System for Testing Different Kinds of Students' Programming Assignments

Ivan Pribela, Mirjana Ivanović and Zoran Budimac

Department of Mathematics and Informatics,
Faculty of Science, University of Novi Sad

Trg Dositeja Obradovića 4,
21000 Novi Sad, Serbia

Phone: +381 21 458 888

Fax: +381 21 6350 458

Email: {pribela, mira, zjb}@dmi.uns.ac.rs

System for Testing Different Kinds of Students' Programming Assignments

Ivan Pribela, Mirjana Ivanović and Zoran Budimac
Department of Mathematics and Informatics,
Faculty of Science, University of Novi Sad
{pribela, mira, zjb}@dmi.uns.ac.rs

***Abstract* – The testing system implemented by using Apache Ant allows students to test their assignments in a controlled manner. The system allows instructor to run the same tests on a set of student assignments. Results of tests are recorded in a log file and are available to students and the instructor. The system accepts any type of files as assignment and the instructor has a great flexibility in specifying how and what is to be tested. The system is independent of underlying platform and programming language used for solutions.**

***Keywords* – Computer supported learning, Computer assisted testing, Programming and problem solving**

I. INTRODUCTION

Most of programming courses at university level are concentrating on practical exercises as a form of continual assessment and examinations. Practical way of continual assessment and evaluation offers friendlier conditions to the student. However, in such circumstances instructors face other problems in assessing student work.

Usually there are a great number of students (between 80 and 150) per course, one programming assignment per week and only up to two instructors. So, it is time consuming to check every aspect of every program written by every student. The only possible way to obtain satisfying quality of practical work and keep assessment efforts in reasonable level is to introduce some kind of automatic or semiautomatic assessment. By transferring part of the assessment process from instructor to computer, students enjoy more objective assessment while instructors can concentrate on more essential parts of assessment (programming style, use of recursion, good choice of data structures and so on).

The development of Testovid[1], an automated testing system, emerged from the mentioned need. The goal of the system is to leverage the effort of assessing assignments during practical exercises especially at our Department, but it could be also included in assignment process of other universities. Some useful suggestions for improvements of the system were the consequence of fruitful discussions within Workshops supported by DAAD project [5]. The system is available to students working on their assignments in computer labs. The testing logic and test data are prepared by the instructors in advance and saved in appropriate files ready for students' usage. There are two main advantages of the system: a) it is platform and language independent and b) it allows great flexibility in what and how will be tested, as well as the file type that can be submitted to testing.

Since the system is built on top of Apache Ant it is very extensible, not dependent on programming language or specific building and compilation logic. It is also modular, and can be quickly adapted to new trends.

The testing system allows students to test their assignments in a controlled manner with the instructor's test data. The instructor creates an Apache Ant [3] script to build and run the student's project. New scripts can be created using parts from previously created scripts, thus simplifying the instructors work. When a student wishes to test her assignment she runs the system which then copies the student's files into a temporary directory, and the instructor's scripts are executed to build the project and test it. The result of each test run is recorded in a log file for the student and the instructor.

The type and number of files that the system can accept is not restricted. Also, there is no limit on the amount of tests being conducted. During one run of the system multiple aspects of the assignment can be tested.

Besides on-line testing provided for students, instructor is offered a batch mode for testing of multiple submitted projects. The system keeps track of results for all students and generates a single file with all the results. Apart from that, one detailed report for each student is generated.

The rest of the paper is organized as follows. Section 2 compares Testovid system with other systems of similar purpose. Sections 3 and 4 discuss system usage and system structure and organization, respectively. Section 5 gives an overview of usage of Testovid in real learning environment. Section 7 focuses on experiences gained using the system and gives a short conclusion.

II. RELATED WORK

Automatic and semi-automatic assessment and testing of student programs written in programming languages dates all the way back to 60's of the last century. Among the first authors were Hollingsworth [10] and Wirth [7] and systems were designed for assembler and Algol. During time many systems were developed [1, 12] and they usually followed modern concepts introduced by new operating systems and new programming languages.

There were also successful contemporary efforts to address problems of electronic submission and assessment. Unfortunately, many such ventures focus on a specific programming language or a specific platform.

The systems like [4, 6, 8, 11] focus on Java programming language, and the system [9], focus on Scheme with a possibility of linking to some other programming languages. There were some attempts to create a system for testing that will encompass wider spectrum of programming languages developed in python [2].

