
# Learning Technology


**publication of**

**IEEE Computer Society**
**Learning Technology Task Force (LTTF)**

http://lttf.ieee.org/learn_tech/

Volume 3 Issue 3 | ISSN 1438-0625 | July 2001

Editorial board | Subscription | Author guidelines

**Advertising in the newsletter**

## Contents

suit a broader spectrum of students and the community at large. The research group expects that the addition of voice and image components will improve the overall communication process between the students as compared to the plain text chat. The metrics for this would be the student and the staff feedback questionnaires.

**Bibliography**


- Shukla, R., Sathu, H. and Tang, Z. "Giving Voice to Discussion Boards" NACCQ 2001.
- SpeaksForItself. http://www.speaksforitself.com/speaksforitself
- Digalo. http://www.digalo.com/
- ReadPlease. http://www.znet.com/
- Bell Labs Text-to-speech. http://www.bell-labs.com/projects/tts/voices.html/
- Microsoft. http://www.microsoft.com/download



**Hira Sathu**
hsathu@unitec.ac.nz

**Ranjana Shukla**
rshukla@unitec.ac.nz

**Zhong Tang**
Ztang@unitec.ac.nz

UNITEC Institute of Technology
Private Bag 92-025
Auckland, New Zealand


*Back to contents*

## Telemachus: A system for submission and assessment of students programs

**Abstract**

The software tool called Telemachus has been developed to test and grade students' programs. Students submit their programs via WWW and then the software compiles, tests and grades them and also generates statistical results. The aim of Telemachus is not only to grade the students' programs but more importantly to provide reliable performance data that could give a reasonable gauge of student knowledge and in this way contribute to teaching programming skills.

## 1. Introduction

In an introductory programming course students write programs in order to develop programming skills. The task of marking the solutions students produce in their programming assignments is laborious and error-prone. Thus, a number of researchers have been investigating the possibility of integrating technology into Computer Science examinations. Among others, Mason and Woit [7, 8] report their experiments from on-line programming examinations; Preston and Shackelford [9] describe a prototype for an on-line assessment software tool; Jackson and Usher [4] developed ASSYST a system for grading student programming exercises; Tinoco et all [11] develop QUIZIT a system for online evaluations in WWW-based courseware; Joy and Luck developed BOSS [5] a system for submission and assessment of students programming assignments.

Most of these systems interested us, but were inappropriate to be used in our courses for two main reasons: such a system must match exactly the requirements of the specific course on which it is intended for use and furthermore, the system must ensure compatibility with the University databases so that electronic marksheets can be integrated into the broader process of assessment administration. Thus we decided to develop a system called Telemachus handling not only submission, program testing and marking students' assignments but also providing reliable performance data that could give a reasonable gauge of student knowledge

Our system consists of two components: the first and the simplest one provides the means by which a student submits a program electronically for grading; the second component, which is used by the tutor, directs the assessment process. In the subsequent sections of the paper, we present the capabilities of our system and we show how technology can help the teaching process. We will show that this electronic marker does not only help in the process of testing and grading programs but can also give data for further didactic research. For example, using students' performance data we can detect their misconceptions and further we can pinpoint any particular notion that students may not have grasped fully.

## 2. Motivation

In the Department of Applied Informatics every year about 130 students are required to attend the CS1 and CS2 courses. These compulsory courses are offered during the first and the second semester and are comprised of a two-hour lecture and a two-hour laboratory session per week. In the laboratory session students solve some programming exercises with the instruction of a tutor. They are given a number of programming exercises as homework, whose solutions they have to submit in the next laboratory session. Almost all the exercises are small or medium sized programming problems and the average number of the programming assignments is 35-40 per course.

Up until now (i.e. before Telemachus), the following rudimentary examination procedure occurred. Since in our department there was only one assistant (like in most Universities of our country, assistants are a rarity) it was impossible to check manually all the students' programs. Thus, what happened was that at the end of the semester the tutor along with the assistant had to examine orally every student on a small number of programs (about 5 in all) as it was humanly impossible to check the entire listings (over 5000).

Obviously, this situation could not satisfy either our students or us and it was the weak point of both courses. We admit that accurate and meaningful assessment is vitally important for many reasons. First, it provides meaningful feedback to students and instructors; quality assessment informs students of their mistakes and successes and informs instructors of student knowledge. Second, it establishes confidence in the measurement of student performance; without accurate assessment, neither students nor instructors have a

reasonable gauge of student knowledge. Third, it provides instructors and administrators with the ability to perform quality control; collecting reliable performance data enables examination of the instructional process for courses. Finally, accurate assessment makes new educational research opportunities possible; customized courses, better use of class time, and student performance trend analysis are a few examples of possibilities [9, 3].

