Teaching Introductory Programming

Agent-based Approach with Pedagogical Patterns for Learning by Mistake

Ljubomir Jerinic
Department of Mathematics and Informatics
Faculty of Science, University of Novi Sad
Novi Sad, Serbia

Abstract—**From the educational point of view, learning by mistake could be influential teaching method, especially for teaching/learning Computer Science (CS), and/or Information Technologies (IT). As learning programming is very difficult and hard task, perhaps even more difficult and extremely demanding job to teach novices how to make correct computers programs. The concept of design pedagogical patterns has received surprisingly little attention so far from the researchers in the field of pedagogy/didactics of Computer Science. Design pedagogical patterns are descriptions of successful solutions of common problems that occur in teaching/learning CS and IT. Good pedagogical patterns could help teachers when they have to design new course, lessons, topics, examples, and assignments, in a particular context. Pedagogical patterns captured the best practice in a teaching/learning CS and/or IT. They could be very helpful to the teachers in preparing their own lessons. In this paper a brief description of special class design of pedagogical patterns, the group of patterns for learning by mistakes, is presented. In addition, usage of *helpful* and *misleading* pedagogical agents, which have been developed in Agent-based E-learning System (AE-lS), based on pedagogical pattern for explanation *Explain*, and pedagogical pattern for learning by mistakes *Wolf, Wolf, Mistake*, is described.**

Keywords—Pedagogical Pattern; Pattern Design; Learning; Programming; Computer science education; Programming; Software agents; Electronic learning; Computer aided instruction

I. Introduction

Conventional pedagogy believes that the one of good way to teach students is to have them repeatedly practice some tasks. In recent work of Lindsey E. Richland, Nate Kornell and Liche Sean Kao [1] the advantages of learning through error was discussed. According to this approach, it is important to avoid mistakes while learning so that our mistakes are accidentally reinforced. That approach assumes that the best way to teach children is to have them repeatedly practice (test for example) as far as it takes.

Once they know (learn or guess or rich somehow) the right answer, that correct response is embedded into the brain. However, this error-free process turns out to be inefficient: Students learn material much faster when they made mistake first, especially in programming. In other words, getting the wrong answer helps us remember the right one.

Nobody likes making mistakes. Nevertheless, unless you want to go through life as a complete recluse, you are guaranteed to make one every now and them. If you learn from mistakes correctly, they could push you forward. You can only learn from a mistake after you admit you have made it, or get the explanation way you have made it.

However, from the educational point of view, learning by mistake could be powerful teaching technique and/or method. If the lecturer[1] create appropriate situation and put student in it, where student can make interesting mistakes, it could be used for educational purpose, and this method is called the learning by mistake technique of teaching. Of course, the lecturer could use some fine facts to make students to made mistake, and after explanation way you made it, you learn, i.e. do not make the same error again.

Joseph Bergin [2] defined pedagogical patterns as follows "Patterns are designed to capture best practice in a specific domain. Pedagogical patterns try to capture expert knowledge of the practice of teaching and learning. The intent is to capture the essence of the practice in a compact form that can be easily communicated to those who need the knowledge. Presenting this information in a coherent and accessible form can mean the difference between every new instructor needing to relearn what is known by senior faculty and easy transference of knowledge of teaching within the community."

This paper covers one point of view in design and implementation of Pedagogical Patterns, the group of patterns for learning by mistakes method in teaching.

The rest of this paper is organized as follows. Section 2 provides an overview of the existing theory and application related to teaching/learning by mistakes. In the field of e-learning and tutoring systems, two categories of software agents are of the special interest: harvester and pedagogical agents. Section 3 provides an overview of the existing work related to e-learning systems and pedagogical agents.

Section 4 introduces pedagogical patterns, pattern language for describing patterns, and pedagogical pattern Explain, and two distinct sub-types of pedagogical agents: helpful and misleading is introduced. Whereas helpful agents provide the correct guidance for a given problem, misleading agents try to steer the learning process in a wrong direction, by offering false hints and inadequate solutions. The rationale behind this approach is to motivate students not to trust the agent's instructions blindly, but instead to employ critical thinking, and, in the end, they themselves decide on the correct solution to the problem in question.

[1] In this paper term lecturer is used to denote teachers, professors, instructors, tutors, i.e. it denotes the person who teach.

In Section 5, a stand-alone e-learning architecture, called Agent-based E-learning System (AE-lS) and some examples are described. AE-lS are designed to help learners in learning programming and programming languages. In Section 7, describe design and definition of pedagogical pattern for learning by mistakes Wolf, Wolf, Mistake. Some examples of use that pedagogical pattern is presented in Section 8. Finally, overall conclusions and future research directions are given in Section 9.

II. Teaching/Learnig by Mistakes

For years, many educators have championed “errorless learning," advising teachers (and students) to create study conditions that do not permit errors. For example, a classroom teacher might drill students repeatedly on the same multiplication problem, with very little delay between the first and second presentations of the problem, ensuring that the student gets the answer correct each time.

People remember things better, longer, if they are given very challenging tests on the material, tests at which they are bound to fail. If students make an unsuccessful attempt to retrieve information before receiving an answer, they remember the information better than in a control condition in which they simply study the information [1]. Trying and failing to retrieve the answer is actually helpful to learning. It is an idea that has obvious applications for education, but could be useful for anyone who is trying to learn new material of any kind.

Lecturer could ask students (students could try to answer) questions at the back of the textbook chapter, or to give them eLearning topic test, before teaching and students could try to answer. If there are no questions available, lecturer could convert the section headings to questions. For example, if the heading is Loop-Control, ask students “What is Loop-Control?” If the answers are wrong, teach the chapter/topic and ask the same questions, when the lecture is finished. If the answers are good lecturer should praise students. If the answers are wrong, lecturer gives instructions, extra questions, hints, and discuss why the answers are wrong. For answers that are very wrong, lecturer gives students additional time to try to learn and master the material lectured. Even if answers are wrong, these mistakes are more useful to the students, much more valuable than just learning the material. Getting the answer wrong is a great way to learn.

