doi: 10.1111/j.1365-2729.2010.00363.x

# Adaptive assessment of student's knowledge in programming courses

**D.I. Chatzopoulou & A.A. Economides**

Information Systems Department, University of Macedonia, 54006 Thessaloniki, Greece

**Abstract** This paper presents Programming Adaptive Testing (PAT), a Web-based adaptive testing system for assessing students' programming knowledge. PAT was used in two high school programming classes by 73 students. The question bank of PAT is composed of 443 questions. A question is classified in one out of three difficulty levels. In PAT, the levels of difficulties are adapted to Bloom's taxonomy lower levels, and students are examined in their cognitive domain. This means that PAT has been designed according to pedagogical theories in order to be appropriate for the needs of the course 'Application Development in a Programming Environment'. If a student answers a question correctly, a harder question is presented, otherwise an easier one. Easy questions examine the student's knowledge, while difficult questions examine the student's skills to apply prior knowledge to new problems. A student answers a personalized test composed of 30 questions. PAT classifies a student in one out of three programming skills' levels. It can predict the corresponding classification of students in Greek National Exams. Furthermore, it can be helpful to both students and teachers. A student could discover his or her programming shortcomings. Similarly, a teacher could objectively assess his or her students as well as discover the subjects that need to be repeated.

**Keywords** assessment of programming, computerized adaptive assessment, computerized adaptive testing, grading, personalized test, programming assessment.

## Introduction

Programming is composed of a broad scientific field that demands not just immaculate theoretical knowledge but also deep understanding of the framework of structured programming. Moreover, students need to have a deep understanding of the syntax of the language they are called upon to learn in order to practice. People involved in programming realize that the science of programming requires perfect handling of the logic behind the idea rather than ability of memorizing the syntax of different languages.

Accepted: 28 April 2010

Correspondence: Dimitra I. Chatzopoulou, Information Systems Department, University of Macedonia, Egnatia 156, Thessaloniki 54006, Greece. Email: chatzopoulou.dimitra@gmail.com

It is not uncommon that several students, upon completing a year of study on programming, exhibit serious shortcomings on basic programming knowledge (McCracken *et al*. 2001). It was found that students with little or no practical work were able to produce a piece of code in the final traditional way of assessment through memorization and achieve a 'good' grade in the course (Woit & Mason 2003). Furthermore, it is difficult to closely observe the progress of a particular student, especially in large classes. This happens because there is not enough available time for the teacher to interact personally with every student. Teaching and learning programming has created significant difficulties to both teachers and students (Wang & Wong 2008). Innovative ways are needed in order to improve the effectiveness of teaching programming. Assessing the students'



 

programming knowledge using computers in a regular and continuous basis could help (Traynor *et al.* 2006). The assessment results could be used for continuous improvement of teaching effectiveness and learning quality (Khamis *et al*. 2008).

The assessment should be carefully designed according to pedagogical theories. Lister and Leaney (2003a) encouraged teachers to design assignments according to the cognitive levels defined in the Taxonomy of Educational Objectives (Bloom 1956). These levels are the following (from lowest to highest) Recall of Data, Comprehension, Application, Analysis, Synthesis and Evaluation. However, it is difficult to categorize a question into the proper cognitive level (Thomson *et al*. 2008). Bloom's Taxonomy can be also used in the course design (Scott 2003). Oliver and Dobele (2007) argued that the lower cognitive levels (Recall of Data, Comprehension and Application) should be gained during the first year of studies. Subsequently, the students could become able to move onto assessments that require higher cognitive levels (Analysis, Synthesis and Evaluation). Otherwise, the assessment will have a negative effect on students to make 'upward progress'.

One of the problems faced by computer science instructors is bridging the following two gaps: the gap between the course and what to teach and the gap between what the students had been taught and how to assess this knowledge (Starr *et al*. 2008). This means that even if two schools offer the same course in computer science, the assessment can be different from one school to other because the teachers' objectives and teaching as well the students' demands may vary. So, the teaching and assessment should be tailored to each particular case.

This study developed PROGRAMMING ADAPTIVE TESTING (PAT), a computerized adaptive testing system for assessing students' programming skills in Greek high schools. The questions are categorized both into three difficulty levels and into three cognitive levels (Recall of Data, Comprehension and Application). If a student answers correctly a question, the next question is more difficult. Otherwise, the next question is easier.

Section 2 presents types of computerized assessment. Section 3 presents PAT, a multiple choice questions (MCQs) testing system that was developed and used in two high school programming classes by novice programmers. Section 4 describes the questions in the question bank as well the adaptive sequence of the questions. Section 5 analyses the results after the use of PAT by 73 students. Section 6 shows that PAT predicts the students' classification in Greek National Exams. Section 7 presents the strengths and weakness of PAT. Finally, section 8 presents the conclusions and future research.

## Computerized testing of programming skills

Computerized assessment offers speed, availability, consistency and objectivity of the assessment (Ala-Mutka 2005). In order to assess programming skills, two types of computerized assessment could be used: code writing and MCQs.

Whalley *et al*. (2006) showed that novice programmers were not yet able to work at fully 'abstract level' (high cognitive level). So students that can not read a short piece of code and describe it are not capable intellectually to write code by themselves. Thus, it is better to assess novice programmers using MCQs. On the other hand, if the students are at an advanced level and the course focus is on developing the students' programming skills then it is better to use code writing assessment. Of course, a combination of both types of assessment could be also used.

