Adaptive assessment in the class of programming

Dimitra I. Chatzopoulou and Anastasios A. Economides

economid@uom.gr
http://conta.uom.gr

Information Systems Department
University of Macedonia,
Egnatia 156, Thessaloniki 54006, Greece

Abstract

This paper presents PAT (Programming Adaptive Testing), 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 consists 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 consisting 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/her programming shortcomings. Similarly, a teacher could objectively assess his/her students as well discover the subjects that need to be repeated.

Keywords: assessment of programming, computerized adaptive assessment, computerized adaptive testing, grading, personalized test, programming assessment, programming assignments, programming examination, programming skills, programming testing, student’s classification.

1. Introduction

Programming comprises 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.

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. 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 & 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): 1) Recall of data, 2) Comprehension, 3) Application, 4) Analysis, 5) Synthesis, and 6) 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 & 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: 1) gap between the course and what to teach, and 2) 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 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.

The next section 2 presents types of computerized assessment. Section 3 presents PAT, a multiple choice questions 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 analyzes 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.

2. 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: 1) Code Writing, and 2) Multiple Choice Questions (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.

2.1. 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 over or under estimate 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 (Ahoniemy et al., 2008).

The main differences between semi-automatic and fully-automatic systems are the following: i) In a semi-automatic system a teacher completes the grading procedure; ii) Semi-automatic systems are mainly used when the students are novice programmers and they need support from a human; iii) The marking in semi-automatic systems is flexible and the teacher can give partial marks to a student even if his/her program is not completely correct (Suleman, 2008). This is not possible in fully-automatic systems; v) 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 (Ahoniemy et al., 2008) bases its objectivity on the use of “rubrics” (Becker 2003) and through 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 th student regarding his/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 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., 2007). At the end it sends the grade and the marking report to the student.

PASS3 provides both immediate feedback to the student regarding his/her submission and a detailed performance statistic report regarding his/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/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/her assessment. So, code writing assessment is more suitable for advanced programmers than for novice programmers.

2.2.Computerized Testing using Multiple Choice Questions (MCQs)

It is a common belief among many (Traynor & Gibson, 2005) in the field of education that multiple choice questions tests are the easy and the lazy way to assess students. However, research (Lister & Leaney, 2003b) has proved that quality multiple choice questions 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, s/he 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 (Jones 1997; Kolstad 1994; Wilson 1991). Research has suggested (Rhodes et al., 2004) that MCQs comprises 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 comprise 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): i) Knowledge questions consisting of questions on theoretical knowledge like gap-filling, true/ false and multiple choice, and ii) Programming ability questions consisting of code behavior 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,
- 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 & Gibson (2005) suggested the following requirements for effective Multiple Choice Questions: i) "Good Quality Code", the code presented to the students should be of high standards. Unstructured code should not be used, ii) "No tricks", the questions should focus on the normal behavior of the programs, and iii) "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 Multiple-Choice Questions 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 s/he may over-

or under-estimate somebody's work. So, Habeshaw et al. (1992) argued that the only way to objectively examine students is using MCQs.

3. Presentation of PAT

PAT (Programming Assessment Testing) 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 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] consist 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.

[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).

Since these students are novice programmers, the most effective assessment method involves the use of Multiple Choice Questions 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 minutes (one teaching hour). Students were assessed on 30 questions at the end of 2nd 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 National Exams. Programming comprises 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 1st school and 30 students from 2nd school) used PAT. Also, they answered evaluation Questionnaires regarding PAT's Environment, Question Content and Usefulness. Results show 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 to traditional exams. This may 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.

4. 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 Sub-Programs.

In PAT, each question is classified to a difficulty level: A = easy question, B = moderate question, 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 Sub-Programs in True/False and MCQ format (difficulty level A, B or C). Such questions examine student's memorization capability.
- **Comprehension**: A piece of code and a question involving the behavior 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 above 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); 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 & 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.

[Insert Table 1]

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.

[Insert Table 2]

4.1 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/her answer. If s/he answers an "A" question correctly then the next question is "B" otherwise it is "A". If s/he answers a "B" question correctly then the next question is "C" otherwise it is "A". If s/he 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/her final score and classification.

5. 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 student's shortcomings per chapter. As far as the teacher is concerned, PAT functions as a means of assessing the students' programming levels which indicates how well they are prepared for Panhellenic (National) exams. Then the teacher could try to help students overcome their weaknesses.

