Jutge.org: Characteristics and Experiences[*]

J. Petit, S. Roura, J. Carmona, J. Cortadella, A. Duch, O. Giménez,
A. Mani, J. Mas, E. Rodríguez-Carbonell, A. Rubio, J. de San Pedro, D. Venkataramani [†]

Abstract

Jutge.org is an open educational online programming judge designed for students and instructors, featuring a repository of problems that is well organized by courses, topics and difficulty. Internally, Jutge.org uses a secure and efficient architecture and integrates modern verification techniques, formal methods, static code analysis and data mining. Jutge.org has exhaustively been used during the last decade at the Universitat Politècnica de Catalunya to strengthen the learning-by-doing approach in several courses. This paper presents the main characteristics of Jutge.org and shows its use and impact on a wide range of courses covering basic programming, data structures, algorithms, artificial intelligence, functional programming and circuit design.

1 Introduction

It is a well-established fact that practice is fundamental to learn computer programming. Whether at secondary, high school or university level, instructors assign programming problems to students in order to help them acquiring this skill. However, correcting programming assignments can be tedious and error-prone. Fortunately for instructors, programming problems are ideal candidates for automated assessment. Reviews on automatic assessment of programming problems can be found in [13,19] and recent experiences include [6,21,23].

Currently, the dominant automatic correction systems are the *online programming judges* on the internet. Online judges are web-based systems that offer a repository of programming problems and enable submitting solutions to these problems so as to obtain a verdict on their correctness and efficiency by applying a battery of unit tests under certain memory and time constraints. Popular online judges include UVa and CodeForces, but these judges are competitively oriented and not adequate for learning. On the other hand, some MOOCs such as the Algorithms courses from Princeton University and the Machine Learning course from Stanford University at Coursera also feature this kind of automatic assessment. However, their repository of problems is much smaller, more focused on a specific topic, and strongly relies on a specific programming language (PL). In a similar vein, many interactive programming educational platforms are also offered on the web, but often these do not focus much on algorithmic problem solving but rather target learning PLs, building games, creating web pages...

The purpose of this paper is twofold: First, it describes Jutge.org, an open educational online programming judge that is the offspring of some teaching projects in several undergraduate programming courses in the Universitat Politècnica de Catalunya (UPC). Second, it presents its use and impact on a wide range of UPC courses spanning the last ten years and covering basic programming, data structures, algorithms, artificial intelligence and functional programming.

The main characteristics of Jutge.org are:

– Jutge.org is designed both for students and instructors. It integrates in a coherent way ideas from traditional online judges and from learning management systems, with a user-friendly interface.

– The repository of problems at Jutge.org covers many topics including basic programming, data structures, algorithms, artificial intelligence, functional programming and circuit design. The problem collection

[*]This work was partially supported by funds from the Spanish Ministry for Economy and Competitiveness (MINECO) and the European Union (FEDER funds) under grants TIN2013-46181-C2-1-R and TIN2015-69175-C4-3-R, and SGR 2014 1034 (ALBCOM) of the Catalan government.

[†]O. Giménez is with Google, Mountain View (California). A. Mani and D. Venkataramani are from SASTRA University, Tanjore (India). All other authors are with the CS Department of the Universitat Politècnica de Catalunya, Barcelona (Catalonia).

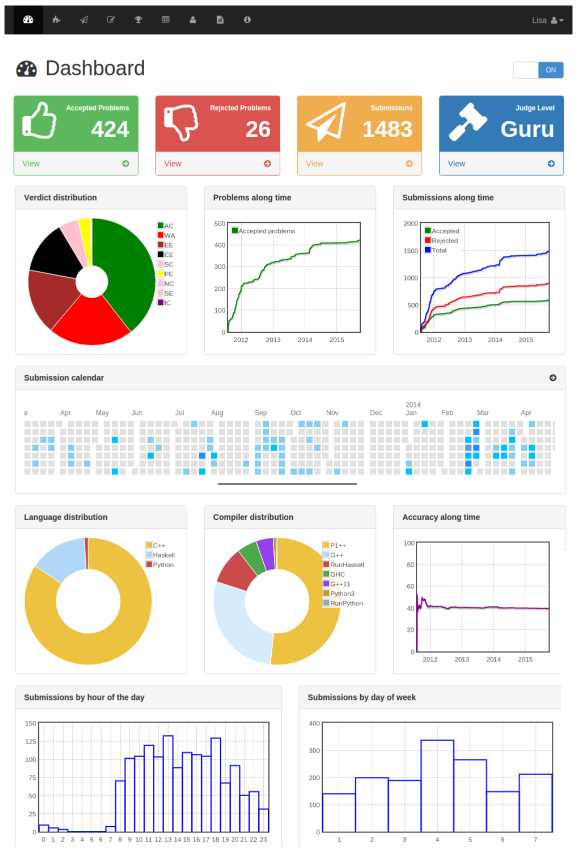


Figure 1: Sample snapshot for the dashboard of a student.

(about 2100 problems) is clearly organized and graded by difficulty, enabling a systematic progress, largely independent of the 22 supported PLs.
– Jutge.org integrates some techniques that go beyond other judges, such as modern verification and formal methods to assess problems on logic circuits, static code analysis to ensure constraints on functional programming problems, and data mining in order to provide appropriate feedback to users' submissions.

As a tool, Jutge.org has propitiated a number of positive changes in the way various programming courses at UPC are taught:
– The judge has become a structuring driving force to modernize the goals, contents, approach and presentation of most programming and algorithms courses.
– Jutge.org places emphasis on the work of the student and is therefore especially useful to reinforce the "learning-by-doing" approach.
– The judge has made possible a semi-automatic grading: In contrast with the old courses where programming was vastly relegated to pseudo-code written on paper, the current courses feature practical exams that take place in front of the computer. This results in a more objective grading than the older one.
– Jutge.org allows conducting alternative motivating and fun assignments where gamification concepts are used to improve learning by leveraging students' natural desires for competition and achievement.
– Other UPC activities such as programming contests, the Spanish Olympiads in Informatics, a Machine Translation MOOC and some summer camps for high school talented students are also powered by Jutge.org and provide added value to our university.

This paper is a coherent full account on the experience of developing and using Jutge.org during the last 10 years. Because of space reasons, some technical details are omitted; the reader is referred to [8, 11, 14, 16, 25] for more information on them.

