Learning by Doing – the Way to Develop Computer Science Professionals

Dimitar Christozov[1], John Galletly[2], Volin Karagiozov[3],

Stoyan Bonev[4]

[1] *American University in Bulgaria, Blagoevgrad 2700, Bulgaria, dgc@aubg.bg*

[2] *American University in Bulgaria, Blagoevgrad 2700, Bulgaria, jgalletly@aubg.bg*

[3] *American University in Bulgaria, Blagoevgrad 2700, Bulgaria, vkaragiozov@aubg.bg*

[4] *American University in Bulgaria, Blagoevgrad 2700, Bulgaria, sbonev@aubg.bg*

Expertise in software development is essential in the field of computer science. Such professionalism in designing and implementing computer programs is achieved solely by promoting practical work in the Computer Science curriculum, i.e. writing programs. This paper shares the experience accumulated in the Computer Science department of the American University in Bulgaria (AUBG) over the last 15 years in the evolution of offering students practical assignments. The emphasis is given to out-of-class practice. The paper also focuses on how to deal with the problems of academic integrity: "copying versus writing", grading individual members of teams, motivating students, etc.

Keywords

Software development, learning-through-cooperation, learning-through-competition, learning-through-challenge, academic integrity

1. Introduction

This paper describes a number of different approaches to teaching Computer Science that have been, and are, used at the American University in Bulgaria (AUBG). Over the last fifteen years, various approaches to creating a learner-centred environment such as blended education [1], learning-through-cooperation [2], learning-through-competition [3] and other skill-development methods have been tried. Our experiences with these approaches are described here.

The major focus of AUBG as an American Liberal Arts institution [5] falls on general education, which limits the offering of professional courses to not more than one third of the curriculum. Under these particular circumstances, some equilibrium between theory and practice must take place in every course. To achieve the required and expected level of professionalism in software development by the students, and at the same time to cover the fundamental theory needed for a successful future career, the practical aspects are left, in a majority of cases, as independent, self-practicing assignments or projects. This limits the ability of the instructor for efficient control on the real contribution of a student on the presented result, usually a developed piece of software, which may cause unfair grading and spoil the training process.

This paper shares the experience, problems, challenges and lessons learned in developing and applying different techniques to motivate students to do by themselves the homework assignments, while at the same time to eliminate the possibilities for cheating, plagiarism, copying, etc. In this way, students have to do by themselves the required assignments, which contribute to their growth as professionals.

2. Objectives and Problems

One of the major objectives in a computer science curriculum is to train the students to become professional software developers. Realistically, this can be achieved only through the students writing software. The approach of "learning-by-doing" is therefore critical in the development of computer science professionals. The curriculum is usually designed to guarantee a steadily increasing complexity of student software development assignments and longer projects.

Those students, who are able to manage their time in a way so as to complete all of the assignments, advance in their professional development and meet even the highest standards and expectations of potential employers. The others, whom, unfortunately, usually are numerous, try to present the required work without actually developing it. They disregard the principle of academic integrity. They have various avenues to do this such as copying from colleagues, downloading from the Internet, etc. Often, the instructor is unable to recognize and distinguish assignments done by a student from those copied by a student. Often, the copied work is of better quality and it is graded with higher mark than the original work. This may cause a better student to get a lower grade then a weaker student, and to discourage students to do the assignments by themselves.

This places the problem of developing practical approaches to motivate students to do the required work by themselves, while at the same time discouraging students to present copied materials, among the major pedagogical problems in the field of computer science.

General Problem of a Student: To do or to find?

Nowadays, the Internet provides a huge repository of information, including solved problems in almost every area of computer science. For many students finding the solution to class assignments replaces their own efforts to solve the problems, which itself nullifies the training effect. There are two issues which need to be addressed:

- Assessment and grading. Students have to know that there is a high probability that any form of cheating will be caught by the instructor and punished accordingly. That an even not complete, but self-made solution, will be graded higher than an excellent, but copied one. In the "Learning-through-Challenge" approach described below, it is really visible whether the student wrote the code or the code was professional, which allows demonstration of this policy.

- Formulating the assignments. The definition of an assignment has to include some elements, which requires personal contribution, at least a comprehensive study of the code of the solution found in Internet. In principle, to consider solutions provided via internet is not bad. The problem arises, when students do not try to understand the code and simply submit, what was found.

The techniques described in Section 4 are oriented to encourage students to work by themselves in solving programming assignments and to avoid simple copying from Internet or other students.

4. Techniques

There are a number of approaches suitable for different courses used to enhance learning-by-doing that we have explored. We describe three of them here.

