

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/291602214>

Towards an Intelligent Environment for Learning Introductory Programming

Chapter · January 1993

DOI: 10.1007/978-3-662-11334-9_11

CITATIONS

10

READS

130

1 author:



Peter Brusilovsky

University of Pittsburgh

507 PUBLICATIONS 17,490 CITATIONS

SEE PROFILE

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



Interactive Intent Modeling for Information Discovery [View project](#)



User-Controlled Hybrid Recommendations [View project](#)

Cognitive Models and Intelligent Environments for Learning Programming

Edited by

Enrica Lemut

Istituto Matematica Applicata, C.N.R.
Via De Marini, 6, Torre di Francia, I-16149 Genova, Italy

Benedict du Boulay

School of Cognitive and Computing Sciences
University of Sussex, Falmer, Brighton BN1 9QH, UK

Giuliana Dettori

Istituto Matematica Applicata, C.N.R.
Via De Marini, 6, Torre di Francia, I-16149 Genova, Italy



Springer-Verlag

Berlin Heidelberg New York London Paris Tokyo

Hong Kong Barcelona Budapest

Published in cooperation with NATO Scientific Affairs Division

Towards an Intelligent Environment for Learning Introductory Programming

Peter L. Brusilovsky

International Centre for Scientific and Technical Information, Kuusinen Str. 216, Moscow 125252, Russia

Abstract: The paper discusses some problems of building intelligent learning environments for novice programmers. An intelligent learning environment ITEM/IP is described. The environment supports a course on introductory programming based on the mini-language Turingal. The mini-language serves as a means of mastering the main concepts of programming, the structures of programming languages and skills in program design and debugging. ITEM/IP integrates a set of tools which support students as they learn to program.

Keywords: Intelligent learning environment, novice programmer, introductory programming, mini-language.

Introduction

Intelligent tutoring systems (ITS) and programming environments can be viewed as two opposite approaches to using advanced computer technologies for learning programming. Intelligent programming environments (intelligent learning environments for programming) attempt to bridge the gap between ITS and programming environments.

Intelligent programming environments are able to offer the flexibility of microworld-like environments accompanied by an artificial intelligent coach or tutor. This paper discusses some problems of building intelligent programming environments, lists some useful components of them and suggests an approach to integrate the components. To illustrate the discussion we describe briefly ITEM/IP environment which supports a course on introductory programming based on the mini-language Turingal. ITEM/IP attempts to integrate the capabilities of a tutoring system, programming environment and on-line reference manual for programming.

1. Intelligent Programming

A variety of computer systems and tools support the stages of learning programming. These include three classes: tutoring systems and programming environments.

A tutoring system supports the whole learning process. It presents teaching material to the student, solving activity and checks the student's solutions. Artificial intelligence (AI) techniques to support learning. Most of these systems are not aimed to decrease the amount of non-creative class work. The systems allow a teacher to have more time for the strongest students.

A programming environment supports the stages of learning. A student is provided with a set of student programs and lets the student modify them in the language. Advanced programming environments use Artificial Intelligence and Cognitive Science techniques to interact with the student.

The following teacher activities \leq techniques:

instructional planning and knowledge representation, task sequencing (BIP [2]), mastery learning (ACT Programming [3]), selection of program example (ELM [4]), scaffolding the student in the process (Lisp-Tutor [1], Bridge [5], GILL [1]), student program checking (BIP [2]), student program analysis or intelligence (ELM-PE [22] and other systems [4]), strategic (meta-level) instruction (M4 [6]).

As we can see, the spectrum of support from one to three of listed features.

The following student activities show

- program design and coding,
- using pre-stored examples and previous solutions,
- program debugging and investigation,
- querying about description of language constructs.



1. Intelligent Programming Environments

A variety of computer systems and tools have been designed to help teachers and students at all stages of learning programming. These systems and tools can be divided roughly into two classes: tutoring systems and programming environments.

