A Tool for Programming Learning with Pedagogical Patterns

Leliane Nunes de Barros and Ana Paula dos Santos Mota
Department of Computer Science.
University of São Paulo
Rua do Matão, 1010, Bloco C. Cidade Universitária. 05508-090.
São Paulo, SP, Brazil.
leliane@ime.usp.br
alanis@linux.ime.usp.br

Karina Valdivia Delgado and Patricia Megumi Matsumoto
Department of Computer Science.
University of São Paulo
Rua do Matão, 1010, Bloco C. Cidade Universitária. 05508-090.
São Paulo, SP, Brazil.
kvd@ime.usp.br
patty@linux.ime.usp.br

ABSTRACT
Programming Patterns help create a shared language for communicating insight and experience about programming problems and their solutions. Inspired by this idea, we developed the ProPAT e-learning tool: an Eclipse IDE that allows students of a first Computer Science course to learn how to program using pedagogical patterns, i.e., a set of programming patterns recommended by Computer Science educators. ProPAT has been implemented as an Eclipse plug-in with two main perspectives: the Teacher Perspective and the Student Perspective. To identify some of the students' mistakes, the ProPAT plug-in also includes a program diagnosis system that uses Model Based Diagnosis techniques from the Artificial Intelligence.

Keywords
Computer-based Learning, Teaching Introductory Undergraduate Programming, Debugging and Testing Tools.

1. INTRODUCTION
Writing a program for a novice involves many difficulties. The attempts to deal with multiple impasses all at once can make this task even worse. Research on programming psychology points out two challenges that a novice programmer has to handle [Winslow, 1996]:

1. learning a new programming language: the student has to learn the syntax and semantics of a new programming language.

2. learning how to program a solution to a given problem: the student has to learn how to transform a hand-written problem solution into a computer program. Sometimes the novice knows how to solve a problem by hand, but he is not able to write a program to solve the same problem, e.g., solving a quadratic equation.

Although a programming language has a lot of details, the first challenge is not the most difficult part. Evidences show that learning a second language is, in general, easier. A hypothesis is that the student has already acquired abilities to solve problems using the computer which is the common skill to learn different languages.

Related to the second challenge, research on cognitive theories about programming learning has shown evidences that experienced programmers store and retrieve old experiences on problem solving that can be applied to a new problem and can be adapted to solve it [Johnson and Soloway, 1984]. On the other hand, a novice programmer does not have any real experiences, but only the primitive structures from the programming language. Inspired on these ideas, a strategy to teach how to program is to present small programming pieces, instead of leaving the student to program from scratch. That is the pedagogical patterns community proposal. This community is a group of experienced educators engaged to recommend programming pieces for novices, also called pedagogical programming patterns or elementary programming patterns [Wallingford, 2001]. Supposing that students who have learned these patterns will in fact construct programs using them, i.e., starting their programs from known pieces of code, an e-learning system could take a number of advantages from this teaching strategy, such as:

• the e-learning system can establish a shared language for communicating insights and experiences about problems and their solutions with the student. This can provide significant benefits to the student since the pedagogical patterns provide the terms and concepts the student needs to know;
• an e-learning module for diagnosing problems in the student program would be able to reason about the patterns in a hierarchical fashion, i.e., to detect programming mistakes in different levels of abstraction.

This is the same idea used by the PROUST system [Johnson and Soloway, 1984] to teach PASCAL programming. The difference with the present work is that PROUST does not open its programming patterns (called programming plans by the authors), i.e., the student has not access to them. Instead, the programming plans are only used internally by the system.

In this paper, we present a new Eclipse IDE for programming learning based on pedagogical programming patterns. Section 2 introduces the idea of pedagogical programming patterns and how they can be used in an introductory course for programming. Section 3 gives a brief definition of Eclipse plug-ins and the perspectives created for ProPAT. Finally, Section 5 describes our current work: a diagnosis module for ProPAT that will use the pedagogical patterns in an intelligent e-learning tool.