4.1 Learning-through-Cooperation.

4.1.1 Project Teams

Commercial software development is performed basically in a team setting. This format is used in several AUBG Computer Science courses by allowing students to work in teams for their software projects. As well as developing individual software development skills, it is also essential for the students to build inter-personal skills in order to cooperate with other team members. Usually, all members of the team receive equal marks for the project, leaving it to students the responsibility of distributing the project work and equalizing their contributions.

A problem arises from the possibility that some students hide behind the work done by the others in the team. The culture of some specific groups of students, coming from countries with traditional and still preserved strong inter-group relationships, makes this problem more severe for institutions with a multinational student body (such as at AUBG). In such groups, the students who do the work cannot refuse the peer pressure of other students in the team just to allow them to “free-wheel” and to pass the course.

The result spoils the working atmosphere in the team and class, and compromises the fairness of the grades assigned by the instructor. Grading team work is a problem – students need to be evaluated according to their individual contribution to the project. An approached adopted at AUBG is to require every member of the team to present the entire project independently to the instructor, which guarantees at least a good understanding of the provided solution, and to receive a grade according to the demonstrated familiarity with the developed software.

4.1.2 Working in Pairs

In many cases, it is impossible to avoid some form of cooperation. Some students are living together and it is natural that they will work on assignments together. In such cases, it is better to allow cooperation in pairs [7]. This practice is heavily applied currently in agile approaches to software development such as extreme programming [6], and also at some universities in their programming courses. Additionally, pair programming allows students to learn by sharing each other’s good practices. Our observation shows that students applying this practice advanced better in their study. In some cases, when the pair is composed of an academically-stronger student and a weaker one, it may happen that the contribution of the stronger one is much higher. But even in such cases, the learning progress of the weaker student is visible.

In these cases it is better to allow cooperation, but to increase complexity and standards in grading. Again, it is important that students declare this cooperation in advance and present their work separately. As a result, they may achieve different marks.

4.2 Learning-through-Competition

Another way to decrease cheating possibilities and to increase students' motivation to work themselves is to place them in a situation of competition. This form encourages students to achieve higher quality in their assignments because they are placed in peer competition. Students are given exactly the same assignments and their results are presented in front of the class in a form of tender, when other students represent the jury. Our assumption is that students know better then instructors about unfair achievements, e.g. when a significant part of the assignment was downloaded. Placed in a situation where they are playing against each other, and not against the instructor, usually increases the students' self contribution.

We have applied this form by allowing other students to judge the quality of an assignment and to select the winner. But the final grade must not depend solely on the students' judgment. However, a small, though significant, percentage of the grade is reserved for this, in order to enhance each student's involvement in the process. Surprisingly, a majority of students prefer that grades be assigned by the instructor instead of by their peers, the students assuming a bias based on interpersonal relationships.

4.3 Learning-through-Challenge

Usually classes are composed of students with quite different backgrounds and abilities. This is visible especially in low-level courses. The complexity of practical assignments is of critical importance. Too complex assignments discourage the less-experienced, but good, students, and they may migrate to another discipline. Too simple assignments do not motivate experienced students to work and, often, results in diminished performance by them in upper-level courses.

To solve this problem – to motivate the more-experienced students to work and not to discourage the less-experienced students - we have adopted the practice of offering a really challenging (even for the best students) problem. This problem requires expertise beyond that expected of the majority of students at that level.

Those who manage to solve the problem achieve self-confidence, which encourages their further progress. The best students, for whom all other problems seem easy, facing such a challenging problem, realize that they need to study further and to work hard to comprehend the discipline.

The problem is with the less-experienced students, who may be discouraged. The role of the instructor is to find the way to use this disappointment to encourage these students to work more. They have to realize that mastering software development requires more effort, but the failure in solving this problem must not discourage them, because its complexity is well beyond the standard expectations for the course.

5. Success Factors

How is the effectiveness of applied strategies measured? What are the factors, which affects the motivation of students to do a required assignment and not to copy it?

Here we will share our good practice:

5.1 In formulating assignments and stating policy, the following factors have to be considered:

- Reflect the studied material. The assignments must follow closely the material studied in class. Even in the "challenging" approach, the problems must allow illustration of the studied material and students have to be able to solve the problems by applying the studied material.
- The process of development must follow the process of course material delivery. The practice is the best way to develop a stable body of knowledge and skills – practical assignments, especially in the form of homework with short deadlines, must be defined in a way to force students to do them in a period that follows immediately the presentation. Usually this is achieved by assigning a failure mark for delivery after the deadline, and deadline is set short enough (e.g. one or two weeks).
- Clear and detailed explanations of what is required and expected by the teacher.

