Supporting Intense Needs of Assessment in Computer Programming Disciplines

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

[1] University of Minho, School of Engineering, Centro Algoritmi
Campus de Azurém, 4800-058 Guimarães, Portugal
mab@dsi.uminho.pt, fss@dsi.uminho.pt

Abstract. After several years of experience teaching computer programming disciplines, the major insight about how to succeed became very clear. Students must work in a weekly flawless base. Instead, students tend to study occasionally with strong peeks of work at assessment eves. However, implementing assessments in a weekly base requires a lot of resources and that is not easy to obtain. At an earlier stage, a sequence of experiments proved the influence of weekly assessment in students' success in computer programming disciplines. A methodology to guide the weekly rhythm was developed and finally an automated assessment tool solved the problem of lack of resources.

Keywords: programming education, e-learning platform, learning and teaching computer programming to novice students, assessment frequency, constructivism, learning management system.

1 Constructivism and Computer Programming Education

Learning to program is mostly about developing the ability of doing it rather than knowing how to do it. We usually illustrate this difference to our students using the metaphor of riding a bicycle. They can know all about the mechanics of it and fully understand how other people accelerate, break, turn, etc., but yet mastering this knowledge will not make them able to ride a bike.

This need of doing it in order to learn, or better saying, the need of doing it to achieve certain capabilities unreachable other way, led us to the foundations of constructivism. This theory justifies our insight of the absolute need of weekly practicing programming.

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 and others' work with the experiential learning paradigm, important works have been developed, such as Kolb's Experiential Learning Model [1], 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 [2].

Meanwhile the discussion was brought to the computer science education field claiming 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. Passive computer programming learning will likely be condemned to failure [3], [4], [5][13]. Indeed, constructivism can even be used to explain the problem of weak students and be part of the solution [6].

2 The Path till Weekly Assessment

Building on the belief of the importance of submitting students to more and more assessments but also facing the strong restriction of human resources to implement those assessments, we progressively introduced more frequent assessments.

A first and strong indicator of the weekly assessment success is the evolution of the percentage of students that stay till the end, i.e. that do not drop at the middle of the semester.

In 2004/2005 a small project was quarterly assessed; i.e. there were two assessment points per semester. The forty-eight per cent of students who did not abandon was clearly insufficient.

During 2005/2006 small problems resolution in computers' lab were added to the assessment on a monthly basis. This assessment paid well in students' success but represented a hard load of work to teachers.

In 2006/2007 a weekly automated theoretical-practical (TP) assessment was implemented and complemented with a quarterly laboratorial (L) assessment. That meant more work in preparing the automated testing batteries but it would be an investment for the future and took us back to quarterly practical assessment. The results were similar to the preceding year, better than two years before but still not satisfactory. The platform adopted was the Learning Management System (LMS) Moodle (http://moodle.org). Although the choice process is not relevant at this point it is important to stress the option for an open source platform as it will be obvious a few paragraphs below.

During 2007/2008 the frequency of laboratorial assessment was increased to monthly. Benefiting of the previous year investment in automated TP assessment we needed to prove the importance of also increasing laboratorial assessment frequency. Finally, the result reached a very acceptable level of seventy-six. This proved our insight about the results of increasing assessment frequency but created a new problem: unbearable teachers' workload.

So finally, 2008/2009 was undoubtedly the toughest year in this process with an investment in a plugin to the Moodle platform in order to perform automated testing of programming procedures. The whole programming tests battery was not completely finish in that year. But finally the whole assessment was automated and weekly performed.

Since then the percentage of students who do not give-up from the discipline continues to increase reaching a peak of eighty-six in the last academic year.

This whole evolution led not only to a supportable workload for teachers but also allowed to engage less experienced teachers to help in classes and also in the assessment process.

The ratio of approval was also growing – 4% in 2005/2006, 2% the year after, 1% in 2007/2008 and finally a huge 13% jump last year with the weekly fully automated assessment – a 20% total improvement in five years! It should be mentioned that this evolution was achieved neither by shrinking the syllabus nor by decreasing the level of rigor imposed to the course over the years.