Thus, we decided to develop Telemachus, a system that tests and grades students' programs and also provides reliable performance data that will help us to detect potential students' misconceptions.

## 3. Description of the system

Telemachus consists of two main components: the one that students see and by which they submit programs electronically and access the results via WWW; and the second one that a tutor views and by which he/she can test and mark programs and obtain statistical results.

Students submit programs or access their results via WWW. Telemachus, in order to permit students' access to these operations, asks for their normal login name and password and then it permits access only to those who are students of the Department and have to attend CS1 and CS2 courses.

The second component, that of the tutor view, is composed of 6 main modules (see figure 3): Exercises, Students, Options, Reports, Marker, New Semester.

### *3.1 The Module "Exercises"*

We can see, in figure 1, the form that handles the exercises' database. We have added until now 200 programming exercises into the database. We have categorized them into different topics (worksheets): basic statements, operations and types, selection structures, repetition structures, arrays, strings, records, files, pointers etc. Every year we choose a number of 35-40 exercises, among those included in the database, that are different from those of the previous year.

At the top of the form of figure 1, we see the buttons, which allow the tutor to add/ delete/ edit or find an exercise. In the section below on the left, we see the worksheet's number that the exercise refers to and the total number of exercises in this particular topic. On the right, the tutor gives the data concerning an exercise, such as: the exercise code (worksheet No, exercise No, Question No); the total number of data sets (input, output data sets); if the exercise is included in the marking process. In the middle of the form, the tutor writes the exercise. At the bottom of the form, the tutor gives some extra settings concerning the data sets.

### *3.2 The Module "Students"*

The system handles students' database with a form similar to that of figure 1. The tutor can add/ delete/ find a student and can edit some elements.

### *3.3 The Module "Options"*

Using the form of figure 2, the tutor sets the options that refer to the electronic marker. This form is divided into 3 sections: the left top section contains information about the exercises that will be marked. In the top right section the tutor can choose the compiler that will be used for compiling the program and by mouse clicking the button "Compile Now" Telemachus starts the compilation process. The results of the compilation process are recorded. The bottom section contains information about students.


*Figure 1.*

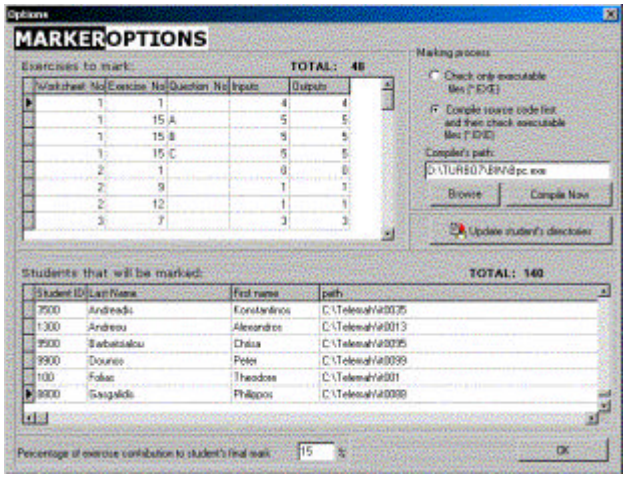


*Figure 2.*

### *3.4 The Module "New Semester"*

The form titled "New Semester" shown in figure 3, initializes students' database every new semester or for each different course. Running this module, the tutor can add all students to the database by giving the total number of students who attend the particular course and the year of their enrollment in the Department (student's IDs in the University's database are formed in this way). This module also creates new students' directories and deletes students' directories of the previous year.

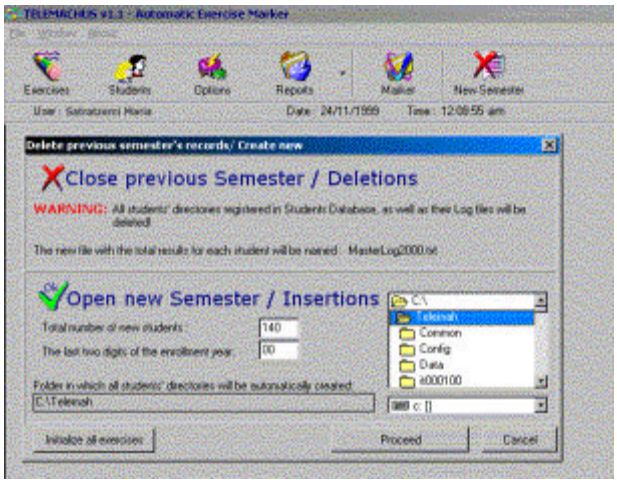


*Figure 3.*

### *3.5 The Module "Marker"*

The button "Marker", in figure 3, runs the module that checks the programs' correctness. The executable code of a student's program (which was previously generated by module "Options") is run against the sets of test data. In the next step the module determines whether the program's output is correct or not and marks the program. The checking approach is to match the student's output produced by every set of the test data with the one produced previously by the system.

