SERBIAN ACADEMY OF SCIENCES AND ARTS
Committee for Education of the Presidency of the Academy

Book of Abstracts

M. Marjanović, Đ. Kadijević
S. Jokić & J. Vučo (Eds.)

ISDTF 2011
20–21 October 2011
Belgrade, Serbia

SERBIAN ACADEMY OF SCIENCES AND ARTS

COMMITTEE FOR EDUCATION OF THE PRESIDENCY OF THE ACADEMY

Book of Abstracts

ISDTF 2011

Improving specific subject didactics at the teacher training faculties

20–21 October 2011**,** Serbian Academy of Sciences and Arts, Belgrade, Serbia

Edited by

Milosav M. Marjanović
Serbian Academy of Sciences and Arts

Đorđe M. Kadijević
Mathematical Institute of the Serbian Academy of Sciences and Arts

Stevan Jokić
Vinča Institute of Nuclear Sciences

Julijana Vučo
Faculty of Philology, University of Belgrade

SASA, Belgrade, 2011

A VIEW ON TEACHING INFORMATICS BASED ON PROGRAMMING AND PROGRAMMING-LIKE ACTIVITIES

Boyko Bantchev
Institute of Mathematics and Informatics, Sofia, Bulgaria
bantchev@math.bas.bg

A programming-based introduction to informatics in some countries is gradually being displaced by instruction in using application software because the majority of students and many of their teachers are viewing programming as both too difficult and boring. Convinced that such a perception stems to a great extent from the unsuitability of the programming languages currently used for teaching, I propose replacing them with more appropriate ones, as well as exploiting other tools whose use is akin to programming.

Popular languages, such as *Basic*, *Pascal*, *C/C++*, and similar are heavily used for educational programming, but are verbose, 'bureaucratic', and hinder direct expression of data and algorithmic structures. Teaching informatics using such languages mostly degenerates into struggling with their syntactic intricacies and imparting the 'incantations' needed for running simple programs. Little room is left for conceptual modeling, building abstractions, thinking over alternatives, and generalization.

Concentrating on exploring fundamental algorithmic and data structures, which I believe is the essence of the introductory teaching in informatics, is much more workable if a dynamic language, e.g. *Ruby*, *Python*, *Lua*, or *Icon* is used. These languages facilitate educational and exploratory programming by featuring rich data structures, well designed algorithm-building blocks, an unobtrusive and readable syntax, and interactive environments for executing and verifying a program code in pieces before assembling it as needed. Significantly more substantial and thematically varied programming can then be experienced, resulting in a straightforward and succinct code.

Functional languages, such as *ML* and *Haskell*, are also worth considering as tools for educational programming. Besides being expressive and concise, they are characterized by higher-level, declarative semantics, treatment of functions as first-class program objects, and rich, polymorphic, yet automatically deducible datatypes. All of these are of great conceptual and practical value, fostering a clear and concise style of expression.

A notable example of a small language combining a functional style with picture composition facilities, intended as a tool for teaching fundamentals of computing to young pupils, is *GeomLab*.

Using programming languages is not the only option for teaching organization and algorithmic processing of information. Carrying out problem-specific computations, such as planning how to build a textual, geometric, purely symbolic or other structure, using scriptable application or tool programs, can be an activity no less meaningful and inspiring than 'true' programming. Even a capable text editor, used knowledgeably in solving

text processing problems, can lead to rewarding intellectual and practical accomplishments.

Regular expressions (regexes) are a good example of an algorithmic tool for text manipulation. Their use spans not only programming languages but also many utility programs, including most modern text editors. Constructing a regex to match a specific text structure – a number, a file name, an e-mail address, a sentence in a natural language – is a kind of small-scale, but potentially demanding programming task.

There are many other utilities widely used for extraction, splitting, rearranging, merging, etc. of textual information. Combining several of them to solve specific problems provides a range of creative and engaging programming activities that can be used as a teaching resource.

Furthermore, the realm of text processing integrates well with other kinds of utilities or application-specific languages, such as those for diagram generation, notably *dot* for abstract graph drawing and *gnuplot* for scientific visualization. These and other programs accept declarative text input describing an image of the respective type, and therefore their use can be automated by piping other programs to them. This also presents a good opportunity to explore links with mathematics and other disciplines.

Dynamic geometry systems (DGS) are widely used for exploration and education in mathematics. Some of them are equipped with a built-in language, allowing geometric models to be constructed programmatically, and thus turning a DGS into an excellent tool for teaching informatics as well. Geometry's clear logical and algorithmic substance becomes explicitly available as a domain of programming activity. A good example is *GeoGebra* with its language, featuring geometric construction capabilities along with high-level general programming semantics.

There is indeed a great variety of problem domains to investigate, and means and tools to employ in teaching informatics, each of them making it possible to discover different forms of organizing and processing information. These options currently remain mostly unused. My belief – based on extensive studying of, and practice with, tens of programming languages and many programmable or programming-like tools – is that a number of these languages and tools can be put to successful use in teaching informatics. This is confirmed by my experience in teaching to university, high-school, and younger students.

Teachers should not limit themselves to a specific programming language, and should avoid those that are syntactically or otherwise complicated in favour of ones fostering straightforward expression of ideas, as well as other environments enabling algorithmic activities. Combined use of tools is possible. The described approach may be challenging to the teachers, as it requires continuously educating themselves, but the results should be worth the effort.