2. PEDAGOGICAL PROGRAMMING PATTERNS IN THE CLASSROOM

"... The goal of patterns within the software community is to create a body of literature to help software developers resolve recurring problems encountered throughout all of software development. Patterns help create a shared language for communicating insight and experience about these problems and their solutions. Formally codifying these solutions and their relationships let us successfully capture the body of knowledge which defines our understanding of good architectures that meet the needs of their users. Forming a common pattern language for conveying the structures and mechanisms of our architectures allows us to intelligibly reason about them. The primary focus is not so much on technology as it is on creating a culture to document and support sound engineering architecture and design.". From Patterns and Software: Essential Concepts and Terminology Web site, by Brad Appleton [Appleton, 2000].

Pedagogical programming patterns, also called elementary programming patterns, are recommended solutions to common problems described in a way to ease their reuse. Patterns are simple, synthetic and recommended by researchers on programming teaching for novices [Porter and Calder, 2003]. A pattern relates a problem to a solution and provides information about the context in which it can be applied. Its potential use in programming teaching has been explored by the pedagogical patterns community. Patterns are available in the Web for C, C++ and Java [Wallingford, 2001], including: selection patterns [Bergin, 1999], repetition patterns [Astrachan and Wallingford, 1998] and others [Bridgeman, 2002]. Porter and Calder [Porter and Calder, 2003] suggest a process to employ programming patterns in the classroom and Proulx [Proulx, 2000] created a first Computer Science course based on these patterns.

Pedagogical programming patterns can help novice programmers in two ways:

1. to learn terms, concepts and general strategies of programming problem solving (in a higher abstraction level);

2. to retrieve the syntax and learn how to use a programming language, since its documentation includes a program, that is an example of the pattern application for a specific programming situation.

On the other hand, pedagogical programming patterns can help a human tutor to:

a. recognize the student’s intentions;

b. establish a better communication with the student, since they provide a common vocabulary about general strategies for programming problem solving.

Some pattern examples are shown in Figure 1, where we show a small part of the documentation of three loop patterns: Loop with Sentinel, Loop with Flag and Counting Loop. Notice that they correspond to distinct elementary strategies, commonly used to solve sequence processing problems, as it is stated in the use/application column of Figure 1. A student that uses a strategy when another one would be more naturally applied, will have difficulties to accomplish his programming task.

The complete structure of pattern documentation used in this work is composed of: pattern name, dependent patterns, use/application, syntax, semantics and situation example. The use/application of a pattern is a text that describes, in general terms, a programming situation for which the pattern can be applied. The syntax of the pattern is a pseudo-code that includes: C code; other pattern names and metadata (text between quotation marks). Metadata are used to describe general programming terms and concepts. Metadata and other pattern names must be replaced by the student while constructing his own program solution. The semantics describes how the pattern really works (in a text format) and its syntax control flow (in a diagrammatic format).

3. THE PROPAT PLUG-IN

ProPAT is a programming learning environment using pedagogical patterns, that has been built as an Eclipse plug-in. ProPAT provides an IDE for a first Computer Science course, i.e., an IDE for novice programmers. In this environment, the student is able to choose a programming exercise and to construct a solution by selecting and adding pedagogical programming patterns into the editor [Delgado and de Barros, 2004]. ProPAT also allows a teacher to add new programming patterns and exercises, whose selection is aimed to motivate the student to use the patterns.

3.1 An Eclipse Plug-in

This project has been first developed for the C programming language. Therefore, some of the PROPAT plug-in features were inherited from the original Eclipse CDT plug-in [CDT Plug-in, 2000], an IDE for C programming, while others were specially developed for this project. An Eclipse plug-in is typically composed of a set of perspectives and views.
A **perspective** is a visual container (a window) for a set of views. A view is used to: (i) open an editor; (ii) navigate in a hierarchy of information (e.g. projects, files, classes and concepts); or (iii) display properties for the active editor. With the exception of the editor, modifications made in a view are automatically saved. The layout of editors and views is controlled by the active perspective.