In what follows, Section 2 presents a guided tour to Jutge.org. Then, Section 3 details the feedback that this judge provides to users' submissions and Section 4 summarizes the original algorithm to produce it. Afterwards, Section 5 presents a selected set of courses that use Jutge.org and, for each of them, details the impact of the judge on the learning and teaching experience. A comparison with similar systems is given in Section 6, before the conclusions.

2 Guided tour to Jutge.org

In this section we first introduce the main types of user and the main teaching resources available at Jutge.org. Afterwards, we present two typical use case scenarios.

Types of user and teaching resource. Users can have different roles: *Students*, *Tutors*, *Instructors* and *Administrators*. All users must register with a valid email address. Additionally, prospective instructors must contact the administrators proving their membership to an educational organization. Most of the features can be tried without registering using the `demo` account. Usage of Jutge.org is free and will remain free. Importantly, Jutge.org sends no spam and personal data is not disclosed to third parties.

Problems are the main resource of Jutge.org. They are identified by a unique short code, and typically consist of a problem statement that describes the task to perform, a correct and efficient reference solution (hidden to users), some public test cases, and some private test cases. For instance, problem P29212 is titled "Modular exponentiation" and can be found at https://jutge.org/problems/P29212. Most of the problems can be solved in any of the 22 programming languages currently available and are written in English. As multilingual problems are supported, many problems are also offered in Catalan and/or Spanish.

The teaching resources of Jutge.org are primarily organized in *courses*. Courses themselves contain *lists of problems*. A problem can be in many lists and a list can be in many courses. Instructors can invite students and tutors to their courses, and students can enroll in or unenroll from such courses. In addition, instructors can also add or edit their courses, lists and problems, and organize *exams* within courses. The system also features *awards*, which are motivating clip-art images rewarded to users when they achieve certain milestones. The system also provides an API to programatically interact with it.

Currently, Jutge.org contains about 2,100 problems, serves more than 15,000 users worldwide and has processed almost two million submissions.

Scenario 1: Student point of view. Let us consider the following scenario: Some student —let us call her Hermione— wishes to use Jutge.org to learn to program. After registering and logging in, Hermione will find her dashboard page that shows some statistics on her work (see Figure 1) and offers her some public courses where she can enroll in.

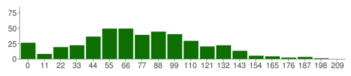
In order to try to solve her first problem, Hermione will read its problem statement and consider its public test cases. Then, Hermione will design and develop a solution for the problem in her own computer, compiling it, and validating it (at least, apparently) using the public test cases and other test cases that she can think about by herself. Finally, Hermione will upload her source code to the online judge, which will return a verdict, in about five seconds on average. To emit its verdict, the engine of the online judge uses private test cases, as exhaustive as possible, to check that the solution submitted by Hermione is efficient enough, and that its output is correct. In the case that her submission is considered correct, Hermione may see a report on software metrics comparing her solution with the reference solution, which may evidence that her solution has room for improvement despite being correct. In the case that her submission is incorrect, Hermione will usually receive a machine generated hint in the form of a short and relevant test case for which her solution fails.

Hermione can submit as many solutions as desired to the judge. All the submitted source code and associated verdicts are permanently stored so that Hermione can retrieve them later.

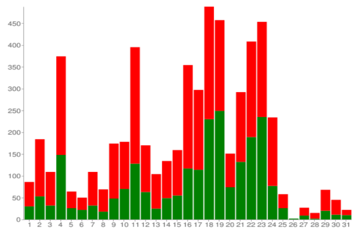
Scenario 2: Instructor point of view. We consider now another typical scenario: During the next semester, Prof. McGonagall will be the responsible of a course on programming at her university. About 400 students (including Hermione) will take that course, and a dozen teachers (including Hagrid) will assist her in the laboratory sessions.

To set up her course, Prof. McGonagall will get an account with *instructor* privileges. Then she will create a new course with some lists of problems that she will populate with the public problems already offered by Jutge.org according to her preferences. The system already offers so many problems that she does not need to create new problems on her own, but she could do it if she wishes so. Afterwards, she will invite her students via their official email addresses at the university. All invited students will receive an email stating that they can enroll in that course at Jutge.org. In the same way, Prof. McGonagall will also invite her teaching assistants, who will be designated as tutors for this course. Naturally, only the invited students and tutors will be allowed to join her course.

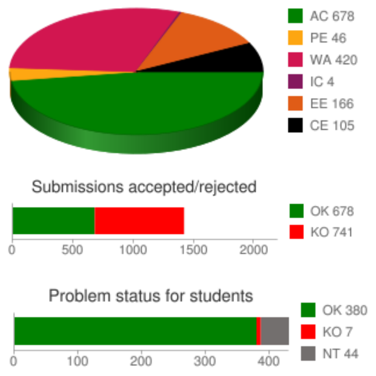
At some point, it may happen that Hermione has some doubts about a particular problem that she is not able to solve. Hermione can contact Hagrid by email to state her difficulties and ask for help. In order to help students, tutors can use a special feature of Jutge.org called *supervision*, which enables Hagrid to visualize Hermione's account and access her problems and submissions. By inspecting her code, Hagrid can spot her mistakes, answer her doubts and offer further guidance. To maintain privacy, tutors can only



(a) Histogram of solved problems by students. Hor. axis: number of problems; vert. axis: number of students.



(b) Number of submissions for each day of May 2011. Green: accepted submissions; red: rejected submissions.



(c) Results for problem "Greatest common divisor" (P67723): distribution of verdicts; proportion of accepted/rejected submissions; number of students with the problem accepted (OK), rejected (KO) and not tried (NT).

Figure 2: Sample snapshots of statistics offered to instructors.

supervise students that have enrolled in their courses, and supervision is limited to their problems.

All through the semester, Prof. McGonagall can gather data on the overall progress of her students. To help her, Jutge.org offers a wealth of diagrams and statistics such as the ones shown in Figure 2. Additionally, Prof. McGonagall will be able to easily create exams for her students. Jutge.org also offers other minor but practical functionalities that somehow alleviate the administrative burden for professors. For instance, the solutions submitted by the students during the exams can be easily distributed to the tutors, so that they can grade those solutions according to their own grading method.

3 Feedback in Jutge.org

The feedback reported to a user's submission is a key characteristic of any programming judge. In this section we explain the Jutge.org's feedback. (Section 5.5 provides particular information on feedback for problems on circuit design.)

In general, Jutge.org emits the following verdicts:

– *Accepted* (AC): The submission passes all public and private test cases.