These are general-purpose strategies for teaching/learning by mistakes, and it is used for design of pedagogical pattern *Wolf, Wolf, Mistake*, described in Section 6. Moreover, this strategy is employed and utilized for *helpful* and *misleading* pedagogical agents, described in Section 5.

III. Teaching Programming With Patterns and Agents

Software agents, or simply *agents*, can be defined as *autonomous* software entities, with various degrees of *intelligence*, capable of exhibiting both *reactive* and *pro-active* behavior in order to satisfy their design goals. From the point of e-learning and tutoring systems, two types of agents are of the special research interest: *harvester* and *pedagogical* agents. Harvester agents are in charge of collecting learning material from online, often heterogeneous repositories [3].

Haake and Gulz [4] define pedagogical agents as “lifelike characters presented on a computer screen that guide users through multimedia learning environments” (p. 28). Heller and Procter [5] points out that main goal of usage of pedagogical agents are to motivate and guide students through the learning process, by asking questions and proposing solutions.

A stand-alone e-learning architecture, called ***A**gent-based **E**-**l**earning **S**ystem* (***AE-lS***). ***AE-lS*** are designed to help learners in learning programming and programming languages. ***AE-lS*** consist of three main components:

- Harvest*e*r agents;
- Classifier module*;* and
- A pair of pedagogical agents.

The harvester agents are in charge of collecting the appropriate learning material from the web. Their results are fed into the *Classifier* module, which performs automatic classification of individual learning objects. Finally, a pair of specially designed pedagogical agents - one *helpful* and one *misleading* - is used to interact with students and help them comprehend the underlying learning material.

The helpful pedagogical agent provides useful hints for the solution of the given problem to the student, trying to direct student to the correct solution, or to help student to understand some topic, giving explanations. On contrary, misleading pedagogical agent try to steer and guide the solving/learning process in the “wrong” direction, giving some hints or explanation which could produce bed results. The student is never sure with which agent (s)he is interacting, this approach encourages students not to follow the agent’s/tutor’s instructions blindly, but rather to employ critical thinking and, at the end, they themselves decide on the proper solution to the given problem or the suitable accepting and understanding presented topic.

Originally, the ideas of using harvester, as well as the two types of pedagogical agents were discussed in [6]. This paper presents a concrete implementation of these ideas, in connection with pedagogical pattern approach.

IV. Pedagogical Patterns

What are Pedagogical Patterns? Patterns are designed to capture best practice in a specific domain. Pedagogical patterns [2] try to capture expert knowledge of the practice of teaching and learning. The intent is to capture the essence of the practice in a compact form that can be easily communicated to those who need that knowledge and experience. In essence, a pattern solves a problem. This problem should be one that recurs in different contexts. In teaching, we have many problems such as motivating students, choosing and sequencing appropriate materials and resources, evaluating students, and the similar.

These problems do recur and in slightly different form each time. Each time a problem, pops up there are considerations that must be taken into account that influence our choice of solution. These forces push us toward or away from any given solution to a problem. A pattern is supposed to present a problem and a solution. The problem together with the forces must apply to make that solution beneficial to the problem.

A. Pattern Languages - The Pattern Format

A pattern language is a set of patterns that work together to generate complex behavior and complex artifacts, while each pattern within the language is itself simple. Pattern languages, on the other hand, promise to drive fundamental and lasting improvements. One very successful pedagogical pattern language is Seminars by Astrid Fricke and Markus Vöelter [2]. It describes how to design and deliver a short course. Little in this language (or any pattern language) is novel, but it brings together in one place expert knowledge that is often forgotten and sometimes overlooked.

Besides its title, a pattern contains at least the following five sections:

- The Context section sets the stage where the pattern takes place.
- The Problem section explains what the actual problem is.
- The Forces section describes why the problem is difficult to solve.
- The Solution section explains the solution in detail.
- The Consequences (positive and negative) section demonstrates what happens when you apply the solution.

The Figure 1 shows the pattern sections and the order in which the pattern should be written.

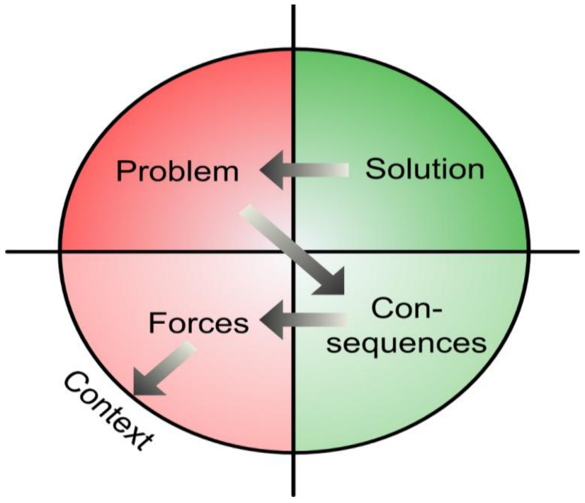


Fig. 1. Pattern language sections and their writing order

B. Explanation Pattern for Explanation in eLearning

Classification and intent. Explanation pattern is based on Builder creational pattern [11]. Its intent is to help separate the construction of a complex object from its representation. Such a separation makes it possible to create different representations by the same construction process.

Motivation. Suppose an eLearning designer wants to develop an explanation generator that can generate explanations for different students. In general, current level of mastering the subject is different for different students at any given moment. That fact is reflected in the student model of each student. Novice students should get more general and easy explanations, while more complex and detailed explanations to more advanced students have to be provided [7]. The problem is that the number of possible explanations of the same topic or process is open-ended.

Using the Builder pattern provides a solution. The explanation generator in eLearning LMS could be designed with an *ExplanationBuilder*, an object that converts a specific knowledge level from the student model to an appropriate type of explanation, which is exposed in Figure 2. In this paper, *ExplanationBuilder* given in [7] is expanded and extended with helpful and misleading suggestions and hints, used for realization of *helpful* and *misleading* pedagogical agents.

The lecturer arranged and organized the appropriate explanations. Whenever the student requires an explanation, the explanation generator passes the request to the *ExplanationBuilder* object according to the student's knowledge level. Specialized explanation builders, like *EasyExplanationBuilder* or *Advanced-ExplanationBuilder*, are responsible for carrying out the request.