5.1 Analysis of the results

If, following the aforementioned structure, the student correctly answers all 30 questions (from 0 to 29), s/he 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

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:

(a) *Total Result (x)*: Number of the correct answers out of 30,

(b) Number of the correct answers per level in relation to the total number of questions per Level,

(c) *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

(d) Classification of student which depends both on the Total Result and on the Final Score.

More specifically the classification is calculated as follows:

if $(0<=x<=17)$ and $(0<=y<=33)$ →

TRY HARDER - LOW PROGRAMMING SKILLS!

if $(16<=x<=20)$ and $(34<=y<=51)$→

GOOD – MEDIUM PROGRAMMING SKILLS!!

if $(21<=x<=30)$ and $(52<=y<=87)$→

VERY GOOD – HIGH PROGRAMMING SKILLS!!!

(e) 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 s/he can study again the chapters in which s/he gave wrong answers.

An example of a Result printout template is presented (Figure 1):

[Insert Figure 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 (6 Questions: A89, A119, A28, A139, A56, A3) and his Total Result is 6 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 s/he answered correctly only 6. So, s/he achieved Final Score = 6 out of 87. It is obvious that s/he is a Low Programming Skills student.

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 students in 3 classes:

5.1.1 High Programming Skills' students

We consider that a student could be classified as a High Programming Skills' student if s/he answers correctly at least 21 questions and obtains Final Score 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 2 correct answers on level A, 7 correct answers on level B, and 12 on level C. So, s/he achieves Final Score = 2*1 + 7*2 + 12*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 (Graph 1).

[Insert Graph 1]

In order to support our argument for High Programming Skills' students, the following Table 3 is presented:

[Insert Table 3]

where Mean is the average number of correct answers per High Programming Skills student and StDev is the standard deviation. On average, High Programming Skills' students answered correctly 15 out of 30 (50%) questions from level C (Table 3).

5.1.2 Medium Programming Skills' students

If a student performed well in Knowledge Questions and at a moderate level in Programming ability Questions, s/he 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 (Graph 2).

[Insert Graph 2]

In order for the student to be classified as a Medium Programming Skills' student s/he 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%).

In order to support our argument for Medium Programming Skills' students, the following Table 4 is presented:

[Insert Table 4]

As we can see from the Table 4 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%).

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

This student has 4 correct answers on Level A, 6 on Level B and 10 on Level C. So, s/he achieves Final Score = 4*1 + 6*2 + 10*3 = 4 + 12 + 30 = 46/87, and Total Result = 20/30. This means that s/he answered correctly questions (10) mostly at Level C.

However, this is not enough to place the student at High Programming Skills. As we can observe the student answers wrongly 10 out of 30 questions and as a result s/he is properly placed as a Medium Programming Skills' student.

5.1.3 Low Programming Skills' students

A Low Programming Skills' student needs to study more. The majority of his/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 problem in questions that requires memorization.

Nevertheless, most of the students' correct answers were on level A questions (recall of data), 31 students out of 40 show that frequency (Graph 3).

[Insert Graph 3]

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.

In order to support our argument for Low Programming Skills' student, the Table 5 is presented:

[Insert Table 5]

The above Table 5 shows that the majority of correct answers are mostly from level A questions. Also, the average number of correct answers from level C questions is very low.

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 7 correct answers at level A questions. So, s/he achieves Final Score = 1*7 = 7/87 and Total Result = 7/30.

6. Prediction of students' classification in National Exams

The results of the 73 students that took the test on PAT (Graph 4) 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 comprehending the logic of programming.

[Insert Graph 4]

The following Table 6 provides the correspondence of the students' classifications using PAT and their expected performance in the Programming course in National Exams.

[Insert Table 6]

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 7 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.

[Insert Table 7]

7. 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 s/he 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.
- 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.
- 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.

8. 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 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 s/he 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.

REFERENCES

Ahoniemi, T. and Reinikainen, T. (2006). ALOHA – A grading tool for semi-automatic assessment of mass programming courses. Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006, Uppsala, Sweden, pp. 139-140.

Ahoniemi, T., Lahtinen, E. and Reinikainen, T. (2008). Improving pedagogical feedback and objective grading. ACM SIGCSE Bulletin, Vol. 40, No. 1, pp. 72-76.