– *Compilation Error* (CE): The submission does not compile.

– *Execution error* (EE): The submission crashes in some test case. Further indications such as *Uncaught exception*, *Division by zero*, *Invalid memory reference* or *Time limit exceeded* complement this verdict.

– *Wrong answer* (WA): The submission successfully executes all test cases but fails to write the expected output for some of them.

In addition to the verdict, Jutge.org also provides more feedback depending on whether the submission has been accepted or not:

Feedback for accepted submissions. Beginners tend to write overly long and complicated solutions. Therefore, the feedback of Jutge.org for accepted submissions focuses on warning them (if necessary) about that possibility.

This is achieved through the comparison of their solution and the reference solution using static code metrics. In particular, the following metrics are reported: number of lines of code, number of tokens, number of comments, number of auxiliary functions, Haelstad's difficulty (DIF) and McCabe's cyclomatic complexity (CCN)[1]. Whereas these code metrics are generally despised as a measure for developer's productivity [31], they have been used in other automatic grading tools [20,27,35] and we think they are useful in the context of small, specific assignments as they provide a nice way to compare beginner solutions to expert solutions without disclosing the latter ones.

Figure 3 shows a snapshot of the code metrics report on a (real) correct submission to the problem of returning the position of the first occurrence of an element in a sorted vector. As the gauges illustrate, the DIF and the CCN metrics of the solution are much higher than these of the reference solution, suggesting that the submitted solution is too complex.

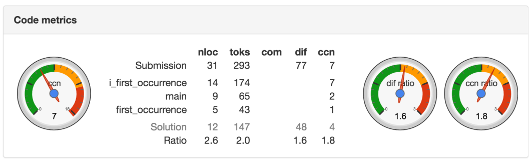
Feedback for rejected submissions. In the event that a submission is incorrect, and within the scope of the "Learning to Program" course (which contains about 200 problems), Jutge.org tries to hint the user with useful counterexamples where the code fails.

In the (too common!) case that failure happens on a public test case, the system warns the user and suggests him less rush and a better testing strategy.

In the case that failures happen on private test cases, the system reveals him up to two *distilled* test cases where the submission fails. Distilled test cases are a small set of relevant and concise test cases automatically computed by the system by a data mining algorithm that takes into account all past wrong submissions for this problem (this algorithm is explained in Section 4). As such, distilled test cases provide a deeper insight than just a dull and long arbitrary counterexample.

Figure 4 shows a snapshot of the two revealed test cases for a (real) incorrect submission to a problem that requires incrementing one second a clock time and printing it in the `hh:mm:ss` format. See how well the two offered hints capture two bugs in the submission: Input `9:59:59` catches a format mistake (presumably

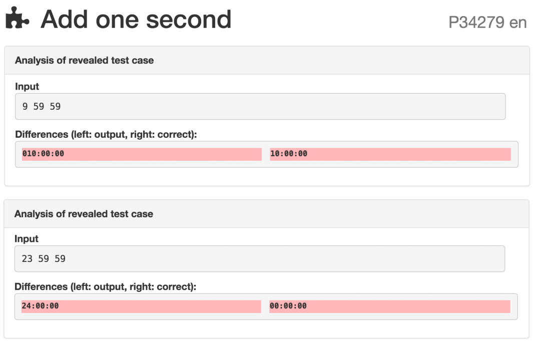
[1] McCabe's cyclomatic complexity [26] gives an indication of the complexity of a program through a quantitative measure of the number of linearly independent paths through its source code. Haelstad's difficulty [17] estimates its difficulty of being understood (e.g., in a code review) through counting the total number and the distinct number of operators and operands.



Code metrics

	nloc	toks	com	dif	ccn
Submission	31	293		77	7
i_first_occurrence	14	174			7
main	9	65			2
first_occurrence	5	43			1
Solution	12	147		48	4
Ratio	2.6	2.0		1.6	1.8

Figure 3: Code metrics reported to a correct submission to problem “First occurrence” (P84219).



Add one second

P34279 en

Analysis of revealed test case

Input

```
9 59 59
```

Differences (left: output, right: correct):

```
010:00:00                               10:00:00
```

Analysis of revealed test case

Input

```
23 59 59
```

Differences (left: output, right: correct):

```
24:00:00                                00:00:00
```

Figure 4: Automatically computed test cases for a wrong submission to problem “Add one second” (P34279).

an incorrect comparison) and input `23:59:59` catches a conceptual mistake (presumably a forgotten edge case). The next submission by that student fixed the bugs and obtained an AC verdict.

For problems that do not belong to the “Learning to Program” course, we consider that students should already have acquired a minimal programming maturity and are able to face negative verdicts such as WA and EE on their own, without hints from the system that would encourage sloppy-thinking programmers, who first recklessly write a wrong program, and afterwards try to patch it (multiple times) using the given counterexamples.

4 Extracting small and relevant test cases

In this section we present the algorithm that Jutge.org uses to automatically reveal a small subset of the private test cases to be used in the feedback of incorrect submissions.

Whereas an obvious solution would be to just show users an arbitrary instance for which their code fails, we feel that providing such a hint would not be effective as it would encourage a “trial and error” approach. Clearly, this is not the right way to proceed. Instead, we believe that the counterexamples provided should be as small and relevant as possible. By “relevant” we mean that these test cases should probably be helpful to most of the users.

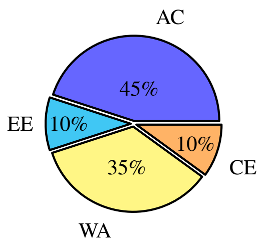


Figure 5: Current distribution of verdicts at Jutge.org.

Solution. Our solution consists in data mining past *incorrect* submissions sent by all the users of the judge. These automatically generated test cases would somehow capture the essence of the problem and, thus, should constitute a great feedback when revealed. This technique, which we call *the distiller* and elaborate in [25], has the virtue to put to use the wrong solutions, which represent about one third of all submitted solutions to Jutge.org (see Figure 5). Indeed, at this point, we are interested in *bad submissions*, defined as

Table 1: Test cases for problem “Validating dates” (P29448).