A tutoring system supports the work of human programming tutor in a traditional learning process. It presents teaching material to the student, guides or coaches the student problem solving activity and checks the student programs. Intelligent tutoring systems (ITS) employ artificial intelligence (AI) techniques to work at the level of a skilled human teacher. However, most of these systems are not aimed to replace the human teacher in the classroom, but rather to decrease the amount of non-creative classroom work of a teacher in the course of programming. The systems allow a teacher to have more time for creative work with the weakest and the strongest students.

A programming environment supports the work of a student and the process of exploratory learning. A student is provided with a set of tools which supports the design and debugging of student programs and lets the student experiment with the programs and the programming language. Advanced programming environments employ human-computer interaction (HCI) and Cognitive Science techniques to increase students' learning abilities.

The following teacher activities can be supported by ITS tools with the use of AI techniques:

- instructional planning and knowledge sequencing (SCENT-3 [16]),
- task sequencing (BIP [2]),
- mastery learning (ACT Programming Tutor [11]),
- selection of program example (ELM-PE [22], Discover [10]),
- scaffolding the student in the process of program design at all levels from plan to code (Lisp-Tutor [1], Bridge [5], GEL [17], Discover [10], SODA [14]),
- student program checking (BIP [2]),
- student program analysis or intelligent debugging (PROUST [21], SCENT-3 [15,16], ELM-PE [22] and other systems [4]),
- strategic (meta-level) instruction (Molehill [20]).

As we can see, the spectrum of supported teacher activities is quite wide, but real systems support from one to three of listed features.

The following student activities should be supported by programming environment tools:

- program design and coding,
- using pre-stored examples and previous solutions to build a new program,
- program debugging and investigation,
- querying about description of language features (on-line help).

Modern commercial programming environments, such as Turbo-Pascal 6.0, support all these activities at some level. Advanced environments use HCI technologies such as structural editors, multi-window interface, program visualization, and hypertext to support the listed activities.

From some point of view, there is no crucial difference between ITS and programming environments. Both kinds of systems can be viewed as sets of tools which support different teacher and student activities in the process of learning programming. At this level we can see good prospects for a marriage between the two kinds of systems, that is between ITS and HCI techniques [10, 15]. Good progress has been made recently in building intelligent programming environments which can be viewed as a result of this marriage. An intelligent programming environment combines the features of ITS and programming environments and employs both AT and HCI techniques.

First, an intelligent programming environment supports both the activities of a teacher and a student, and includes tools from both the above lists. The following systems provide good examples of this: BIP [2] (task sequencing and program visualization), Lisp-Tutor [1] and ACT Programming Tutor [11] (structural editor and guided program design), Bridge [5] and Discover [10] (guided program design and program visualization), ELM-PE [22] (selection of examples, intelligent debugging, and program visualization), GIL [17] (visual editor, guided program design and visualization), and Molehill [20] (program explanation and on-line adaptive help).

Secondly, an intelligent programming environment attempts to employ both AT and HCI techniques to support all kinds of activities listed above. In particular, it means using AT techniques within the "student" list and using HCI techniques within the "tutor" list. Good examples of this are intelligent program visualization [12] and guided visual programming [17].

There are two groups of problems in the design of an intelligent programming environment: how to use the AT and HCI techniques to design a tool which supports the given student or tutor activity, and how to integrate the given set of tools into a complex environment. The above mentioned systems provide us with good examples of solving problems of the first group. Note that applying AT techniques appears to be more difficult and less cost effective than applying HCI techniques. As a result, most of the existing intelligent programming environments use both AT and non AT tools to support the required set of activities.

The problems of building an integrated intelligent programming environment seem to be less investigated. We referred above to a number of environments which are made up of several specialised tools. However, some of these environments are more boxes of independent tools than really integrated Systems. For example, the task sequencing component and visualizer in BIP [2] are absolutely independent tools. An integrated system ought to be more than just the total sum of a number of specialised **components**. Components of an integrated system should be interrelated. First, one component should be able to use the capabilities of another component as well as exchange or share data with it. Secondly, the results of students' work

with any of the components during the components and used in adapting to the 1 student. Careful design and both AT and features.