Ala-Mutka, K.M. (2005). A survey of automated assessment approaches for programming assignments. Computer Science Education, Vol. 15, No. 2, June 2005, pp.83-102.

Arnow, D. and Barshay, O. (1999). On-line programming examinations using Web to teach. ACM SIGCSE Bulletin, Vol. 31, No. 3, pp. 21-24.

Bakali, A., Giannopoulos, I., Ioannidis, N., Kilias, C., Malamas, K., Manolopoulos, J. and 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, Vol. 35, No. 3, pp. 253.

Berry, R.E. and Meekings, B.A.E. (1985). A style analysis of C programs. Communications of ACM, Vol. 28, No. 1, pp. 80-88.

Bloom, B.S. (1956). Taxonomy of educational objectives. Handbook I. Cognitive Domain. Longmans, Green and Company, pp. 201-207.

Brusilovsky, P. and Sosnovsky, S. (2005). Engaging students to work with self-assessment questions: A study of two approaches. ACM SIGCSE Bulletin, Vol. 37, No. 3, pp. 251-255.

Califf, M.E. and Goodwin, M. (2002). Testing skills and knowledge: Introducing a laboratory exam in CS1. ACM SIGCSE Bulletin,Vol. 34, No. 1, pp. 217-221.

Carter, J., Ala-Mutka, K., Fuller, U., Dick, M., English J., Fone, W. and Sheard, J. (2003). How shall we assess this? ACM SIGCSE Bulletin, Vol. 35, No. 4, pp. 107-123.

Choy, M., Lam, S., Poon, C.K., Wang, F.L., Yu, Y.T. and Yuen, L. (2008). Design and implementation of an automated system for assessment of computer programming assignments. Advances in Web Base Learning, Proceedings of the 6th International Conference on Web-based Learning (ICWL 2007), Springer, LNCS 4823, pp. 584-596.

Daly, C. and Horgan, J. (2001). Automatic plagiarism detection. Proceedings of the International Conference in Applied Informatics, pp. 255-259.

Daly, C. and Waldron, J. (2004). Assessing the assessment of programming ability. ACM SIGCSE Bulletin, Vol. 36, No. 1, pp. 210-213.

Denenberg, A.S. (1981). Test construction and administration strategies for large introductory courses. ACM SIGCSE Bulletin, Vol. 13, No. 1, pp. 235-243.

English, J. (2002). Experience with a computer-assisted formal programming examination. ACM SIGCSE Bulletin, Vol. 34, No. 3, pp. 51-54.

Habeshaw, S., Gibbs, G. and Habeshaw, T. (1992). 53 problems with large classes- Making the best of a bad job. Technical and Educational Services Ltd., Bristol, U.K.

Hwang, W-Y., Wang, C-Y., Hwang, G-J., Huang, Y-M. and Huang, S. (2008). A web-based programming learning environment to support cognitive development. Interacting with Computers, Vol. 20, No. 6, pp. 524-534.

Jackson, D. (2000). A semi-automated approach to online assessment. ACM SIGCSE Bulletin, Vol. 32, No. 3, pp. 164-168.

Jackson, D. and Usher, M. (1997). Grading student programs using ASSYST. ACM SIGCSE Bulletin, Vol. 29, No. 1, pp.335-339.

Jones, A. (1997). Setting objective tests. Journal of Geography in Higher Education, Vol. 21, No. 1, pp. 104-106.

Joy, M., Griffiths, N. and Boyatt, R. (2005). The BOSS online submission and assessment system. Journal on Educational Resources in Computing, Vol.5, No 3, pp. 1- 28.

Khamis, N., Idris, S., Ahmad, R. and 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, Vol. 7, No. 9, pp. 1427-1436.

Kolb, D.A. (1984). Experiential Learning: Experience as the source of learning and development. Prentice Hall: Englewood Cliffs, N.J.

Kolstad, R.K. and Kolstad, R.A. (1994). Applications of conventional and non-restrictive multiple-choice examination items. Clearing House, Vol. 56, No. 4, pp. 153-155.

Lister, R. (2001). Objectives and objective assessment in CS1. ACM SIGCSE Bulletin, Vol. 33, No. 1, pp. 292-296.

Lister, R. and Leaney, J. (2003a). First year programming: let all the flowers bloom. Proceedings of the fifth Australasian conference on Computing Education, Vol. 20, Adelaide, Australia, ACM, pp. 221-230.

