repertoire of practice, and the basis for the process is the knowledge structure of the field of programming. The problem solving process is guided by methods and environments. Compared to the previously mentioned approaches, this one stresses the input and outcome of the learning process in terms of knowledge and personal experience. It is therefore more a model of learning than of a teaching strategy. A teaching strategy for problem solving could be the "change a program" step in the cognitive objectives taxonomy approach, or the whole course could be organized as a set of problems to cope with.

3. Interview

Teachers of introductory programming subjects at university departments in an

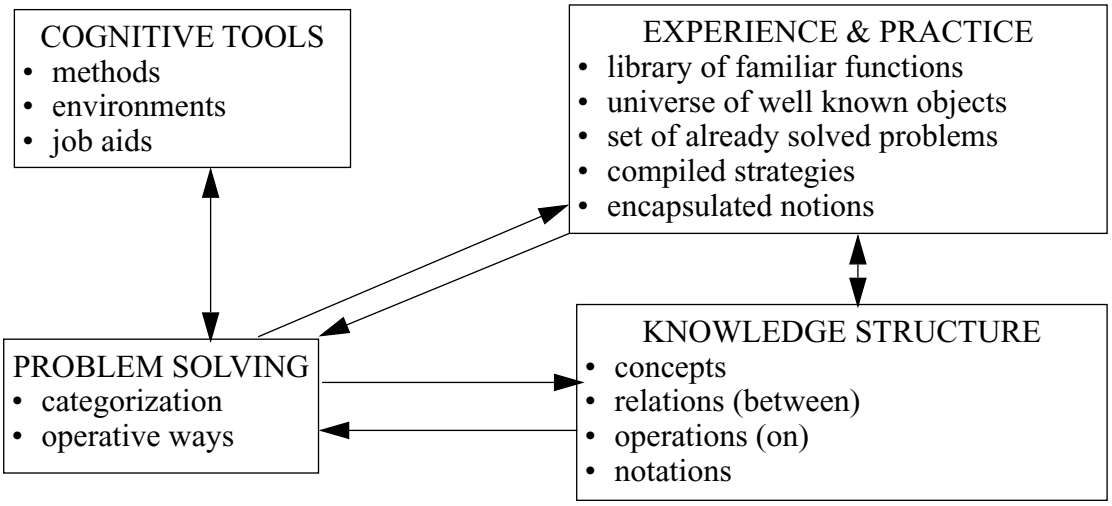

Figure 3. Acquisition of programming knowledge and skills (Rogalski and Samurçay, 1990) .

Australian city were interviewed late 1997 in order to explore teachers' preferences with regard to teaching models and their views on the three models described in the previous section. Since the aim was to initially explore the area, qualitative interviews were chosen.

In a quantitative survey the number of respondents can be calculated based on the level of statistical validity that is required. The explorative investigation aims at covering the breadth of responses rather than a representative selection of respondents. An indication of the number of interviewees required for explorative purposes is that when nothing new appears in a couple of consecutive interviews, no more can be gained from further interviewing. This situation appeared after the third and forth interview. However, appointments for a fifth interview had already been made, so this was also carried out. Surprisingly, this interview brought up new issues, such that more interviews where these issues were considered should have been added. However, due to practical reasons, no more interviews could be carried out.

At two of the institutions, more than one teacher participated, such that these interviews had more character of group discussions. The interviewer took notes during the interviews, and no other means of recording was used. The interview guide is found in appendix, and the models in section 2 were also explained by means of figures.

First, questions regarding the aim, size and duration of the subject, and the teachers' experience were asked to understand their setting. The teachers were lecturers or senior lecturers and had 1–10 years of experience in teaching the subject. All except one subject lasted for one semester with 1/4 workload for the students and 4–5 hours lecturing, tutoring, and labs. The number of students varied from 80–400, with a corresponding number of tutors involved.

The subjects aimed at skills and understanding of programming and problem-solving. The exams consisted of writing or changing programs in addition to multiple choice questions. Assignments usually counted as parts of the grade.

