Journal of Universal Computer Science, vol. 20, no. 9 (2014), 1304-1326
submitted: 8/12/13, accepted: 29/8/14, appeared: 1/9/14 © J.UCS

FLOP: A User-Friendly System for Automated Program Assessment

Luis Llana, Enrique Martin-Martin, Cristóbal Pareja-Flores
(Universidad Complutense de Madrid, Madrid, Spain
llana@sip.ucm.es, emartinm@fdi.ucm.es, cpareja@sip.ucm.es)

J. Ángel Velázquez-Iturbide
(Universidad Rey Juan Carlos, Madrid, Spain
angel.velazquez@urjc.es)

Abstract: Currently, automated systems for program submission and assessment play a central role in the teaching of programming. A number of such systems have been developed in the last two decades. However, their adoption in regular programming teaching presents an obstacle: the overhead work required for the design of each problem, for compilation of problem collections, and for mundane management tasks. An open challenge is to make these systems easier to use and to reduce to a minimum the number of management tasks. In this article we present the FLOP system, which was developed to satisfy this goal. The contribution of the article is twofold. On the one hand, we present the FLOP system itself and its user-friendly features. On the other hand, we present in detail the user-centered design process used to design and enhance the ease of use of FLOP. Several actions were undertaken to inquire users' concerns and needs, with a usability evaluation of FLOP conducted with students being the most fruitful action.

Keywords: Learning of programming, automated assessment systems, FLOP, user-centered design
Categories: D.2.5, K.3.1, K.3.2, L.0.0, L.3.6.

1 Introduction

Automatic submission and grading of programs is currently considered a good practice for programming education. Quoting Carter [03, section 3.1], "Using Computer Aided Assessment (CAA) can provide an increased amount, quality and variety of feedback for students as well as providing new assessment tools for educators. The benefits include speed, consistency, and availability 24 hours per day. Such tools can support a variety of different learning habits and needs amongst the students. Well-written CAA tools can allow teachers more time for advanced support, assessment and education design tasks rather than basic support. Digitally formatted assessments can also offer possibilities for reuse and resource sharing between teachers."

On the other hand, educational concern to emphasize testing as a means to assure code correctness is growing. Students often test each new piece of software by trial and error. This practice is obviously insufficient, consequently we find in recent years different proposals to integrate testing and automated assessment systems in introductory programming courses [Edwards 04]. Automatic grading reinforces the

Test-Driven Development (TDD) approach to programming learning because problem statements include a set of test cases to satisfy.

However, some authors express different concerns about automatic grading of programming assignments. For instance, Ruehr and Orr [02] considered interactive demonstration as the best assessment method for programming assignments and based their claims on didactic arguments. They considered personal contact a key aspect of the learning process, thus automatic assessment does not satisfy this requirement.

Currently, the adoption of testing in programming courses is less common than it could be expected [Desai 09]. An important reason is the heavy workload that these tools pose on instructors. Quoting Carter [03, section 3.1] again: "The setup phase for a new system or new assessments requires more time and resources than the development of traditional assessments." The difficulties to integrate testing into programming courses have been pointed out by some authors [Georgouli 11], particularly the lack of appropriate courseware materials that can be directly used by instructors [Elbaum 07]. Certainly, the high quantity of work needed for each assignment requires a lot of work, as does organizing even a small set of problems. Using these tools will also require management work from the instructor, such as tracking students' activity, controlling plagiarism, collating the grading results, and integrating them with other grades for each student, etc. Any instructor can manage this workload occasionally, but unlikely on a daily basis work.

A satisfactory solution to this problem should automate the handling of these tasks in daily teaching. In this article we present FLOP, an automated assessment system designed to help with the tasks mentioned above. The contribution of the article is twofold. On the one hand, we present the FLOP system itself and its user-friendly features. FLOP is freely available for students and for instructors. It is easy to use, it is complemented by a large collection of problems and therefore it is ready to be used with low handling effort. On the other hand, we present in detail the user-centered design process used to enhance FLOP. Several actions were undertaken to inquire users' concerns and needs, with a usability evaluation of FLOP conducted with students being the most fruitful action.

The structure of the article is as follows. In next section, we review the main underlying concepts and related work. In Section 3, the FLOP system is described, with an emphasis on those features most relevant to the instructor and to the student. Section 4 describes the user and instructor centered design. An outstanding element of FLOP's user-centered design was to satisfy students' needs, thus Section 5 describes the aforementioned usability evaluation conducted with students. Then, Section 6 contains a discussion on the most relevant features that were integrated into FLOP, as well as a comparison with other systems. Finally, Section 7 summarizes our conclusions and outlines some lines of future work.

2 Background

We present in this section the most relevant background to properly understand the article contributions, namely TDD and automated assessment systems, programming problems, and collections of such problems.

2.1 TDD and Automated Assessment Systems

The TDD methodology of program design [Beck 02, Shepard 01] has become very popular, both for programming education and for professional software development. In TDD, the specification of any part of a program includes a set of test cases that completely specify the intended behavior of the new piece of code. From the point of view of a programming student, the new piece of code must pass these test cases. TDD is also advocated in the professional world by the eXtreme Programming (XP) methodology [XP 09]. Thus, the key elements of the methodology are the same for both communities, with the exception of one minor difference. Software developers must design test cases as part of their work, while students either can be given the test cases as part of the problem statement or may be required to create their own.

There are two major testing techniques [Myers 04]: black-box testing and white-box testing. In the former testing technique, a program is compiled and executed against a set of input cases, whose output is known. The program is labeled depending on the stage achieved: It compiles successfully, it runs without execution errors, it runs and each input yields its corresponding output, and it runs successfully with the space and time consumed within the established bounds. In the white-box model, not only is the program behavior considered but also the source program. The success depends on the fulfillment of certain criteria: code coverage, style, and organization. Whilst the information provided by black box testing is more limited, white box testing is very dependent on the programming language and the amount of work required to develop a programming exercise is much more time-consuming. Our approach is based on the black-box model, so we refer to this in the rest of the article.

Automatic submission and assessment of programs is based on these testing techniques and is used in two different scenarios: programming contests, and regular teaching. Systems used to support both kinds of usage are similar, although they are most often called judges and automated assessment systems, respectively. More information about these systems can be found in a number of surveys [Ala-Mutka 05, Douce 05, Ihantola 10].

We note that assessment may play two different and important roles, which should be supported by automated assessment systems: formative and summative. Formative assessment gives feedback to the student by means of a message –often called a verdict– in a standard format on the program submitted. Verdicts may also help the instructors to improve their teaching. Summative assessment allows the teacher to assign each student activity a grade that will contribute somehow to their final grade in the course.