To investigate the ways of building ar have designed the Intelligent Tutor, Envi (ITEM/113) [7]. ITEM/IP is an intelllligem Turingal which is used in an introductory this simple language, let us design a coml concentrate our efforts on problems of inti tutor activities: knowledge sequencing, student program checking, counterexaml examples and previous solutions, prograr integrate all the listed tools we used ITS (s techniques. Briefly speaking, the results c reflected in the overlay student model and the knowledge level of the particular s description of system components are prs here the details of ITEM/IP development

2. Learning Introductory Prol

The successful application of Logo "turtly approach towards learning introductory V while studying how to control an exec microworld environment and can change commands. The executive can be contra addition to these commands, includes m condition, etc.). The environment, as well displayed on the screen and thus, the studs immediately. The Karel language [18] language.

While learning a mini-language, the s the informative programs that control fundamental notions of programming are development and debugging are acquires using this language, as well as the ger achieving the desired level of "computer li

with any of the components during the session should be taken into account by all the components and used in adapting to the knowledge level and style of work of the particular student. Careful design and both AI and HCI techniques should be used to achieve these features.

To investigate the ways of building an integrated intelligent programming environment we have designed the Intelligent Tutor, Environment and Manual for Introductory Programming (ITEM/IP) [7]. ITEM/IP is an intelligent programming environment for the mini-language Turingal which is used in an introductory programming course (see the next section). Taking this simple language, let us design a complete environment with a good set of tools as well as concentrate our efforts on problems of integration. ITEMIP supports the following student and tutor activities: knowledge sequencing, task sequencing, selection of program examples, student program checking, counterexamples selection, program editing, access to pre-stored examples and previous solutions, program debugging and investigation, and on-line help. To integrate all the listed tools we used ITS (student models [19]) and HCI (adaptive interfaces [3]) techniques. Briefly speaking, the results of students' work with the pedagogic component are reflected in the overlay student model and used by system components as a basis for adapting to the knowledge level of the particular student. A brief overview of ITEM/IP and a short description of system components are presented in the following sections. We do not discuss here the details of ITEM/IP development which can be found in the paper [7].

2. Learning Introductory Programming with Turingal

The successful application of Logo "turtle graphics" has stimulated the development of a new approach towards learning introductory programming. Beginners learn what programming is, while studying how to control an executive (the robot, the turtle, etc.) which acts in a microworld environment and can change its position and the environment according to a set of commands. The executive can be controlled by means of a simple mini-language which, in addition to these commands, includes more or less complicated structural statements (loop, condition, etc.). The environment, as well as the position of the executive in it, are permanently displayed on the screen and thus, the student is able to see the result of the command execution immediately. The Karel language [18] serves as a good example of an educational mini-language.

While learning a mini-language, the student, from the first steps, begins to write and debug the informative programs that control the executive. During this process many of the fundamental notions of programming are learned easily and quickly, and the skills of program development and debugging are acquired. For many students, the experience they get while using this language, as well as the general feeling of programming, is quite enough for achieving the desired level of "computer literacy".

The method of learning introductory programming with the aid of mini-languages is quite well developed in Russia now. A number of mini-languages are used to support introductory programming courses in Russian schools and universities. The mini-language Turingal (Turingal = Turing machine + Pascal) was specially designed to support an introductory programming course for the first-year students of the Department of Applied Mathematics and Cybernetics of the Moscow State University. This language is an alternative way to control the well-known system of algorithmic theory - Turing Machine works with a tape of symbols. The elementary operations of the mini-language are simple and visual: the movements of the machine head left and right along the tape and typing the symbols on the tape. To control the machine head, Turingal offers a wide range of control structures (conditional statement, loops, case, subroutines) with syntax and semantics similar to the structures of Pascal. The mini-language approach and the language Turingal are described in more detail in [9].

