Work in Progress: Exploring Programming Anti-patterns as Emphasis on Creativity in the Teaching of Computer Programming

Programming Anti-patterns as Emphasis on Creativity

Deller James Ferreira
Informatics Institute
Federal University of Goiás
Goiânia, Brazil
deller@inf.ufg.br

Abstract—**Computer programming is a creative activity. However, while computer scientists have devoted their work to solving complex problems and developing new technology, few have written on the creative process behind their innovations. This work explores programming anti-patterns as beneficial detour to the creation of good programming designs. Examples of programming anti-patterns are: failure to combine linear traversal and a flag variable to search a given value in a uni-dimensional array; to draw wrong analogies from natural language; not consider uncommon cases in a solution; to overlook minor parts of the problem; to assume that a program construct will work in the same manner in all situations; not be flexible in understanding new code, not being aware that there are many different right solutions to a problem. In this research, the teacher of introductory programming course promotes discussions among students involving the bad and good features of program anti-patterns, when there are programming anti-patterns code examples and discussions based on situations having the potential to prevent programming anti-patterns, when programming anti-patterns describe a lack of programming skill. An initial experiment showed improvements in creation of good program designs by students, when the teacher boosts discussions exploring program anti-pattens.**

Keywords-programming anti-pattern; creative programmer; collaborative programming

I. Scientific and Educational Feasibility of the Project, Expected Outcomes and Evaluation Plan

Computer science is a creative field to work, innovation is a necessary condition to create new products and technology and creativity is evident in software design processes. Although in computer programming creativity is required, pedagogic methods applying tasks that address creative aspects of programming are scarcely reflected in computer science education. Students should be fostered to develop and apply their creativity and computer science education researchers and teachers should approach creativity more often.

Our approach is under a creative perspective for programming. In this research, the teacher anticipates programming anti-patterns in order to encourage better critical analysis and creation of programs by means of explorations of good and bad features of a program anti-pattern. Sax and Dix [1] has investigated the exploration of bad ideas as a way to support creativity in human-computer interaction. They advocate that bad ideas encourage both divergent thinking and a more structured analysis of the problem, pulling the designer to a new and unpredictable place within the design space.

In this research, students are motivated to discuss about what is bad concerning a programming anti-pattern, why this feature is illogical, inadequate, and ill-crafted, if there is a good design possessing this feature, if so what is the difference, and if there is a situation where it could be considered well-crafted. Students also discuss about what is good concerning a programming anti-pattern, why this feature is logical, adequate, and well-crafted, if there is a bad design possessing this feature, if so what is the difference, and if there is a situation where it could be considered ill-crafted.

Here we apply collaborative programming. In collaborative programming, a group of students jointly perform the same task. Computer programming has traditionally been taught and practiced individually. Nevertheless, computer science educators have recently adopted different collaborative learning practices such as programming in pairs and team projects. By means of these collaborative activities, students can produce better programs and improve their performance and programming skills [2]. However, the teacher must help the students to develop the collective ability to use dialogs for learning, fostering productive interactions during argumentation in instructional settings. Discourse must be

This work is supported by FAPEG – Fundação de Amparo à Pesquisa – Goiás.

978-1-4673-1352-0/12/$31.00 ©2012 IEEE

facilitated aiming creative and innovative processes and products.

Collaborative programming combined with programming anti-patterns examination, constitutes a potentially powerful learning method for computer programming that provides profitable exploration of possibilities among the participants and provide a framework for social interaction that favors fruitful idea generation and program design discussion.

In this research, we expect students to produce program codes correctly in a greater percentage. Also, we aim student´s better programs understanding as a result of discussions concerning a great repertoire of not perfect program solutions. We expect to prevent students misunderstandings as well, going further in discussions based on situations having the potential to prevent programming anti-patterns, scrutinizing previous recurrent students bottlenecks.

For at least four semesters of computer science introductory courses, there will be one class taught by traditional method (control group) and another one taught by the proposed method (treatment group). Students performance will be confronted by questionnaires measuring their understanding of program codes, their grades, the percentage of the occurrence of correct and incorrect code generated by the students, as well as the percentage of the occurrence of programming anti-patterns classified by different types.

II. Importance to the Education Community

Traditional teaching proceeds by generating programs that meet problem constraints and best fit good design constraints. An usual approach is to guide students toward effective strategies for programming. However, understanding such strategies and program design demands a high level of comprehension and abstraction [3]. Novices are supposed to make connections beyond the scope of question, be able to transfer knowledge to a new situation, make a lot of interconnections among program components, and to integrate language constructs in a logical way in order to provide a computational solution to a problem.

Our motivation for the application of programming anti-patterns is that novice programmers usually cannot find their way out confronting a programming problem. They sometimes say “I don´t even know how to fail.” On the other hand, there are better students able to produce fully correct or almost right solutions. Recurrent and almost right solutions are then used to help students that the answer is blank or totally wrong.

To overpass student’s cognitive load and lack of programming knowledge and skills, instead of considering bad designs as problems to be overcome, students are encouraged to analyze and discuss them. Student´s bad program design is related to the student prior knowledge, contains correct parts, and is a naive solution. So, programming anti-patterns are easier to be understood and their exploration serve as springboards for the understanding and creation of good programming designs.

III. Project Status and Preliminary Results

During one semester of an introductory programming course, two classes containing forty under graduate students from food engineering course were analyzed considering students´ generation of language-based anti-patterns, such as wrong program sequence. Second, we analyzed design problems, i. e., students´ inability to properly combine, specialize, optimize, and students´ incapacity to visualize details during correct code adaptations. Third, we checked students´ skills to develop solutions to new problems, measured by means of the quantity of innovative program code generated.

The programming anti-patterns considered in this research were: inability to combine language features generating new programs; incapacity to improve programs; difficulties in identifying particular cases of known problems; bad usage of previous knowledge during programs interpretation; difficulties in delimiting control structures; inability to predict unexpected cases in a problem; incapacity to understand the sequence of a program; difficulties in decompose a problem in sub-problems; incapacity to identify and correct errors in programs; incapacity to understand step-by-step the program code; false belief that programming constructs have the same form in every program; false belief that there is only one correct solution for a computational problem; failure to combine correct program code; failure to adapt correct program code; failure to generalize examples to conceive a correct program code; failure to detect details during correct program code adaptation; inability to understand the program code considering correct program code combinations.

The experiment results showed that in the second class, where the teacher facilitated discussions involving programming anti-patterns, the occurrence of programming anti-patterns and code adaptation problems was smaller and the occurrence of innovative program code generated was bigger as shown in Table I.

TABLE I. Statistical Overview of the Results Obtained

Data Considering Programming Anti-patterns Application	**Percentage of Occurrence**
Programming Anti-patterns and Code Adaptation Problem	40.00% smaller
Innovative Program Code Generated	20.00% bigger

References

[1] C. Sax and A. Dix. Exploring the design space. In *DIS Workshop: Exploring Design as a Research Activity*. Proceedings of DIS Workshop: Exploring Design as a Research Activity, Penn State, USA, 2006.

[2] E. Verdú et al.. A distributed system for learning programming on-line.*Computers and Education.* Vol. 58, pp. 1-10, 2012.

[3] B. Haberman and O. Muller. Teaching abstraction to novices and ADT-based problem-solving processes. In 38th ASEE/IEEE Frontiers in Education Conference. pp. 1-6 October, 2008.