2.2 Programming Problems and Their Assessment

Programming problems typically have the same structure in automated assessment systems. First, a problem statement contains an explanation of the task that must be solved, if necessary including figures to illustrate key issues in the problem. Then, there are two sections explaining carefully and in detail the format of the input that students' programs must read and the format of the output that the programs must generate. Format instructions are very strict, including the usage of lower/upper letters, blanks and other constraints. Finally, there are two sections showing some input/output examples that students can use to test their programs and also to get

familiar with the format. See Fig. 1 for the problem of computing the *n*-th Catalan number, as presented by FLOP.

It is very important to meticulously explain for each problem the format of input and output, because students' programs will be checked against test cases that strictly follow those formats. Any difference or misunderstanding in input format by students, such as separating elements by a different character or showing the results in a different order, will most probably lead to a runtime error or an erroneous output. Furthermore, the system will consider the program to be wrong, although the algorithm used could be correct. Finally, the set of input-output pairs provided must be complete, covering all the possible cases, including special situations. In summary, the design of programming assignments for automated assessment systems is a time-consuming task, which hampers their usage in daily programming teaching.

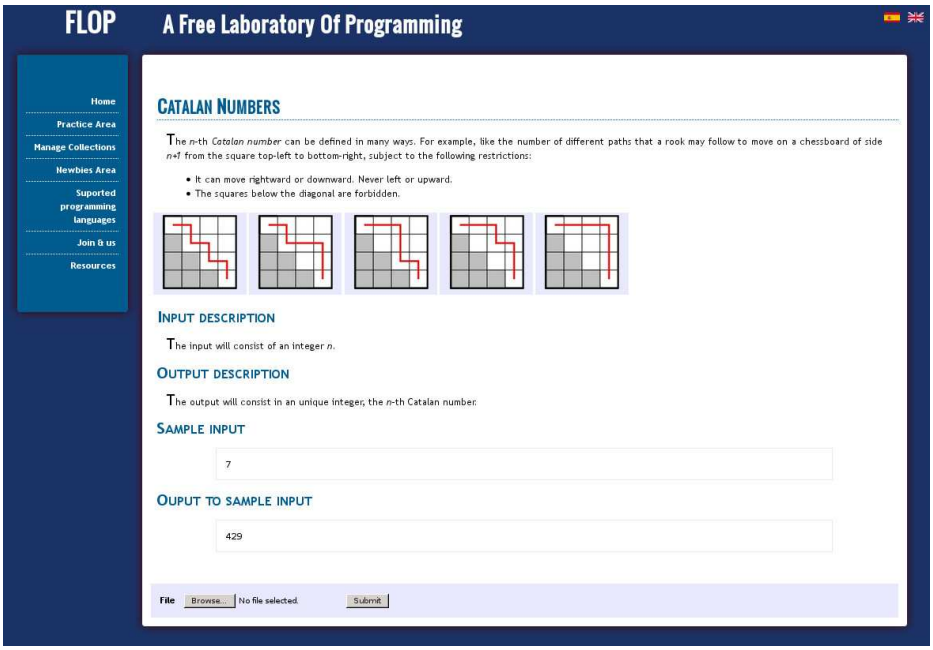

Figure 1: A problem statement available in FLOP

The set of possible grades that an automated assessment system can give to a submitted program is limited and has become a *de facto* standard:

- "Accepted". The program passed all the test cases, each output was *exactly* the expected one.
- "Presentation error". The program passed all the test cases and the output is the expected one but with some differences. For example, the expected output must separate pairs with one space but the program produces pairs separated by tabs, or the expected output should contain a blank line between pairs but the program generates them one after another. In these cases, the problem is considered correct, but the system informs about these minor differences.

- "Wrong answer". The program passed the test cases without triggering any runtime error but the output differs in more than blanks or similar characters. Consequently, the system considers that the student algorithm computed a wrong answer.
- "Runtime error". The student's program has risen an error (e.g. division by zero, array index out of bounds, or null pointer access) for some test cases.
- Other less frequent and typically less important messages are "time limit exceeded", "memory limit exceeded" and "output limit exceeded".

2.3 Collections of Problems

Due to the interest in programming contests and to the public availability of collections of problems, some web sites have become widely popular, e.g. USACO [Kolstad 09]. We especially cite the UVA Online Judge [Revilla 08] that gathers the large collection of programs proposed in ACM programming contests. Each problem in the collection was carefully designed and was difficult enough to serve in local, regional or final world programming contests. These features grant the problems great interest, so some of these web sites receive thousands of visits and submissions each month.

Contest problems are especially interesting to brilliant programmers, but normal students may also benefit from them if they are adequately adapted. In educational settings it makes more sense to offer problems in a range of difficulty levels, and to provide both summative and formative assessment. There are several initiatives with large collections of assignments intended for learning and teaching, e.g. Jutge.org [Giménez 12] or CodingBat [CodingBat].

3 The FLOP System

The FLOP system [FLOP] is a web system intended to ease the practice of programming for both students and instructors. We had previous experience in using judges in the first stages of several programming contests and in the use of Mooshak [Leal 03] for four years in regular programming courses [Rubio-Sánchez 14]. We first constructed a prototype of FLOP [Llana 12] primarily designed to serve in our own programming teaching. Based on our experience using the prototype and on the results of the user-centered actions presented in the article (including a usability evaluation involving students), we refined and extended FLOP, resulting in the version we present here.

In the first subsection we introduce the philosophy and basic features of FLOP. Then, we show the system functionality for registered students. Finally, we present the functionalities FLOP offers to instructors for managing collections of problems and statistics about students' usage and performance.

3.1 Basic Features of FLOP and Open Usage

Our ultimate goal on designing FLOP [Llana 12] was to provide a system that could be usable with low effort by both students and instructors. Our rationale follows:

- Students. We wanted to foster students' practice with programming problems anytime and anywhere. Consequently, the FLOP System was conceived as an open-access website. Users do not have to either install any program or plug-in on their client side or to log in to the system. They only have to enter into the web site, choose any programming problem and submit their solutions. This openness encourages users to browse the collection of problems and to try to solve them without fear of any consequences from mistakes. Consequently, we expected that students would feel free to practice at their own pace.
- Instructors. FLOP also was intended to be attractive to instructors by demanding very little time for using it in their courses. Instructors can create collections of problems from the set of problems provided by FLOP with just several clicks. In addition, a URL associated with the new collection is given to the instructors so that they just have to e-mail it to students. Finally, FLOP is an open source code system distributed under a GPL license, so enthusiastic instructors may adapt it to their needs.

Let us briefly illustrate the open use of FLOP. From its home page, users may access the "Practice Area" (see Fig. 2) from the main menu (at the left-hand side), where they can find two collections of problems. Currently, FLOP hosts about 130 problems, all of them written both in English and Spanish, covering different topics and levels of difficulty. They are intended to support an introductory course to programming (CS1) and a course on data structures and algorithms (CS2) currently under development. Thus, any user (either a student willing to practice or an instructor willing to try with the system) could choose any programming problem and submit a solution. To facilitate the selection of problems, the statement of any problem emerges at the right of the mouse cursor by just putting it over the problem title.