When asked what models or strategies of teaching they used, and how they conceived student learning, most of the teachers did not come up with any answers. The interviewer then presented the three models from section 2, and asked whether any of those corresponded to the way they organized their teaching. The models triggered extensive answers and discussions within the groups. Each model was preferred by at least one

teacher, and some said that their teaching method contained aspects of two of the models. No model was included by all teachers.

In the last department visited, the teachers explained their main approach as a software development cycle. Teaching the relation between specification, program and tests was central to their approach.

In order to find out how the teachers coped with the topics being most difficult to learn, they were first asked which topics they thought were difficult, and then how they taught these issues. Parameters were mentioned by four teachers and other language related topics like pointers, loops, and I/O were also emphasized. The teachers used different tricks of instruction to teach these issues, eg, two of the teachers told that they made the students go shopping as an exercise in function calls and parameter passing. The interviewer did not ask for relations between the model of teaching and the difficult issues, and none of the teachers who mentioned parameter passing brought up the issue.

The exception was again in the department with the software development approach, where problem solving and

> "Students don't know where to start"

were the difficult topics emphasized by the teachers. These are problems related to the programming process and not to language features. The starting problem seems to be in alignment with Spohrer and Soloway's (1986) findings, which emphasized "putting things together" as a main learning threshold, rather than misconceptions about language constructs.

The main lesson from the interviews was that none of the three suggested models had significant advantages, and that a forth model based on software development process should be developed.

The interviews also addressed issues regarding course organization, but these are not considered in this paper.

4. Discussion

4.1 Problem solving revisited

Barnes, Fincher and Thompson (1997) have developed a four-stage process of problem-solving which they have used in computer science teaching:

- Understanding
  - Structuring and dividing
  - Clarifying
  - Finding sample I/O
- Design
  - Finding related problems and solutions
  - Checking against I/O
- Writing
  - Completing
  - Adaptation to problem
- Review
  - Testing
  - Summarizing lessons

This model was not known to the interviewer. In the first step the students analyze the problem description through asking questions about it, challenge unclear aspects, write sample input/output etc. In "design," they look for related problems and solutions, adapts these to the problem given, and checks for completeness and consistency by means of their input/output samples. In the writing stage, the adapted design is completed to

executable code, which will constitute a simplified solution. More details are added or modified to make the program match the problem description. In the final "review" phase, the students test and look back at the finished product and summarize the lessons learned.

Their model seems similar to the software development cycle model brought up in the last interview.

Compared with the cognitive objectives taxonomy, Barnes et al's (1997) model incorporates reading and changing programs through design and writing. Through focusing also on introductory analysis and concluding evaluation, the problem solving model may be even better suited for students to see how the learning goals are related to the teaching.

In the problem solving approach, human and organizational issues may be included in the problems to be solved and the methods, and thereby in the teaching. The programs and models may be made less abstract through selecting problems with which the students are familiar, both through their knowledge of the domain and through having seen similar programs before.

An objection against the problem solving approach is that it resembles a waterfall model of program development, thereby hinting that programs are developed without iterations of the phases.

Barnes et al (1997) also raise the issue that the strategy of programming conveyed by their model is more of a "hacking" manner of copying and adaptation than a "structured, stepwise refinement." They counter this argument by saying that students who don't know what the solution should be cannot apply the structured approach.

This issue seems to point to the process of learning to program being different from the process of software development, thus requiring reconsideration of the software development model in teaching.

4.2 A learning process

In her study of introductory students, Booth (1992, pp.119ff) found that they conceived learning programming in four ways, in advancing order:

- Learning a programming language
- Learning to write programs in a programming language
- Learning to solve problems in the form of programs
- Becoming part of the programming community

While the two former stages concern coding skills, the third one requires that the student is also able to apply the skill in the appropriate manner when confronted with a problem not being expressed in programming terms. "Becoming part of the programming community" means, eg, considering oneself as one member of a programming team that also interacts with users. In Booth's study, only some of the second term university students conceive programming this way.