Next, both types of computerized testing of the students' programming skills are presented.

### Computerized testing using code writing exercises

There are many ways to answer an exercise in a programming language and more specifically, in high level programming languages. So many instructors prefer to correct manually the 'solutions' given by the students. Ala-Mutka (2005) found that 74% of instructors preferred the 'practical work of assessment'. However, the manual inspection of the code is inefficient and the possibility to overestimate or underestimate a student is increased (Kolb 1984) depending on the number of students.

Computerized testing could help in achieving accurate estimation of the student's knowledge. However, the design of a Code Correction and Assessment system presents many difficulties regarding to its objectivity (Schwieren *et al*. 2006).

Code writing assessment systems could be divided into fully automatic and semi-automatic systems (Ahoniemi *et al*. 2008).

The main differences between semi-automatic and fully automatic systems are the following: (1) in a semi-automatic system, a teacher completes the grading procedure; (2) semi-automatic systems are mainly used when the students are novice programmers and they need support from a human; (3) the marking in semi-automatic systems is flexible and the teacher can give partial marks to a student even if his or her program is not completely correct (Suleman 2008), which is not possible in fully automatic systems; and (4) the quality and efficiency of the source code are very hard or unfeasible to be evaluated via a fully automated system. A fully automatic system can not examine a student's program at an abstract level (e.g. meaningfulness of variables).

Next, the following semi-automatic tools are presented: ALOHA, ASSYST, EMMA, Sakai and TRY.

ALOHA (Ahoniemi *et al*. 2008) bases its objectivity on the use of 'rubrics' (Becker 2003) and statistical analysis of the achieved grades distribution (Ahoniemi & Reinikainen 2006).

In ASSYST (Jackson & Usher 1997), the students submit their assignments via e-mail. Instructors run the system, which tests and marks the submitted programs.

EMMA is a Web-based tool (Tanaka-Ishii *et al*. 2004) where students' programs are executed and tested on different inputs.

Sakai (Suleman 2008) can compile, test, execute and score the student's program without human intervention. If the output is not correct, then a feedback is given to the student regarding his or her produced output and the expected one.

In TRY (Reek 1989), the students submit their programs and the instructors test them. The output evaluation is based on textual comparison.

In the aforementioned tools the tutor defines the grading process and some template feedback phrases.

Next, the following fully automatic tools are presented: BOSS, Marmoset, Oto, and PASS3.

BOSS (Joy *et al*. 2005) supports both submission and testing of programs in various programming languages.

Marmoset monitors the student's progress and sends a feedback to both the student and the instructor (Spacco *et al*. 2006).

Oto is a marking tool that provides support for submission and marking of assignments in a programming course (Tremblay *et al*. 2008). At the end, it sends the grade and the marking report to the student.

PASS3 provides both immediate feedback to the student regarding his or her submission and a detailed performance statistic report regarding his or her progress (Choy *et al*. 2008). The differences with the previous version of PASS (Yu *et al*. 2006) are that there are multiple levels of difficulty, and a student selects the level according to his or her capabilities (Wang & Wong 2008).

The aforementioned tools helped to the creation of the xlx System (Schwieren *et al*. 2006). The student's code can be evaluated through static and dynamic control. The static control checks the source code for syntactic errors. The dynamic control additionally examines the code's performance, structure and output produced after its execution in relation to a standard code.

The common disadvantages of both semi-automatic and fully automatic tools are that both instructors and students should become familiar with such a system and the student must follow strict steps in order to complete his or her assessment. So code writing assessment is more suitable for advanced programmers than for novice programmers.

### Computerized testing using MCQs

It is a common belief among many (Traynor & Gibson 2005) in the field of education that MCQ tests are the easy and the lazy way to assess students. However, research (Lister & Leaney 2003b) has proved that quality MCQs is by no means 'the work of the lazy'.

According to Lister (2005), assessment through MCQs can be effectively administered to beginner programmers who have acquired basic skills. If a student scores poorly or averagely on basic skills, he or she is bound to fail on final exams, which are comparatively more demanding and require more knowledge. However, well-structured MCQs testing can be successfully used to test more complex skills (Wilson & Coyle 1991; Kolstad & Kolstad 1994; Jones 1997). Research has suggested (Rhodes *et al*. 2004) that MCQs are composed of a feasible assessment method, if the questions are qualitative in order to provoke students' knowledge and understanding of teaching material. Furthermore, according to Habeshaw *et al*. (1992), the objectivity can be achieved only through MCQs.

Denenberg (1981) stressed the need that evaluation results, questioning and structure must all be based on

quality; otherwise, the assessment results are of little value. MCQs are composed of a reliable evaluation method, not only in the theoretical field of information science but also in programming. In addition, the test's complexity could be increased by increasing the number of suggested answers or by the addition of short-length answer questions.

MCQs are divided into two categories (Denenberg 1981): knowledge questions composed of questions on theoretical knowledge such as gap-filling, true/false and multiple choice, and programming ability questions composed of code behaviour questions to examine the capability of students to comprehend the logic of programming. More specifically, Denenberg (1981) suggests that students should be able to read a program (e.g. find the output of the program), read a logical diagram (comprehension of its flows and operations), convert a logical diagram to a code and write a program (e.g. find commands from missing code).