(a) Public sample test cases

Sample input	Sample output
30 11 1971	Correct Date
6 4 1971	Correct Date
4 8 2001	Correct Date
29 2 2001	Incorrect Date
32 11 2005	Incorrect Date
30 11 2004	Correct Date
-20 15 2000	Incorrect Date

(b) Distilled input test cases

29 2 4900	13 9 1800	29 2 1870
7 2 1900	28 0 1852	31 2 1964
31 8 8298	31 1 7709	1 12 2002
29 1 1942	31 12 5741	30 1 3157
30 2 1864	29 2 6592	0 2 1910
31 7 3535	31 9 1948	29 2 2000
31 2 2000	32 12 2008	0 3 2000
5 13 2000	31 11 2000	

Table 2: Test cases for problem “Brackets and parentheses” (P96529).

(a) Public sample test cases

Sample input	Sample output
`[]([]())`	yes
`(([])()`	no
`[](`	no
`(([]))`	yes

(b) Distilled input test cases

`(`	`[[[`
`[]`	`([)[]`
`][`	`[((()))]]`
`]]`	`[[(())](`
`)]`	`[(()[])][([]]])(())]`

the submissions that pass all public test cases but fail in some private test cases. We are not interested in submissions that pass all test cases (because they do not help finding relevant test cases) nor in submissions that do not pass the public test cases (because these do not even reflect a minimum quality).

The distillation process works as follows. Consider a bipartite graph whose left vertices correspond to test cases, right vertices to submissions, and edges to failure to report the correct output for a particular submission on a particular test case. Then, discovering the relevant test cases consists in finding a minimum subset S of left vertices such that every right vertex has some neighbor in S. This corresponds to solving the *Set Cover* problem, a classical hard problem in computer science. If the length of the input test cases must also be taken into account, one can enrich this graph formulation by weighting the left vertices by their length, and requiring a set cover of minimum weight. Similarly, it is also possible to cover just a fraction of the bad submissions.

To build the bipartite graph, the distiller must extract all individual private test cases of the problem. This is not a trivial task, as many test cases are contained in a single file, and some splitting must be performed to access them. Afterwards, the distiller solves the Set Cover problem using Integer Linear Programming (ILP). Despite the theoretical intractability of the Set Cover problem, we have seem that ILP work quickly in all our instances; heuristics could be used to reduce execution time if necessary, as optimality is really not needed.

Results. We have been able to distill almost all problems in the “Learning to Program” course using submissions of about 7200 users since September 2006. The results indicate that our solution is general, efficient, and generates high quality test cases. Let us consider a couple of examples:

• *“Validating dates”* (P29448). This problem asks the student to tell whether several triples of integers are valid dates in the Gregorian calendar. Its public test cases are shown in Table 1(a). This is an easy problem, but the thorough case analysis for the validation on the number of days for each month (including coping with leap years) can be tricky for beginners. This problem has 129458 private test cases. It has received about 7000 submissions, from which about 3700 are bad. The distillation process took about 35 seconds and produced a distilled set with 23 test cases, shown in Table 1(b). As could be expected, those test cases feature limiting values for the number of the day (such as 0 and 32) and stress years (such as 2000) for which the leap year rule may be wrongly coded.

• *“Brackets and parentheses”* (P96529). This problem asks the student to tell whether words made up of brackets and parentheses close correctly. Its public test cases are shown in Table 2(a). In this case, the distillation process produced 11 relevant test cases out of the 1766 private test cases. However, the distilled set had a big issue: its total size was quite large (18360 characters). Using weights, we obtained 13 test cases, whose total size is 1254 characters, and where three test cases still had more than 350 characters each. By relaxing the ILP to catch 90% of the bad submissions, we obtained a workable distilled set with just 10 inputs and 63 characters; see Table 2(b).

Conclusions. The distillation process extracts in an automatic way a small, concise and relevant subset of private test cases that captures the difficulty of the problem for many users. Revealing these distilled test cases to beginners helps not discouraging them and increases their own ability to identify and address edge

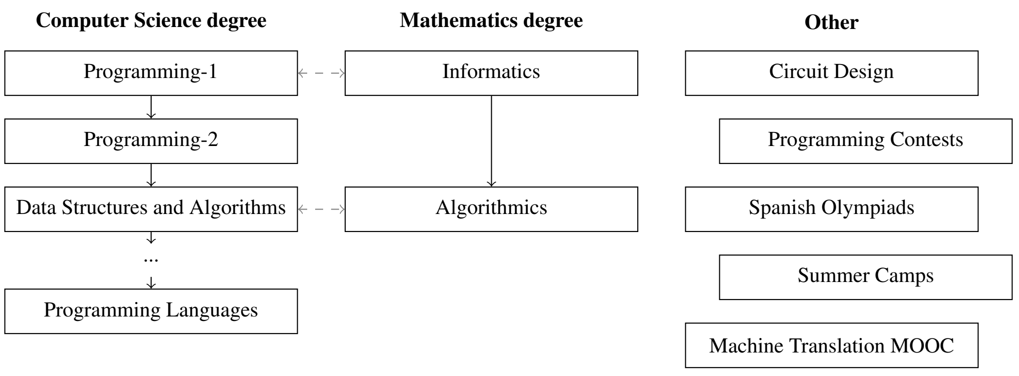


Figure 6: Courses at UPC using Jutge.org.

cases. Taking into account the amount of effort spent by problem setters in creating huge and exhaustive private test cases, it is a bit deceiving to see *a posteriori* that only few of these test cases are, in fact, relevant.

5 Experiences of courses using Jutge.org

In this section we detail how Jutge.org has been used in several courses at UPC. In particular, we focus on the Computer Science and Mathematics Schools: the Facultat d'Informàtica de Barcelona (FIB) and the Facultat de Matemàtiques i Estadística (FME).

Within the Computer Science degree at FIB, the judge is used in Programming-1 (PRO1), Programming-2 (PRO2) and Data Structures and Algorithms (EDA), which correspond to its three first compulsory courses on programming and algorithmics, and also in Programming Languages (PL), which is elective. Within the Mathematics degree at FME, the judge is used in Informatics and Algorithmics, which are both compulsory courses. These two courses are somehow similar to PRO1 and EDA respectively, so we shall not elaborate on them. Additionally, Judge.org has been used in other transversal activities at UPC, including Circuit Design, Programming Contests and the Spanish Olympiads in Informatics. A MOOC course on Machine Translation has also used Jutge.org [8]. The overall structure of these courses is shown in Figure 6.

5.1 Programming-1

PRO1 is a first-year, 7.2 ECTS-credits, compulsory, one semester programming course taught at FIB that introduces computer programming. This course is taught by 15 faculty members to approximately 450 students per semester.

