Computers in Human Behavior 30 (2014) 623–628

Contents lists available at ScienceDirect

Computers in Human Behavior

journal homepage: www.elsevier.com/locate/comphumbeh

Assessment frequency in introductory computer programming disciplines

Miguel A. Brito *, Filipe de Sá-Soares [1]

Department of Information Systems, Centro Algoritmi, School of Engineering, University of Minho, Campus de Azurém, 4800-058 Guimarães, Portugal

ARTICLE INFO

Article history:
Available online 19 August 2013

Keywords:
Assessment frequency
Computer programming
Constructivism
Learning
Novice students
Programming education

ABSTRACT

Introductory computer programming disciplines commonly show a significant failure rate.

Although several reasons have been advanced for this state of affairs, we argue that for a beginner student it is hard to understand the difference between know-about disciplines and know-how-to-do-it disciplines, such as computer programming. This leads to failure because when students understand they are not able to solve a programming problem it is usually too late to catch all the time meanwhile lost.

In order to make students critically analyse their progress, instructors have to provide them with realistic indicators of their performance.

To achieve this awareness and to trigger corrective actions in a timely manner there is a need to increase assessment frequency. This paper discusses how this can be done, analyses benefits of the proposed approach and presents data on the effects of changes in assessment frequency for a university first year course in fundamentals of computer programming.

1. Introduction – reasons for failure

The literature identifies several reasons for students failure when learning to program (Robins, Rountree, & Rountree, 2003), with some addressing the curriculum, others focusing on methodologies (Cook, 2008), on resources, or on the best first programming language to use (Duke, Salzman, Burmeister, Poon, & Murray, 2000). The inherent difficulty of programming (Teague & Roe, 2008; Winslow, 1996) and controlling what kind of mistakes students are more likely to make (Spohrer & Soloway, 1986) are also addressed.

However, a well-known empirical truth about teaching programming is that a motivated student needs little guidance so he will succeed no matter how bad are the overall conditions, teachers included. Similarly, a student not motivated to spend some weekly hours practicing will fail no matter what the teacher says or how well the teacher explains all about computer programming.

Our teaching practice also led us to note that many students do not have a realistic idea about their effective study performance.

- Some think that understanding the solutions presented by the teacher or book is enough. They will probably try to solve their first problem during the assessment itself.
- Others go a little further and do some exercises but stop training as soon as they reach a solution for a certain kind of problem. However, being able to solve a 10-min problem in a couple of hours is far from sufficient and probably the student is not even sure how the solution works.
- A more insidious kind of problem emerges with students that always study in a (same) group. Frequently, what happens is that although everyone follows and contributes to the solution, it is always the same student that performs the first step from the problem statement. The others just follow the lead and genuinely believe they can alone solve the problem from the very beginning.

In this paper the main concern is to address the need to persuade students to critically analyse their study methodology and progress over the semester. So the focus is on failure reasons that can be overcome by tuning computer programming methodology of study.

Papers about difficulties in computer programming learning can be found in the literature (Jenkins, 2002). In (Gomes & Mendes, 2007) difficulties are divided into five categories:

- The teaching methods – this addresses the lack of personalized supervision and immediate feedback. Teachers must ensure students follow the most appropriate learning approach while respecting different learning styles, such as individual vs. group study.

* Corresponding author. Tel.: +351 253510319.
E-mail addresses: mab@dsi.uminho.pt (M.A. Brito), fss@dsi.uminho.pt (F. de Sá-Soares).
[1] Tel.: +351 253510319.

0747-5632/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.chb.2013.07.044

The focus on syntactical details instead of promoting problem solving capabilities also needs to be addressed.

- The study methods – learning computer programming is a very experimental and intense task, much different of other disciplines students are used to that are based on formulas or procedures memorization. Programming demands a lot of extra class work.
- The student's abilities and attitudes – there is a students' generalized lack of problem solving capabilities. They read the problem description instead of analyzing it and jump to solve something prior to understand the problem. They also lose many learning opportunities because when facing a difficulty they simply give up or ask for help which basically leads to the same outcome.
- The nature of programming – the high level of abstraction needed and the syntax complexity are also mentioned as difficulties.