The students in Booth's (1992, pp.207ff) study approached the problem of producing code based on a textbook exercise in four ways. In an order of being more advanced, the approaches were:

- Expedient. Producing a complete program from the outset by making use of an existing program.
- Constructional. Recognizing details of the problem in terms of features of the programming language.
- Operational. First interpretation of the operations that the program has to perform. Then writing the program.

- Structural. First interpretation of the problem within its own domain. Then structuring. Finally coding.

During their study, the students in general moved to more advanced levels, and those who did not move tended not to pass the course. This observation implies that the teaching should train the students in interpreting the problem in its domain and structuring the program. Learning this way of programming may be strengthened by presenting programming as problem solving.

However, interpretation and structuring does not necessarily help the students overcoming their starting problem. Without knowing the pieces from which a program can be constructed and the common patterns of putting these together, the students have no chance of structuring a program, even if they are able to discuss the problem domain. A way out is to look for similar problems and programs and adapt these to the problem at hand. In Booth's terms, that is going back to the expedient or constructional approach. Pinpointing this, we may say that learning is hacking, while programming in a structured way means that you have no more to learn.

When starting to learn programming, learners makes mistakes of the type Pea (1986) calls the "superbug:" they use the programmng language as if the computer were a person. Eg, a student at our introductory programming course wrote

```
while > 0 do ...
```

and could not understand why the compiler refused the code. The tutor asked the student: "What is going to be greater than zero?"

Student response: "It"
Tutor: "What it?"
Student: "There is only one x in the program"

Since not having learnt otherwise, the students apply the human principles of thinking and acting to the computer. We may therefore say that the first step of learning programming is going from human thinking to programming concepts. This includes understanding the way programs are executed, both in terms of internal variables and external files and I/O.

Putting pieces of code together was reported by Spohrer and Soloway (1986) as the main obstacle in learning programming. Therefore, there seems to be a step from understanding the programming concepts to mastering the skill of coding.

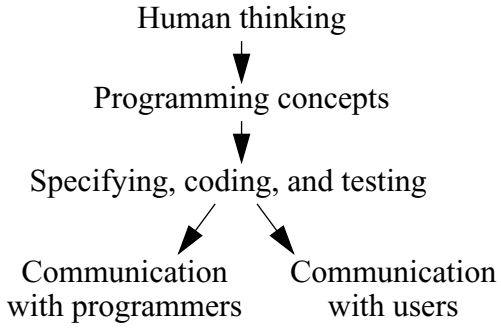

Figure 4. Learning programming

According to Booth, regarding programs as means of communication between programmers is a more advanced level of competence. Correspondingly, understanding programs as means for communication between programmers and users is another level of understanding.

The stepwise model of the process of learning programming shown in Figure 4 is therefore suggested.

4.3 Teaching for learning programming

Building on Barnes et al's (1997) model and the software development cycle from the last interview, and iterative software develoment approach to teaching is suggested, as illustrated in Figure 5. The model is not intended to deal with the sequence of instruction, rather being a map of activities and corresponding competence. The main idea to teaching is to enable the students to learn an iterative process of construction and

evaluation through teaching its structure and parts. The cycle is driven by problems, which could include requests for changes in existing programs, as in the cognitive objectives taxonomy model. The construction (to the right) and evaluation (left) phases are supported by specific methods and concepts.

4.4 Models as tools for teaching

Explicit models of learning and teaching of may help the teacher coming to grips with underlying assumptions of specific ways of teaching.

Students and tutors develop their attitude and behavior of programming and of learning and teaching programming, regardless of whether these issues are taught explicitly or not. Explicit models can help learners conceptualize the issues in a more useful and uniform way. Student discussion is also a good learning activity, and providing the students with concepts concerning the teaching and learning processes may give them better ways of verbalizing the issues.