In September 2006, the PRO1 course was redesigned and its fundamental objective was set "to ensure students passing PRO1 to be able to confidently code correct, readable programs to solve problems of elementary difficulty." The course syllabus is similar to that of other analogous courses: It includes basic elements of programming (types, variables and control structures), functions and procedures, recursion, one and two-dimensional arrays, strings, and aggregate types. Some well-known algorithms are also introduced, such as Euclid's algorithm, the sieve of Eratosthenes, binary search, basic sorting methods, and merge-sort. The programming language of choice is (a small subset of) C++.

For the sake of acquiring the required programming skills, students were encouraged to solve as many problems as possible. During laboratory sessions, teachers helped students. However, it was expected that students should work most of the time on their own, using a complete and well-organized problem collection. To apply this methodology, the PRO1 programming judge —which would later evolve into Jutge.org— offered up to 300 programming problems.

The online judge was also used in exams, to reject wrong solutions while giving the chance to the student to identify and correct mistakes (multiple submissions were allowed). After the exam, only the correct solutions were additionally graded by human instructors, mainly to assess their adequacy to general quality criteria.

The results of applying this methodology during the first four semesters were not as successful as expected: Although there was a clear correlation between the amount of problems the students solved and the grades they obtained (see Figures 7 and 8), the data collected on the judge showed that the effort that most

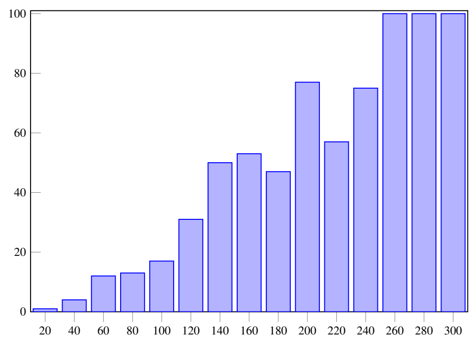


Figure 7: % of students passing the course w.r.t. the number of problems solved during the course. The trend is that the more solved problems, the higher the probability of passing.

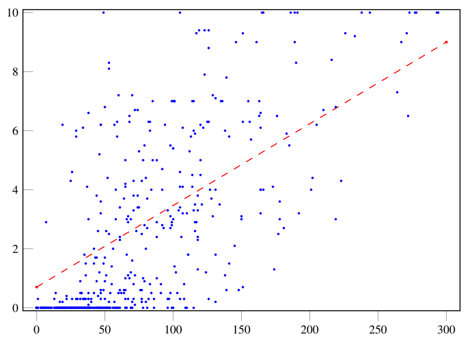


Figure 8: Final grade (between 0 and 10) vs the number of solved problems with a linear regression.

students devoted to the course was way less than the workload required according to its ECTS-credits. As a sad but logical result, a high number of those students failed PRO1. More details can be found in [16].

As a consequence, during the subsequent years, several adjustments were progressively applied to the course with the intention to turn the situation around. Basically, these modifications involved increasing the weight of continuous assessment as a way to boost students' work, reintroducing some partial exams on paper, and relaxing the condition to pass the private data sets to have a problem exam graded. However, it appears that despite these adjustments, the rate of success of the PRO1 course has not yet increased as much as desired. Figure 9 shows the evolution of the success rate from 2007 up to 2015, which roughly rose from 20% to 40%.

The opinion of students with respect to the usage of the judge in PRO1 has evolved since its introduction. Whereas students in 2006 officially protested to the Dean by claiming that "*the judge demotivates and thwarts students*" and that "*exams in front of the computer are unfair*" [12], current students claim the opposite: "*programming exams on paper are unfair!*". Figure 10 shows the answers of students to an opinion survey conducted in December 2015 to get a clearer view on the matter. The range of answers is from 1 (strongly disagree) to 5 (strongly agree). From the survey, we can see that a huge majority of students accepts that the judge system helps them to improve their programming skills, that they admit not using it as much as they should, and that they would object to the course without it. The survey also demystifies that the judge highly thwarts students: the average response to this question is 3.6, and the dual question that asks whether an "Accepted" verdict motivates them has an average response of 4.5. Likewise, the poll results indicate that the additional stress of exams in front of the computer with respect to traditional paper exams is not very significant (one third of the answers are 1 or 2, one third are 3 and, another third 4 or 5).

Furthermore, all the coordinators and teachers of the PRO1 course acknowledge the usefulness of the online judge. As a learning tool, it is a handy device to learn programming, since it contains an extensive

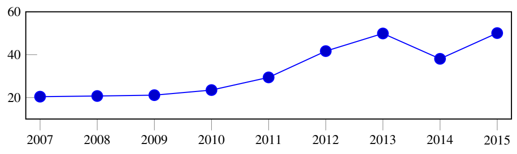


Figure 9: Percentage of students passing the course by year.

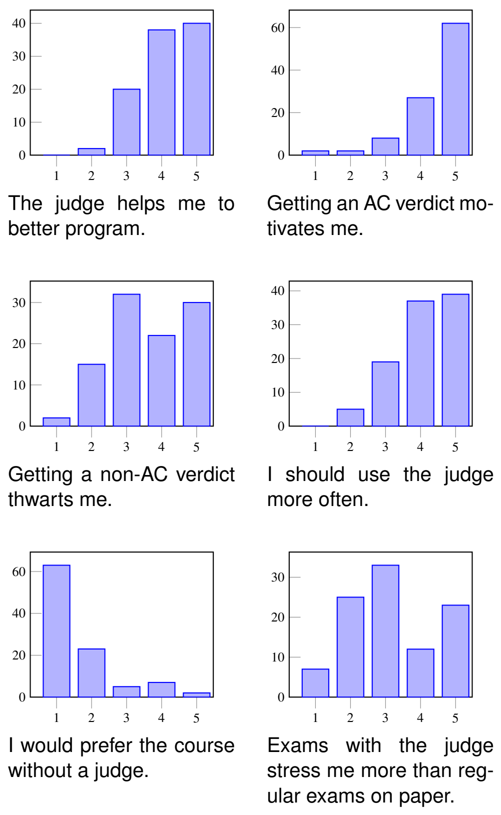


Figure 10: Results of the survey about the judge on PRO1 (1: Strongly disagree → 5: Strongly agree). Values shown as percentages.