There are also successful attempts of bringing automated assessment of programming to learning management systems (LMS). The assessment system described in [15] addresses

[1] Testovid is part of wider submission environment called Svetovid. Svetovid is named after the pagan Slavic god that sees everything. The name Testovid is then chosen to resemble Svetovid.

assessment of SQL Select queries and short programming assignments in LMS environment. The solution provides support for multiple programming languages but is limited to the LMS for which it was implemented and lacks support for large programming projects.

A truly programming language independent testing system is [14], a system built for UNIX platform, also based on command scripts, but a little outdated and without network support.

The main advantage of Testovid, in comparison to mentioned systems is that it is built on Apache Ant and thus it is modern and not dependent on programming language or specific building and compilation logic. Furthermore, the system can be used in a wide variety of situations and environments, and is very extensible, modular, and can quickly adapt to new trends.

III. Usage of Testovid in Assessment Process

Testovid system supports testing of any aspect of student solutions written in any programming language. It is also not limited to assessment of programming solutions; it can accept textual files, images, and other documents.

The main motivation for developing the system and its most common usage, however, is to check student programs for compilation errors, code style guidelines adherence, implementation correctness and performance. At our Department, students have several courses which focus on programming exercises (using Java, C#, Delphi, Modula-2, Scheme) as a main technique for continual assessment of practical work. During the practical exercises students work in computer laboratories on the given assignments and instructor assesses their practical skills. As this way of continual assessment is very time consuming and wearying for both the student and the instructor, students utilize Testovid to check their solutions before submitting, and instructors to increase reliability and speed up the assessment.

Apache Ant is a tool that can automate the application build process, with strongest support exactly for the described cases. The build script can perform compilation, documentation generation, packaging, and if during the build process there is an error the build fails. In a failed build a detailed message describing the reason of failure is produced. Proposed testing system, built on the Apache Ant, can capture details of failed build and show appropriate advice to the student in order to improve the solution.

Look at an example where the instructor wants to automatically check if a student solution compiles successfully. The instructor should write one configuration file telling how many points to award for successful compilation (Fig. 1), how to name the aspect being tested and to give description that will be displayed to the student. In the example (Fig. 1) the tested aspect is named “testCompile”, it is worth one point and the description displayed to the student is “Compilation”.

```
targets.all=testCompile
testCompile.name=Compilation
testCompile.score=1
```

Fig. 1 Configuration file for compilation testing

The list of actions to be performed during testing also has to be set up. This is done by writing the body of an Apache Ant target named like the tested aspect (in the example “testCompile”). One example of the target body is shown in Fig. 2. It tries to compile all files in the current directory (line 4) and if all went well set the advice given to the student to an empty string (line 5). If an error occurs during compilation, advice is set accordingly (lines 8 and 9).

```
 1 <target name="testCompile">
 2   <trycatch>
 3     <try>
 4       <javac srcdir="." destdir="." fork="true"
   source="1.5" target="1.5"/>
 5       <property name="testCompile.advice" value=""/>
 6     </try>
 7     <catch>
 8       <property name="testCompile.advice"
 9        value="Compilation failed, check language
   documentation."/>
10     </catch>
11     <finally/>
12   </trycatch>
13 </target>
```

Fig. 2 ANT target for the compilation test

When a student invokes the system to test student’s solution, she will receive a message about success (Fig. 3) or failure (Fig. 4) of testing that particular program. This message contains information about the course and assignment, list of all tested aspects with their friendly names displayed, and states the amount of points awarded. If the test was unsuccessful an advice given by the Ant target will be displayed to the student (Fig. 4).

```
Course: Object oriented programming
Assignment: Inheritance assignment no 2
Student: John Smith

Test: Compilation
Success: 1 point.

-+-+-+-+-+-+-+-+-
Total: 1 point.
```

Fig. 3 Successful test attempt

```
Course: Object oriented programming
Assignment: Inheritance assignment no 2
Student: John Smith

Test: Compilation
Failure: Compilation failed, check language documentation.

-+-+-+-+-+-+-+-+-
Total: 0 points.
```

Fig. 4 Unsuccessful test attempt

Compilation of Java programs comes out of the box with Apache Ant; however, any other language and compiler can be easily supported using Ants ability to run any operating system command (any native program).

The real power of the system comes from wide spread and adoption of Apache Ant as build process automation tool. Many commercial and open source tools that are targeted to help in software build process are developed with Apache Ant integration in mind. Tools for style checking, code coverage,

software metrics, documentation generation, image processing and other purposes are available and many more are being developed [3].