The steps from problem analysis to program test in the iterative software development model are intended to aid the learning process from human thinking to coding. According to Barnes et al, letting the students work with a problem in their mother tongue before coding, reduces the distance between what the students are familiar with and the coding process. Making the students specify variables and data structure also delays the actual coding.

Inspection and use tests are intended to aid learning the advanced levels of programming competence: communication. Code inspection is an activity that can support learning of programs as means for communication between programmers. Learning communication with users is supported by user testing and negotiating specifications with users.

The hacking style of coding that learners tend to adopt is supported in the model by explicit teaching of finding, copying and modifying existing programs.

Compared to the cognitive taxonomy approach, iterative software development also demonstrates how evaluation provides knowledge about programs. The students thus get exposed to a way in which knowledge of programs is developed in informatics. When learning in this manner, the learning process is turned from knowledge consumption to knowledge generation. Ben-Ari (1998) and Hadjerrouit (1998) have argued that such a constructivist approach to programming education is essential for proper learning.

5. Conclusion

Five exploratory interviews with teachers of introductory programming in universities indicated that suggested models for teaching had not been adapted, and that most courses had no clear teaching model.

In this paper, models of two kinds have emerged: A learning model (Figure 4) and a teaching model, and how knowledge can be produced during software development (Figure 5).

Empirical studies are necessary to validate and develop these. Since providing students with concepts for verbalization in general supports learning, including such models in teaching may be beneficial regardless of whether the model is optimal.

Acknowledgement

Thanks to the lecturers for spending their time and sharing their experience.

References

[1] Barnes, D. J.; Fincher, S.; Thompson, S. (1997) Introductory Problem Solving in Computer Science. In Daughton, G. and Magee, P. (eds.) *5th Annual Conference on the Teaching of Computing,* Dublin City University, pp.36–39