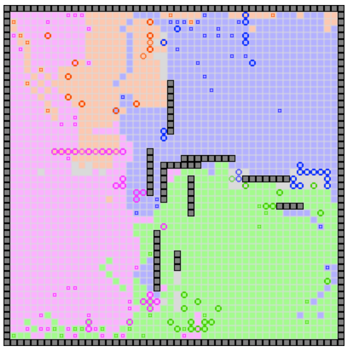
list of exercises, automatically evaluates the solutions submitted by students and gives an immediate verdict about their correctness. As a tool for helping on the evaluation process, the judge makes it possible to grade students precisely in the competence that they must acquire, that is, solving simple programming problems. This includes reading and understanding the statement of a problem, thinking about an algorithm solving it, coding it, checking that it works, and making the appropriate changes, if needed. As a tool for monitoring students progress, the online judge provides a source of objective, untainted data from a hitherto unknown depth at UPC.

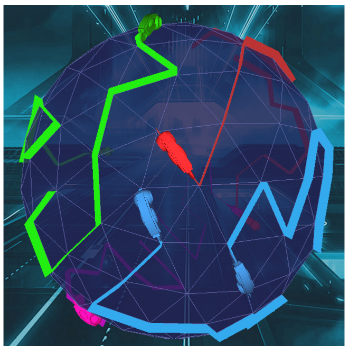
5.2 Programming-2

The PRO2 course is a natural continuation of PRO1. With 7.5 ECTS-credits, it introduces modular and object-oriented design; presents new data structures such as stacks, queues, lists and trees paired with their implementations with pointers, and emphasizes the reasoning about the correctness of solutions and the improvement of inefficient solutions. The collection of problems in Jutge.org for PRO2 expands the one in PRO1 with problems that ask to code (or expand) a set of C++ classes in different files with a given interface, and ensuring their compilation with a makefile.

5.3 Data Structures and Algorithms

The Data Structures and Algorithms (EDA) course consists of 6 ECTS-credits. It combines basic theoretical aspects of algorithm design and analysis together with several programming features. The course starts by introducing algorithm analysis, together with its basic mathematical tools. Then, these bases are used to analyze several implementations of classic algorithms and data structures (sorting and searching, divide and conquer, backtracking, dynamic programming, hashing, balanced binary trees, graphs, shortest paths, among others). Within this course, the online judge is used as the basis of the programming assignments and exams in a similar way as in the PRO1 and PRO2 courses previously described. Interestingly, in the EDA course, Jutge.org is also used to manage a new and motivating activity: the *EDA Tournament*.


Battle Royale: Each player controls several knights and peasants to colonize cells and fight adversary knights.


Tron: Each player controls some light cycles that should never crash against the trails that light cycles leave behind.

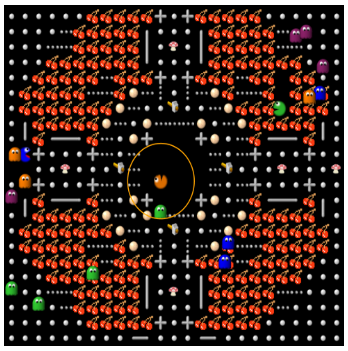
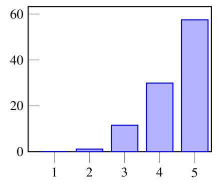
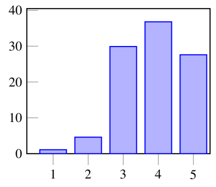

PacMan: Each player controls one pacman and four ghosts.

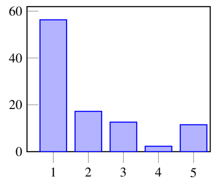
Figure 11: View of some games.



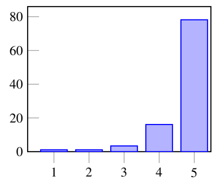
I enjoyed the game.



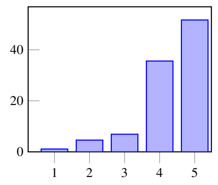
Beating the dummy player was easy.



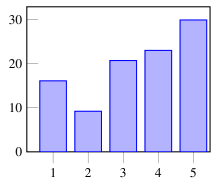
I would rather spend my time studying theory than programming my player.



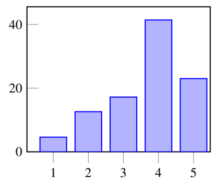
The game motivates me more than traditional programming projects.



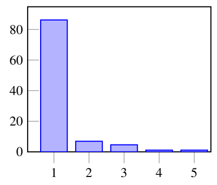
Competing against my mates motivates me.



I improved my player after winning the dummy.



The documentation of the game is right to me.



I would rather prefer this course without game.

Figure 12: Results of the survey for the PacMan game (1: Strongly disagree → 5: Strongly agree). Values shown as percentages.

The EDA Tournament is a programming project where students compete among themselves by designing and implementing strategies that control the movements of several agents according to some rules of a game. There is no human interaction during a match; this tournament is played by programs. Unlike most exercises of this and previous courses, it is an "open" project: it is not about solving a specific problem, but about designing a plan, studying the opponent's strategies and, if required, adjusting or rebuilding one's own coded plan. It emphasizes the algorithmic aspects taught in the course, requires careful programming, encourages good coding habits and, moreover, is fun, which is particularity important because it intends to overcome the poor motivation of many students for the theoretical contents of the course.

Tournament. We briefly comment on the three main stages in the development of the EDA Tournament: qualifying, tournament and grand final. More details can be found in [14]. In order to qualify for the tournament —thus obtaining a passing grade in this project— students have three weeks to individually work on their own strategy and come up with a player that consistently beats the *dummy*: a basic player programmed by the course lecturers. This is a mandatory requirement to ensure a minimal coding effort by every student.

Afterwards, the tournament itself begins, with all the qualified players. It lasts for around two weeks, and consists of the necessary amount of rounds to eliminate all players but 16. A round consists of several matches. Every match is publicly visible and involves (usually) four programs. The worst player of each round is eliminated. The longer a student stays in the game, the higher is her grade.

When there are only 16 players left, a grand final takes place in the conference room. The surviving players are grouped into four rounds, and the winner of each one competes in a last round to determine the absolute champion. In the ceremony, the programmers of the best players are invited to give a public short speech about their strategies. See https://jutge.org/flm/jocs.mp4 for a video of the event.

Several games have been released, trying to make them as attractive and enjoyable as possible. All the games feature a board where several agents controlled by four different players interact for several rounds to get a final score. The board is organized as a collection of cells that induce a graph. At each round, the movements of the agents are governed by the players' strategies and invoked through the game API. Every player decides its movements for the next round independently of the other players, and there is a randomization process by which possible collisions are resolved; see Figure 11 for some snapshots.