Structure. Figure 2 shows the general structure of the *Explain* pattern, based on Builder pattern. Unlike similar form, given in [7], *Explain* pattern is extended with helpful and misleading suggestions, hints, and clues.

Consequences. Using *Explanation* pattern lets designers vary a product's internal representation, e.g., the contents of the explanation. The pattern provides isolation of the code for representation from the code for construction. Construction of the product is a systematic process, and is under the director's control.

Known uses. Examples of using the *Explanation* pattern in Intelligent Tutoring Systems (ITS) design include different generators, such as explanation generator, exercise generator, and hint generator. In GET-BITS model [8], explanation generator is can construct explanations for a predefined set of users, which is configurable (e.g., beginners, midlevel, advanced, experts...) [9]. Hints for solving problems are generated in much the same way. In *Eon* tools, different contents are presented to the student during the teaching process depending on different Topic levels, which represent different aspects or uses for the topic (e.g., introduction, summary, teach, test, beginning, difficult,…) [10]. Extended *Explain* pattern is used in ***A****gent-based* ***E****-****l****earning* ***S****ystem* (***AE-lS***) [6].

Related patterns. Builder pattern is similar to the Abstract Factory pattern [11]. Explain pattern is based on *Expose the Process* [2].

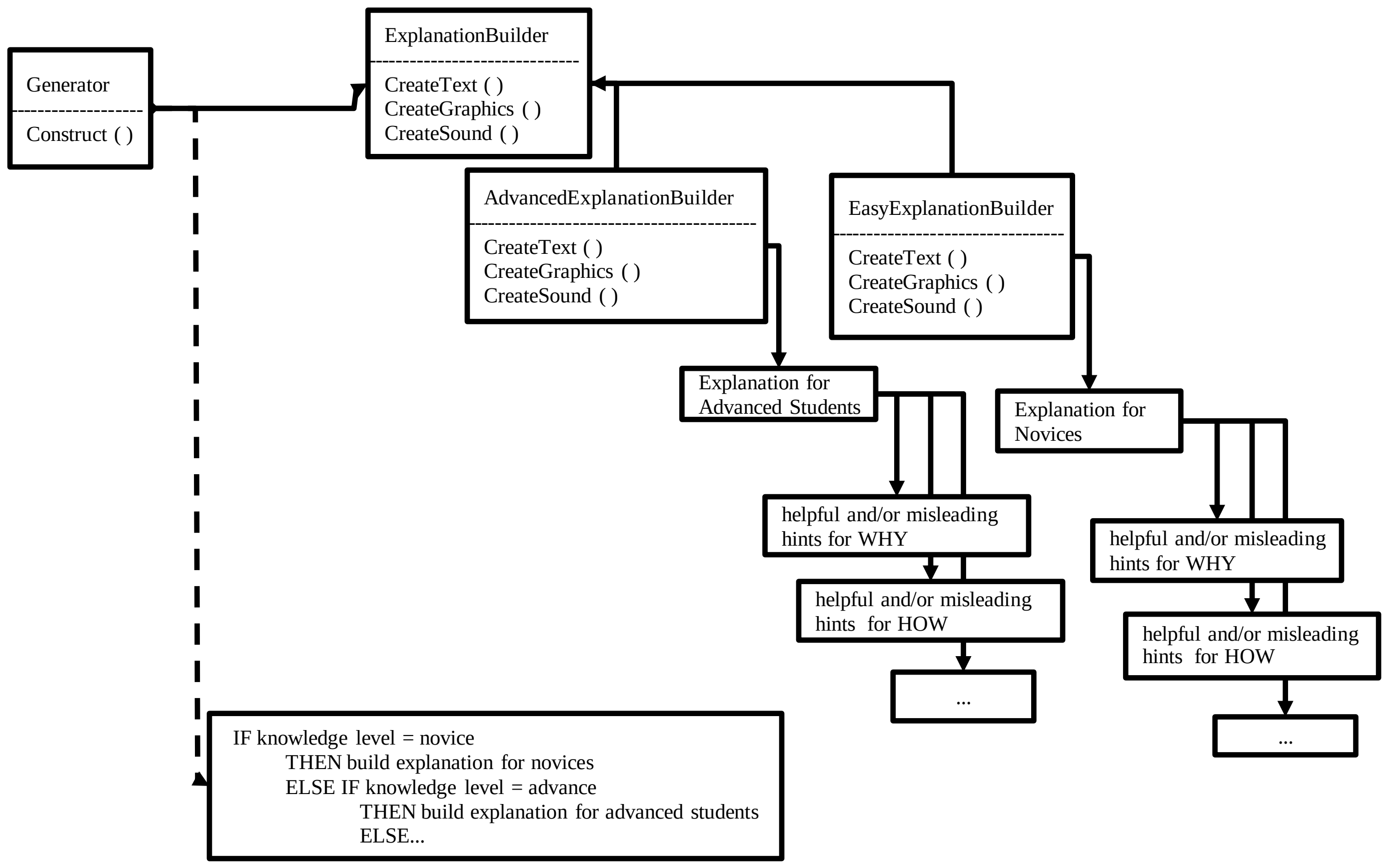

Generator
Construct ( )
ExplanationBuilder
CreateText ( )
CreateGraphics ( )
CreateSound ( )
AdvancedExplanationBuilder
CreateText ( )
CreateGraphics ( )
CreateSound ( )
EasyExplanationBuilder
CreateText ( )
CreateGraphics ( )
CreateSound ( )
Explanation for Advanced Students
Explanation for Novices
helpful and/or misleading hints for WHY
helpful and/or misleading hints for HOW
helpful and/or misleading hints for WHY
helpful and/or misleading hints for HOW
...
...
IF knowledge level = novice
THEN build explanation for novices
ELSE IF knowledge level = advance
THEN build explanation for advanced students
ELSE...