3. Learning with ITEM/O

The following styles of learning programming are supported by ITEM/IP:

1) Exploratory learning. The student can make use of a complete set of tools for program design and debugging known as the programming laboratory. One of the functions of the laboratory is to support visual program execution. With the help of the laboratory the student can observe the "behaviour" of programs, experiment with them and gradually learn from his/her observations and mistakes. The ITEM/IP programming laboratory, like BIP [2], and the ReGIS [13] laboratories, support the exploratory style of learning: design the program - experiment (run the program) - observe - change the program - experiment again, etc. This style of work enables the student to learn the programming concepts and construction from the experience, to "discover" features, and to get an idea of programming technology.

2) Guided learning and free learning. To the inquiry of the student, ITEM/IP selects the currently optimal teaching operation and presents it to the student: it explains new programming concepts, demonstrates semantics of programming language construction, or tests the understanding of it, presents an example solution for a relevant programming problem or suggests a problem for the student to write a program solution. The optimal operation is selected using a built-in strategy that takes into consideration the knowledge that ITEM/IP has about its domain and the student. If the student is not satisfied with the embedded strategy, s/he can also choose a teaching operation from a list of relevant ones suggested by the system. The student uses the programming laboratory to experiment with presented examples and to solve problems.

3) Learning by repeating and on-line explanation of past material, including previous problems. The extent to which material is repeated is determined by the student's current student knowledge about the material. This material is more concise than the original one. This mode of learning is used for the final material of the course.

Students spend most of the time in the laboratory. With the help of the main menu, the student can select a new teaching operation, information, or a problem. ITEM/11? ensures the continuity of the student's learning. Each of the ITEM/IP components are designed as components to adapt behaviour to the knowledge.

4. An Architecture of ITEM/11?


The main components of ITEM/IP are the conceptual kernel, which includes the teaching operations.

4.1 Conceptual Kernel

The domain model in ITEM/IP is the network of conceptual and procedural knowledge. It includes programming concepts (for example: loop, conditional statement) and skill (for example: *while* statement) and skill (for example: using negative condition in *while*). The domain model is general/specific (IsA), part-of and usage. It is used to construct itself (for example: the mention of a "negative condition" constructed by a

The Student Model, a kind of overlay on the Domain Model concept, The Student Model indicates the extent to which the student's knowledge is correct. The Student Model includes optimal difficulty level. The Student Model is always kept up-to-date and supports

The Base of teaching operations is mentioned earlier, example frames and



3) Learning by repeating and on-line help. At any time a student can demand a re-explanation of past material, including presentation of related examples and analysis of related problems. The extent to which material is explained by ITEMAP is inversely proportional to the current student knowledge about the material. Therefore a repeated explanation is usually more concise than the original one. This mode offers an on-line reference access to the learned material of the course.

Students spend most of the time in ITEM/IP working with the programming laboratory. With the help of the main menu, the student can at any moment call the pedagogic component for a new teaching operation, information, help, or checking the solution of the solved problem. ITMM ensures the continuity of the student's work: the results of the student's work with each of the ITEM/IP components are stored by the Student Model and used by other components to adapt behaviour to the knowledge level of the student.

4. An Architecture of ITEM/IP

The main components of ITEM/IP are the pedagogic module, the programming laboratory, and the conceptual kernel, which includes the domain model, the student model and the base of teaching operations.

4.1 Conceptual Kernel

The domain model in ITEM/IP is the network which contains interconnected elements of conceptual and procedural knowledge. We use three kinds of nodes in the network: high level programming concepts (for example: loop), constructs of the studied programming language (for example: *while* statement) and skills for using the constructs in a context (for example: using negative condition in *while*). The nodes are linked with the following relations: general/specific (IsA), part-of and usage. The latter relationship links a construct-usage with the construct itself (for example: the mentioned skill is linked to *while* by an IsA link and is linked to a "negative condition" constructed by a usage link).

