1 CONCISE SUMMARY OF THE DISSERTATION

In the first chapter of the dissertation I outline the history of subject ‘informatics’: what the purpose of teaching was, what tools it had for teaching before informatics became a subject, what changed after that, and how it is different now. I refer to well-known, but arguable examinations (PISA- [“Knowledge and Skills for Life – first results from the OECD PISA 2000”, OECD, 2001; http://www.pisa.oecd.org/], IAE-report [„IEA Newsletters Number 34”, IAE, 1999; http://www.iea.nl/Home/IEA/iea.html], minutes of the meeting of “Országos Köznevelési Tanács” [OM in the column “EU és oktatás” 2003/2: “Gondolkozz Európában! – A tudás uniója”; http://www.om.hu] etc.) that criticize the state of education in Hungary at the turn of the century. I survey the methods for teaching programming [SzPZsLD, SzPZsLM]. Each of these departs from a characteristic idea or concept in programming (e.g. algorithm, data, programming language etc.), and deduces every question of detail from this. I qualify these methods: what advantages and disadvantages they have. In my esteem, the specification has a significant role in programming; hence I dedicate a chapter to its concept [SzPPPrSp]. In this chapter I deal with several basic concepts of programming, moreover the causes and the facts of their formation, and their history. The concepts include algorithm, code, program and problem specification, proof of correctness, programming theorem and transformation, data type, object, data class specification etc.

The 2nd chapter is intended for didactic investigation of program development in “classical”, so-called Neumann languages. The attribute “classical” indicates, on the one hand, the category of classical languages, and, on the other hand, my decision to deal with the didactics of the one-module programs only. The didactical problems, however, enmesh the whole process of program development, so they affect multiple-
module programming as well. I list the following concepts to **mental toolkit**:

- lingual abstraction;
- analogy;
- algorithmic abstraction;
- decomposition;
- superposition;
- conversion;
- intuition;
- variation.

The body of this chapter is made up of the particular analyses of their role and significance in programming [SzPÖ, M12, M18, M19, M21, SzPPrSp, SzPPrSp, SzPGyPr]. In the argumentation I apply the results of the related disciplines. I refer, for example, to Pólya’s mathematical-didactical works, Láboš’s and Lénárd’s studies on the psychology of teaching-training and on the exploration of thoughts, and Chomsky’s bestsellers, in which he reflects on the connection between language and thinking. In addition, I build my arguments on such notable authors as Knuth, Dahl, Dijkstra, Hoare, Cormen, Rivest, Horowitz, Koster etc., who made the basis of programming and clarified its idea and concepts.

The central issue of the 3rd chapter is **data abstraction**. I examine the concept ‘type (class)’, which embodies the abstraction. I regard the concept ‘module’ as a tool of implementing the type. In this sense, it can be declared that this chapter deals with developing the multiple-module program, that is, **modular programming**. In addition, the informal introduction of a formalism elaborated by the author is located in this chapter. This formalism makes type-defining fit into the medium of our algorithmic language. I put the emphasis on demonstration, not on the exact definition, of the following concepts: ‘export-module’, ‘series-language’, ‘representation-implementation module’, ‘memory models’ etc. [SzPTSp, M34, MSz6, PGSzPADT]. I unusually introduce **data recursion**: it is one of the type-constructors [M27]. My conception of the recursion is **explicit**, unlike in other programming languages that define and use it implicitly only. This is evidently a didactical advantage of my formalism. In this way, I define several type-constructor, namely ‘files’, ‘graph’ [M38] and ‘table’ [MSz8, M34]; and I introduce the concept of a new type-constructor, namely that of the ‘abstract series’ type-constructor [MSz6]. Here I broach a connection between algebraic and functional specification [MSz6, M34/Appendix].

Here I broach a connection between **algebraic and functional specification**. [MSz11]

In the 4th chapter I refine **formalisms** that I have mentioned in the previous chapter, which are the ‘core-language’, the extension of the module,
and the algebraic type-specification language. After describing them, I analyse some concepts, including invariant state, programming theorem and transformation [HSzZs, SzPPTr], because they have a great didactical importance. For the same reason I sketch my method for checking the compatibility of the algebraic and functional specification of the type (class).

The last chapter is intended for demonstrating the benefit of programming at school. I discussed:

- the role of the ‘lingual abstraction’;
- the connection between ‘creativity’ and ‘encyclopaedity’, in special consideration of the programming theorems;
- the significance of the ability of making algorithm in promoting the ability of making models [M1, M9, M17, SSzZs, SzZsBio]; and
- the benefit of the type- (or object-) oriented programming in problem solving.