### *3.6 The Module "Reports"*

Button "Reports" (figure 3) produces 3 different types of electronic reports. i) An extended report for every student, where he/she can see the following information: which programs were not successfully compiled; for every set of test data which program produced a correct or an incorrect output. ii) A report for all students with their grades. iii) A statistical report where the tutor can see for every exercise and every set of test data the rate of programs that were correct or had errors or were unsuccessfully compiled or did not gave an output due to an infinite loop.

"File" menu (figure 3, main form) handles the produced reports (open, print a report etc).

## 4. Test Data Adequacy & Students' Conceptions

As we have already mentioned in the introduction, not only does Telemachus help the tutor in grading students' performance but also in providing useful data concerning students conceptions.

As it is known, many errors in students' programs [1, 2, 10] have an element of chance and are thus unpredictable. There are some errors, however, which are more systematic and more persistent and since they are due to students' misconceptions can be predicted. Students produce programs that are correct for most of the cases but when these programs are tested for some data sets they beget incorrect results since students do not take into consideration all the cases. Telemachus validates students' programs, running them against a number of predefined data sets rather than against random data sets so as to detect logical errors. We chose adequate data sets in such a way that a program with logical errors will produce an incorrect output or it will have an incorrect performance (infinite loop). Therefore, some data sets will cause students' programs to give incorrect outputs whereas other data sets will cause correct outputs. Of course, the combination of incorrect and correct outputs does not guarantee the detection of a misconception; there are other types of errors that might be associated with the same combination of correct and incorrect outputs. Nevertheless, the combinations of the chosen data sets give valuable insight into students' conceptions. Following, we give two examples in order to show the above.

### *4.1 Example 1*

The first example is a series of programs which deal with the binary search [6]. We present 3 programs. The first is correct and the other two are incorrect. Table 1a gives the data sets and Table 1b summarizes the results.

| No | Elements | Searching element |
|---|---|---|
| 1 | 1 2 3 4 | 1 |
| 2 | 1 2 3 4 | 2 |
| 3 | 1 2 3 4 | 3 |
| 4 | 1 2 3 4 | 4 |
| 5 | 1 2 3 4 | 5 |

*Table 1a*.Data sets

| *Programs* | *Results* |
|---|---|
| start:=1; fin:=n; found:=false;<br>while ((not found) and (start <= fin)) do begin<br><br>l:= (start+fin) div 2 ;<br>if matrix[l]=element then found:=true<br>else<br>if matrix[l]<element then start:=l+1 else fin:=l-1<br><br>end;<br>if found then writeln(l) else writeln('not found'); | It gives correct output for all data sets |
| start:=1; fin:=n; found:=false;<br>while ((not found) and (start <= fin)) do begin<br><br>l:= (start+fin) div 2 ;<br>if matrix[l]=element then found:=true<br>else<br>if matrix[l]<element then start:=l else fin:=l<br><br>end;<br>if found then writeln(l) else writeln('not found'); | Infinity loop for data set No 4 and 5 |
| start:=1; fin:=n; found:=false;<br>while ((not found) and (start <= fin)) do begin<br><br>l:= ((start+fin) div 2 );<br>if matrix[l]=element then found:=true else<br>if matrix[l]<element then start:=l else fin:=l;<br>if fin-start=1 then<br><br>if matrix[start]=element then<br>begin l:=start; found:=true end<br>else begin l:=fin; found:=true end<br><br>end;<br>if found then writeln(l) else writeln('not found'); | Incorrect answer for data set No 5 |

*Table 1b*. The programs and their performance

The second program gives, according to our observations, the most common error students make, while the third program shows the most usual modification that they make to the second program when students realize that it is incorrect.

### *4.2 Example 2*

The second example shows a more trivial but equally frequent error in students' programs. The proposed problem was to write a program which erases from a string any leading and trailing blank characters. Obviously the use of the appropriate repetition structure gives correct output and vice versa. Table 2 summarizes the results given when the applied code is the following:

```
Readln(st);
repeat delete(st,1,1); until copy(st,1,1)<>#32;
repeat delete(st,length(st),1); until copy(st,length(st),1)<>#32;
```

Symbols used: S =any string without any leading and trailing blank characters, B= a string of blank characters

| Given String | Results |
|---|---|
| S | Error: erases the first and the last character of S even though are not blank characters |
| ÂS | Error: Correctly erases the leading blanks B but also the last character of S even though it is not blank |
| SÂ | Error: Incorrectly erases the first character of S even though it is not blank but correctly erases the trailing blank characters B |
| ÂSÂ | Correct |

*Table 2.*

## 5. Conclusions

Telemachus is very simple to use. Students use their normal email login names and passwords to log into the system, submit their programs electronically and access their report. Submitting programs electronically helps students save time, otherwise they would have to print their programs, which is a time consuming task. In addition in receiving a report on their submitted programs it gives them feedback on their mistakes and successes.