The Student Model, a kind of overlay model, is based on the Domain Model. For every Domain Model concept, The Student Model contains several integer counters for the purpose of indicating the extent to which the student has mastered a concept. Besides a set of counters, the Student Model includes optimal difficulties for all kinds of teaching operations. The Student Model is always kept up-to-date and supports adaptive work of all the environment modules.

The Base of teaching operations is a set of frames of three kinds: the concept frames mentioned earlier, example frames and problem frames. The pedagogic module uses these



frames to build teaching operations of five kinds. Independence of frames makes it possible to expand the base with ease.

Example frames and problem frames are the two main kinds of frames. An *example frame* includes the text of the example (the language construct) and a set of input data "tests" by which this construct can comprehensively be demonstrated. Using this data, the pedagogic module organises a visual demonstration of the construct's semantics and "mental execution" tests checking the acquisition of semantics.

A *problem frame* includes a title, a problem text, a set of test data, a variant of the plan, a model solution and the problem complexity. A problem can be analysed as an example or given to the student to be solved. A special slot is used to link the Base frames with the Domain Model. This slot lists all Domain Model concepts related to the given frame. This list is called the spectrum of the frame.

4.2 Programming Laboratory

The programming laboratory is a set of special "software tools". The tools support the design, debugging and investigation of Turingal programs by the student. The laboratory includes a structured editor which makes it easier to enter and correct a program, and a visual interpreter which enables the student to run programs step by step and observe its work. The interpreter visually displays all machine head moves and symbol typing on the tape and also marks the current operator.

The student uses the programming laboratory in the *independent* mode for experiments and solving problems presented by the pedagogic module. The pedagogic module uses the laboratory in the *controlled* mode to perform teaching operations (see next section). In both these modes, the programming laboratory components adapt their work to a specific student using the student model. It is on the basis of the student model in particular, that the extent to which various language constructs must be visualized is determined: the less studied the construct is, the more detailed the visualization.

For example, when the construct of compound conditions is introduced, it will be visualized in detail as sequence of "steps" of interpretation. Each of the elements of the compound conditions will be visually checked in a separate step and the result (true or false) will be also visually presented in a separate step. On the next level of visualisation granularity stepwise condition checking is substituted with single-step checking, then the visual effect of checking switched off, and follow on. If the compound condition is already studied completely (last level) it will not be visualized at all and just will be passed by the flow of control pointer.

Error handling by the interpreter and error message generation also depends on the student model. More details about adaptive error handling and visualization in ITEM/IP can be found in [8].

4.3 Pedagogic Module

The pedagogic module controls the process. On the "macro" level the pedagogic module controls the teaching operation at each moment and chooses the optimal teaching strategy based on the current student model and the built-in teaching strategy complexities [6].

On the "micro" level the pedagogic module controls the performance of the teaching operation by the student and her/himself. When the teaching operation is completed, the results of the students' work with the teaching operation are stored. Performing a teaching operation, the pedagogic module uses the laboratory tools. The student model is used to select the given student.

The pedagogic module deals with the teaching operation, demonstration, test, problem analysis and evaluation.

Presentation introduces (or reminds) the student of a programming concept or construct. While presenting, as well as information about its relations with other concepts. Each fragment of a concept is stored in the concept's frame as a separate fragment. Each fragment is connected to the concept by counters. The fragment is not presented if its counter is zero. According to the method given above, demonstrating a concept, descriptions are usually more detailed than in the concept. If a concept is not stored but is generated by the pedagogic module, more complete information about relations with other concepts have been studied.

Demonstration of language constructs is a process of understanding of the given construct and its relations with other constructs. The pedagogic module uses example frames stored in the Base of the pedagogic module. The chosen construct is first demonstrated by the student, then experimenting with the example. In a test, the student and then, having mentally executed the teaching operation, the result is then compared to the result obtained by the pedagogic module.