This part concludes with the results of an examination, the object of which was measuring the progress in the knowledge of some student-groups.

2 SCIENTIFIC RESULTS

I group my scientific results according to the following considerations: (1) original approach, conceptual systematisation; (2) integrated aspect, uniform formalism; (3) methods, didactic handholds; and (4) the examination of progress with statistical calculations.

2.1 Original approach, conceptual systematisation

So far, scientifically accurate didactic examinations concerning programming have been an open and undiscovered area of science. Scientific literature widely related to programming (e.g. psychology, cognitive psychology, and psycho-linguistics) deals with these didactic topics in general, only. I refer mainly to Pléh’s works, for they include such examinations, but his questions and answers are undeniably put from the psychologist’s point-of-view. In addition, if a work happens to touch on some topics of programming, it does so far too cursorily. Krajcsi’s paper can be mentioned as a characteristic example. In this work, the author studies the success of two student-groups in the Wason’s selection task. The first group consists only of students interested in humanities, while the second includes only students with technical interests.
I adapt the concepts of particular mental tools (like lingual abstraction, and analogy) from other branches of science to algorithmic thinking, and I define some other concepts (like decomposition-superposition, conversion, and variation); and I complete all of these with further well-known concepts after Dahl, Dijkstra, Hoare and Knuth (algorithmic and data abstraction etc.).

2.2 Integrated aspect, uniform formalism

Programmers use several idioms, most of which are designed to only a phase of program development. Consequently, they can function as a methodological aid for just a few steps. (For example the flow-chart, the Jackson’s diagram, the structogram, Z, VDM, and so on.) There are some efforts, however, in which formal systems have arisen not only to one, but to each phase of program development. (An example is UML.) But these are too complicated or too specific to serve as a tool in the instruction of programming in Hungary.

During Zsákó’s and my researches, we have elaborated a formal system to write algorithm for beginners, which became named as “core language”. My efforts made the extension of the language crystallize, so that it can be used as an exact tool for defining types (classes).

An elaborate didactics belongs to this idiom, which can be found in several publications from me and my colleagues. In these issues, we analyze the didactical topics, dividing them into items. The topics include: programming theorems, data types, string and data processing, etc.

2.3 Methods, didactic handholds

The paper sets up several important didactic principles and applies a few as methods, the primary of which is the multiple-level method of incremental and iterative program development.

- core language – classical concepts of structured programming, programming theorems, program transformations and steps of proving program correctness; in addition, the method, based on the rules of code transformations, of translation into a programming language;

- types (classes) – algebraic tool (see in the chapter “2.2.2. Algebrai megközelítés – algebrai specifikáció” at the “módszertani fogó-dzók”); the export (see in the chapter “2.2.3. Első algoritmus megközelítés – félület szerinti specifikáció, exportmodul” at the “módszertani fogódzó”) and the representation-implementation module, equipped with the asserts of functional specification; and finally, the
code (see the chapter “3.5. A kódolási szabályokról...”); in addition, the method for examining the consistency between algebraic and functional specifications presupposes that the students are familiar with the Prolog language.

- The method for the examination of the consistency between algebraic and functional specification supposes that the students know Prolog language. (See the chapter “2.2.5. Kapcsolat az algebrai és a funkcionális specifikáció között” and “3.6.3. Kapcsolat az algebrai specifikáció és a modulspecifikáció között”)

- Coding is necessary during the use of both the core language and the development of types. Therefore, the knowledge of the conceptions and practices in coding is important on each level of programming. (See the chapter 3.5.)

2.4. The examination of progress with statistical calculations
The paper reveals the actual state, and consequently, the actual problems of the teaching of programming.

The paper sets up several hypotheses on the connection between the subject “Informatics” and other subjects and skills. Finally, I examine some of these hypotheses with the tools of mathematical statistics.

3 PUBLICATIONS IN THE TOPIC OF THE DISSERTATION
The referred publications are marked with red character on grey background.


Szlávi Péter: Didactic problems of Program Development


[MSz1] Szlávi P.: „A tűzifogalom fejlődése az algoritmus nyelvekben”, Útőgia Szilánkok1, ELTE TTK Informatikai Tanszékcsoport, 1992


[MSz6] Szlávi P.: „Adatok, adattípusok”, Útőgia Szilánkok6, ELTE TTK Informatikai Tanszékcsoport, 1992
Szlávi Péter: Didactic problems of Program Development


In the Web:

Szlávi Péter: Didactic problems of Program Development