The help that Telemachus offers to the tutor is likewise invaluable. Besides the fact that the system saves the tutor from the laborious task of checking and marking students' programs manually, it also gives information that a human might have completely missed: it spots errors that could be difficult to pinpoint from the visual examination of listings. Furthermore, students' performance scores give the tutor the possibility to evaluate the success of a course. Finally, collecting reliable performance data over a long period of time the hypothesis concerning students' errors will empirically be confirmed.

## 6. Acknowledgments

The "Operational Program for Education and Vocational Training" of the Second Community Support Framework, EC, financially supports this work. We acknowledge the significant help given by Theodore Folias and Maria Myari during the development process.

## References

[1] Hoc J. M. Analysis of beginners' problem-solving strategies in programming, in Psychology of Computer Use, Green T.R.G., Payne S.J., van den Veer G.C. [eds], Academic Press, (1983), 143-158.

[2] Hoc J., Green T., Samurcay R., Gilmore D., Psychology of Programming, Academic Press , (1990).

[3] Hopkins K., Educational and Psychological measurement and Evaluation, Allyn & Bacon, Boston, (1998), 2-25.

[4] Jackson D., Usher M., Grading Student Programs using ASSYST, In Proceedings of SIGCE'97, ACM, 335- 339.

[5] Joy M., Luck M., Effective Electronic Marking for On-line Assessment, In Proceedings of ITiCSE'98, ACM, 134-138.

[6] Lesuisse R. Some Lessons Drawn from the History of the Binary Search Algorithm, The Computer Journal, Vol. 26, n° 2, (1983), 154-163.

[7] Mason D., Woit D., Integrating Technology into Computer Science Examinations, In Proceedings of SIGCE'98, ACM, 140-144.

[8] Mason D., Woit D., Providing Mark-up and Feedback to Students with Online Marking, In Proceedings of SIGCE'99, ACM, 3-6.

[9] Preston J., Shackelford R., Improving On-line Assessment: an Investigation of Existing Marking Methodologies, In Proceedings of ITiCSE'99 (Crakow Poland), ACM , New York, July 1999, 29-32.

[10] Soloway E., Spohrer J., Studying the Novice Programmer, Lawrence Erlbaum Associates, 1989.

[11] Tinoco L., Fox E., Barnette D, Online Evaluation in WWW-based Courseware, In Proceedings of SIGCE'97, ACM, 194-198

**Maya Satratzemi**
Dept. of Applied Informatics
University of Macedonia
156 Egnatia Str., P.O.Box 1591
54006 Thessaloniki
Greece
maya@uom.gr

**Vassilios Dagdilelis**
Department of Educational and Social Policy
University of Macedonia
dagdil@uom.gr

**Georgios Evangelidis**
Dept. of Applied Informatics
University of Macedonia
gevan@uom.gr

*Back to contents*

# A Pattern Language for Architectures of Intelligent Tutors

**Abstract.** This paper introduces PLAIT, a specific pattern language for architectures of intelligent tutors. A pattern language is a structured collection of interrelated patterns in a specific domain. PLAIT is based on the idea of using patterns in the architectures of intelligent tutors, as well as on a number of patterns that have been discovered in the existing architectures of intelligent tutoring systems.

## 1. Introduction

In software engineering, patterns are attempts to describe successful solutions to common software problems [6]. Software patterns reflect common conceptual structures of these solutions, and can be applied over and over again when analyzing, designing, and producing applications in a particular context. Each pattern has a context in which it applies. When several related patterns are woven together, they form a *pattern language*. Pattern languages cover particular domains and disciplines, such as concurrency, distribution, organizational design, business and electronic commerce, human interface design and many more.

There are also patterns in intelligent tutoring systems (ITSs). Such patterns are, however, mostly implicitly present in ITSs. Patterns exist in architectures of ITSs, in the way learners learn from such systems, and in the way ITSs convey domain knowledge to the learners. This paper describes explicitly some patterns that exist in ITS architectures. The patterns described are all interrelated, and together represent the core of *PLAIT*, a Pattern Language for Architectures of Intelligent Tutors.