Lister, R. and Leaney, J. (2003b). Introductory programming criterion - referencing, and Bloom. ACM SIGCSE Bulletin, Vol. 35, No. 1, pp. 143-147.

Lister, R. (2004). Teaching Java first experiments with a pigs-early pedagogy. Proceedings of the sixth conference on Australasian computing education, Vol. 30, Dunedin, New Zealnd, ACM, pp. 177-183.

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. Proceedings of the 7th Australasian conference on Computing education, Vol. 42, Newcastle, New South Wales, Australia, ACM, pp. 155-164.

Mason, D.V. and Woit, D. (1998). Integrating technology into computer science examinations. ACM SIGCSE Bulletin, Vol. 30, No. 1, pp. 140-144.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D.,Laxer, C., Thomas, L., Utting, I. and Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. ACM SIGCSE Bulletin, Vol. 33, No. 4 , pp.125-180.

Oliver, D. and Dobele, T. (2007). First year courses in IT: A Bloom rating. Journal of Information Technology Education, Vol. 6, pp. 347-359.

Reek, K. (1989). The TRY system-or-how to avoid testing student programs. ACM SIGCSE Bulletin, Vol. 21, No. 1, pp. 112-116.

Rhodes, A., Bower, A. and Bancroft, P. (2004). Managing large class assessment. Proceedings of the sixth conference on Australasian computing education, Vol. 30, Dunedin, New Zealand, ACM, pp. 285-289.

Schwieren, J., Vossen, G. and Westerkamp, P. (2006). Using software testing techniques for efficient handling of programming exercises in an e-Learning platform. The Electronic Journal of e-Learning, Vol. 4, No. 1, pp. 87-94.

Scott, T. (2003). Bloom’s taxonomy applied to testing in computer science classes. Journal of Computing Sciences in Colleges, Vol. 19, No. 1, pp. 267-271.

Spacco, J., Hovemeyer, D., Pugh, W., Emad, F., Hollingsworth, J.K. and Padua-Perez, N. (2006). Experiences with Marmoset: Designing and using an advanced submission and testing system for programming courses. ACM SIGCSE Bulletin, Vol. 38, No. 3, pp. 13-17.

Starr, C.W., Manaris B. and Stavley R.H. (2008). Bloom’s taxonomy revisited: Specifying assessable learning objectives in computer science. ACM SIGCSE Bulletin, Vol. 40, No. 1, pp. 261-265.

Suleman, H. (2008). Automatic marking with Sakai. 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, Wilderness, South Africa, ACM, pp. 229-236.

Tanaka-Ishii, K., Kakehi, K. and Takeichi, M. (2004). A Web-based report system for programming course – automated verification and enhanced feedback. ACM SIGCSE Bulletin, Vol. 36, No. 3, pp. 278-285.

Thomson, E., Luxton-Reilly, A., Whalley, J.L., Hu, M., Robbins, P. (2008). Bloom's taxonomy for CS assessment. Proceedings of the tenth conference on Australasian computing education, Volume 78, Wollongong, Australia, ACM, pp. 155-161.

Traynor, D. and Gibson, J.P. (2005). Synthesis and analysis of automatic assessment methods in CS1. ACM SIGCSE Bulletin, Vol. 37, No. 1, pp. 495-499.

Traynor, D., Bergin, S. and Gibson, J.P. (2006). Automated assessment in CS1. Proceedings of the 8th Austalian conference on Computing education, Vol. 52, Hobart, Australia, ACM, pp. 223-228.

Tremblay, G., Guerin, A., Pons, A. And Salah, A.(2008). Oto, a generic and extensible tool for marking programming assignments. Software: Practice and Experience, Vol. 38, No. 3, p.p. 307-333

Wang, F.L. and Wong T.L. (2008). Designing programming exercises with computer assisted instruction. Proceedings of the First International Conference on Hybrid Learning and Education (ICHL 2008), Hong Kong, China, August 13-15, 2008 Springer, LNCS 5169, pp. 283-293.

Whalley, J.L., Lister, R., Thompson, E., Clear, T., Robbins, P., Kumar, P.K.A. and Prasad, C. (2006). An Australasian study of reading and comprehension skills in novice programmers, using the Bloom and SOLO taxonomies. Proceedings of the 8th Austalian conference on Computing education, Vol. 52, Hobart, Australia, ACM, pp. 243-252.