3 The Learning Methodology

Weekly assessment induces a regular weekly rhythm. In order to fully take advantage of it a learning methodology was developed ([7]). As part of this methodology we have a set of supposedly good advices and tutorial support during classes but in this context the focus is on the method the students need to follow each week.

Some authors [8], [9] agree on the merits of frequent assessment but have a small enough number of students or a big enough number of hours x teachers.

We do not have such resources and have almost two hundred students and only a laboratorial teacher. Nevertheless, a third factor in the equation has proved to be the automated assessment of the problems solved by students.

4 The programming plugin

The mentioned need of automatizing the programming questions led to the implementation of a quiz plugin with some additional features.

Fig. 1 presents the high-level architecture of the plugin which is composed of five main modules (specific modules related to the setting of the plugin into the system are not depicted) and three storage resources.

The two modules “Student Interface” and “Teacher Interface” provide the interface to the plugin to students and teachers, respectively. These modules interact with plugin’s CSS definitions and with the language localization of the plugin (currently there are two languages available, namely Portuguese and English, although the extension of the plugin to other languages is very easy).

When the student is undertaking the assessment an extended interface is displayed for the programing questions.

In this kind of question, the student uses a *Resposta* (answer) box to write his/her solution to the problem. To avoid misspelled procedure names the procedure header is already in the *Resposta* box and the student only fills the respective body and other eventual auxiliary procedures.

If somehow the student loses control about what he already did it is possible at any time to press the *Reinicializar* (reset) button in order to restart from scratch and the box is cleared again with just the original content.

The button *Testar* (test), allows the student to evaluate that particular question. So at any time the student can ask for an assessment of his/her answer. The button *Testar*

is only available for the programming questions. Besides verifying basic syntax correction (for instance, if there is an unbalanced number of open versus close parentheses the system informs the student of that situation), the use of the *Testar* button just provides an indication of how good the students' answer currently is. This is done by providing a percent value indicating the grading that the answer would get if the student submitted that current answer.

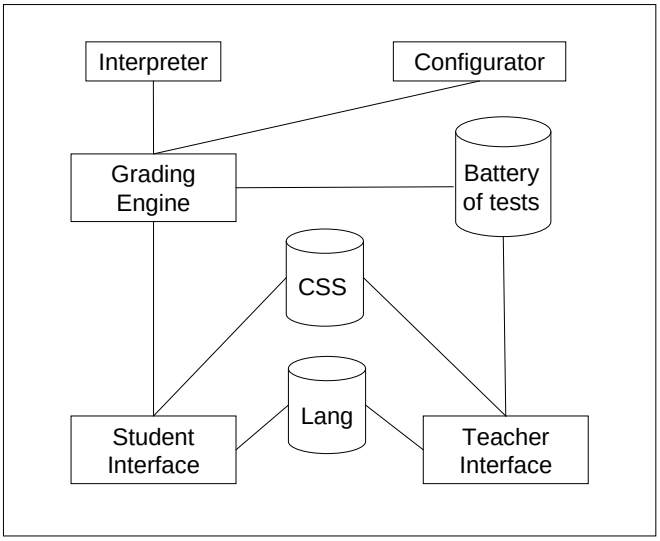

Fig. 1. High-level architecture of programming plugin

When defining a programming question an extended interface is presented to the teacher. Besides the standard fields related to the definition of questions, those that are specific to the plugin are:

- Answer Header – the text box where the teacher defines what will be displayed to the student
- Hidden Code – any code not to be displayed to the student that may be needed to the evaluation of the answer or to support the answer of the student, such as additional procedures or data structures (optional field)
- Procedure Name – the name of the main procedure whose correctness will be evaluated
- Allow Test – a boolean field that indicates if the student will be allowed to test (*Testar* button) the answer to this particular programming question
- Number of Input/Output Pairs plus Input/Output Pairs – the interface to the database that stores the battery of tests. The number of pairs and the pairs' contents should be careful selected, so they may consist of a thorough, complete, and non-obvious set of test cases for evaluating the correctness of the student's answer.