4.3 Pedagogic Module

The pedagogic module controls the process of learning at the "macro" and "micro" levels. On the "macro" level the pedagogic module compiles a list of teaching operations relevant at a given moment and chooses the optimal teaching operation. The optimal operation is chosen using the student model and the built-in teaching strategy, which takes into account the tasks' spectra and complexities [6].

On the "micro" level the pedagogic module manages a dialogue with the student during the performance of the teaching operation, chosen at the "macro" level or by the student her/himself. When the teaching operation is complete, the pedagogic module evaluates the results of the students' work with the teaching operation and updates the student model. Performing a teaching operation, the pedagogic module widely uses the programming laboratory tools. The student model is used to adapt the performance to the knowledge level of the given student.

The pedagogic module deals with the five main kinds of teaching operations: presentation, demonstration, test, problem analysis and problem to solve.

Presentation introduces (or reminds the student of) a piece of conceptual knowledge: programming concept or construct. While introducing a concept (construct), its description as well as information about its relations with other concepts is presented. The description of each concept is stored in the concept's frame. The text of the description can be broken up into fragments. Each fragment is connected with the threshold value of one of the student model counters. The fragment is not presented if the counter for the concept exceeds this threshold. According to the method given above, during a re-presentation (presentation as repetition) of concept, descriptions are usually more laconic. Information about the relations of a given concept is not stored but is generated by the system. While presenting a concept, information is generated only about relations with previously studied concepts. During the re-presentation, more complete information about relations is usually given, since by this time, a greater number of concepts have been studied.

Demonstration of language construct semantics and *Tests* that check the student's understanding of the given construct are performed with the help of the interpreter using example frames stored in the Base of teaching operations. In a demonstration, the behaviour of the chosen construct is first demonstrated on a set of tests and then the student has a chance of experimenting with the example. In a test, the student is given the initial state of the tape (test), and then, having mentally executed the construct tested, s/he can enter the result. The answer is then compared to the result obtained by the interpreter.

Programming problem is the most important kind of teaching operation. The text of the given problem is displayed and then explained in the form of the "test-result" pairs. Additional explanations, a plan for solving the problem and hints as to how the program should work (visualization of model program actions without presenting the text of the actual program), can be requested as help. Solutions are checked out by the interpreter which consecutively compares the student's program's results on the given set of tests with that of the model program. If the results of any of the tests do not coincide, the mistake is fixed and "explained" by a visual demonstration of the student's program wrong behaviour on this counterexample test. Our experience has shown that this is a very effective way of remediation.

Programming problem analysis gives the student a practical idea as to how to use the skills s/he has learned. During the analysis the student is consecutively given the text of the problem, offered a plan of action and in the end given a model solution, whose work is demonstrated on tests. Next, the model program is presented to the student for experimentation. Any previously analysed or solved problem can be re-analysed on demand.

The sequence of language study in ITEM/IP is not fixed. The system can generate an optimal sequence of teaching operations for the given student. The human teacher can tune the teaching strategy to match her/his way of tutoring by setting an individual teaching order for any student [8]. The individual teaching order is a sequence of subsets of domain model nodes. At the beginning of teaching the nodes of the fast subset are activated. Then the activated subset of nodes is taught with the use of the teaching operation selected by the tutoring component until the moment when the values of all counters related to the activated nodes exceed the mastery threshold or until the moment when all the relevant teaching operations are exhausted. Then the next subset of nodes are activated in the same way.

5. Progress and Status

The version of ITEM/IP described in this paper was designed and implemented for the Soviet mainframe computer BESM-6. We have tested ITEMIP for a year in the learning process among fast year students of the Moscow State University and 14 years old students of Moscow schools. A more detailed description of ITEM/IP and some experience with the system can be found in [7]. In particular, we have found that the ITEM/IP visual interpreter could strongly help the teacher to solve the problem of debugging student programs. In about four fifths of all cases to identify the bug it was quite enough for a student to run the wrong program visually with a test data suggested by the system. Only in one fifth of cases did the student (usually a weaker student) need the teacher to help to understand the bug.