Fig. 2. Using the Builder pattern in designing explanation generator

C. Pedagogical Agents

The link between the student and the set of code completion tasks is provided in form of pedagogical agents. As noted earlier, two different types of pedagogical agents are used – one helpful, and one misleading in designing of ***Agent-**based **E-l**earning **S**ystem* (***AE-lS***).

As a crucial design step, both agents are hidden from the student behind the same interface, and take turns in interacting with the student at random time intervals. Therefore, the student is never sure with which agent he/she is interacting. The rationale behind this approach is straightforward: to motivate students not to trust the agent's hints blindly. Instead, they should critically analyze both the problem in question and the proposed hint, and, in the end, they them-selves decide on the proper solution.

In much of the scientific literature, as well as the actual software products, it is common to represent pedagogical agents as lifelike, animated characters. On the contrary, we feel that there is no real value in this approach. Primarily, many re-sources need to be put into designing and implementing a visually appealing character. However, although maybe "fun" to look at in the beginning, over the time the visual character and its built-in animations stand in the way of getting the job done. They distract the user/student from concentrating on the problem in question, and, in the extreme case, may negatively affect his/her willingness to use the system.

Pedagogical agents helpful and misleading are designed to increase student's productivity as primary goal. Consequently, no special attention for visual representations is considered. Purely, well-known characters from Office Assistant gallery, Clippy and Scribble, are used.

Both pedagogical agents are capable of adapting to each individual student. The agents track a set of information about the student, including his/her personal data, the ratio of correct and incorrect solutions to each code completion problem, and student's grade for each topic.

Based on the accumulated data, the agents can mediate if the student's success rate becomes unacceptable. For example, if the student gives to many wrong answers to questions regarding for loops, the pedagogical agent will recommend additional learning material or new examples, of course easier.

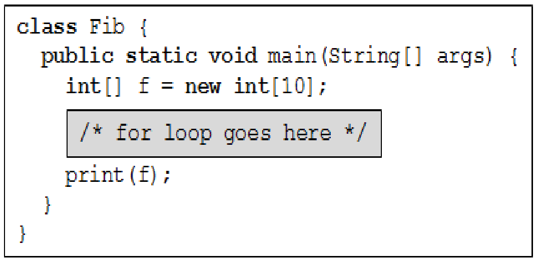
V. AE-LS EXAMPLE

Several important implementation requirements can be drawn from the functionality of ***AE-lS*** described earlier. For example, harvesting is a process that can and should be distributed and executed in parallel. Then, students should be able to interact with and use ***AE-lS*** through a web interface. Moreover, like all web-based systems, ***AE-lS*** should be resilient to hardware and software failures, malicious attacks, etc. Given these implementation requirements, and its popularity in developing software agents and multi-agent systems, Java has been chosen as the implementation platform for ***AE-lS***.

A. Helpful and misleading hints

In order to provide the reader with a better insight into the evaluation of *AES*, some examples of the prepared code completion tasks are given next. The two given tasks are tailored to the topics on "*For Loops*" and "*Classes*" in Java, respectively. Helpful and misleading hints assigned to each task are also presented and discussed.

The task tailored to the topic on "*For Loops*" in Java requires the student to complete a program for calculating the first 10 members of the *Fibonacci* sequence. The skeleton program presented to students is shown in Figure 3 [6].



```
class Fib {
  public static void main(String[] args) {
    int[] f = new int[10];
    /* for loop goes here */
    print(f);
  }
}
```

Fig. 3. Code completion task related to for loops.

Based on this skeleton, a set of helpful and misleading hints for pedagogical agents have been prepared. The helpful agent uses the following set of hints:

H1. *for (int i = ?; i < 10; i++){}* "What should be the starting index? Remember that the first element of the *Fibonacci* sequence has the index 0, while the expression for calculating other elements is $f_i = f_{i-1} + f_{i-2}$"

H2. *for (int i = 0; i <= ?; i++){}* "What should be the ending index? Although you need 10 numbers, remember that the index of the first element is 0."

H3. *for (int i = 0; i < 10; ?){}* "Should you use *++i* or *i++* to modify the value of *i*? Remember that this modification is always executed at the end of the for loop"

The misleading pedagogical agent uses the following set of corresponding hints (Ivanovic et al., in press):

H4. *for (int i = ?; i < 10; i++){}* "What should be the starting index? Hint: the first element of the *Fibonacci* sequence is often denoted as f_0"

H5. *for (int i = 0; i <= ?; i++){}* "What should be the ending index? Hint: look at the initialization of the array *f* - how many elements does it have?"

H6. *for (int i = 0; i < 10; ?){}* "Should you use *++i* or *i++* to modify the value of *i*? Remember that the instruction *++i* first increases the value of *i*, and then uses the new value in an expression."

By suggesting that f_0 is the first element of the *Fibonacci* sequence in hint H4, the misleading agent tries to suggest the improper usage of 0 for the initial value of *i*. In the general expression $f_i = f_{i-1} + f_{i-2}$ this decision would cause the index to go out of the array bounds. Similarly, in hint H5, the agent suggests that the student should use 10 as the final value of *i* (note the expression *i<=?*), disregarding the fact that Java array indexes are 0-based. The final hint H6 is just trying to confuse the student (i.e. to check whether the topic "*For Loop*" mastered with comprehension or not), since obviously both *++i* and *i++* are correct.

The example given in Figure 3. is extended as following. Lecturer should pay special attention in assembling and incorporating the suitable and appropriate examples and tasks for learning and testing the student's knowledge. For example, instead to give the usual task for realizing the concept of array and the sum of some numbers (the use of topics "*For Loop*" and/or "*Recursion*" in problem solving), the following problem (task) is given to the students:

"One mad scientist wants to make the chemical chain, made of plutonium and lead atoms. However, if two atoms of plutonium are side by side, the chain reaction and atomic explosion will be. How many of ways the safe chain could be constructed of the length N, if the mad scientist has N atoms of lead and N atoms of plutonium?",

The goal of above task is to practice the recursive technique of programming and to compare their results with previously done. This problem is given instead the ordinary problem like:

"Write Java method to realize the following mathematical function: $f_n= f_{n-1}+3$, $f_0=1$."