When a problem is selected, its statement is displayed (e.g. see Fig. 1). Then, the user may submit a program intended to solve the problem in a file by using the form at the end of the page (see Fig. 1). After a few seconds, the system shows a page with the results of each test case and one message summarizing the assessment of the program (as explained in Subsection 2.2).

The current version of FLOP provides two enhancements with respect to black-box testing, which were repeatedly demanded by students. Firstly, the "Presentation error" message has been replaced by "Correct but format". We consider that the new warning is fairer with respect to the student achievement, thus being less demotivating and more formative. Notice that this situation is very common given the very strict format of output that is imposed for ease of automatic assessment rather than for algorithmic or instructional reasons.

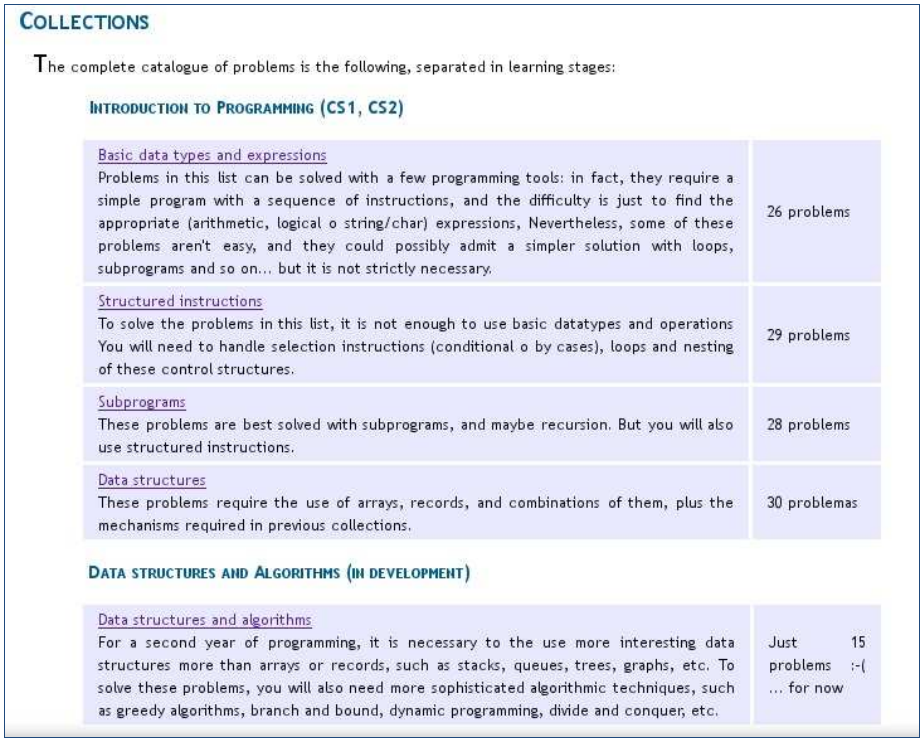

Figure 2: The Practice Area of FLOP

Secondly, FLOP tries to assist students by showing the first test case where their program failed. It shows the expected output in this case and the output generated by the student program (see Fig. 3).

Currently, the FLOP system still provides open access, but it has been extended with some features to make more useful for formal education. These features are mainly intended for teachers, so that the effort otherwise needed to integrate automatic assessment systems into a programming course can be dramatically reduced. These extensions are presented in the ensuing subsections.

3.2 Registration of Students

This facility allows an instructor to have her students solving a collection of problems and then assigning them the grade given by the system. First, the instructor must publish the URL of a FLOP page where they only have to register their e-mail address (i.e. no additional information is demanded from students). Once a student registers, she receives an e-mail containing the URL where they have available the collection of problems created by the instructor. This page also contains information about the student activity and performance: submitted programs, accepted programs, etc. (see Fig. 4).

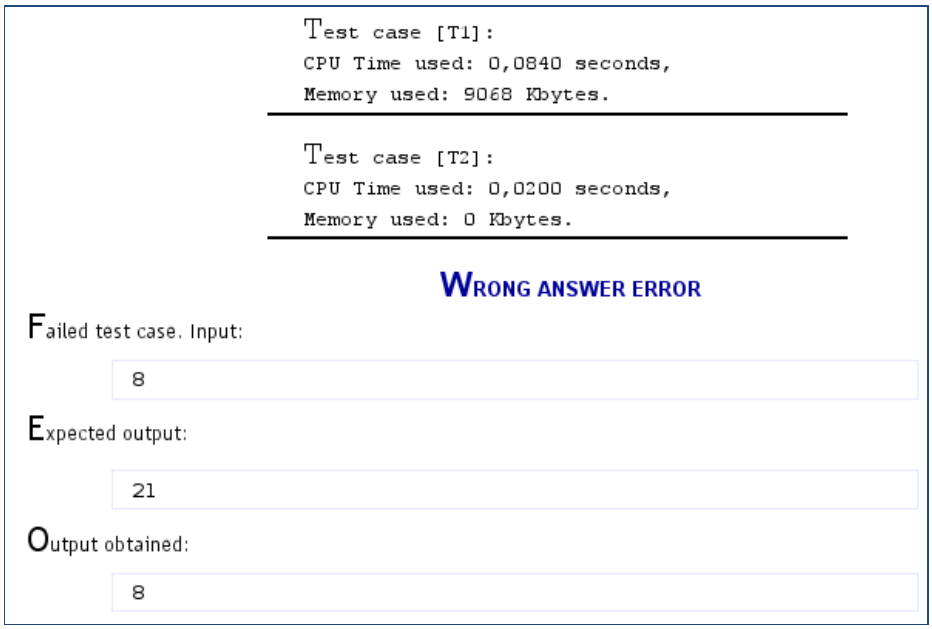

Figure 3: Message delivered by FLOP for a "Wrong answer" error

Every collection page contains a button to "Set Anonymous Mode". In this mode, all attempts to solve a problem will be recorded anonymously, i.e. without recording the student e-mail. Students can change between anonymous and non-anonymous mode at any time. Of course, information about the activity or performance of a student working in anonymous mode is neither available to her nor to the instructor.

When a student accesses a collection page, the left menu shows three additional entries. The "Current Collection" link allows her to return to the collection page. The "Collections" entry allows any student to see all the collections where she is registered with her e-mail address. For each collection, she may consult its information (similar to the collection page), may browse the collection (possibly changing the current collection where they are logged) and may see all the files submitted as solutions. Finally, the "Logout" link closes the session linked to the e-mail address.