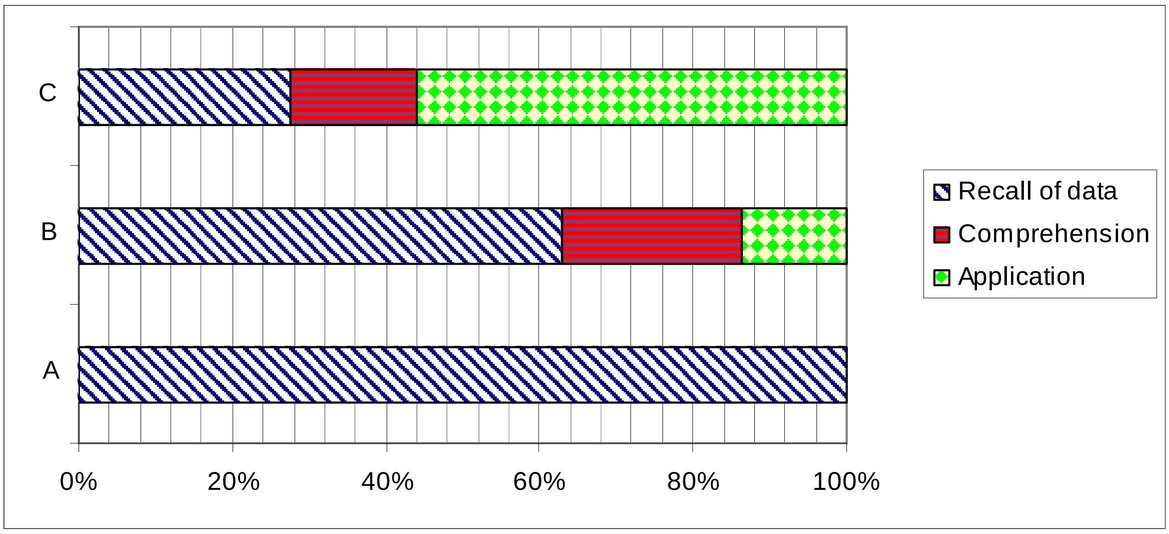
Woit, D. and Mason, D. (2003). Effectiveness of online assessment. ACM SIGCSE Bulletin, Vol. 35, No. 1, pp. 137-141.

Wilson, T.L. and Coyle, L. (1991). Improving multiple choice questioning: Preparing students for standardized test. Clearing House, Vol. 64, No. 6, pp. 421-423.

Yu, Y.T., Poon, C.K. and Choy, M. (2006). Experiences with PASS: Developing and using a programming assignment assessment system. Proceedings of the Sixth International Conference on Quality Software (QSIC'06), IEEE, pp. 360-368.

	A	B	C	Total # Questions per Bloom's Level
Recall of data	186	93	30	**309**
Comprehension	0	35	18	**53**
Application	0	20	61	**81**
Total # Questions per difficulty level	**186**	**148**	**109**	**443**

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



Graph 1: Percentages of questions which respect to Blooms' Taxonomy and difficulty level.

Level of Questions	Possible Answers in Multiple Choice Questions	True/ False Questions (2 possible answers)
A	3	v
B	4	v
C	5	---