The student's task is to write a method that calculates some function similar to the methods used in example for *Fibonacci* sequence. The helpful agent uses the following set of hints:

H7. Try to remember what we have done last two classes? Something about calculating "Fib… seq…" and "Rec… method."

H8. First, try to make model, i.e. appropriate series, of the sequence of the atoms.

H9. Use that initial value is 1. What is the next value? Find the connection between the first and the second value.

H10. Try $f_n = f_{n-1} + 3$, $f_0 = 1$

The misleading pedagogical agent uses the following set of corresponding hints:

H11. Try to remember what we have done last two classes? Something about calculating "*rectangle*…" and "*Rec*… method."

H12. It easy, you could try $f_i = f_{i-1} + f_{i-2}$. Yeah, that is model of the sequence of these atoms.

H13. Use that initial value is 0. What is the next value? Find the connection between the first and the second value.

H14. Get stuck? Try $f_n = f_{n-1} + 3, f_0 = 0$

VI. Pedagogical Patterns for Learning by Mistakes

Learning by mistakes is very fine teaching techniques or teaching method. In teaching Computer Science, Informatics, Information Technologies, and similar disciplines based on technique or technologies, and it is used very often. Joseph Bergin proposed couple of general Pedagogical Patterns, which are directly involved in learning by mistake method of learning, with special implications in usage of them in teaching Computer Science [12].

They are:

- ***Mistake* -** Students are asked to create an artifact such as a program or design that contains a specific error. Use of this pattern explicitly teaches students how to recognize and fix errors. We ask the student to explicitly make certain errors and then examine the consequences.

- ***Grade It Again Sam* -** To provide an environment in which students can safely make errors and learn from them, permit them to resubmit previous assignments for reassessment and an improved grade.

In addition, some other general Pedagogical pattern could be used to explore the method of learning by mistakes, with smaller modification [12]:

- ***Fixer Upper* -** the lecturer makes the errors and the students correct them.

- ***Test Tube*** – the lecturer ask for explorations. Here lecturer could ask for explorations of specific errors.

Couple of Composite Pedagogical Patterns could be used, like:

- ***Design-Do-Redo-Redo (DDRR)* -** pattern by Marcelo Jenkins [15], used in teaching Object-Oriented Programming (OOP) to senior students based on a multi-language approach. The idea is to teach OOP concepts such as encapsulation, abstraction, and polymorphism, independently of the OOP language used. To do that, a Design-Do-Redo-Redo (DDRR) pattern is used, in which students design an OOP solution to a programming assignment and then implement it in three different languages. They have to elaborate differences and possible errors.

- ***Design-Implement-Redesign-Re-implement (DIRR)*** – pattern by Steve Houk [16]. The pattern could be used to bridge the gap from an old paradigm to a new paradigm (from procedural to object-oriented), emphasizes common programmers mistakes when they tried to "compile" solutions form procedural point of view to object-oriented directly, for example.

In the next chapter, one new Pedagogical Patterns for using the learning by mistake method in teaching Computer Science will be presented.

VII. Pedagogical Pattern Wolf, Wolf, Mistake

Topic, which is taught, is divided into smaller pieces called subtopics or fragments. Fragments are introduced systematic using ***Spiral*** [12] or ***Semiotic Ladder*** [13] patterns. The goal of the topic is to show usage of these fragments in solving certain problems. After the whole material is presented, some examples of implementation these fragments (or the methods based on them) are shown to the students. They have active participation in constructing the solutions. At the end, an artifact such as a program, object and/or design, with a particular error has been realized. Lecturer knows that mistake is made, but say nothing about that. At the end of the class lecturer just says that all examples have to be tested and verified as homework assignment. Next time, lecturer asks students do they found something in their homework assignments. Lecturer is interested about their opinions on the correctness of the solution that he presented last time. Students should explain the nature and possible consequences of the error, if they were find the mistake at all. Lecturer just conducts the discussion. Using this form, students learn how to recognize specific errors of construction and design, as well as the importance of testing software.

In the rest of this Section, the definition of Pedagogical Pattern Wolf, Wolf, Mistake is presented.

Title: ***Wolf, Wolf, Mistake***

Problem/Issue: Novice students make mistakes in programming, design, and particularly in problem solving. Moreover, they are aware of that. Students "believe" that teacher is a person who always tells the truth, so they accept the facts and solutions without checking them. Moreover, the students take and accept some facts without checking the source of them, from Internet for example. Students often do not know how to interpret the error messages, or what to do to solve problems that are diagnosed. Debugging and Testing are an essential skill, whether done with a sophisticated debugger, or just by comparing actual outputs or results with expectations, as well as to have the whole picture of the problem and test properly the given solution from teacher.

Audience/Context: This is very applicable to the early stages of learning programming. Syntax and semantic errors are frequent and students need to become familiar with the messages produced by compilers and run-time systems. In addition, the students have to understand what these errors indicate about the program. More over this pattern is good in learning the students about importance of proper testing the solution in problem solving. The pattern could also be used in an analysis or design course in which certain specific, but common, errors could be made easily.

Forces: Students, make errors in problem solving, more than professionals and/or teachers. They are not prepared to see the whole picture, yet. Students do not accept easily the fact that testing the solution is very important.

Teachers usually help students to pass up possible errors in problem solving techniques, telling them about all cases that have to be considered, before the solution is constructed. Moreover, teachers know how to test the solution properly. Therefore, the students became passive, not active participant in learning process. They simply accept and memorize the solution, instead to construct it, in sense to create new knowledge of some topics.

Solution: Some carefully chosen example in problem solving technique is presented to the students. Teacher creates solution from the beginning (understanding of the problem) to the end (making the code). The given solution has certain (hidden) specific errors (usually a single error).

Teacher then asks students to carefully consider and explore given solution, to test it, and to find is it good or not.

When the students find the error, give them the chance to elaborate and discuss the cause and the consequences. Use ***Gold Star*** [12] for the reward.

If students do not find the error, tell them that the solution is not good in some cases. Give them extra time and/or some hints, trying to activate them. Repeat the process until the solution is found.