3.3 Management of Collections by Instructors

Problems included in FLOP were designed for teaching to a wide range of students, ranging from brilliant to novices. Its collections of problems are based on eXercita [Gregorio 01], a large collection of assignments developed using a system to archive and manage in a collaborative environment, and its major compilation of problems [Gregorio 02]. Currently, FLOP hosts 130 problems, most of them intended for a first course on imperative programming and a few of them constituting a seminal collection for a second course on data structures.

Figure 4: Student's view of a collection

One goal of the FLOP system is making things easy for users, so an instructor who wants to create and provide a new collection of problems for her students only has to follow three simple steps:

1. Click on the "Manage Collections" (using the left hand menu) of FLOP.
2. Create a title for the collection and give an e-mail address to manage it, and click on "Create New Collection". The instructor will receive an e-mail containing two links: one link for managing the collection (e.g. to add/remove problems or to see statistics and submissions) and one link to distribute to her students.
3. Problems can be added to the collection by navigating the "Practice Area", choosing the box nearest to the problem titles wanted, then clicking on the button "Update User Collection". Alternatively, a problem can be added to the current collection by pressing the button "Add to Collection" available on its statement page.

Afterwards, the collection of problems will be available in the "Current Collection" entry of the left menu and the instructor may deliver its access link to students. The instructor may also perform several additional actions on the "Current Collection" page:

- Remove problems.
- Rearrange the problems order by typing their position in the text areas near the problem names and then pressing the button "Update Collection".
- Delete the collection.
- Consult summary data of the collection.

At any time, the instructor may add problems to or remove them from any collection by navigating the "Practice Area", checking/unchecking the boxes near the problem titles and pressing the button "Update User Collection".

Finally, instructors have an entry "Collections" in the left hand menu to access and manage all her collections (see Fig. 5). The collections page contains a list of collection tabs. If a collection tab is selected, a summary of its associated data is shown: creation date, success rate (correct submissions divided by number of attempts) and number of accesses, submissions and correct submissions per problem. The collection tab also contains the link to distribute to students to access the collection. In addition, five buttons allow additional actions to be performed (see Fig. 5):

- The "Show Collection" button shows the main page of the collection and makes it current.
- The "Get submissions" button allows the viewing of all the programs submitted by students.
- The "Statistics" button shows the statistics of the collection (described in the following subsection).
- The "Remove" button deletes the collection, removing all its related data from the system database.
- The "CSV" button, which produces statistical information in this format for each student and for each problem.

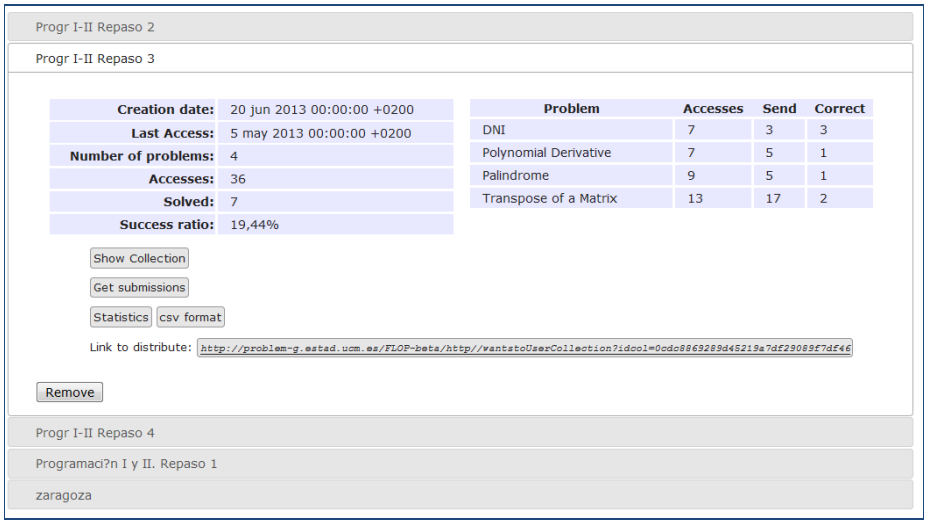

Figure 5: Page showing all the collections of an instructor, with the information of the second collection (out of five collections) visible

3.4 Statistics of User Usage

FLOP provides instructors with the statistics to show the usage of any collection. So they can monitor the progress of their students and check the effectiveness of a

collection. Statistics are available through from the "Statistics" entry in the left hand menu (and then selecting the collection) or the "Statistics" button of a collection.

The FLOP system displays statistical data grouped by problem or by student. Switching between both modes is possible with two buttons available at the top of the page. If the first option is selected, FLOP will show a list of tabs, one per problem. For each problem, the following information is displayed (see Fig. 6):

- A link to view all the programs submitted for the problem.
- The results of the submissions, 1) differentiating the programming languages used, and 2) not differentiating the programming languages used.
- Number of submissions per programming language.

To facilitate the viewing of statistics, this data is accompanied by interactive pie charts with highlight area showing the associated value when the mouse passes over it.

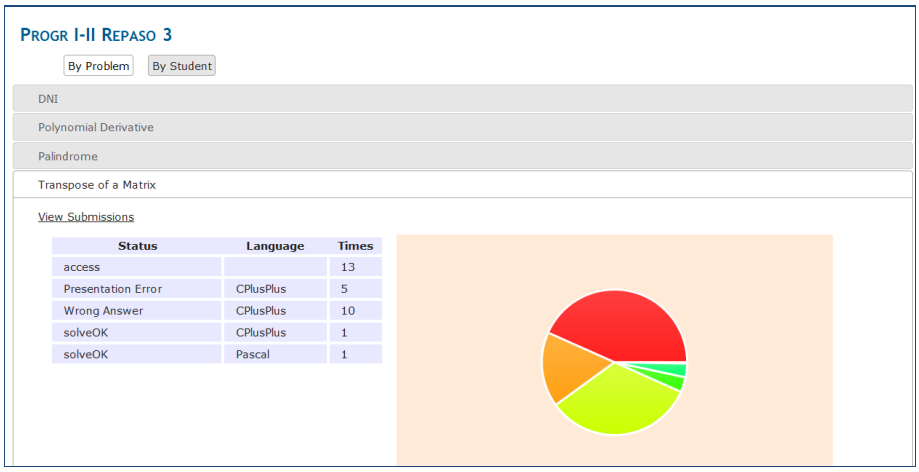

Figure 6: Statistics page per problem

Similarly, grouping statistics by students, the FLOP system shows a list of tabs (see Fig. 7), each one containing:

- A link to view all the student submissions (independently of the problem).
- A table containing the number of accesses and submissions of the student, grouped by the results obtained and the programming language used.
- An interactive pie chart linked to the table.