Before exams are carried out, students should be fully informed on what they are supposed to do and how they are supposed to be graded.

Furthermore, Traynor and Gibson (2005) suggested the following requirements for effective MCQs: 'Good Quality Code', the code presented to the students should be of high standards and unstructured code should not be used; 'No Tricks', the questions should focus on the normal behaviour of the programs; and 'Quality Distracters', the erroneous answers given as alternatives should be appropriate and of high feasibility so as to ensure the sense of correctness in answers.

So many researchers believe that MCQs could be used not only for the students' assessment but also for students' practice on basic knowledge of programming. Moreover, the fact that correction and evaluation are carried out through the use of a computer renders the results objective and precise. For example, when a teacher has 100 papers to correct, there is the slight chance that he or she may overestimate or underestimate somebody's work. So Habeshaw *et al*. (1992) argued that the only way to objectively examine students is using MCQs.

## Presentation of PAT

PAT is a Web-based fully automated assessment system. It was developed with the use of a FLASH MX tool. The programming was conducted in ActionScript and the final files were extracted in html format. PAT can be used in the school's computer laboratory or via the Web from anywhere. It was tailored to the course of 'Application Development in a Programming Environment' (Bakali *et al*. 2004). This course is an introductory computer programming course in Greek high schools. This course is taught twice a week on a theoretical level, and if there is enough time, students are encouraged to carry out practice training, i.e. code writing in a real programming environment[1] or other pedagogical software. Instructors assess students in two semesters.[2] The second semester tests include the whole year teaching material. Semester tests and Panhellenic (Greek National) exams[3] are composed of paper tests, involving true/false, correspondence, output finding from a given code, conversion of logical diagrams into code or the opposite and code writing questions.

The emphasis concerning Semester tests or the Panhellenic (Greek National) exams is placed more on programming ability and knowledge questions (60%) than code writing (40%). It should be pointed out that students are examined in code writing only on paper. So most of the students do not have the experience of solving problems in a real programming environment.

Since these students are novice programmers, the most effective assessment method involves the use of MCQs instead of code writing. As we have already mentioned, code writing requires for students to exhibit an advanced level of knowledge in order to cope with the demanding material.

PAT was used in the schools' computer lab under the supervision of the teaching staff. The test takes approximately 45 min (one teaching hour). Students were assessed on 30 questions at the end of the second Semester and before the Panhellenic (Greek National) exams.

PAT is not only a software tool to assess novice students in programming, but it can also predict their classification in the programming course in the National Exams. Programming is composed of a core course in the technological direction of the General Lykeion (high school). Students are examined in programming in order to be admitted to Greek Universities (not necessarily only to enter computer science departments). Furthermore, PAT could be used in a Summative Assessment (Khamis *et al*. 2008), which could be used to assess the level of learning at the end of the course.

PAT was approved by the Greek Pedagogical Institute to be used in Greek high schools. During May 2009, 73 students from two schools (44 students from the first school and 30 students from the second school) used PAT. Also, they answered evaluation questionnaires regarding PAT's environment, question content and usefulness. Results showed that 61 students out of 73 found the experience positive and the tool very useful to increase their depth of knowledge in programming course and that they have been helped to discover their shortcomings. However, most of them think that they were underestimated by PAT in comparison with traditional exams. This may have happened because they do not have the experience in computerized exams. Most of the students (especially, low-performance students) preferred to use PAT for learning and self-assessment than for testing.

Next, several reasons are presented for using PAT:

- Students will be able to practice and be assessed in Knowledge and Programming Ability Questions.
- Most of the teachers who teach the programming course complain about the fact that teaching hours suffice only for teaching the exams material, leaving little time for practice. Through PAT, students will be able to practice more frequently, not just in the laboratory environment but also via the Web.
- Through the use of PAT, students will be able to discover their shortcomings in order to be prepared for the National Exams.
- PAT's friendliness will attract students of all levels to participate and practice as frequently as possible in order to increase their programming skills.

## Questions in PAT

The book's structure is such so that the exam material is repeated (Bakali *et al*. 2004). Chapters 1, 4 and 6 provide the theory and serve as an introduction on the necessity of programming; chapter 7 refers to the basic Programming elements and presents the pseudo-language (Glossa); chapters 2 and 8 provide an introduction to the structure of sequence, choice and repetition; chapters 3 and 9 present data structures, with an emphasis on tables; finally, chapter 10 deals with subprograms.

In PAT, each question is classified to a difficulty level: A = easy question, B = moderate question and C = difficult question. In addition, the question's content was developed according to the low levels of Bloom's Taxonomy (Bloom 1956).

The following categories of questions were developed:

- Recall of data: knowledge questions on the theory of the course, the syntax and function of frameworks of structured programming and of subprograms in true/false and MCQ format (difficulty level A, B or C). Such questions examine a student's memorization capability.
- Comprehension: a piece of code and a question involving the behaviour of the code (finding the output after the execution of a program). Such questions have been found efficient (Lister 2001) as far as student's assessment on their ability to read and comprehend the code's semantic (difficulty level B or C).
- Application: exercises to examine students' skills to apply prior knowledge to new problems. Three types of exercises were used: (1) a piece of code, which can be realized through a structure of process or choice or repetition, where a student is called to choose an equivalent command for the execution of the aforementioned functions (difficulty level B); (2) also a logical diagram is given where the student is called upon to find the equivalent command to express one or more functions (difficulty level C); and (3) gap filling in a piece of code or program according to some expressions (Lister & Leaney 2003a). Program gap filling difficulty (level B and mostly level C) is the most difficult activity and needs much more consideration and capabilities. Also, it helps students in increasing their power of solving sub-problems (Hwang *et al*. 2008).