Discussion/Consequences / Implementation: Students become more familiar with testing the given solutions. They understand why the error occurs, and how to correct it. Discovering the error, students could learn to avoid making it. The goals are to teach students how to analyze the problem properly as well as importance of the testing.

Examples for the use of this pattern should be carefully prepared. Otherwise, if there are too many errors or mistakes are too obvious, contra-effect could be produced.

This pattern can be used in many situations. In design part of Software Engineering course, problem solving courses, Object-oriented courses, and the like, the pattern could be successful. Moreover, it can be used in introductory programming course.

Special Resources: The instructor simply needs knowledge of the problem he thought; therefore, he could hide the trap.

Related Patterns: ***Fixer Upper*** **[12]**, ***Test Tube*** **[12]** and ***Mistake*** **[12]**.

Example/Instances: This pattern could be used effectively in teaching some introductory CS course. If you wish to teach the students about importance of analyzing the boundary cases in program design, and why the testing software is not an easy job, you may use this pattern.

For example, the pattern was used in Basic of Computer Literacy course for non-professionals (like students with major in Geography) at the University of Novi Sad. Topic on data types and potential problems with them (such as division by zero for numbers, for example) was taught at the beginning of the course. After a while, branching and control structures were done, and their usage in solving some problems is presented. The students together with lecturer solve some problem using these branching and control structures. The lecturer conducted the output. Nevertheless, students, i.e. for the particular data entry the program could crush, do not see the “hidden” special case. They miss to observe the case, which leads in dividing by zero. This case lecturer "wisely" ignore in the analysis of the task. Next class, if the students still did not notice the mistake, and lecturer admitted her/his “sin”, and explains the reason and consequences of mistake. Couple weeks later, students get the assignment very similar to previously, but in some other context. They all do the assignment without a single mistake.

In addition to those mentioned above, this pattern could be used effectively to teach students about pointers in languages like C or C++, by having them make all of the common pointer errors purposely. This particular use is somewhat dangerous on computers that have memory mapped I/O and unprotected operating systems. Both syntax and semantic errors can easily be explored using this pattern.

One exercise from an old book [14] was to write a program that produced every diagnostic mentioned in the manuals for a given (FORTRAN) compiler. This is, not surprisingly, very difficult to do. Impossible, for some compilers, as the documentation and the compiler are not parallel.

Contraindications: Do not use this pattern too often. You all know the fairy tale about a boy who cried wolf, wolf when there was none – everybody believed because he is a little boy, and they do not know to lie. He does it too many times, so when the wolf came, no one believed him. You could lose confidence and authority of experts in the eyes of students.

VIII. EXAMPLE OF PEDAGOGICAL PATTERN WOLF, WOLF, MISTAKE

“Our goal is to transform how children learn, what they learn, who they learn from.” (Mitchel Resnick, A Media Lab for Kids: $27 Million from Isao Okawa Creates Center for Future Children at MIT, MIT News. November 18, 1998.)

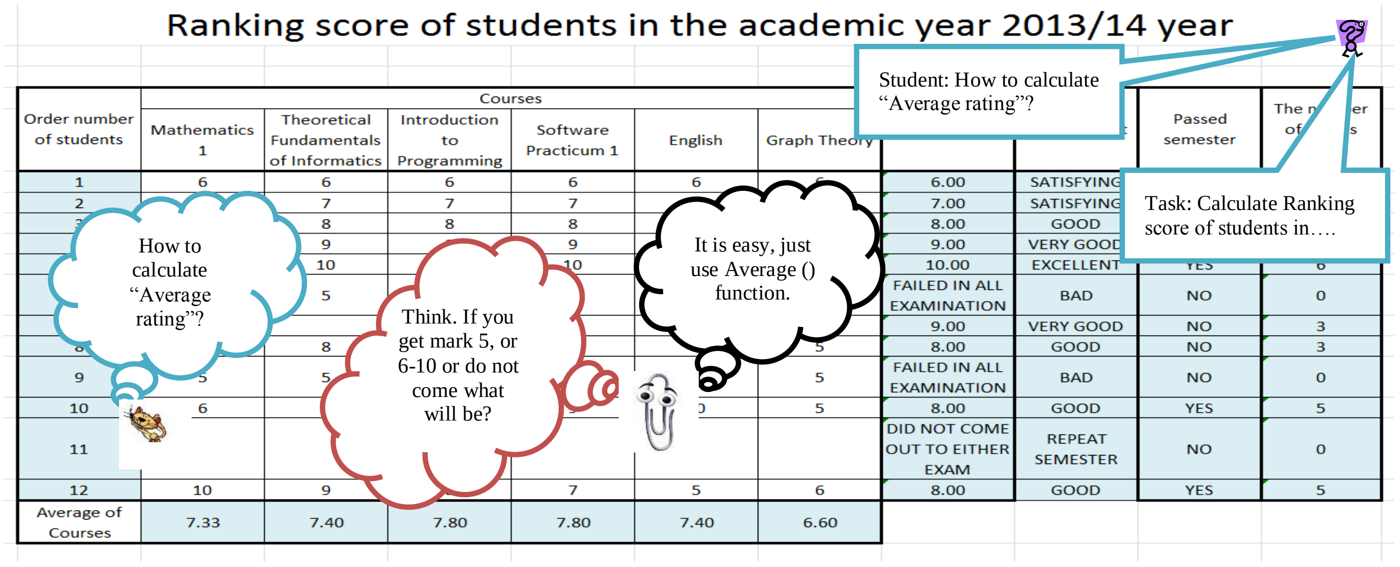
Therefore, our starting points are:

- We strongly believe that teaching is ART.
- Therefore, our first advice is to be a first-class artist on your stage (the classroom).
- It means, try to be different from others teachers in your environment, and engage your students to actively participate in lecture.
- Use a constructivist approach rather than objectivist in teaching.
- Use games and tools in teaching.

In addition, provide some home works for the students. For example, you finished classes about word processing in some course for computing literacy. After some time, give to the students your CV generating by Research Gate (for example), and ask them “How many times does my name appear in that document?”