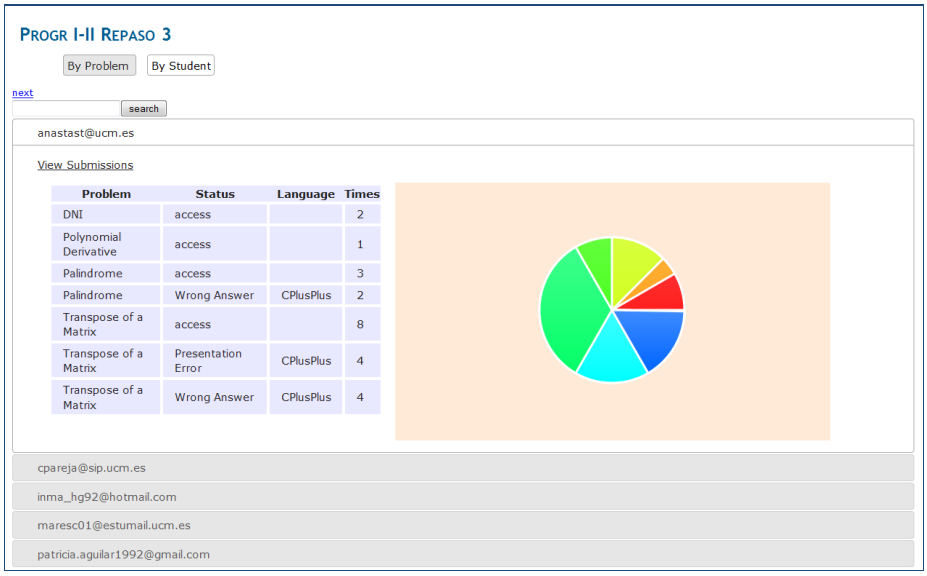

Figure 7: Statistics page per student

4 User- and Instructor-Centered Design

We outline in this section the process followed to develop FLOP, based on user-centered design. We also describe the actions conducted to satisfy to one class of users, i.e. instructors. In Section 5, we address the actions conducted to satisfy the second class of users, i.e. students.

4.1 User-Centered Design

FLOP has been developed using a user-centered design methodology. This is a general term used in human-computer interaction [Norman 13] but also in other product design fields. The major difference from other design methodologies is that user-centered design tries to enhance the product around how users use the product, rather than forcing the users to change their behavior to accommodate the product.

In brief, we constructed a first version of FLOP based on several key instructional principles and then we iteratively enhanced it in a cycle of evaluation-refinement steps. The real use of FLOP by end users was a key concern both in its initial development and in subsequent evaluations. We do not consider FLOP as a finished product, but we present here a stable version that we consider satisfies many of its users' concerns. We further explain some of these steps.

We started the development of FLOP after several years of experience using the automatic judge Mooshak in programming or algorithms courses [Rubio-Sánchez 14]. Based on this experience, we became aware of the heavy workload that automated assessment systems impose on instructors. As a consequence, we identified the major

feature an automated assessment system should exhibit from the instructor's point of view: to ease the use of the tool without any extra effort on the user (either an instructor or a student).

Consequently, the main design principles underlying the initial version FLOP were [Llana 12]:

- It was an on-line, free web service which could be used without requiring the installation of any software at the computer.
- It allowed anonymous access, fostering free access by students and instructors.
- It was developed under GNU license, therefore any instructor could download it, install it in her server, use it with her students, modify it or add new exercises.

Since, we have conducted several kinds of evaluations intended to enhance it. Evaluations by experts in usability have been conducted by the developers to identify early the obstacles to usability and to fix them. More relevant here are evaluations of FLOP by end-users. In the following subsection we describe an evaluation aimed at satisfying instructors' needs; in Section 5 we describe an evaluation whose aim was to satisfy students' needs, which was much more elaborate and yielded more interesting results.

4.2 Instructor-Centered Design

We tried to address instructors' needs and concerns by making presentations to instructors of programming courses. In particular, we made one presentation to five instructors of the Universidad Rey Juan Carlos. The goal of the presentation was two-fold: to encourage them to adopt FLOP in their courses and to gather suggestions for improvement from the instructors' point of view.

The session consisted of a presentation and a focus group meeting. First, we introduced FLOP, including some hints for practical use, for about 20 minutes. Then, an unstructured conversation was held, where the instructors asked questions. Most suggestions could be satisfied in different ways, but others could not be supported by the status of FLOP development.

We systematically gathered these suggestions, thus we may highlight two main contributions:

- Any instructor should be able to contribute to the system collection with new problems.
- The instructor should be able to import the list of the students enrolled in her course in a standard format to ease students' grading.

5 Student-Centered Design

We addressed students' needs by means of an evaluation of FLOP usability based on questionnaires. Not only did we try to inquire about the usefulness of the system but also the relevance to students and whether it suits their needs. We present an evaluation in this section. We first describe the evaluation design and then we present

the results of different parts of the questionnaires. Full details can be found in a technical report [Llana 13].

5.1 Evaluation Design

Feedback from students was gathered by means of a questionnaire measuring the perceived usability of FLOP. This evaluation was held in May 2013, in the mandatory courses *Computing* and *Programming II*, in the first academic year of the degrees of Mathematics and Applied Statistics, respectively.

FLOP was used in the last six weeks of the courses. Students were introduced to test-driven development and attended a preliminary session of familiarization with FLOP. In particular, the standard structure of assignments was stressed, with an emphasis on the strict input/output format and the corrector error messages.

After this initial session, a sequence of small collections was successively delivered to the students. Each collection was presented in a laboratory session, but students had a one-week deadline to solve them. Each collection consisted of four to six exercises: some of them easy to solve, some of medium difficulty and zero or one difficult problem.

Let us remember that, when a student receives an e-mail message for a sheet of problems, she must self-identify to be able to access to it. This way, the instructor may track students' activity. However, students may deliver solutions anonymously, both for a given worksheet and for the complete collection.

Students were given a questionnaire after the exam consisting of nine closed questions and six open questions. Answers to close questions were in a Likert scale ranging from 1 (very bad) to 5 (very good). Students received extra credits for using FLOP (up to 1 point over 10, depending on the amount of work and its quality) and for filling in the questionnaire. Consequently, students were encouraged to submit their questionnaire identifying themselves, but anonymous questionnaires also were accepted. We collected 26 questionnaires, respectively 15 and 11 from each group. We gathered 21 questionnaires from males, 10 from females and 5 anonymous questionnaires. An analysis of students' answers follows, first the answers to closed questions and then answers to open questions.

5.2 Responses to Close Questions

The first question asked how often they had used the FLOP system. The results are shown in Table 1. We noticed that half the students (13 out of 26) used FLOP often or very often.