These students were novice programmers. So they were examined at the lower levels of Bloom's Taxonomy. Oliver and Dobele (2007) showed that the pass rates of courses with higher cognitive demands (Analysis, Synthesis and Evaluation) were increased in relation with lower cognitive demands (Recall of Data, Comprehension and Application). This means that if a first year experience in programming demands a high cognitive level of assessment, then weaker students are prevented to continue their studies in this science.

The following Table 1 shows the number of questions with respect to Bloom's Taxonomy and difficulty level.

Table 1. Number of questions which respect to Bloom's Taxonomy and difficulty level.

| | A | B | C | Total number of questions per Bloom's level |
|---|---|---|---|---|
| Recall of data | 186 | 93 | 30 | 309 |
| Comprehension | 0 | 35 | 18 | 53 |
| Application | 0 | 20 | 61 | 81 |
| Total number of questions per difficulty level | 186 | 148 | 109 | 443 |

Table 2. Number of possible answers per difficulty level.

| Level of questions | Possible answers in multiple choice questions | True/false questions (two possible answers) |
|---|---|---|
| A | 3 | v |
| B | 4 | v |
| C | 5 | – |

Another factor that increases the difficulty of question is the number of possible answers. A student should have deeper knowledge of the subject in order to answer correctly a question with many possible answers than with few possible answers. Table 2 presents the number of possible answers per difficulty level.

### Model structure

PAT presents to a student 30 questions from various chapters of the exam material, depending on the students' level. Each student is tested on different questions at different levels. This ensures the quality of the exams as far as cheating is concerned since students sit in close proximity in computer laboratories.

The student moves from one difficulty level to another according to his or her answer. If he or she answers an 'A' question correctly, then the next question is 'B'; otherwise, it is 'A'. If he or she answers a 'B' question correctly, then the next question is 'C'; otherwise, it is 'A'. If he or she answers a 'C' question correctly, then the next question is 'C'; otherwise, it is 'B'.

At the end of the test, PAT shows the student's total number of correct and wrong answers per chapter and level. Also, it shows the student's total number of correct answers out of 30, his or her final score and classification.

## Grading

Significant effort was placed on feedback. PAT seeks to serve both the teacher and the student. As far as the student is concerned, PAT not only serves as a means of practice on the exam material but also as a means of feedback on the student's shortcomings per chapter. As far as the teacher is concerned, PAT functions as a means of assessing the students' programming levels, which indicate how well they are prepared for Panhellenic (National) exams. Then the teacher could try to help students overcome their weaknesses.

### Analysis of the results

If, following the aforementioned structure, the student correctly answers all 30 questions (from 0 to 29), he or she will obtain the following best performance sequence of question levels: A, B, C, C, C, C, C, C, C, C, C, C, C, C, C, C, C, C, C, C, C, C, C, C, C, C, C, C, C, C.

On the contrary, if the student answers all 30 questions incorrectly, the worst performance sequence of question levels will be as follows: A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A, A.

In the results printout, the answers given by the student are characterized by the letter of the difficulty level and the corresponding question number, LQn, where L is the difficulty level (A, B or C) and Qn is the number of the question at the corresponding level (Qn = 0.185 for level A, Qn = 0.147 for level B, and Qn = 0.108 for level C). For example, the following questions sequence appeared at a student's results printout: A5, B7, C3, B33, A12, B1, C77, C4, C100, B18, C5, C7, B22, A23, A27, A34, A47, A61, B75, C62, C55, C59, B81, C80, C19, B9, C0, C41, B29, A30.

This questions' sequence helps the teacher to immediately recognize which questions the student failed.

Regarding the example's questions sequence, the student answered wrongly the following questions:

- C3: because a level B question follows;
- B33: because a level A question follows; and
- also, C100, C7, B22, A23, A27, A34, A47, C59, C19, C41 and B29.

At the end of the test, the following results are presented for each student:

**1** *Total result (x)*: number of the correct answers out of 30.
**2** Number of the correct answers per level in relation to the total number of questions per level.
**3** *Final score (y)* given by the following formula:
Final score = 1 × number of correct answers at level A+
2 × number of Correct Answers at level B+
3 × number of Correct Answers at level C;
**4** Classification of student, which depends both on the total result and on the Final score.
More specifically the classification is calculated as follows:
if $(0 \leq x \leq 17)$ and $(0 \leq y \leq 33)$ → TRY HARDER – LOW PROGRAMMING SKILLS!
if $(16 \leq x \leq 20)$ and $(34 \leq y \leq 51)$ → GOOD – MEDIUM PROGRAMMING SKILLS!!
if $(21 \leq x \leq 30)$ and $(52 \leq y \leq 87)$ → VERY GOOD – HIGH PROGRAMMING SKILLS!!!
**5** Analytical results section contains all questions presented to the student per chapter during the test and the total wrong answers per chapter. This facilitates the student's study as he or she can study again the chapters in which he or she gave wrong answers.