The job of programming requires skills such as analytical thinking, creative synthesis, and attention to detail. An additional fundamental skill is the ability to abstract, consisting in "the process of identifying common patterns that have systematic variations" (Gabriel, 1989). A programmer needs to apply abstraction when analyzing computational problems, as well as to instantiate abstract programming concepts and techniques to solve particular computational problems.

The syntax complexity of programming languages raises several difficulties to novice programming students, since they may have to simultaneously struggle with the inherent complexity of the problem that has to be solved and the syntactical specificities of the programming language used to convey the computational solution to the problem.

- Psychological effects – the lack of motivation for several possible reasons, being well known that an unmotivated student will hardly succeed. This is aggravated by the fact that programming courses usually gain a reputation that passes from student to student of being difficult that so many of them start already feeling defeated. There is also the fact that computer programming is usually taught at the very beginning of higher education courses, coinciding with a transition and instability period in the students' life (Teague, 2008).

2. Constructivism in education

Constructivism is a learning theory in which Jean Piaget argues that people (and children in particular) build their knowledge from their own experience rather than on some kind of information transmission.

Later, based on Piaget's insights on the experiential learning paradigm, important works have been developed, such as Kolb's Experiential Learning Model (Kolb & Fry, 1975) which reinforces the role of personal experiment in learning and systematizes iterations of reflection, conceptualization, testing and back again to new experiences. A rich set of works about constructivism in education can be found in (Steffe & Gale, 1995).

Meanwhile, the discussion was brought to the computer science education field, with the claim that real understanding demands active learning on a lab environment with teacher's guidance for ensuring reflection on the experience obtained from problem solving exercises. In other words, passive computer programming learning will likely be condemned to failure (Ben-Ari, 1998; Hadjerrouit, 2005; Wulf, 2005).

Indeed, effective learning according to the constructivist perspective demands the mental construction of viable models. As argued by (Lui, Kwan, Poon, & Cheung, 2004), learning to program is a difficult endeavour because the learning process is susceptible to several hazards. Although all students face these hazards (Lui et al., 2004) observe that weak students get stumbled and stalled more easily when they encounter such hazards. From our experience of teaching computer programming courses, we think that the characteristics of weak students that magnify the impacts of those learning hazards are, to some extent, shared by the average novice student of computer programming – the student population target of this paper. Those characteristics include lack in training in abstraction processes, lack of a prior foundation for anchoring the construction of new knowledge, and low levels of confidence in themselves, in the teachers, and in the study materials or practices.

The main reason for the authors' concern with constructivism is the conviction that teaching must always be focused on the people learning process. Even in organizations in general, there is already the belief that the integration of knowledge has achieved limited success primarily because it has focused on treating knowledge as a resource instead of focusing on the people learning process (Grace & Butler, 2005).

3. Why weekly assessment

An interesting study that crosses students' individual cognitive level with the types of errors made (Ranjeeth & Naidoo, 2007) suggests the need to adopt innovative strategies in order to counter the seemingly perpetual rate of failure and at the same time increase the intensity for students with better cognitive level.

Whatever the reason, the ultimate truth is that a lot of things can go wrong when learning computer programming, especially in undergraduate courses.

So it is virtually impossible to prevent them all, mainly because students tend to overestimate their own understanding (Lahtinen, Ala-Mutka, & Järvinen, 2005) and usually they are not very open to follow teacher's good advices, especially if following those advices means more work.

This leads to the only winning strategy we have found so far: fail fast to learn sooner. If frequent assessment opportunities are given to the student, no matter the reason why he is eventually performing badly, two important goals are immediately achieved:

- there is still time to change student's study methodology and
- the teacher finally gets some real attention from the student to his good advices.

Even for students who do not need to fail to correct eventual study errors, we observe that weekly assessment is an extra motivation for not postponing study and consequently they will also attend next class better prepared (Becker & Devine, 2007).

Other medium term issues related with tuning the discipline from year to year are also addressed by weekly assessment. The ordering of different concepts by difficulty (Milne & Rowe, 2002) can be inferred from final examinations or by directly asking students and teachers, but if we have automated weekly assessments this kind of data is readily available. So it is easier tuning the classes' distribution along the year, dedicating more time and exercises to those issues we know students need more time to assimilate and eventually to concentrate easier subjects in less classes.

4. How weekly assessment

Weekly assessment involves several dimensions. First we need a methodology and then implementation resources. For the methodology we have a set of supposedly good advices and