Table 2: Number of possible answers per difficulty level

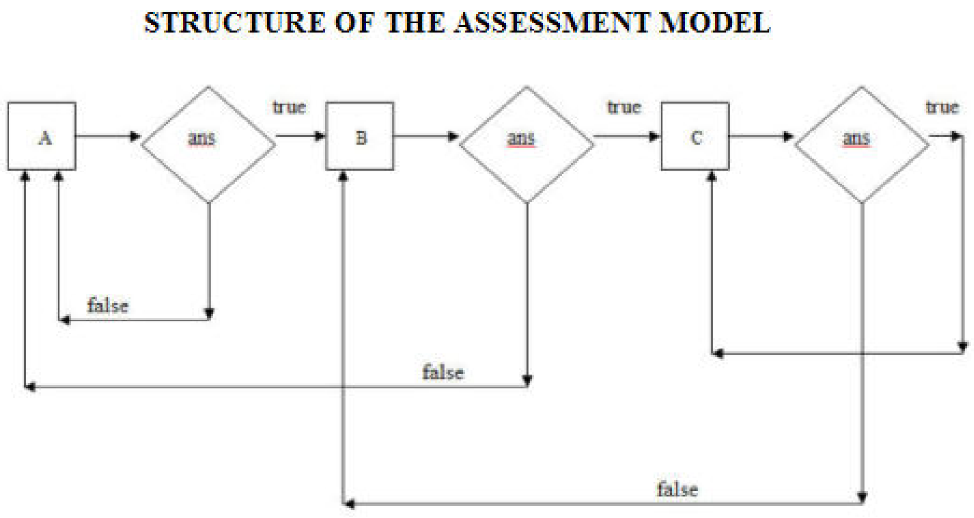
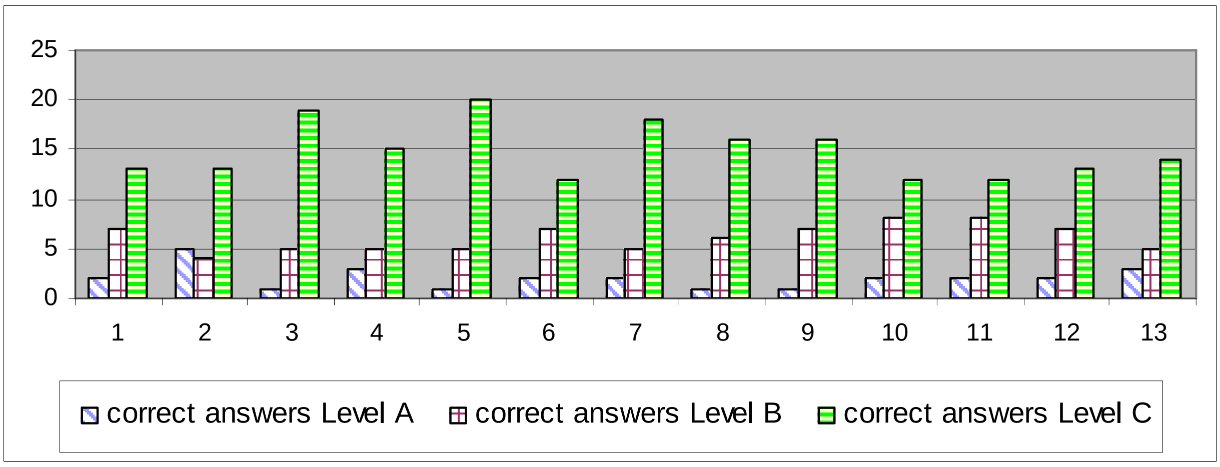


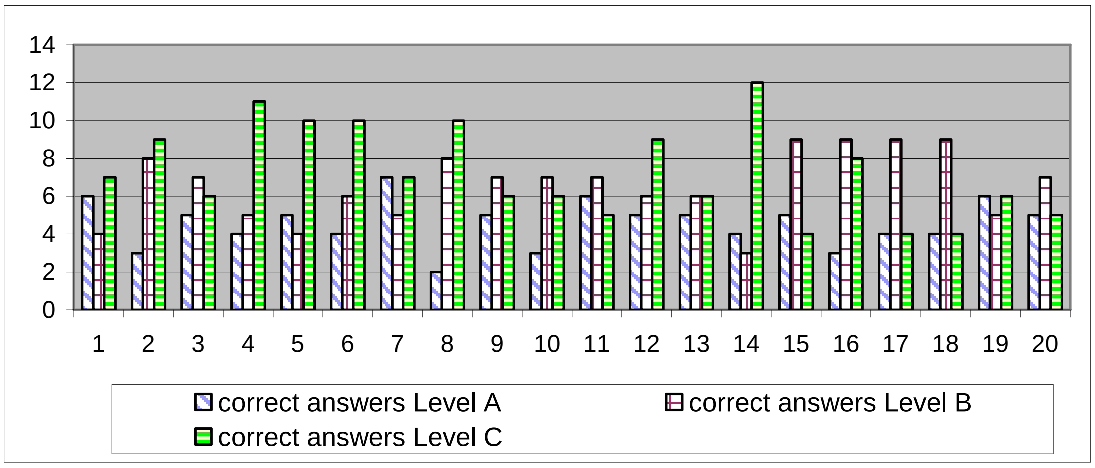
Figure 1: Adaptive Sequence of question in PAT



Graph 2: Correct answers per level by High Programming Skills' students (13 students out of 73)