An example of a result printout template is presented (Fig 1): upon closer examination of the result printout, it can be inferred that the majority of this student's correct answers belong to level A questions (six questions: A89, A119, A28, A139, A56, A3) and his or her total result is six correct answers out of 30 questions. So the student was unsuccessful in most of the questions.

This student's final score is 6/87 (6%). More specifically, out of the 24 level A questions, he or she answered correctly only six. So he or she achieved a final score = 6 out of 87. It is obvious that he or she is a low programming skills student.

Name: John Surname: Papadopoulos Class: G2

Your Total Result is: 6/30

TRY HARDER!
LOW PROGRAMMING SKILLS SPECIFICALLY

| | QUESTIONS | WRONG ANSWERS |
|---|---|---|
| CHAPTER 1 | 2 | 2 |
| CHAPTER 2 | 7 | 6 |
| CHAPTER 3 | 3 | 2 |
| CHAPTER 4 | 1 | 1 |
| CHAPTER 6 | 3 | 2 |
| CHAPTER 7 | 1 | 1 |
| CHAPTER 8 | 3 | 1 |
| CHAPTER 9 | 7 | 5 |
| CHAPTER 10 | 3 | 3 |

Final Score: 6/87=6%

Correct Answers in A level: 6/24
Correct Answers in B level: 0/6
Correct Answers in C level: 0/0

| QUESTIONS/ANSWERS | | | |
|---|---|---|---|
| A62 | a | A139 | d |
| A177 | b | B2 | a |
| A60 | c | A103 | d |
| A185 | a | A56 | b |
| A105 | a | B135 | c |
| A133 | b | A141 | a |
| A89 | a | A75 | a |
| B16 | c | A3 | a |
| A119 | a | B89 | b |
| B145 | b | A60 | c |
| A71 | a | A72 | d |
| A28 | c | A32 | a |
| A70 | d | A13 | b |
| A24 | a | A72 | a |
| A51 | c | A114 | c |

Fig 1 Result template: TRY HARDER (low programming skills).

Finally, the application menu also includes the choice 'teacher'. Through this choice, the teacher can read or print all the questions according to level and per chapter in order to evaluate the students' shortcomings in detail (which questions and what chapters).

Based on our investigation using 73 students, we classify the students into three classes.

*High programming skills' students*

We consider that a student could be classified as a high programming skills student if he or she answers correctly at least 21 questions and obtains a final score of at least 52/87 (60%).

*Example: high programming skills student with the lowest total result and final score*

A *A B* C B A *A B* C *B C C C C* C *B C* C *B C C C* C *B C C B C C C*

This student has two correct answers on level A, seven correct answers on level B, and 12 on level C. So, he or she achieves a final score = $2 \times 1 + 7 \times 2 + 12 \times 3 = 2 + 14 + 36 = 52/87$ and total result = 21/30. The majority of answers correctly answered (12) belong to level C.

A high programming skills' student will answer correctly questions mostly at level C (Fig 2). On average, high programming skills' students answered correctly 15 out of 30 (50%) questions from level C.

*Medium programming skills' students*

If a student performed well in knowledge questions and at a moderate level in programming ability questions, he

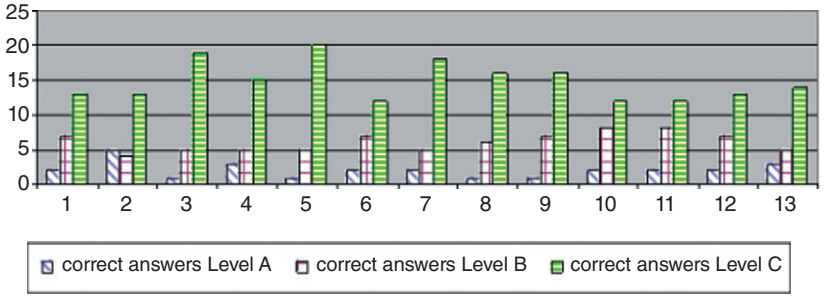


Fig 2 Correct answers per level by high programming skills' students (13 students out of 73).

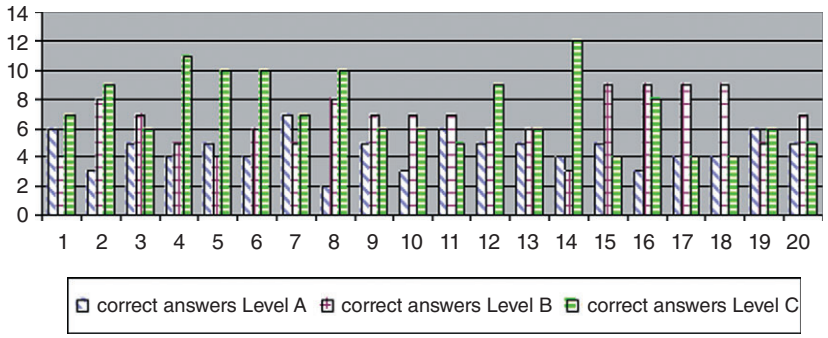


Fig 3 Correct answers per level by medium programming skills' students (20 students out of 73).

or she will be classified as a medium programming skills student. In our sample, most of the students answered correctly questions mostly to Level B and C (Fig 3).