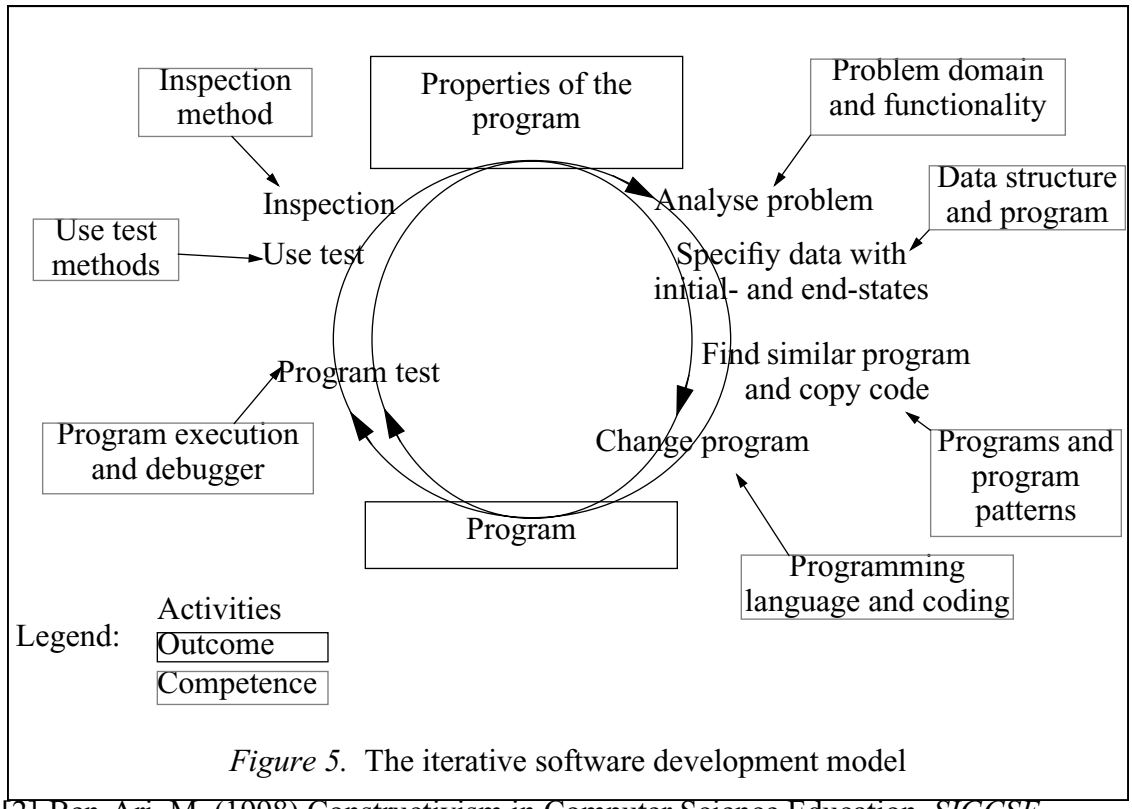

Figure 5. The iterative software development model

[2] Ben-Ari, M. (1998) Constructivism in Computer Science Education. *SIGCSE Bulletin,* 30, 1, 1998, 257–261

[3] Booth, S. (1992) *Learning to program: a phenomenographic perspective.* ACTA Univ. Gothenburg studies in educational science 89

[4] Brusilovsky, P.; Calabrese, E.; Hvorecky, J.; Kouchnirenko, A.; Miller, P. (1997) Mini-languages: a way to learn programming principles. *Education and Information Technologies,* 2, 1, 65–83

[5] Carbone, A. and Kaasbøll, J. (1998) A survey of methods used to evaluate computer science teaching. *Proc. of the 6th Annual Conference on the Teaching of Computing.* Dublin City University, 41–45

[6] Hadjerrouit, S. (1998) A Constructivist Framework for Integrating the Java Paradigm into the Undergraduate Curriculum. *Proc. 6th Annual Conference on the Teaching of Computing.* Dublin City Univ., 105–107

[7] Kirkerud, B. (1996) "Use—Read—Change" A new introductory course in informatics. In Krogdahl et. al. *Norsk Informatikkonferanse.* Høgskolen i Finmark, 173–180

[8] Koffman, E.B. (1986) *Turbo Pascal: A Problem Solving Approach.* Addison-Wesley, Reading, Mass.

[9] Liebenau, J. and Backhouse, J. (1990) *Understanding information : an introduction.* London, Macmillan;

[10]Linn, M.C. and Clancy, M.J. (1992) The case for case studies of programming problems. *Commun. ACM,* 35, 3, 121-132

[11]Papert, S. (1980) *Mindstorms: Children, Computers and Powerful Ideas.* New York, Basic Books

[12]Pea, R.D. (1986) Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research,* 2, 1, 25–36

[13]Reinfelds, J. (1995) Logic in first courses for computer science majors. In Tinsley,

J.D. and van Weert, T.J. (eds.) *World Conference on Computers in Education VI: Liberating the Learner.* Chapman & Hall, 467–477

[14]Rogalski and Samurçay (1990) Acquisition of programming knowledge and skills. In Hoc, J.-M (ed.) *Psychology of programming.* London, Academic Press

[15]Spohrer, J. and Soloway, E. (1986) Novice mistakes: are the folk wisdoms correct? *Commun. ACM,* 29, 7, 624–632

[16]Stodolsky, S.S. (1988) *The subject matters: classroom activity in math and social studies.* Chicago: University of Chicago Press

APPENDIX: Interview guide

- Subject
  - Pre requirements, credits, hours lecturing, tutoring, lab
  - Number of students, lecturers, tutors
  - The lecturer's practice of teaching the course
- Learning
  - Main educational goal, competence aimed at
  - Teaching model or concepts, skills, structure of these
  - Learning models, theories, image, metaphors of how students learn. Eg, Sequence, conceptual, practice
- Difficult topics to learn
  - Which ones? Threshold
  - Ways of teaching these topics. Visualization?
- Getting the tutors aligned to the teaching ideas
  - Preparation before the course, meetings
  - Topics being discussed
- Exam
  - What do you intend to measure?
  - Question and students' answers