Alternatively, novice students make mistakes in programming, design, and particularly in problem solving. Moreover, they are aware of that. Students “believe” that teacher is a person who always tells the truth, so they accept the facts and solutions without checking them. Moreover, the students take and accept some facts without checking the source of them, from Internet for example. Students often do not know how to interpret the error messages, or what to do to solve problems that are diagnosed. Debugging and Testing are an essential skill, whether done with a sophisticated debugger, or just by comparing actual outputs or results with expectations, as well as to have the whole picture of the problem and test properly the given solution from teacher. For example, the pattern was used in Basic of Computer Literacy course for non-professionals (like students with major in Geography) at the University of Novi Sad. Topic on data types and potential problems with them (such as division by zero for numbers, for example) was taught at the beginning of the course. After a while, branching and control structures were done, and their usage in solving some problems is presented. The students together with lecturer solve some problem using these branching and control structures. The lecturer conducted the output. However, students, i.e. for the particular data entry the program could crush, do not see the "hidden" special case. They miss to observe the case that leads in dividing by zero. This case lecturer "wisely" ignore in the analysis of the task. Next class, students still did not notice the mistake, and lecturer admitted her/his "sin", and explains the reason and consequences of mistake "she/he made". The usage of pedagogical agents is provided, helpful and misleading. Therefore, the students could try to re-solve task (Figure 4.).

Couple weeks later, students get the assignment very similar to previously, but in some other context. They all do the assignment without a single mistake.


Ranking score of students in the academic year 2013/14 year
Student: How to calculate "Average rating"?
Task: Calculate Ranking score of students in….
How to calculate "Average rating"?
Think. If you get mark 5, or 6-10 or do not come what will be?
It is easy, just use Average () function.

Order number of students	Courses: Mathematics 1	Theoretical Fundamentals of Informatics	Introduction to Programming	Software Practicum 1	English	Graph Theory			Passed semester	
1	6	6	6	6	6	6	6.00	SATISFYING		
2		7	7	7			7.00	SATISFYING		
3		8	8	8			8.00	GOOD		
		9		9			9.00	VERY GOOD		
		10		10			10.00	EXCELLENT	YES	6
		5					FAILED IN ALL EXAMINATION	BAD	NO	0
							9.00	VERY GOOD	NO	3
8		8				5	8.00	GOOD	NO	3
9	5	5				5	FAILED IN ALL EXAMINATION	BAD	NO	0
10	6					5	8.00	GOOD	YES	5
11							DID NOT COME OUT TO EITHER EXAM	REPEAT SEMESTER	NO	0
12	10	9		7	5	6	8.00	GOOD	YES	5
Average of Courses	7.33	7.40	7.80	7.80	7.40	6.60				

Fig. 4. The usage of pedagogical agents is provided, helpful and misleading.

The lecturer have to provoke with the right questions/tasks, to determine how students have progressed and understand what you are teaches, as well to engage your students to participate in lecturing actively, with aim of pedagogical patterns and agents.

At the end, good lecturer do not forget to use good old methods in teaching, like use of some physical device, such as a toy, that has some of the characteristics of the concept being taught. For example, use the Frisbees™, to explain the concept of a parameter passing in function and the difference between value and reference parameters in function calls - elementary programming course.

IX. Conclusion and Further Work

The Pedagogical Pattern Wolf, Wolf, Mistake is described. The pattern could be systematized in category of General Pedagogical Patterns. The example of usage of the pattern is presented. In further work realization some more Pedagogical Patterns for learning by mistake method will be realized, like Blow-Up, Crash the System etc. Also the important part of teaching Computer Science (and other scientific fields), Explanation Pedagogical Patterns and will be investigated. In addition, the pedagogical approach "Gradual Improvement and Stepped Development with Fine-tuning" [17] based on pedagogical patterns will be further researched and developed, for teaching programming.

The agent technology has been recognized as a useful tool in a wide variety of domains. From the point of view of e-learning and tutoring systems, harvester and pedagogical agents are of the special interest. Some examples of using e-learning system named ***AE-lS*** that efficiently incorporate both harvester and pedagogical agents based on pedagogical patterns approach are given.

A more important functionality, however, is achieved by defining two new sub-types of pedagogical agents - helpful and misleading. As noted, the helpful pedagogical agent provides correct suggestions and hints for the problem in question. On the other hand, the misleading agent tries to steer the problem solving process in a wrong direction, by offering false suggestions and hints. The main motivation for this approach is to motivate students not to follow the agent's directions blindly, but instead to analyze both the problem and the suggestions thoroughly, employ critical thinking, and, in the end, they themselves find the solution to the problem. According to our knowledge, none of the existing e-learning systems uses this kind of helpful and misleading pedagogical agents, in combination with pedagogical patterns.

The further work will be in two directions. First direction is definition of teaching agents. They help the teacher to build eLesson based on Constructivism. Constructivism offers a sharp contrast to teaching/learning [18]. First, the modern education is based on active student and student-centered teaching. Constructivism is a theory of learning based on the idea that knowledge is constructed by the knower based on mental activity. This approach is our contribution in replacing objectivistic learning theory at University of Novi Sad in teaching programming. In this view, students passively "absorb" programming elements, commands and structures presented by lecturer and documented in textbook, presentation, blackboard, etc. Teaching consists of transmitting sets of established facts, skills, and concepts to students. This is classical objectivistic approach in teaching.

The second direction is definition of “new pedagogy/didactics for teaching programming”, called ePedagogy/eDidactics of programming, based on pedagogical patterns: ***Gradual Improvement***, ***Stepped Development*** and ***Fine-tuning*** [17], to promote Constructivism. If you examine the tables of contents of most eLearning systems, you find that the underlying educational philosophy is one of Objectivism. This theory holds that the student's mind is an empty slate that the lecturer/teacher/instructor fills up. The systems approach to this kind of eEducation has the creator of that system examine the subject to be taught, divide it up into small bits, sequence the bits in some logical order, and then put all students through the same process of learning the material in that order.