As we can see from Fig 3, the number of correct answers is spread across all difficulty levels questions, but the majority belong to levels B and C questions (14 out of 30, approximately 50%).

In order for the student to be classified as a medium programming skills' student he or she will have to achieve a final score of at least 34/87 (39%) and a total result of at least 16/30. For a medium programming skills student, the highest total result is 20/30 and the highest final score is 51/87 (58.6%).

*Example: medium programming skills student with most correct answers from level C questions*

A *A B* C *B* C *B C* B A *B C C* C B *A B* C B *A B C C C C C C C C C C*

This student has four correct answers on Level A, six on Level B and 10 on Level C. So, he or she achieves a final score = $4 \times 1 + 6 \times 2 + 10 \times 3 = 4 + 12 + 30 = 46/87$, and total result = 20/30. This means that he or she answered questions (10) correctly mostly at level C. However, this is not enough to place the student at high programming skills. As we can observe, the student answered wrongly 10 out of the 30 questions, and as a result, he or she is properly placed as a medium programming skills student.

*Low programming skills' students*

A low programming skills student needs to study more. The majority of her or her correct answers do not necessarily belong to level A. However, the percentage of level C correct answers is lower than that of levels A and B. Otherwise, the student has a problem in questions that requires memorization.

Nevertheless, most of the students' correct answers were on level A questions (recall of data). Thirty-one students out of 40 show that frequency (Fig 4).

The highest total result that can be achieved by a low programming skills student is 17/30. Also, the highest final score is 33/87.

*Example: low programming skills' student with most correct answers from level A questions*

A A A A A A *A* B *A* B A *A* B A A A A A *A* B A *A* B A A *A* B A *A* B

This student only has seven correct answers at level A questions. So, he or she achieves a final score = $1 \times 7 = 7/87$ and a total result = 7/30.

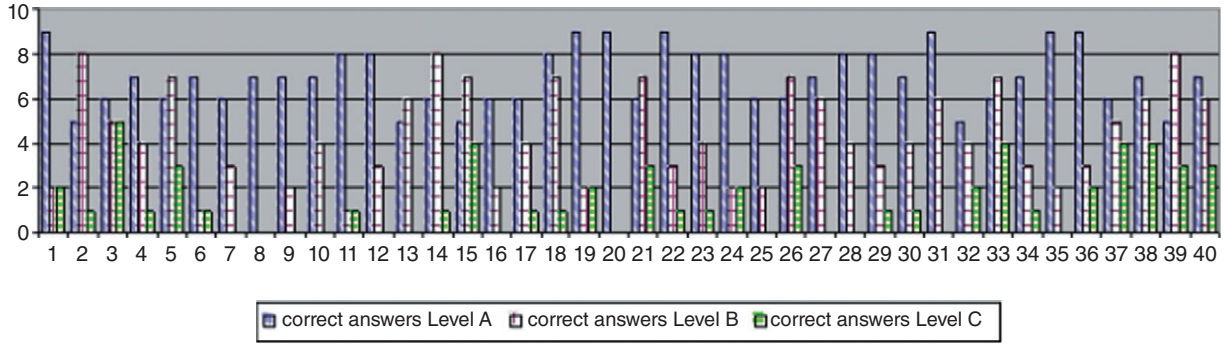


Fig 4 Correct answers per level by low programming skills' students (40 students out of 73).

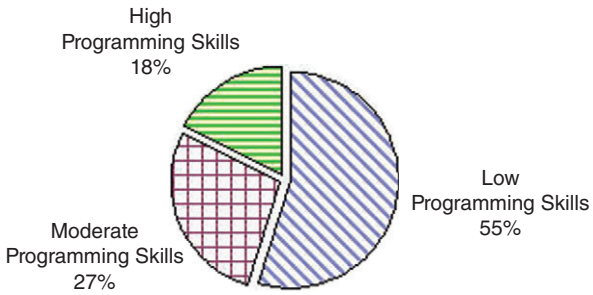


Fig 5 Classification of students using PAT.

Table 3. Correspondence between PAT students' classification, total result, final score and performance in national exams.

| Programming skills | Total result | Final score | Performance in national exams |
|---|---|---|---|
| High | 21–30 | 52–87 | 18–20 |
| Medium | 16–20 | 34–51 | 12–17.9 |
| Low | 0–17 | 0–33 | 0–11.9 |

## Prediction of students' classification in national exams

The results of the 73 students that took the test on PAT (Fig 5) indicate that 45% of them performed well. However, 55% of the low programming skills students need to practice more in order to achieve better grade in Panhellenic (National) exams. According to their teachers, most of the students do not practice often. They memorize instead of comprehend the logic of programming.

The following Table 3 provides the correspondence of the students' classifications using PAT and their expected performance in the programming course in National exams.

Table 4. Correspondence between PAT assessment and national exams assessment.

| Programming skills | Performance in national exams | PAT classification (%) | | National exams classification (%) | |
|---|---|---|---|---|---|
| High | 18–20 | 18 | 45 | 17 | 44 |
| Medium | 12–17.9 | 27 | | 27 | |
| Low | 0–11.9 | 55 | | 56 | |

Using PAT classification, in the two high schools where this study was carried out, we predicted that in the 2009 National Exams (computer programming course), 55% of the students will score below 12, 27% of them between 12 and 18 and 18% of them between 18 and 20 (maximum possible grade).