### 3.2 ProPAT

The ProPAT plug-in inherited some views from the original Eclipse CDT plug-in to compose two new perspectives: the **Student Perspective** where the student can choose programming exercises and develop solutions for them, through pattern selections or write his own code; and the **Teacher Perspective** used by the teacher to specify new exercises and patterns that will be available to the student through the Student Perspective. These two perspectives are described in more details in the following sections.

An important feature of the ProPAT plug-in is the database of patterns and exercises. ProPAT uses an XML database. By using the Document Object Model (DOM) to create and manipulate a hierarchy of data objects from an XML document, it is possible to have a random access and modification of its contents. There are two XML data models in ProPAT: one for patterns and another for exercises.

### 4. HOW TO USE PROPAT

#### 4.1 The Student Perspective

There are eight views available in the Student Perspective: the Program Editor View, the Navigator View, the Patterns View and Pattern Info View.

---

<table>
<thead>
<tr>
<th>pattern name</th>
<th>use/application</th>
<th>syntax</th>
</tr>
</thead>
</table>
| Loop with Sentinel | You want to repeat a set of actions while a condition is true. In general, the set of actions is related to the processing of a sequence of elements or numbers. The amount of elements is unknown but the end of the sequence is indicated by a sentinel value. The elements can be read or generated. | `<INITIALIZATIONS> 
  <SENTENTIAL VARIABLE INITIALIZATION>` 
  while `<SENTENTIAL VARIABLE CONDITION>` { 
    `<READ GENERATION OF A SEQUENCE ELEMENT>` 
    `<PROCESS ELEMENT>` 
    `<UPDATE THE SENTENTIAL VARIABLE>` 
  } |
| Counting loop | You want to repeat a set of actions a determined number of times. In general, the set of actions is related to the processing of a sequence of elements or numbers. The number of elements must be known. The elements can be read or generated. | `<INITIALIZATIONS>` 
  `<COUNTER INITIALIZATION>` 
  while `{COUNTER CONDITION}` { 
    `<READ GENERATION OF A SEQUENCE ELEMENT>` 
    `<PROCESS ELEMENT>` 
    `<UPDATE THE COUNTER>` 
  } |
| Loop with flag | You want to repeat a set of actions while a condition is true. In general, the set of actions is related to the processing of a sequence of elements or numbers. However, during this process, you need to verify if a certain property holds in order to: (1) interrupt the loop and/or (2) to perform an action that depends on that property, after the loop. For this purpose, a variable called flag is used to indicate if the property holds or not. | int flag; 
  `<INITIALIZATION>` 
  flag = 0; 
  while `{COUNTER CONDITION} && flag == 0` { 
    `<READ GENERATION OF A SEQUENCE ELEMENT>` 
    `<PROCESS ELEMENT>` 
    if `{FLAG CONDITION}` { 
      flag = 1; 
    } else { 
      `<SET OF ACTIONS>` 
    } |

---

**Figure 1:** Pedagogical Programming Patterns Example

**Figure 2:** ProPAT Student Perspective - Patterns View and Pattern Info View
Each exercise is supposed to be related to one or more patterns. The student’s challenge is to find out what are the patterns to be used to solve the problem in a more suitable way (i.e., the patterns that makes the program more clear and/or simpler).

The Exercise Description View shows to the student the description of a selected exercise in an HTML format and contains the exercise name, the exercise problem specification, and a set of bench tests.

The Student Perspective is organized in such a way that the program editor and all its views can be visualized in the same window, which is very convenient for the student. In addition to this, there is a button for compiling and running a program in the Perspective, making it easy for the novice to test his solutions to the exercises through the Message Console View, applying the bench test data from the exercise descriptions.