For example, eTextbooks (most of eLearning materials are some kind of electronic textbooks and called Tutorials) for learning elementary programming suggest that IF statements MUST come before LOOPING statements and so they contain chapters devoted to everything about selection, before anything is seen of repetition. These eLearning systems are reference works, not learning materials. The objectivist theory ignores the fact that such a methodology is deadly boring to most students. First, it forces them to "learn" things they already know. Second, it ignores any individual difference in learning style or preference.

Constructivist educational philosophy, on the other hand, views the student as knowledgeable and task driven. New things are learned by integrating them into what is already known and it is done primarily so that meaningful (to the person) tasks may be carried out.

At the end, the “future” of using computers in education is the last direction. Instructional computer programs (or the usage of computers in education) are being developed since the early ‘70s. Rapid development of Information Communication Technology, introduction of computers into schools, and daily use of computers by people of different vocation, education and age, has made education a very important field to researchers. Their main goals have been to develop programs that can teach humans and to achieve individualization of the educational process. The methods and techniques of Artificial Intelligence have been successfully used in these systems, since the end of last century. Hierarchical modeling, interoperable and reusable software components, and ontology are modeling techniques that have only recently penetrated into the eLearning. In addition, these methods are used in new “field” called "eEducation", a new approach to education with the help of Information and Communication Technologies and Computer Science. The following questions have to be answered:

- Could we described "eEducation" = "eLearning" + "eTeaching", by this “simple” equation? Alternatively, do we need more "+"?
- Are we all (researchers, teachers and students) have succeeded in eEducation (eLearning) so far? Do "users" of eEducation (eLearning) systems are "better" than traditional students are, in a since of learning gain?
- Do we have right pedagogy (teaching methods/strategies) for eEducation (eLearning)?
- Do we have right learning strategies (models/theories) for eEducation (eLearning)?
- At the end, what is the future of eEducation (eLearning)?

REFERENCES

[1] Richland, L. E., Kornell, N. and Kao, L. S., “The Pretesting Effect: Do Unsuccessful Retrieval Attempts Enhance Learning?” Journal of Experimental Psychology: Applied, 2009, Vol. 15, No. 3, 243-257.

[2] Bergin, J., Eckstein, J., Manns, M. L., Sharp, H., Maraquardt, K., Chandler, J., Sipos, M., Völter, M. and Willingford E., “Pedagogical Patterns - The Pedagogical Patterns Project”, in Bergin J. (Ed.) *Pedagogical Patterns: Advice fir Educators*, Joseph Bergin Software Tools, 2012. (ISBN: 978-1-4791718-2-8)

[3] De la Prieta, F. and Gil A. B., “A Multi-agent System that Searches for Learning Objects in Heterogeneous Repositories,” in *Proc. PAAMS Special sessions and workshops: Trends in Practical Applications of Agents and Multiagent Systems*, 8th International Conference on Practical Applications of Agents and Multiagent Systems, Salamanca, Spain, 2010, pp. 355-362.

[4] Haake, M. and Gulz, A., “Visual Stereotypes and Virtual Pedagogical Agents,” *Educational Technology & Society*, vol. 11 no. 4, pp. 1-15, Oct. 2008.

[5] Heller, B. and Procter M., “Animated Pedagogical Agents and Immersive Worlds: Two Worlds Colliding,” in *Emerging Technologies in Distance Education*, G. Veletsianos (Ed.), Athabasca, Canada: AU Press, 2010, ch. 16, pp. 301-316.

[6] Ivanovic, M., Mitrovic, D., Budimac, Z., Vesin, B. and Jerinic, Lj., “Different Roles of Agents in Personalized Learning Environments,” In *Proc. of the 10th International Conference on Web-Based Learning - ICWL 2011*, Hong Kong, Dec. 8-10, 2011.

[7] Jerinic, Lj. and Devedzic, V., OBOA Model of Explanation Module in Intelligent Tutoring Shell. SIGCSE Bulletin, Vol. 29, Number 3, September, ACM PRESS, 133-135.

[8] Devedzic, V. and Jerinic, Lj., “Knowledge Representation for Intelligent Tutoring Systems: The GET-BITS Model”, In: du Boulay, B., Mizoguchi, R. (Eds.) Artificial Intelligence in Education, IOS Press, Amsterdam / OHM Ohmsha, Tokyo, 1997, 63-70.

[9] Devedzic, V. and Jerinic, Lj., “Explanation in Intelligent Tutoring Systems”, *Bulletins for Applied Mathematics*, 1196/96, 1996, 183-192.

[10] Jerinic, Lj. and Devedzic, V., An object oriented shell for intelligent tutoring lessons. Lecture Notes in Computer Science Vol. 1108, 1996, 69-77.

[11] Gamma, E., Helm, R., Johnson, R. and Vlissides, J., "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Reading, MA, 1994.

[12] Bergin, J.: Fourteen Pedagogical Patterns. In M. Devos and A. Rüping (Eds.): Proceedings of the 5th European Conference on Pattern Languages of Programms (EuroPLoP '2000), Irsee, Germany, July 5-9, 2000. UVK - Universitaetsverlag Konstanz 2001 ISBN 978-3-87940-775-0, pp. 1-49, 2000.

[13] Kaasbøll, J. J., "Exploring didactic models for programming", Tapir, pp. 195-203, 1998.

[14] Teague, R., "Computing Problems for FORTRAN Solution", Canfield Press, 1972.

[15] Jenkins, M., "Pedagogical Pattern #13: Design-Do-Redo-Redo (DDRR) Pattern", in Bergin J. (Ed.) *Pedagogical Patterns: Advice fir Educators*, Joseph Bergin Software Tools, 2012. (ISBN: 978-1-4791718-2-8)

[16] Houk, S., "Design-Implement-Redesign-Re-implement (DIRR) – Pattern", in Bergin J. (Ed.) *Pedagogical Patterns: Advice fir Educators*, Joseph Bergin Software Tools, 2012. (ISBN: 978-1-4791718-2-8)

[17] Jerinic, Lj. "Pedagogical Approach 'Gradual Improvement and Stepped Development with Fine-tuning' in Teaching Programming", in print.

[18] Jonassen, D., "Objectivism vs. Constructivism", Educational Technology Research and Development, 39(3), pp. 5-14, 1991.