The "Grading Engine" is the module responsible for grading the student's answer. It is invoked when the student hits the *Testar* button (if displayed) or when the system is grading the student's answer. This module was designed according to the following top level algorithmic steps:

1. Strip answer of programming language comments;
2. Check answer syntax;
3. Evaluate the answer by submitting code to the battery of tests;
    a. For each pair of arguments (input)/expected return value (output), apply the procedure by calling the interpreter and accumulate partial grades;
    b. Prevent too long to accomplish processes or infinite loops;
    c. Calculate percentage of correctness;

The "Interpreter" is the programming language interpreter (or compiler) that is invoked to test student's answer.

The module "Configurator" allows some low level configuration of the "Grading Engine", such as programming language comment opening character sequence and comment ending character sequence; interpreter/compiler to use, its location and calling options; and time to wait before considering computational process is taking to long to complete.

5 Conclusions

Weekly assessment proved to induce the solution to a major problem in computer programming teaching, transforming high percentages of students' failure into high rates of approval.

However, it also brought a whole new kind of issues to address. Some of these issues are mainly security issues which were addressed by strict procedures during assessment and even a browser specially developed for the purpose of these assessments. How these issues were addressed is a whole new story that does not fit in this document but will certainly be addressed in a future one.

For this time the resources issue that came with weekly assessment was solved with resourcefulness with the construction of a Moodle quiz plugin which automates tests assessment.

The implementation of this plugin solved the issue of available resources, with the whole list of advantages inherent to the weekly assessments, which were already exposed above in this document. However two additional not negligible advantages were found:

- It became possible to use less qualified teachers at the laboratorial classes in which first half-hour the assessments are done.
- It is possible to tailor a specific programming assessment to the needs of particular groups of students in a very short period of time (for instance, students who cannot attend practical classes during the semester).

In the future it would be very useful to extend this plugin to other programming languages. However this demands to find very robust language interpreters in order to insure the test does not blow up during assessments. Other languages also demand an extended syntactic checker to provide students more support where heavy syntax is present. Another interesting improvement would be a first approach to the evaluation of programming style.

References

1. Kolb, D. and R. Fry, *Toward an applied theory of experiential learning*, in *Theories of Group Process*, C. Cooper, Editor. 1975, John Wiley: London.
2. Steffe, L.P. and J. Gale, eds. *Constructivism in Education*. 1995, Lawrence Erlbaum Associates: Hillsdale, New Jersey, United States.
3. Ben-Ari, M., *Constructivism in computer science education.* SIGCSE Bull., 1998. **30**(1): p. 257-261.
4. Hadjerrouit, S., *Constructivism as guiding philosophy for software engineering education.* SIGCSE Bull., 2005. **37**(4): p. 45-49.
5. Wulf, T., *Constructivist approaches for teaching computer programming*, in *Proceedings of the 6th conference on Information technology education*. 2005, ACM: Newark, NJ, USA.
6. Lui, A.K., et al., *Saving weak programming students: applying constructivism in a first programming course.* SIGCSE Bull., 2004. **36**(2): p. 72-76.
7. Brito, M.A. and F. Sá-Soares, *Computer Programming: Fail Fast to Learn Sooner*, in *Technology Enhanced Learning. Quality of Teaching and Educational Reform*, M.D. Lytras, et al., Editors. 2010, Springer Berlin Heidelberg. p. 223-229.
8. Duke, R., et al., *Teaching programming to beginners - choosing the language is just the first step*, in *Proceedings of the Australasian conference on Computing education*. 2000, ACM: Melbourne, Australia.
9. Costello, F.J., *Web-based Electronic Annotation and Rapid Feedback for Computer Science Programming Exercises*, in *Case Studies of Good Practices in Assessment of Student Learning in Higher Education*, G. O'Neill, S. Huntley-Moore, and P. Race, Editors. 2007: Dublin. p. 61-63.