With support already provided by Ant and the tools targeted to supplement missing features Testovid system can test code style, code coverage, use unit testing paradigm, and not be limited only to Java programming language. More and more tools are available for C#, Python, and other languages.

For example, if the instructor wanted to check code style of student solutions, he would have to change only few lines in the above example to use his favorite code style checking library. The name and point allotment can be setup similarly to the already given example, while the Ant target implementing the test is different but only slightly. Figure 5 shows a target that tests proper usage of spaces and tabs for indentation in Java programs. In comparison to the previous example, line 4 is different: it uses the "style check" library to check style adherence instead of calling the Java compiler; besides that, only the advice is different.

```
 1 <target name="testStyle">
 2   <trycatch>
 3     <try>
 4       <checkstyle config="tabs_check.xml"
   file="Assignment3.java"/>
 5       <property name="testStyle.advice" value=""/>
 6     </try>
 7     <catch>
 8       <property name="testStyle.advice"
 9    value="Improper indentation, use spaces not tabs."/>
10     </catch>
11     <finally/>
12   </trycatch>
13 </target>
```

Fig. 5 Ant target for code style adherence test

Implementation correctness of student solutions can also be checked with minimal effort using unit testing or testing with prepared input and output data. Apache Ant natively supports JUnit and has the ability to run user applications with input and output redirected to external files. With many available performance and code coverage libraries designed for Apache Ant, testing of these aspects of student solutions can also be easily accommodated.

IV. Structure and Organization of Testovid

The system is designed and organized in such a way that it can be executed in two modes: interactive and batch. Interactive mode allows students to test their programs and to be informed how close is their solution to a correct one. In this way the system can be invoked only from computer laboratory (Fig. 6). Batch mode is intended to be used by the instructor. Instructor uses this mode usually after the submission deadline.

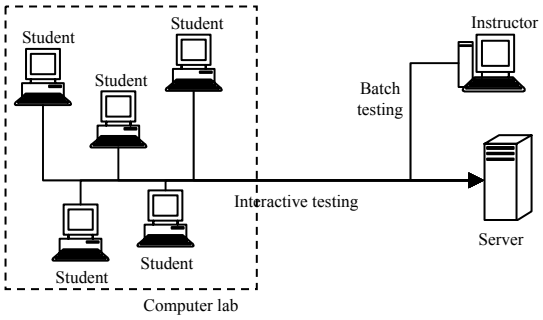
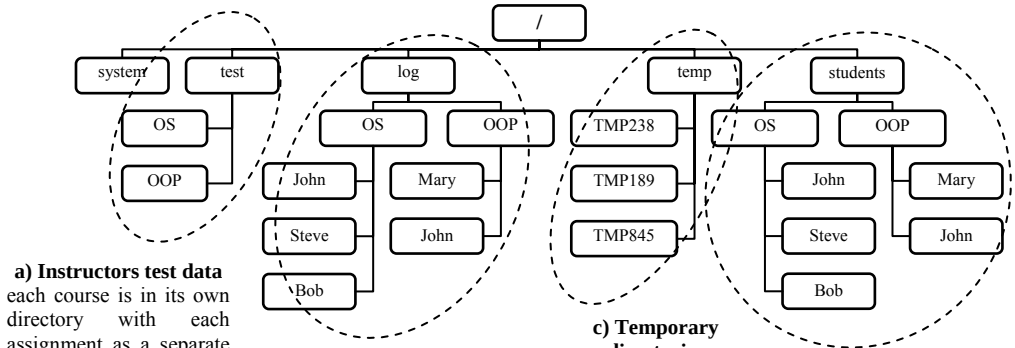


Fig. 6 Network layout of the system

Testing system may be used to test any type of program in any language, shell scripts, make files, etc. The word "project" will be used to refer to the student solution of specific task or problem, in the framework of the course.