	Mean	StDev
correct answers on level A questions	2,077	1,115
correct answers on level B questions	6,077	1,320
correct answers on level C questions	14,846	2,764

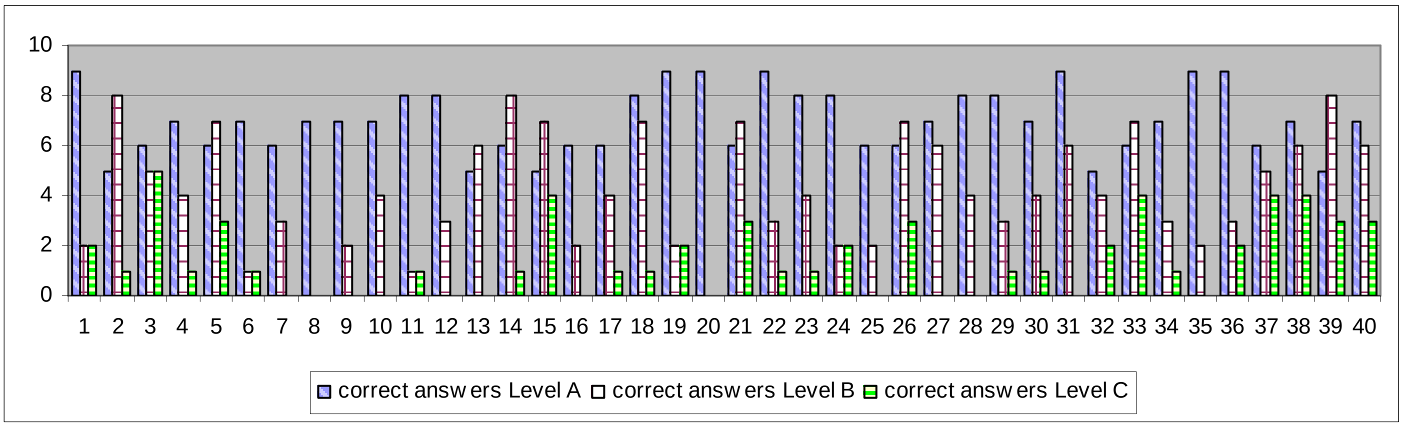
Table 3: Correct answers per level by the 13 High Programming Skills' students



Graph 3: Correct answers per level by Medium Programming Skills' students (20 students out of 73)

	Mean	StDev
correct answers on level A questions	4,55	1,234
correct answers on level B questions	6,55	1,82
correct answers on level C questions	7,25	2,468

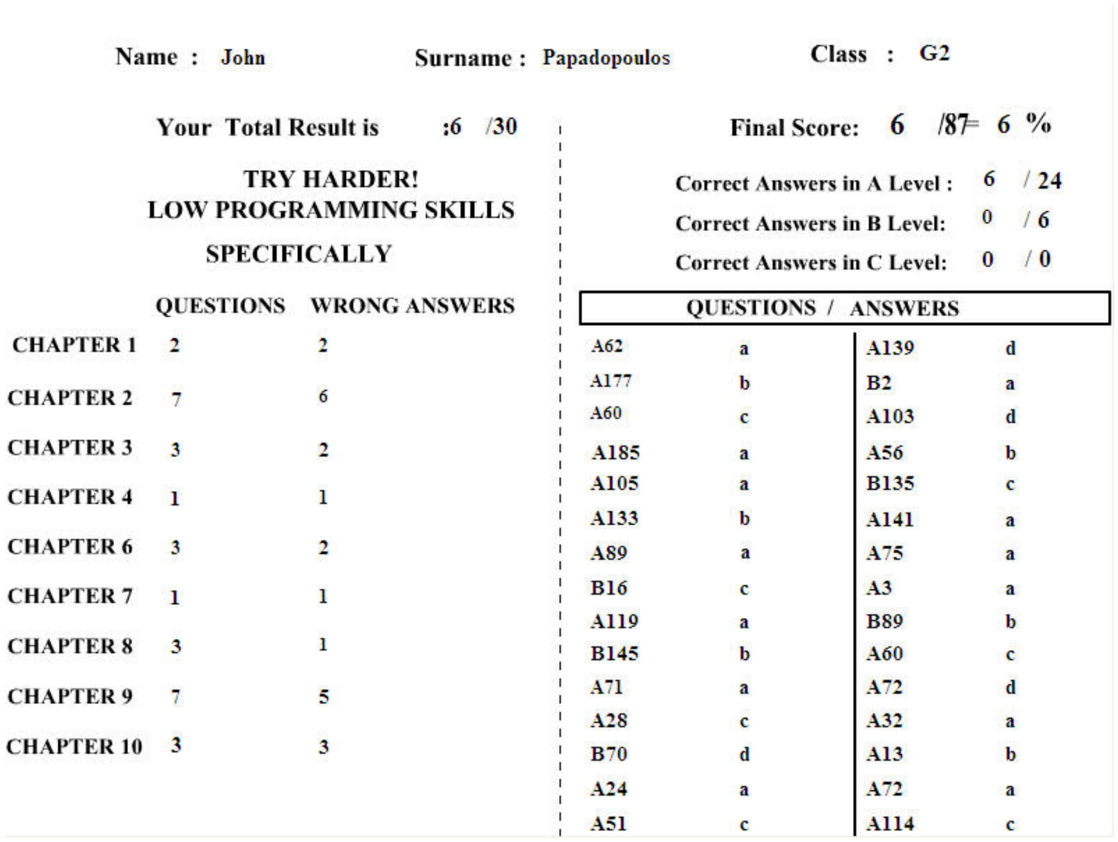
Table 4: Correct answers per level by the 20 Medium Programming Skills' students