No answer	Nothing	Rarely	Often	Very often
2 (7.7%)	2 (7.7%)	9 (34.6%)	11 (42.3%)	2 (7.7%)

Table 1: Results about usage of FLOP

We also asked students their opinion about different issues regarding the system. The questions, the mean and the mode of the answers are shown in Table 2, sorted in decreasing order of mean. We gathered 25 answers for each question, as one student misunderstood the expected format for answers.

Statement	Mean	Mode
Organization of exercises clear	4.52	5
Organization of web site adequate	4.39	4
Exercises carefully designed	4.17	4
FLOP easy to use	4.00	4
Amount of problems proposed reasonable	3.96	3.5
FLOP contains exercises of adequate difficulty (for you, among other problems)	3.95	4
Overall, do you like FLOP?	3.83	4
FLOP has aided you in learning to program, in proportion to the use you have made of it	3.50	4
Global	**4.04**	**4**

Table 2: Results of general, closed questions

Notice that the best results are obtained in the three questions regarding system quality. The results that follow are about system acceptance and problems. The lower rank corresponds to perceived system utility.

5.3 Responses to Open Questions

We asked students six open questions (to gather a detailed opinion of students) about the positive and negative features of FLOP. The two students who acknowledged that they had not used FLOP (see Table 1) left all their answers blank. Table 3 contains the number of answers given for each open question.

We filtered the answers gathered before analyzing them. Some answers were filled in some cells but it was obvious that they corresponded to other questions. Other answers were not informative (e.g. "not detected" for negative features). Sometimes, several answers given by a student were redundant. In some cases, an answer was multiple, given that it contained several pieces of information. Finally, we catalogued suggestions for improving negatives features. Then, we filtered and re-catalogued the answers, resulting in the numbers shown in the last column of Table 3.

Question	Received answers	Catalogued answers
Positive features	21	27
Useful features that FLOP lacks	14	16
Negative features	17	17
Useless features you would discard	2	–
Other interesting features	6	–
Other comments	6	–
Total	**68**	**60**

Table 3: Number of answers to open questions

We iteratively analyzed the answers using coding and content analysis methods of qualitative research [Cohen 11] and classified them into seven categories. We include a short review of each category, composed of a description and one literal quote (S*n* denotes the questionnaire of the *n*-th student), in decreasing order of frequency:

- Assistance to detect errors (12 answers). These students demanded more information from FLOP about the nature of errors, where to locate them and hints how to solve them. "It would be very useful if FLOP indicated somehow why your file is not accepted, so it can be modified, and also later provide a possible solution if it is not accepted several times." (S12).
- Strict format (10 answers). Most of these comments refer to the strict input/output format for data that is common in automatic correctors and also in FLOP. "It is not flexible at all for testing programs. Any deviation in the format gives an error" (S14). A few answers refer to the format of solutions, the presentation of exercises or to the programming language used.
- Other functions of the system (10 answers). They are very heterogeneous suggestions.
- Usefulness for self-study (9 answers). "FLOP provides problems of increasing difficulty that help correct progress from the beginning, besides the available problems are very diverse, so there is no need to repeat the same kind of problem" (S24).
- Usefulness as a corrector (8 answers). "Sometimes it only works for some values and you do not realize, so the system is useful because it tests your program with a wider range of cases." (S18).
- Usefulness in general for the course (7 answers). "It helps with revision on the syllabus especially during the summer holydays" (S8).
- General comments on the system (4 answers). "It is well organized, clear and accurate" (S6).

In Table 4, answer categories refer back to the question:

- Positive features refer to 3 ways of using FLOP and general comments about it.
- Negative or those needing improvement correspond to assistance in detecting errors, in formatting and other functions.
- Useful features that FLOP lacks: these answers correspond to assistance in detecting errors and other functions.

Question	Positive features	Negative or to-improve features	Useful features FLOP lacks
Assistance to detect errors		2	10
Strict format		10	
Other functions		5	5
Utility for self-study	8		
Utility as a corrector	8		
Utility in general for the course	7		
General comments on FLOP	4		
Total	**27**	**17**	**16**

Table 4: Types of answers to open questions

6 Discussion

In this Section we discuss the results of students' opinions and how to incorporate this experience and analysis to FLOP.

6.1 Students' Opinions

The results obtained from students' evaluations were overall very positive. With respect to closed questions, half the students used FLOP often or very often. As they only used FLOP for a short time and the date of their final exams were imminent, this showed a high acceptance of FLOP. Students found it useful and easy to use.

The scores for several issues were also high. The overall mean was 4.04 over 5. The highest scores were related to the perceived quality of the system and the materials, followed by statements about ease of use and likeness. The poorer results were its usefulness. We judged these scores very positively given the circumstances of the evaluation: the system was only used for a few weeks and it did not contain some of its current features (most of which were implemented following students' suggestions). The low score regarding usefulness is easily explained by the relatively short time students used it in the course.

Open answers consistently reinforced these results. Most students who used the system pointed out its potential usefulness for a programming course. Some students also made positive overall judgments on the system.

6.2 Improvements Incorporated into FLOP

Suggestions of improvements came from three sources: The developers' experience as instructors, from other instructors, and from students' opinions. We discuss here which suggestions were taken into account and which ones were not. They are classified into five groups of improvements. The two first groups relate to the instructors' needs, whilst the ensuing three ones deal with the students' needs.

- Adoption and use in a course. Using an educational system is not an easy task: it must be searched, installed, adapted to the course and put into use. In the introduction we cited some authors who pointed out this critical issue. Open access presents advantages for universal use but it does not directly address these mundane issues.

  The most evident need of instructors using an automated assessment system was the effortless generation, maintenance and delivery of collections of exercises. On addressing the integration of FLOP into a course, we detected a compatibility problem with open access. Our solution consisted of allowing open access to the general catalogue, while asking students the minimum identification in those cases it was necessary, such as access to the course collections of problems.

  Another suggestion enabled instructors being able to import the list of the students enrolled in their course in a standard format to ease students' grading. FLOP generates a CSV file that can be imported from any Excel-like spreadsheet.
- Tracking students' performance and activity. Since the advent of LMS, tracking students is an important concern, commonly named learning analytics [Siemens 12]. However, the information displayed in an automated assessment system needs not be as comprehensive as in an LMS.

  Consequently, we incorporated data and descriptive statistics for the two most relevant criteria for an instructor: student and problem. In addition, two properties were gathered for each of these criteria: status of submissions and programming language used.
- Assistance in detecting errors. Students demanded more information from the system about their errors and how to fix them. This is a non-trivial issue because two conflicting learning concerns collide. On the one hand, formative evaluation is successful not only if students are given information about how many errors they have made but also which ones. On the other hand, a risk in giving all the information to students about their errors is that they may use the system as a compiler or a debugger [Malmi 05].