The second extended version of ITEB is important that ITEMIP-II will be able to u activity in designing and debugging progr

Summary

An intelligent programming environment teacher and student activities in the proces how to integrate the given set of tools un just the total sum of a number of compone

We have designed an intelligent progi teacher and student work in the course of

ITEM/IP components are tightly intel other components and the results of stud the student model and used by all com particular student. The student model of t functions of a student model, as well as si model.

The approach described above is not design an intelligent programming enviro

References

1. Anderson, J.R. and Reiser, B.: The
2. Barr, A., Beard, M. and Atkinsoi Stanford BIP project. International (1976)
3. Benyon, D., Murray, D.: Experience (5), 465-473 (1988)
4. Bertels, K., Vanneste, P. and De E novice programming analysis. In P Conference on Emerging Compute 40. Moscow: ICSTI 1992
5. Bonar, J. and Cunningham, R.: programming. In J. Self (ed.), *Art, computer aided instruction*, pp. 391
6. Brusilovsky, P.L.: A framework sequencing. In C. Frasson, G. Gai systems. Proc. of II Internat. Conf Science, vol. 608, pp.499-506. Berl
7. Brusilovsky, P.L.: Intelligent b programming. *Educational Technok*

The second extended version of ITEMAP is being developed for personal computers. It is important that ITEMAP-II will be able to update the student model while observing free student activity in designing and debugging programs.

Summary

An intelligent programming environment can be viewed as a set of tools that supports different teacher and student activities in the process of learning programming. An interesting problem is how to integrate the given set of tools into a complete environment that ought to be more than just the total sum of a number of components.

We have designed an intelligent programming environment ITEM/IP, which supports both teacher and student work in the course of introductory programming.

ITEMAP components are tightly integrated. One component widely uses the capabilities of other components and the results of students' work with pedagogic components are stored in the student model and used by all components in adapting to the knowledge level of the particular student. The student model of the ITEM/IP performs the traditional elaborative [19] functions of a student model, as well as some adaptive interface support [3] functions of a user model.

The approach described above is not limited to learning Turingal only. It can be used to design an intelligent programming environment for Pascal, C, or another language.

References

1. Anderson, J.R. and Reiser, B.: The LISP tutor. *Byte*. 10 (4), 159-175 (1985)
2. Barr, A., Beard, M. and Atkinson, R.C.: The computer as tutorial laboratory: The Stanford BIP project. *International Journal on Man-Machine Studies*. 8 (5), 567-596 (1976)
3. Benyon, D., Murray, D.: Experience with adaptive interfaces. *The Computer Journal*. 31 (5), 465-473 (1988)
4. Bertels, K., Vanneste, P. and De Backer, C.: An evaluation of different approaches in novice programming analysis. In P. Brusilovsky and V. Stefanuk (eds.) *Proc. East-West Conference on Emerging Computer Technologies in Education, 1992, Moscow*, pp. 36-40. Moscow: ICSTI 1992
5. Bonar, J. and Cunningham, R.: Bridge: an intelligent tutor for thinking about programming. In J. Self (ed.), *Artificial Intelligence and human learning. Intelligent computer aided instruction*, pp. 391-409. London: Chapman and Hall 1988
6. Brusilovsky, P.L.: A framework for intelligent knowledge sequencing and task sequencing. In C. Frasson, G. Gauthier and G. I. McCalla (eds.), *Intelligent tutoring systems. Proc. of II Internat. Conference ITS'92, Montreal. Lecture Notes in Computer Science*, vol. 608, pp.499-506. Berlin: Springer 1992.
7. Brusilovsky, P.L.: Intelligent tutor, environment and manual for introductory programming. *Educational Technology and Training International*. 29 (1), 26-34 (1992)