Graph 4: Correct answers per level by Low Programming Skills' students (40 students out of 73)

	Mean	StDev
correct answers on level A questions	7	1,301
correct answers on level B questions	4,2	2,301
correct answers on level C questions	1,45	1,449

Table 5: Correct answers per level by the 40 Low Programming Skills' students



Name : John **Surname :** Papadopoulos **Class :** G2

Your Total Result is :6 /30

TRY HARDER!
LOW PROGRAMMING SKILLS

SPECIFICALLY

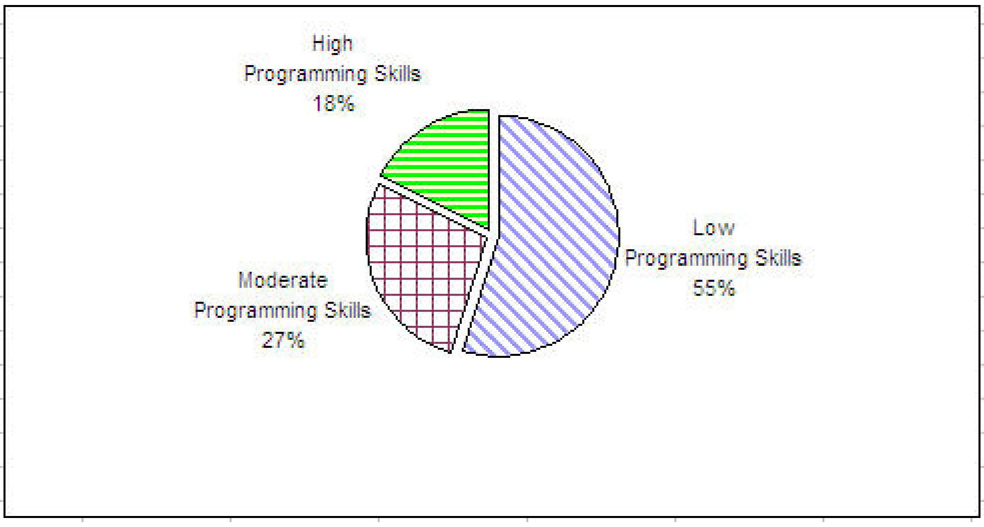
	QUESTIONS	WRONG ANSWERS
CHAPTER 1	2	2
CHAPTER 2	7	6
CHAPTER 3	3	2
CHAPTER 4	1	1
CHAPTER 6	3	2
CHAPTER 7	1	1
CHAPTER 8	3	1
CHAPTER 9	7	5
CHAPTER 10	3	3

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
B70	d	A13	b
A24	a	A72	a
A51	c	A114	c

Figure 2: Result template: TRY HARDER (Low Programming Skills)



Graph 5: Classification of students using PAT

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

Table 6: Correspondence between PAT students' classification, Total Result, Final Score and Performance in National Exams

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%**

Table 7: Correspondence between PAT Assessment and National Exams Assessment