  We adopted an intermediate solution consisting in giving students information about the first test case that caused their program to fail. A message is shown briefly explaining the test case (the input and its corresponding output) and the output actually given by the program.
- Strict input/output format. The most common way for specifying test cases consists of providing input/output textual descriptions. Judges typically enforce strict adherence to this specification, including blank spaces. However, educational correctors are often more flexible [Ala-Mutka 05, Ihantola 10]. In particular, there is a risk of contradiction between programming concepts learned in the course and the assessment system behavior, given that students learn that blanks are not important in programs (but for aesthetical purposes).

  Our decision was to replace the message "Presentation error" by "Correct but format". The new message acknowledges that the student's algorithm is correct, identifying output format as the only obstacle to the submission being considered correct.

- Other functions. This category comprises a number of heterogeneous suggestions. Some suggestions even came from students ignoring some of the system capabilities (e.g. they thought that the system needed an e-mail address to use the collections). The lesson learned here is that we must devote more time to students' training in FLOP.

  Other suggestions require very different degrees of effort to implement. At one extreme we find some trivial and distracting suggestions (e.g. enhancing the system logo); at the other extreme we find very demanding suggestions that are worth considering but demand a heavy development effort (e.g. integrating FLOP and an IDE). The latter suggestion has already been incorporated in other systems, e.g. in COALA as an Eclipse plug-in [Jurado 09]. However, this solution contradicts our philosophy of letting FLOP universally available without requiring the user to install any software. Both kinds of suggestions have not been prioritized at the present stage.

  We have considered those suggestions that may produce a substantial improvement in the students' perception of FLOP usefulness without losing user-friendliness. The most relevant suggestion we have incorporated refers to showing the status of the problems that the student has already tried to solve.

6.3 Related Systems

As Ihantola [10] notices, the number of automated assessment systems constructed in past years is very large. Consequently, it is very difficult, if possible at all, to identify "the best" system. However, we may compare the different systems with respect to those features that try to satisfy users' concerns and needs, with an emphasis on reducing instructors' effort for using a system in a course.

Figure 5 contains the result of comparing FLOP with a subset of relevant systems. Each row contains a different system. Some of them were selected because of their relevance in the academic community (e.g. BOSS or Web-CAT) while others were selected because of their similarity to FLOP, at least in some features (e.g. CodingBat or Jutje.org). The selection is relatively arbitrary but it is representative of a wide range of systems, from older to newer, from contest judges to educational systems.

The meaning of the columns is as follows:

- **Access** (Free or restricted access): does the corresponding system allow free access or the user must be registered and identified with a user name?
- **Open source**. Is the system source open and universally available?
- **Built-in collection**: in case of providing a set of problems, this column roughly indicates the number of available problems.
- **User collections**: does the system allow users to build their own collections?
- **Statistics** (User/problem statistics): does the system provide usage statistics about either users' activity or regarding specific problems?
- **Meaningful feedback**: does the system give the users meaningful explanations regarding their activity, especially about their errors?
- **Flexible checking**: does the system allow any relaxation with respect to output format of checking rules?

System \ Feature	**Access**	**Open source**	**Built-in collection**	**User collections**	**Statistics**	**Meaningful feedback**	**Flexible checking**
BOSS [Joy 05]	R	✓				✓	✓
CodingBat [CodingBat]	F / R		>100		U	✓	
CourseMarker [Higgins 05]	R				U	✓	✓
Jutge.org [Giménez 12]	R		>1,500	✓	U / P		✓
Mooshak [Leal 03]	R	✓			U / P		✓
TRAKLA [Malmi 04]	R	✓	≈50			✓	
USACO [Kolstad 09]	R		>100			✓	
UVA online Judge [Revilla 08]	R		>5,000		U / P		✓
Web-CAT [Edwards 04]	R	✓				✓	✓
FLOP	F / R	✓	≈130	✓	U / P	✓	✓

Table 5: Comparison of features provided by a selection of automated assessment systems

Notice that there is no system that provides all of the selected features to potential users but FLOP. In particular only two systems support free access and management of user problem collections. However, our conclusion is not that FLOP is more adequate than other systems to users' needs. Some concerns could be analyzed into greater detail. More importantly, notice that the range of potential issues and concerns is very wide. The users interviewed or tested by us expressed specific concerns. We think that some concerns are very relevant but we guess that others might arise in different contexts.

7 Conclusions

Automatic submission systems are invaluable tools in programming education. Nevertheless, the excessive effort they demand instructors is a major obstacle to use in daily instruction. The FLOP system has been developed and refined to be useful in an educational setting while alleviating this problem, both for students and instructors. We have described in this article its two main contributions: the FLOP system itself and the user-centered process followed to refine the system and ultimately to achieve its intended goals.

We summarize here the evolution of the most relevant features of FLOP. Initially, the main design principles underlying FLOP were:

- To be an on-line, free web service which can be used without requiring the installation of any software at the computer.
- To allow anonymous access, fostering free access by students and instructors.
- To be free under GNU license, therefore any instructor can download it, install it in her server, use it with her students, modify it or add new exercises.

The refinements introduced into FLOP were the result of considering both the instructors' and the students' points of view. This approach has proved to be successful in other experiences [Velázquez-Iturbide 13]. The main features added are as follows:

- Creation, maintenance and delivery of collections of exercises.
- Statistical information available to instructors and students about their submissions and performance.
- More information about the errors found in student's submission, thus reinforcing formative assessment.
- Format errors considered less severe by giving a more encouraging and more acknowledging message to students.

The collection has been extended, so that it currently hosts about 130 problems, available both in English and in Spanish. We are disseminating FLOP in a number of forums, so we hope that in the next academic year it will be adopted by a number of groups for their teaching in programming courses. This will allow us to see whether we adopted right design decisions in our user-centered refinement of FLOP.

There are different challenges that we would like to address in the near future. Some challenges refer to FLOP. For instance, we cited in Section 4.2 that an instructor suggested free contribution to the collection of exercises. This feature could even be considered for students, leading to "contributing student learning" [Hamer 10]. However, it requires major design decisions regarding control of problem quality [Fincher 10]. A number of other enhancements also are scheduled, ranging from technical improvements to additional evaluations of the system in actual usage.

In a more general setting, we would like to study the features that lead to less user effort in using an automated assessment system, as well as to user adoption of a system. We consider that our experience has been valuable, but more studies should be conducted and more general conclusions should be achieved.

Acknowledgements

This work has been supported by research grants TIN2011-29542-C02-01 and TIN2012-36812-C02-01 of the Spanish Ministry of Economy and Competitiveness, and S2009/TIC-1465 and SICOMORO-CM (P2013/ICE-3006) of the Regional Government of Madrid.

References

[Ala-Mutka 05] Ala-Mutka K. M.: A survey of automatic assessment approaches for programming assignments. Computer Science Education 15, 2 (2005) 83-102.

[Beck 02] Beck, K.: Test Driven Development by Example, Addison-Wesley Professional, 2002.