The following Table 4 testifies that PAT can predict the students' classification in National Exams (Computer Programming course). Indeed, 56% of the students scored below 12, 27% of them between 12 and 18, and 17% of them between 18 and 20.

## Strengths and weaknesses of PAT

PAT is a Web-based adaptive testing system for assessing high school students' programming knowledge. It is based on a graphical environment and is user-friendly. Its item bank contains a large number (443) of MCQs at various difficulty levels and Bloom's taxonomy levels. It presents to a student 30 randomized questions adapted to the student's programming knowledge. So every student receives different questions from the other students and cheating becomes almost impossible. The adaptive sequence of questions increases the student's motivation since he or she is challenged by the questions' levels.

PAT provides:

- adaptation to the student's programming skills;
- successful classification of the students;.
- prediction of students' performance in Greek National Exams;
- automated assessment process;
- speed in results production;
- a large library of questions – possibility of test repetition with renewed interest;
- memorization of questions by students is rendered difficult;
- indication of students' sufficient preparation for participation in Panhellenic (Greek National) exams;
- exposure of students' weaknesses per chapter of the exam curriculum;
- pleasant and usable graphic work environment (it was developed using FlashMx);
- convenience of practice either in school laboratories (local) or via the Web; and
- the execution of PAT software requires only the installation of a browser and one can run PAT from any hard disk device even without Internet connection.

However, PAT presents also some shortcomings. It contains items to test only beginners in programming. Also, it was developed to test student's programming skills on 'Glossa', a pseudo-language for Greek students.

## Conclusions and future research

Different schools in different countries have different requirements for teaching and assessing students' computer programming skills. PAT was developed to help Greek high school students and teachers to evaluate students' programming skills. PAT is not only an adaptive assessment tool, but it can also predict the students' classification in the corresponding course in National Exams.

Future work will further validate PAT's objectivity and reliability to accurately classify students. PAT will be extended to support the assessment of other programming languages (e.g. Java, Visual Basic) as well as code writing exercises. Then, it will be used by students at various schools as well as university departments in introductory programming classes.

PAT could be used as a self-assessment too. It will be extended to let a student choosing the chapter and the level that he or she wishes to be examined. Also, it could be extended to enable the teachers to upload their own questions. Finally, various statistical results regarding a question, a student and a class will be available to the student and the teacher.

## Notes

[1]They use a pseudo-language named 'Glossa', which can be best described as a Greek translation of PASCAL.

[2]At the end of the year, the average between first and second semester is computed, which determines the final grade for this lesson in the school certificate.

[3]These exams determine if the students are going to continue their studies in a high educational institution (university or technological educational institution).

## References

Ahoniemi T. & Reinikainen T. (2006) ALOHA – a grading tool for semi-automatic assessment of mass programming courses. In *Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006* (eds A. Berglund & M. Wiggberg), pp. 139–140. Uppsala University, Uppsala.

Ahoniemi T., Lahtinen E. & Reinikainen T. (2008) Improving pedagogical feedback and objective grading. *ACM SIGCSE Bulletin* **40**, 72–76.

Ala-Mutka K.M. (2005) A survey of automated assessment approaches for programming assignments. *Computer Science Education* **15**, 83–102.

Bakali A., Giannopoulos I., Ioannidis N., Kilias C., Malamas K., Manolopoulos J. & Politis P. (2004) *Application Development in Programming Environment – Third Class in General Senior High School of Greece*, 5th edition. Organization of School Books Publications, Athens.

Becker K. (2003) Grading programming assignments using rubrics. *ACM SIGCSE Bulletin* **35**, 253.

Bloom B.S. (1956) Taxonomy of educational objectives. In *Handbook I. Cognitive Domain* (ed. B.S. Bloom), pp. 201–207. Longman, White Plains, NY.

Choy M., Lam S., Poon C.K., Wang F.L., Yu Y.T. & Yuen L. (2008) Design and implementation of an automated system for assessment of computer programming assignments. In *Advances in Web Base Learning, Proceedings of the 6th International Conference on Web-Based Learning (ICWL 2007)* (eds H. Leung, Q. Li, F. Li & R. Lau), pp. 584–596. Springer, Berlin/Heidelberg. LNCS 4823.

Denenberg A.S. (1981) Test construction and administration strategies for large introductory courses. *ACM SIGCSE Bulletin* **13**, 235–243.

Habeshaw S., Gibbs G. & Habeshaw T. (1992) *53 Problems with Large Classes – Making the Best of A Bad Job*. Technical and Educational Services Ltd, Bristol.

Hwang W.-Y., Wang C.-Y., Hwang G.-J., Huang Y.-M. & Huang S. (2008) A Web-based programming learning environment to support cognitive development. *Interacting with Computers* **20**, 524–534.

Jackson D. & Usher M. (1997) Grading student programs using ASSYST. *ACM SIGCSE Bulletin* **29**, 335–339.

Jones A. (1997) Setting objective tests. *Journal of Geography in Higher Education* **21**, 104–106.

Joy M., Griffiths N. & Boyatt R. (2005) The BOSS online submission and assessment system. *Journal on Educational Resources in Computing* **5**, 1–28.

Khamis N., Idris S., Ahmad R. & Idris N. (2008) Assessing object-oriented programming skills in the core education of computer science and information technology: introducing new possible approach. *WSEAS Transactions on Computers* **7**, 1427–1436.