When a student concludes that her program (project) is completed, she invokes the system through a workstation in computer laboratory using command line. The system creates a new temporary (scratch) directory and copies all files from the student's directory (Fig. 7c). In the scratch directory, system runs the Ant script provided by the instructor to build and test student's project. This may do compilations, builds, formatting, testing the project in any way or whatever is appropriate. After that the new log file is created in appropriate directory (Fig. 7b) and the result of the run is recorded in it. When the test has been executed and completed, a copy of the created log file is also given to the student, since he cannot access the scratch directory.



a) Instructors test data each course is in its own directory with each assignment as a separate subdirectory (assignment directories not shown)

b) Log files containing test results each course has in its own directory with each student as a separate subdirectory, log files are named after assignments

c) Temporary directories one for each concurrent test run

d) Student directories for storage of latest attempts each course has in its own directory with each student as a separate subdirectory

Fig. 7 Directory structure on the server

A side effect of running the system interactively is that all files, from directory where the test is run, are also copied to a directory designated for that particular student and that particular assignment. The files are kept for later manual inspection by the instructor, and only the files from the latest attempt are kept (Fig. 7d).

On the other hand, if the system is not intended to be used interactively by students there is a possibility for instructor to test multiple projects at once.

The batch mode can be used if students are working on their assignments at home or if interactive test ability is not desired or not allowed. In this case all projects must be arranged in proper directories before start of testing. Usually the projects will be gathered by a separate submission system, and manually saved under Testovid directory tree. Each project must be saved in a directory named after the student to which it belongs (or after a student id) and as a subdirectory of corresponding assignment and course directories.

When invoked in batch mode the system will create a new scratch directory for each student and copy all files from the corresponding prearranged directory. From within the scratch directory, system runs the Ant script provided by the instructor to build and test the student projects as it would be in interactive mode. Projects are tested one at a time and results are saved in log files (one for each student) within log subdirectory. Also, one CSV (coma separated values) file containing results for all students is maintained in directory corresponding to the assignment being assessed.

V. Testovid in Real Learning Environment

In this section a quick overview of usage of Testovid system in a concrete learning environment is given.

Consider, for example, a case where students are given an assignment to implement a double ended queue using a data structure of their choice. The assignment is aimed to assess student skills of pointer manipulation in "Data structures and algorithms" course. The solution should be written in Modula-2 in a form of implementation module that implements given definition module.

The best choice of data structure would be a double linked list, and a part of a typical solution is given in Fig 8. Procedures for addition and removal of elements from front of the list are analogue to the addition and removal from the rear and are omitted from the example.

```
PROCEDURE AddToRear(VAR R: Type;
      VAR Element: INTEGER);
  VAR Tmp: Type;
  BEGIN
    IF R = NIL THEN RETURN; END;
    Tmp:= TmpElement(Element);
    Tmp^.Prev:= R^.Prev;
    R^.Prev^.Next:= Tmp;
    R^.Prev:= Tmp;
    IF R^.Next = NIL THEN
      R^.Next:= Tmp;
    END;
    INC(R^.Size);
  END AddToRear;

PROCEDURE RemoveFromRear(VAR R: Type;
      VAR Element: INTEGER);
  VAR Tmp: Type;
  BEGIN
    IF (R = NIL) OR (R^.Next = NIL) THEN
      RETURN;
    END;
    Tmp:= R^.Prev;
    R^.Prev:= Tmp^.Prev;
    IF R^.Prev = NIL THEN
      R^.Next:= NIL;
    ELSE
      R^.Prev^.Next:= NIL;
    END;
    DEC(R^.Size);
    Element:= Tmp^.Info;
    DISPOSE(Tmp);
  END RemoveFromRear;

PROCEDURE Count(VAR R: Type): CARDINAL;
  BEGIN
    IF R = NIL THEN RETURN 0; END;
    RETURN R^.Size;
  END Count;
```

Fig. 8 One typical solution for double ended queue

In a typical case the teacher would pay attention to the choice of data structure, check if all procedures conform to the given specifications, and that the memory and pointers are well handled. Having this in mind, an automated test would try to add few elements to the structure, then retrieve them back, and check if elements are retrieved as expected. Memory and pointer handling could be checked by comparing available memory before and after the operations, and by the fact that null pointer dereference will cause run time error.