4.2 The Teacher Perspective

The Teacher Perspective (Figure 3) gives to ProPAT the characteristics of an authoring learning tool, i.e., an e-learning tool whose content can be defined by the teacher. It gives flexibility and facilities to the teacher who is using pedagogical programming patterns in his courses. This perspective inherits all the features available in the Student Perspective plus the necessary tools to edit, remove and add new patterns and exercises. For this purpose it was created three extra views: the Pattern Editor View, the Exercise Editor View and the Metadata Editor View.

In the Pattern Editor View, to edit or create a pattern, the teacher must fill in a form, which corresponds to the pattern documentation structure described in Section 2. Besides that, there is an extra information that the teacher must provide: the pattern insertion constraints, also called, pattern insertion rules. This information is used by the Program Editor View to check out some student’s mistakes and to give him some explanations.

The exercises has also two new fields: program solution example (in C code) and patterns suggestions made by the teacher as the best recommendation for the exercise. Both fields can be edited by the teacher through the Exercise Editor View. One of the uses of the teacher program solution is to run it over some input bench test data and calculate the expected output data (e.g. for automatic generation of bench tests). Although the patterns suggestions information has not yet been used, it will allow a richer interaction with the student.

The Metadata Editor View is a text editor that allows the teacher to define the concepts underlying a metadata, which is the text that will be displayed to the students and other teachers. This view can also help the teacher to use the same metadata, as much as possible, while creating new patterns. This is going to be a more important view in future versions of ProPAT when it will be used, together with the pattern names, to generate an ontology of programming knowledge, i.e., a formal conceptualization of the programming knowledge (also called in some educational tools as a conceptual map) [ ].
5. DIAGNOSIS

We are currently working on adding a diagnosis module to the ProPAT plug-in. This will be used to detect non-syntactic errors in the student program (i.e., after it has been compiled successfully).

The basic idea for diagnosing programs is to derive a component/connection model directly from the student program and the programming language semantics. This model must identify components, connections, the program structure and the system description. While in the automatic model-based diagnosis of physical devices, a model of the device has to be specified, called system description for diagnosing programs the system description is the actual student program behavior which reflects its errors. The observations are the incorrect outputs in the different points of the original program code. The predictions are not made by the system, but by the student and therefore this is the situation where the student must communicate his programming goals to the system.

To derive the component/connection model from the student program we built a parser in ANTLR [Parr, 1989], a language tool that provides a framework for constructing recognizers, compilers, and translators from grammatical descriptions containing Java, C, C++, or Python actions.

We propose an addition to the diagnosis method described in [Mateis et al., 2000] so that Programming Patterns can also be modeled as new components. Thus, the ProPAT diagnosis module will be able to reason about patterns in a hierarchical fashion, i.e., to detect program faults in different levels of abstraction.

6. CONCLUSIONS AND FUTURE WORKS

In this work, we have presented the ProPAT e-learning tool: an Eclipse IDE that allows students of a first Computer Science course to program using pedagogical patterns. ProPAT has been implemented as an Eclipse plug-in with two main perspectives: the Teacher Perspective and the Student Perspective.

In the current version of the ProPAT system, teachers can gather, insert and remove patterns and exercises, while the students can solve a list of proposed exercises by inserting pedagogical programming patterns in the editor. Students can also compile and test their solutions based on the sample inputs and outputs (bench tests) provided by the teacher in the exercise descriptions.

An important part of the ProPAT plug-in is the independent Databases for Patterns and Exercises, represented as an XML database. To create and manipulate a hierarchy of data objects from an XML document, we used the Document Object Model (DOM), allowing random access and modification of its contents.

In order to test ProPAT with students of a first Computer Science course, one of the difficulties we are currently facing is the decision on which collection of elementary patterns to use. This decision requires the teacher commitment on adding the elementary patterns by himself, according with a set of problems. This is our effort to implement a course using ProPAT for the 1st semester of 2006.

6.1 Acknowledgment

We would like to thank Fabio Kon for all the support he has given for this project.

7. REFERENCES