[Carter 03] Carter, J., English, C., Ala-Mutka, K., Dick, M., Fone, W., Fuller, U., Sheard, J.: How shall we assess this? In Proc. 8th Annual Conf. Innovation and Technology in Computer Science Education (ITiCSE 2003), Working Group Reports, 17 pp.

[CodingBat] CodingBat, http://codingbat.com/

[Cohen 11] Cohen, L., Manion, L., Morrison, K.: Research Methods in Education, Routledge, Abingdon, UK, 7th ed., 2011.

[Desai 09] Desai, C., Janzen, D.S., Clements, J.: Implications of integrating test-driven development into CS1/CS2 curricula. In Proc. 40th ACM Technical Symp. Computer Science Education (SIGCSE'09) 148-152.

[Douce 05] Douce, C., Livingstone, D., Orwell, J.: Automatic test-based assessment of programming: A review. Journal of Educational Resources in Computing, 5, 3 (2005).

[Edwards 04] Edwards, S. H.: Using software testing to move students from trial-and-error to reflection-in-action. In Proc. 35th SIGCSE Technical Symp. Computer Science Education (SIGCSE'04), 2004, 26-30.

[Elbaum 07] Elbaum, S., Person, S., Dokulil, J., Jorde, M.: Bug hunt: Making early software testing lessons engaging and affordable. In Proc. 29th International Conf. Software Engineering (ICSE'07), 2007, 688-697.

[Fincher 10] Fincher, S., Kölling, M., Utting, I., Brown, N., Stevens, P.: Repositories of teaching material and communities of use: Nifty Assignments and the Greenroom. In Proc. 6th International Workshop on Computing Education Research (ICER 2010), 2010, 107-114.

[FLOP] FLOP, http://problem-g.estad.ucm.es/.

[Georgouli 11] Georgouli, K., Guerreiro, P.: Integrating an Automatic Judge into an Open Source LMS. International Journal of E-Learning 10, 1 (2011), 27-42.

[Giménez 12] Giménez, O., Petit, J., Roura, S.: Jutge.org: An educational programming judge. In Proc. 17th Annual Conf. Innovation and Technology in Computer Science Education (ITiCSE 2012), 2012, 445-450.

[Gregorio 01] Gregorio, C., Llana, L.F., Martínez, R., Palao, P., Pareja-Flores, C., Velázquez-Iturbide, J.Á.: EXercita, A system for archiving and publishing programming exercises. In Computers and Education, Towards an Interconnected Society. Kluwer Academic (2001), 187-198.

[Gregorio 02] Gregorio, C., Llana, L.F., Martínez, R., Palao, P., Pareja-Flores, C.: Ejercicios de programación creativos y recreativos en C++, Prentice-Hall, 2002.

[Hamer 10] Hamer, J., Luxton-Reilly, A., Parchase, H.C., Sheard, J.: Tools for "contributing student learning". In Proc. 15th Annual Conf. Innovation and Technology in Computer Science Education (ITiCSE 2010), Working Group Reports, 14 pp.

[Higgins 05] Higgins, C.A., Gray, G., Symeonidis, P., Tsintsifas, A.: Automated assessment and experiences of teaching programming', ACM Journal on Educational Resources in Computing, 5, 3 (2005) article 5.

[Ihantola 10] Ihantola, P., Ahoniemi, T., Karavirta, V., Seppälä, O.: Review of recent systems for automatic assessment of programming assignments. In Proc. 10th Koli Calling International Conf. Computing Education Research (Koli Calling 2010), 86-93.

[Joy 05] Joy, M., Griffiths, N., Boyatt, R.: The BOSS online submission and assessment system. In ACM Journal on Educational Resources in Computing 5, 3 (2005), article 2.

[Jurado 09] Jurado, F., Molina, A. I., Redondo, M. Á., Ortega, M., Giemza, A., Bollen, L., Hoppe, H. U.: Learning to program with COALA, a distributed computer assisted environment. Journal of Universal Computer Science 15, 7 (2009) 1,472-1,485.

[Kolstad 09] Kolstad, R.: Infrastructure for contest task development. Olympiads in Informatics, 3: (2009) 38–59.

[Leal 03] Leal, J. P., Silva, F.: Mooshak: a web-based multi-site programming contest system. Software Practice & Experience 33 (May 2003), 567-581.

[Llana 12] Llana, L., Martín-Martín, E., Pareja-Flores, C.: FLOP, a Free Laboratory of Programming. In Proc. 12th Koli Calling International Conf. Computing Education Research (Koli Calling 2012), 93-99.

[Llana 13] Llana, L., Martín-Martín, E., Pareja-Flores, C., Velázquez-Iturbide, J.Á.: Una evaluación de usabilidad de FLOP. Internal report DLSI1-URJC 2013-01, Universidad Rey Juan Carlos, Spain.

[Malmi 04] Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppäla, O., Silvasti, P.: Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. Informatics in Education 3, 2 (2004) 267–288.

[Malmi 05] Malmi, L., Karavirta, V., Korhonen, A., Nikander, J.: Experiences on automatically assessed algorithm simulation exercises with different resubmission policies. ACM Journal on Educational Resources in Computing 5, 3 (2005), article 7.

[Myers 04] Myers, G. J. Art of Software Testing. John Wiley & Sons, New York, 2nd ed., 2004.

[Norman 13] Norman, D.: The Design of Everyday Things, Basic Books, New York, revised ed., 2013.

[Revilla 08] Revilla, M.A., Manzoor, S., Liu, R.: Competitive learning in informatics: The UVA Online Judge experience. Olympiads in Informatics, 2 (2008) 131–148.

[Rubio-Sánchez 14] Rubio-Sánchez, M., Kinnunen, P., Pareja-Flores, C., Velázquez-Iturbide, J. Á.: Students perception and usage of an automated programming assessment tool. Computers in Human Behavior 31 (2014) 453-460.

[Ruehr 02] Ruehr, F., Orr, G.: Interactive program demonstration as a form of student program assessment. Journal of Computing in Small Colleges 18 (2002) 65-78.

[Shepard 01] Shepard, T., Lamb, M., Kelly, D.: More testing should be taught. Communications of the ACM 44, 6 (June 2001), 103-108.

[Siemens 12] Siemens, G.: Learning analytics: Envisioning a research discipline and a domain of practice. In Proc. 2nd International Conf. Learning Analytics and Knowledge (LAK'12), 4-8.

[Velázquez-Iturbide 13] Velázquez-Iturbide, J. Á., Pérez-Carrasco, A., Debdi, O.: Experiences in usability evaluation of educational programming tools. In Student Usability in Educational Software and Games. IGI Global (2013) 130-143.

[XP 09] Extreme programming, a gentle introduction. http://www.extremeprogramming.org/, 2009.