5.2 Motivation and Grading

- To have fun. When the assignment constitute such a problem, it is more likely that students will try to do it by themselves; otherwise, if the assignment is a routine, boring one, it is more likely that students will try to find an acceptable solution, instead of doing it.
- Clear, transparent and fair grading. Students must to feel that their efforts will be awarded and that the probability to be punished when cheating is high enough.

The best example of such an assignment is the Sudoku solver given in the "Fundamental Data Structures" course. This course is taken usually as the only second programming course. As an illustration of the data structures, stack and queue, the homework with a two weeks deadline is to write two Sudoku solvers – one applying a depth-first search (DFS) and the second a breath-first search (BFS) algorithm, and to compare their performance, measured by steps. To make the problem doable, the Sudoku matrix is the simplest one – a 4x4 matrix, which does not create trouble requiring sophisticated memory management. The solutions available on Internet are too complex and usually do not apply directly the DFS or BFS algorithms. The available Internet solutions, which use some forms of these algorithms, never use both of them. This makes it visible in the code of the two solvers, presented by students, whether they did the homework by themselves.

6. Examples

In this section, we illustrate the above ideas by presenting parts of the syllabi, addressing practical assignments:

Plagiarism and cheating (learning-trough-cooperation: pairs):

"Any attempt for plagiarism and/or cheating will lead to immediate expulsion from the course. For homework assignments, cooperative work of up to 2 is possible. In such case the first student who submitted the work must attached a letter explaining who is the other contributor and what is his/her particular contribution. If such a letter is missing, or not provided by the first student, both students will be considered as practicing plagiarism and expelled from the course."

“Fundamental Data Structures” course

Class Project (learning-through-competition):
“The project is to present a proposal to develop and implement an Information System to support managerial activities of a company (writing a specification for the development of MIS) selected from a list provided by the instructor. The proposal has to be presented in a form of an offer for tender. The system has to support managerial activities at all levels of the company, in its branches, horizontal (between units on the same hierarchical level) and vertical (between levels) communication as well as communication between separate entities. If relevant, the system has to support the relationship between the company and its clients and suppliers. Two alternative teams will present offers for the same company in the same session. The two competitors have to agree about the time and order of presentation. The rest of the students in the class have to choose the winner of the tender.”
“Management Information Systems” course

Conclusion

As an American liberal arts educational institution, operating in Bulgaria, AUBG faces challenges to meet quality standards required by the two accreditation systems – one accepting a broader education and the other emphasizing a more narrow professionally-oriented training. The second accreditation system sets high standards on the content and comprehensiveness of the curriculum, which usually require much larger number of courses directly addressing computer science issues. In the case of AUBG, the same content must be covered in much smaller number of courses. This explains the necessity of moving as much as possible practice out of the class.

Additionally it is expected that graduates will be well prepared for both business and graduate studies, which makes intensive practice unavoidable. Building practical skills is essential for computer science professionals. Learning-by-doing is the only recognized way to achieve competence in software development.

To meet these challenges, we set up a flexible curriculum schema [4] which allows different pathways in passing through the program. In this scheme, the required Senior Project (capstone) course serves to demonstrate that students have ability to solve independently a real-world problem, which requires research, analysis, design and development of a significant software package. This course allows students to integrate and apply the knowledge and skills acquired in the course of their study.

The paper shares the expertise in offering out-of-class practice, accumulated during the fifteen years of training Computer Science at the American University in Bulgaria.

References

1 Uskov Vladimir, Student-Centered Learning in Online and Blended Education on Computer Information Systems, 33rd ASEE/IEEE Frontiers in Education Conference, November 5-8, 2003, Boulder, CO, p. T4F-17

2 Tammy Daub, Learning through cooperation, http://www.dukechronicle.com/

3 Grant, Kevin P., Achieving Organizational Learning Through Team Competition, Engineering Management Journal, March, 2004

4 Bonev S., Christozov D., Galletly J., Karagiozov V., Evolving a Computer Science Curriculum in a Liberal Arts Settings, Elektrotechnica & Elektronica, Volume 42, (1-2/2007), p. 20.

5 Bonev S., Christozov D., Galletly J., Karagiozov V. Computer Science Curriculum in a Liberal Arts Institution: Transition from ACM/IEEE Curriculum Model 1992 to 2001, Computer Science'2005, Technical University of Sofia, Posidi, Greece, 2005, pp. 213-217

6 Kent Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley, 2000

7 Williams, Laurie & Robert Kessler, All I Really Need to Know about Pair Programming I Learned in Kindergarten, Communications of the ACM 43(5): 108–114