An example of test configuration for Testovid system would be a sequence of five test aspects (Fig. 9). First aspect would test memory allocation and if the number of elements in the structure is correct. Second and third checks would test the usage of the structure as a plain queue, while fourth and fifth would test the usage of the structure as a stack. All tests are checking memory allocation, and will fail on a null pointer dereference or on retrieval of wrong elements. This five checks cover conformance to the given specifications, and memory and pointer handling for our example assignment. Only thing left for human inspection is the choice of data structures.

```
targets.all=testElementCount,testQueue1,testQueue2,t
    estStack1,testStack2
testElementCount.name=Element count
testElementCount.score=1
testQueue1.name= Add to front, remove from rear
testQueue1.score=1
testQueue2.name= Add to rear, remove from front
testQueue2.score=1
testStack1.name= Add to front, remove from front
testStack1.score=1
testStack2.name= Add to rear, remove from rear
testStack2.score=1
```

Fig. 9 Specifications of test aspects

The implementation of all five test targets is very similar (Fig. 10). Each test target depends on a compilation target (Fig. 10, line 7), and after compilation tries to run corresponding program module (Fig. 10, line 11). The program module is responsible for performing the test logic; calling student procedures (Fig. 11, lines 8 and 11) and monitoring memory usage (Fig. 11, lines 16, 20, 21 and 22), since Apache Ant can have no insight in memory usage of native processes. At the end, the advice is produced and written to a file named

"advice.adv" (Fig. 11, lines 12, 20, 21 and 22) using a helper procedure.

```
 1 <target name="testCompile">
 2   <exec dir="." executable="tsc" timeout="50000"
   failonerror="no">
 3     <arg line="*.mod /m /zq"/>
 4   </exec>
 5 </target>
 6
 7 <target name="testQueue1" depends="testCompile">
 8   <trycatch>
 9     <try>
10       <delete file="advice.adv"/>
11       <exec dir="." executable="test02.exe"
   timeout="50000" failonerror="yes"/>
12     </try>
13     <catch/>
14     <finally>
15       <property file="advice.adv"
   prefix="testQueue1."/>
16     </finally>
17   </trycatch>
18   <property name="testQueue1.advice"
   value="${crash.advice}"/>
19 </target>
```

Fig. 10 Implementation of test aspects

The helper procedure "Check.Condition" checks its first parameter and if it is false writes the advice (its second parameter) and terminates the program. After termination, Ant target loads the advice from the file (Fig. 10, line 15). If the program crashed (due to a bad pointer dereference), the file would not have been written, and advice will not be loaded from the file, but set appropriately (Fig. 10, line 18).

```
 1 MODULE Test02;
 2   IMPORT ...
 3   PROCEDURE Check(VAR R: Dequeue.Type);
 4   VAR i, pom: CARDINAL;
 5   BEGIN
 6     pom:= 0;
 7     FOR i:= 0 TO 500 DO
 8       Dequeue.AddToFront(R, i);
 9     END;
10     FOR i:= 0 TO 500 DO
11       Dequeue.RemoveFromRear(R, pom);
12       Check.Condition(pom = i, "Wrong element
   removed from dequeue.");
13     END;
14   END Check;
15 BEGIN
16   HeapSize:= HeapTotalAvail(NearHeap);
17   R:= Dequeue.New();
18   Check(R);
19   Dequeue.Destroy(R);
20   Check.Condition(
21     HeapSize = HeapTotalAvail(NearHeap),
22     "Memory leak, check deallocation usage.");
23 END Test02.
```

Fig. 11 Module performing the tests

Testing of a student solution with a missing DEALLOCATE call for removing elements from the rear and a broken addition to the rear will produce the output shown in Fig. 12. Third and Fifth test aspects are failing because of the broken procedure, and second is failing since memory has not been released.

```
Course: Data structures and algorithms
Assignment: Double ended queue
Student: John Smith

Test: Element count
Success: 1 point.

Test: Queue: Add to front, remove from rear
Failure: Memory leak, check ALLOCATE DEALLOCATE
   usage.

Test: Queue: Add to rear, remove from front
Failure: Wrong element removed from dequeue.

Test: Stack: Add to front, remove from front
Success: 1 point.

Test: Stack: Add to rear, remove from rear
Failure: Wrong element removed from dequeue.

-+-+-+-+-+-+-+-+-

Total: 2 points.
```

Fig. 12 Partially successful test attempt

With this test configuration acting as a filter for screening aspects that are tedious and cumbersome to inspect manually, the only aspect that is left to the teacher is the more philosophical question of the choice of the data structure.

VI. Conclusion

The presented system has been designed to be part of an existing submission environment [13] with a goal to provide benefits of great degree of automated testing, for both the student and the instructor. It provides a solid base for automating assessment of student programs.

At our Department, we constantly needed such kind of tool because, there are great number of courses at Computer science directions, in which continual assessment of practical work is organized during the whole semester. Apart from that, another motivation was to obtain objective, efficient and timely way of assessment of programming solutions in circumstances when there are many students per an instructor.