Analysis. The EDA Tournament activity has been implemented as described above for the last nine semesters. It has a large percentage of participation (always more than 90%) and almost all students who tried were able to qualify a player into the tournament, consequently passing the project. Moreover, since the results in the tournament provide extra points for the course's grade, typically around 20% of students pass the whole course thanks to this bonus.

A survey was conducted to the EDA students in December 2013. During that course, when the PacMan game was played, 143 students submitted a total of 2929 programs, 749 of them beating the dummy player. A total number of 7761 matches were disputed. Figure 12 shows the answers of the students. The range of answers was from 1 (strongly disagree) to 5 (strongly agree). From the survey, we can infer that almost all students enjoyed the game activity, that they did not find too many problems in beating the dummy player, and that they preferred it over standard assignments. Also, since most of them liked to compete against each other, this activity was fun, attractive and motivating.

5.4 Programming Languages

Functional programming is introduced as part of the Programming Languages elective course (6 ECTS-credits). It introduces new programming constructions such as anonymous functions, higher-order functions, richer and stronger type systems, and monads. The language of choice is Haskell [33]. This language is ideal for introducing functional programming, as it provides all these constructions plus lazy evaluation. Being a pure functional language, it prevents students from using any construction based on having side effects, and increases the skill of students to define functions by recursion or by composition of existing functions.

Since functional programming represents only half of the Programming Languages course, it is specially important to provide students with good tools for training on their own since the very beginning of the course.

Along the course, different features of functional programming are tought. Therefore, each exercise specifies what kind of constructions are allowed to solve it. Although students are responsible for doing their job properly, it turned out that (as will be shown below), in a relevant amount of cases the submissions do not meet the given requirements. For this reason, a *code inspector* was recently developed and integrated into Judge.org.

Table 3: Results of the evaluation with and without the Haskell inspector.

	P93632	P31745	P93588	P90677
ACs without inspector	41.4%	54.3%	77.5%	31.1%
ACs with inspector	30.5%	32.3%	72.8%	16.8%
Submissions with inspector diagnostics	28.4%	54.7%	9.5%	37.4%
Average score without inspector	70.7	79.5	91.0	75.4
Average score with inspector	58.8	43.3	85.7	63.1

The Haskell code inspector. The inspector parses the Haskell submission and produces an *abstract syntax tree* (AST) which is analyzed to check if all the requirements hold (test cases are still used to judge correctness). This tool is written in Haskell itself (using haskell-src-exts) and checks the following requirements:

• *Use of recursive definitions.* The use of recursion may be forbidden in some problems, whereas it may be mandatory in others. For instance, a problem could ask for the factorial function expressed as

$$fact\ \ n\ =\ foldl\ \ (*)\ \ 1\ \ [1..\, n]$$

rather than recursively as

$$fact\ \ 0\ =\ 1$$
$$fact\ \ n\ = n\ *\ fact\ \ (n{-}1)$$

Based on the AST, the inspector can detect the presence or absence of recursive definitions (direct, indirect or mutual).

• *Use of list comprehensions.* Some problems are intended to teach list comprehensions. For instance, in Haskell, Pythagorean triplets could be defined using a list comprehension such as

$$pythag\ \ n\ =\ [(x,y,z)\ \mid\ x{\leftarrow}[1..n],\ y{\leftarrow}[x..n],$$
$$z{\leftarrow}[y..n],\ x\hat{\ }2\ +y\hat{\ }2\ ==z\hat{\ }2]$$

As these problems can be solved as well using recursion or predefined higher-order functions, the inspector checks for the use of list comprehensions.

• *Use of predefined functions.* Some problems ask for functions already in the Haskell standard library (for instance, implementing the ubiquitous *map* function). To rule out solutions such as *myMap* = *map* or variations thereof, the inspector analyzes the possible invocation of such functions (not just its appearance).

Results. During the first four semesters of using Jutge.org in the Programming Languages courses, the Haskell problems were corrected only via test cases, without the code inspector. Consequently, we possess a set of submissions big enough to analyze how the inspector discriminates non-compliant submissions. The results presented here are for some paradigmatic problems, namely "Use of higher-order functions 1" (P93632), "Use of higher-order functions 2" (P31745), "Use of comprehension lists" (P93588), and "Definition of higher-order functions 1" (P90677).

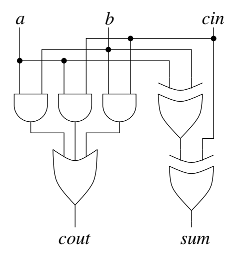
Table 3 shows, for each of the selected problems, the number of AC verdicts with and without the inspector, the number of submissions that received some inspector diagnostic, and the average problem score (over 100) with and without the inspector. Note that, for all problems, there is a significant portion of submissions that do not follow the requirements of the statement. Corresponding, the average score decreases about a 20% when using the code inspector.

Taking into account that the requirements are intended to help training, it is surprising that so many students tricked themselves by submitting functionally correct but non-compliant solutions (even when they knew that their exam solutions would be manually graded). Fortunately, since the integration of the Haskell inspector in the judge, the number of such non-compliant solutions has dropped abruptly: in the last two semesters inspector diagnostics were only issued for a 6% of the Haskell submissions.

5.5 Circuit Design

Somehow surprisingly, online programming judges can also be useful in courses on logic circuit design. Indeed, the times of schematic captures in circuit design have evolved towards a more reliable and productive methodology based on Hardware Description Languages (HDLs) [34]. Within this approach, designers describe their circuits with HDLs and implement them using CAD tools that automatically generate logic netlists, which are later transformed into layouts using physical synthesis tools. Consequently, problems on circuit design can also be understood as programming problems where circuits are described using an HDL, such as Verilog.

Jutge.org uses formal verification techniques and tools to prove the correctness of logic circuits. For that, the problem setter is asked to design a reference circuit that is assumed to be correct. Correctness *(a) Schematics*



(b) Verilog logic gates

```
module full_adder(a,b,cin,sum,cout);
  input a, b, cin;
  output sum, cout;
  wire x1, x2, x3;
  xor (sum, a, b, cin);
  and (x1, a, b);
  and (x2, a, cin);
  and (x3, b, cin);
  or (cout, x1, x2, x3);
endmodule
```

(c) Verilog logic equations

```
module full_adder(a,b,cin,sum,cout);
  input a, b, cin;
  output sum, cout;
  assign sum = a ^ b ^ cin;
  assign cout = a & b | cin & (a | b);
endmodule
```

Figure 13: Possible implementations for the full-adder.