Kolb D.A. (1984) *Experiential Learning: Experience as the Source of Learning and Development*. Prentice Hall, Englewood Cliffs, NJ.

Kolstad R.K. & Kolstad R.A. (1994) Applications of conventional and non-restrictive multiple-choice examination items. *Clearing House* **56**, 153–155.

Lister R. (2001) Objectives and objective assessment in CS1. *ACM SIGCSE Bulletin* **33**, 292–296.

Lister R. (2005) One small step toward a culture of peer review and multi-institutional sharing of educational resources: a multiple choice exam for first semester students. In *Proceedings of the 7th Australasian Conference on Computing Education*, Vol. 42 (eds A. Young & D. Talhurst), pp. 155–164. ACM, Newcastle, NSW.

Lister R. & Leaney J. (2003a) First year programming: let all the flowers bloom. In *Proceedings of the Fifth Australasian Conference on Computing Education*, Vol. 20, pp. 221–230. ACM, Adelaide, SA.

Lister R. & Leaney J. (2003b) Introductory programming criterion – referencing, and Bloom. *ACM SIGCSE Bulletin* **35**, 143–147.

McCracken M., Almstrum V., Diaz D., Guzdial M., Hagan D., Kolikant Y.B.-D., Laxer C., Thomas L., Utting I. & Wilusz T. (2001) A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin* **33**, 125–180.

Oliver D. & Dobele T. (2007) First year courses in IT: a bloom rating. *Journal of Information Technology Education* **6**, 347–359.

Reek K. (1989) The TRY system-or-how to avoid testing student programs. *ACM SIGCSE Bulletin* **21**, 112–116.

Rhodes A., Bower A. & Bancroft P. (2004) Managing large class assessment. In *Proceedings of the Sixth Conference on Australasian Computing Education*, Vol. 30 (eds R. Lister & A. Young), pp. 285–289. ACM, Dunedin.

Schwieren J., Vossen G. & Westerkamp P. (2006) Using software testing techniques for efficient handling of programming exercises in an e-Learning platform. *Electronic Journal of eLearning* **4**, 87–94.

Scott T. (2003) Bloom's taxonomy applied to testing in computer science classes. *Journal of Computing Sciences in Colleges* **19**, 267–271.

Spacco J., Hovemeyer D., Pugh W., Emad F., Hollingsworth J.K. & Padua-Perez N. (2006) Experiences with Marmoset: designing and using an advanced submission and testing system for programming courses. *ACM SIGCSE Bulletin* **38**, 13–17.

Starr C.W., Manaris B. & Stavley R.H. (2008) Bloom's taxonomy revisited: specifying assessable learning objectives in computer science. *ACM SIGCSE Bulletin* **40**, 261–265.

Suleman H. (2008) Automatic marking with Sakai. In *Proceedings of the 2008 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries: Riding the Wave of Technology* (eds Ch. Cilliers, L. Barnard & R. Botha), pp. 229–236. ACM, Wilderness.

Tanaka-Ishii K., Kakehi K. & Takeichi M. (2004) A Web-based report system for programming course – automated verification and enhanced feedback. *ACM SIGCSE Bulletin* **36**, 278–285.

Thomson E., Luxton-Reilly A., Whalley J.L., Hu M. & Robbins P. (2008) Bloom's taxonomy for CS assessment. In *Proceedings of the Tenth Conference on Australasian Computing Education*, Vol. 78 (eds S. Hamilton & M. Hamilton), pp. 155–161. ACM, Wollongong.

Traynor D. & Gibson J.P. (2005) Synthesis and analysis of automatic assessment methods in CS1. *ACM SIGCSE Bulletin* **37**, 495–499.

Traynor D., Bergin S. & Gibson J.P. (2006) Automated assessment in CS1. In *Proceedings of the 8th Australian Conference on Computing Education*, Vol. 52 (eds D. Tolhurst & S. Mann), pp. 223–228. ACM, Hobart.

Wang F.L. & Wong T.L. (2008) Designing programming exercises with computer assisted instruction. In *Proceedings of the First International Conference on Hybrid Learning and Education (ICHL 2008)* (eds J. Fong, R. Kwan & F.L. Wang), pp. 283–293. Springer, Berlin/Heidelberg. August 13–15, 2008, LNCS 5169.

Whalley J.L., Lister R., Thompson E., Clear T., Robbins P., Kumar P.K.A. & Prasad C. (2006) An Australasian study of reading and comprehension skills in novice programmers, using the Bloom and SOLO taxonomies. In *Proceedings of*

*the 8th Australian Conference on Computing Education*, Vol. 52 (eds D. Tolhurst & S. Mann), pp. 243–252. ACM, Hobart, Tas.

Wilson T.L. & Coyle L. (1991) Improving multiple choice questioning: preparing students for standardized test. *Clearing House* **64**, 421–423.

Woit D. & Mason D. (2003) Effectiveness of online assessment. *ACM SIGCSE Bulletin* **35**, 137–141.

Yu Y.T., Poon C.K. & Choy M. (2006) Experiences with PASS: developing and using a programming assignment assessment system. In *Proceedings of the Sixth International Conference on Quality Software (QSIC'06)* (ed. H. Mei), pp. 360–368. IEEE, Computer Society, Washington, D.C.