The system has been used as help in assessing students' solutions for several courses on different years of study at Computers Science directions: "Introduction to programming", "Object-oriented programming", "Data structures and algorithms" and "Operating systems". For these courses students had written their solutions in Modula-2 and Java programming languages. They endorsed providing some test data to help them submit assignment only when they are satisfied with their results.

Another aspect that was discovered during the use of this system is the tendency of students to create solutions that will pass the test, and not to develop universal solutions for the given problem. Because of this phenomenon the system is used only as assessment aid and not as a fully automated testing system. The instructor is the one that is making the final decision and assessing other aspects of student solutions (style, design issues), the system acts as a "filter" that checks cumbersome and time consuming properties.

To use the system, the instructor must invest time to design test data and write ant scripts. If student programs are intended to be tested, the data must be created anyway and, frequently, command scripts can be adapted from other assignments and courses with little modification. Consequently, the only opened issue is the creation of a pool of commonly used scripts. In time, when the pool grows large enough, it will be the base for easy creation of test scripts.

The system has been used as a help in assessing student solutions for only few years (from school year 2006/07 until now) and in spite of such short period of usage we have obtained satisfactory results. Students' satisfaction was, more or less, the same as in completely manual manner of assessment, which is by our opinion excellent outcome of the system. On the

other hand, instructors significantly shorten time necessary for assessment of great number of students' solutions, and as a consequence they managed their work and duties in more efficient and more systematized way. Of course, we need to have exploitation of the system for several more years, which will give us better and more precise feedback results of efficiency and usefulness.

REFERENCES

[1] (2003) E-Learning Framework website. [Online]. Available: http://www.elframework.org/projects/asap, 2003.

[2] M. Amelung, M. Piotrowski, D. Roesner, "EduComponents: Experiences in E-Assessment in Computer Science Education", Proceedings of the Eleventh Annual Conference on Innovation and Technology in Computer Science Education ITiCSE'06, June 26–28, 2006, Bologna, Italy, pp. 88-92.

[3] (2010) The Apache Ant project. [Online]. Available: http://ant.apache.org/

[4] J. Bull, "Supporting Computer-based Assessment", Teaching and Learning Directorate, University of Luton, 1999, pp. 14.

[5] (2000-2010) DAAD project, "Software Engineering: Computer Science Education and Research Cooperation". [Online]. Available: http://www2.informatik.hu-berlin.de/swt/intkoop/daad/

[6] J. Dempster, "Web-based assessment software: Fit for purpose or squeeze to fit?", Interactions Online Journal, vol. 2, no. 3, 1998.

[7] G. Forsythe, N. Wirth, "Automatic grading programs", *Comm. of ACM*, vol. 8, no. 5, May 1965, pp. 275-278.

[8] T. Hawkes, "An Experiment in Computer-Assisted Assessment", Interactions Online Journal, vol. 2, no. 3, 1998.

[9] J. Hext, J. Winings, "An automatic grading scheme for simple programming exercises", *Comm. of ACM*, vol. 12, no. 5, May 1969, pp. 272-275.

[10] J. Hollingsworth, "Automatic graders for programming classes", *Comm. of ACM*, vol. 3, no. 10, October 1960, pp. 528-529.

[11] M. S. Joy and M. Luck, "The BOSS System for On-line Submission and Assessment of Computing Assignments", *Computer Based Assessment (Volume 2): Case studies in Science & Computing*, ed. Dan Charman and Andrew Elmes, SEED Publications, University of Plymouth, 1998, pp. 39-44.

[12] U. von Matt, "Kassandra: The automatic grading system", ACM SIGCUE Outlook, vol. 22, no. 1, January 1994, pp. 26-40

[13] I. Pribela, N. Ibrajter, M. Ivanović, "Svetovid – Special Submission Environment for Students Assessment", Proceedings of the second Balkan Conference in Informathics, Ohrid, FYR Macedonia, November 17-19, 2005.

[14] K. Reek, "The TRY system -or- how to avoid testing student programs", Proceedings of the twentieth SIGCSE technical symposium on Computer science education, 1989, ACM Press New York, NY, USA, pp. 112-116.

[15] I. Botički, I. Budišćak, N. Hoić-Božić, "Module for online assessment in AHyCo learning management system", Novi Sad Journal of Mathematics, vol. 38, no. 2, 2008, pp. 123-139.