8. Brusilovsky, P.L.: Student models and flexible programming course sequencing. In I. Tomek (ed.), Supplementary Proc. 4-th Internat. Conference on Computers and Learning, Wolfville, Canada, pp. 8-10. Wolfville 1992
9. Brusilovsky, P.L.: Turingal - the language for teaching the principles of programming. Proc. 3-rd European Logo Conference 1991, Parma, pp. 423-432. Parma: A.S.I. 1991
10. Ramadhan, H. and du Boulay, B.: Programming environments for novices. This volume
11. Corbett, A.T. and Anderson, J.R.: Student modeling in an intelligent programming tutor. This volume
12. Eisenstadt, M., Price, B.A and Domingue, J.: Redressing ITS fallacies via software visualization. This volume
13. Heines, J. and O'Shea, T.: The design of rule-based CAI tutorial. *International Journal of Man-Machine Studies*. **23** (1), 1-25 (1985)
14. Hohmann, L., Guzdial, M. and Soloway, E.: SODA: a computer-aided design environment for the doing and learning of software design. In I. Tomek (ed.), Computer Assisted Learning. Proc. of 4-th Internat. Conference ICCAL '92, Wolfville, Canada. Lecture Notes in Computer Science, vol. 602, pp. 307-319. Berlin: Springer 1992
15. McCalla, G.I. and Greer, J.E.: Two and one-half approaches to helping novices learn recursion. This volume
16. McCalla, G.I., Greer, J.E. and Scent Research Team: SCENT-3: An architecture for intelligent advising in problem-solving domains. In C. Frasson, G. Gauthier (eds.), Intelligent Tutoring Systems: At the crossroads of artificial intelligence and education, pp. 140-161. Norwood: Ablex Publishing 1990
17. Merrill, D.C., Reiser, B.J., Beekelaar, R. and Hamid, A.: Making process visible: scaffolding learning with reasoning-congruent representations. In C. Frasson, G. Gauthier and G. I. McCalla (eds.), Intelligent tutoring systems. Proc. of II Internat. Conference ITS'92, Montreal. Lecture Notes in Computer Science, vol. 608, pp. 103-110. Berlin: Springer 1992
18. Pattis, R.E.: Karel - the robot, a gentle introduction to the art of programming. London: Wiley 1981
19. Self, J.: Student Models: What use are they?. In P. Ercoli and R. Lewis (eds.), Artificial intelligence tools in education. Proc. of the IEP TC3 working conference on AI tools in education, 1988, pp. 73-85. Amsterdam: North-Holland 1988
20. Singley, M.K., Carrol, J.M. and Alpert, S.R.: Incidental reification of goals in an intelligent tutor for Smalltalk. This volume
21. Soloway, E.: PROUST. Byte, 10 (4), 179-190 (1985)
22. Weber, G.: Analogies in an intelligent programming environment for learning Lisp. This volume

Programming Envi

Haider Ramadhan* and Benedict du

School of Cognitive and Computing SciE
*permanent address: Math. and Compu
Oman.

Abstract: We describe DISCOVER, which combines a free phase, in which students program of their own, with a guided phase, monitored. The system embodies a machine. When setting a problem, the system presents the problem, the student solves the problem. A pilot experiment is described during the guided phase, of providing

Keywords: Intelligent Tutoring Systems

Introduction

Opponents of intelligent programming environments have not fulfilled their early promise of abstraction and difficulty found in programming. They come up with alternative approaches. Programming systems, they claim, should be based on learning by example [14] techniques rather than rules (e.g. [2,9]).

The way forward, therefore, should be towards microworlds and ITS. A practical example of a discovery programming system DISCOVER is an issue of visualisation as simply a question of visualisation [13]) nor of displaying different representations of a high-level, integrated, concept to relate problem-solving with the concept. To achieve this, DISCOVER integrates an integral dynamic model of the underlying concept with the novice user is interacting [10].

An Overview of DISCOVER

DISCOVER is an intelligent programming environment for microworlds with automatic, dynamic, and works in two phases. In the free phase