(a) Reference solution

```
module mod3_counter (cnt, clk, rst);
 input clk, rst;
 output reg [1:0] cnt;
 always @(posedge clk)
  if (rst | (cnt == 2)) cnt ⇐ 0;
  else cnt ⇐ cnt + 1;
endmodule
```

(b) Wrong candidate solution

```
module mod3_counter(cnt, clk, rst);
 input clk, rst;
 output [1:0] cnt;
 wire [1:0] nxt;
 wire cout;
 adder #(2) _add_ (cnt, 1, 0, nxt, cout);
 register #(2) _reg_ (nxt, cnt, clk, rst);
endmodule
```

(c) Wrong solution report

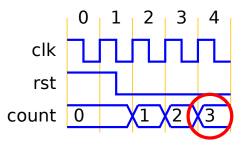


Figure 14: Mod-3 counter: reference and wrong solutions and waveform report for the wrong solution.

is proved by checking the equivalence of the reference and the candidate circuits. Under the hood (more details can be found in [11]), there is a formal verification engine, NuSMV [7], that performs equivalence checking. The technology behind NuSMV is based on the theory and algorithms for sequential equivalence checking that were proposed and developed during the nineties [29]. In these algorithms, Binary Decision Diagrams are often used to represent sets of states and symbolically compute the set of reachable states of a sequential system. For the set of reachable states it is proved that two systems generate the same output when they receive the same input stimuli.

The main limitation of these techniques arises during the verification of sequential circuits. The complexity of the verification procedure is mostly dominated by the number of state signals. Still sequential circuits with few dozens of flip-flops can be handled in an affordable computation time.

This approach offers two benefits: For students, the judge can provide a counterexample trace for incorrect circuits that describes the input stimuli required to lead the circuit from the initial state to the failure state; see Figure 14(c). For problem setters, formal verification involves less work to create problems on circuits than for general programming problems: in the former case no test cases must be written.

Thanks to this innovative correction engine, Jutge.org offers an introductory course on digital circuit design that includes the following topics:

• *Combinational circuits:* Simple controllers, multiplexers, voting systems, priority encoders, properties on numbers, etc.

• *Arithmetic circuits:* The topic starts with 1-bit adders/comparators, and progresses to n-bit adder/subtractors, comparators, incrementers and small ALUs.

• *Sequential circuits:* The topic includes various up/down counters, sequence recognizers, simple control circuits (e.g., for a traffic-light) and some sequential arithmetic (e.g., a generator of Fibonacci series, or a calculator of the gcd using Euclid's algorithm).

• *A simple CPU:* By combining a few components that must be designed as separate problems, students can create a simple 8-bit CPU with small instruction and data memories.

A sample combinational circuit. As a first example, consider problem X12983 (the second problem in the Arithmetic Circuits list), which asks for the implementation of a full adder. Figure 13 shows several possible solutions: with schematics, in structural form (using logic gates) or in dataflow form (using logic equations). Interestingly, when a student submits an incorrect implementation (either in structural form or in dataflow form) the judge offers a set of values for the input stimuli that causes such a wrong answer.

A sample sequential circuit. As a second example, problem X05944 proposes the design of a (mod 3)-counter that generates the sequence $0, 1, 2, 0, 1, 2, \ldots$ The reference solution may implement the counter using a behavioral conditional statement, as in Figure 14(a). In the case that a student misunderstands the specification of the problem and implements a conventional (mod 4)-counter with a 2-bit adder and a register as in Figure 14(b), the judge would report a *"Wrong answer"* verdict and, in this case, would also provide *the shortest trace* from the initial state to an erroneous state. This trace, reported in graphical form, indicates with a circle the unexpected value according to the specification of the problem; see Figure 14(c).

Conclusions. While using formal verification to fully prove software correctness is still an utopia, it is becoming increasingly practical in hardware design. As we have shown, current technology on formal verification allows us to build an introductory course on digital circuit design around a collection of problems that are automatically corrected by our judge with no need for explicit test cases and providing suitable feedback for wrong submissions. As a similar approach has demonstrated [2], students using the platform were a 20 percent more likely to pass the course than students who did not use it.

6 Comparison to related work

Without intending to be exhaustive, in this section we compare Jutge.org to various types of systems that share similar characteristics. To do so, we classify them into six categories: massive open courses[2], interactive programming educational platforms[3], judges and automated assessment systems[4], AI-supported systems[5], programming games platforms[6] and circuit design platforms.

Massive online courses. In its current state, Jutge.org cannot be considered a MOOC platform, as it is intended to complement presence-based courses and does not feature multimedia lessons. That said, using Jutge.org's back-end to assess programing problems in a MOOC is easily doable, as shown in [8].

Whereas assessment in most MOOCs is usually based on peer review or on multiple choice quizzes, Jutge.org features automatic correction of programming assignments. Only few algorithmic MOOCs start to offer such assessment; e.g., the Algorithms courses and the Machine Learning course at Coursera. Yet, these courses only feature few exercises (one or two for each topic among a dozen of topics) and without choice of PLs.

As offering personal advice is not possible in MOOCs, automatically providing useful feedback becomes a necessity. Jutge.org copes with this challenge through data mining past wrong solutions, an approach further developed in Codewebs [28] on a wider setting.

Interactive programming educational platforms. Jutge.org shares some characteristics with interactive programming educational platforms: problem lists, automated-testing, static program analysis and achievements. However, Jutge.org is geared towards a more mature audience (high school and university students) than most platforms that are, in general, more oriented toward kids (such such as Code, TKP, Scratch), offers many more PLs (sites such as CodingBat just offer about six PLs) and places more emphasis on algorithmic topics than platforms that focus on building games, web sites or apps (such as Codecademy, edX and Khan Academy). Furthermore, whereas most systems in this category follow a freemium pricing strategy (e.g., Code Avengers, Code School, Udemy), Jutge.org is free to use.

Judges and automated assessment systems. Jutge.org belongs to the category of online judges which strongly focus on algorithmic problem solving. Nevertheless, the main purpose of such systems is to help users to train themselves for programming contests (e.g., UVa or Codeforces), or to help companies recruiting programmers (TopCoder). Even though experiences with competitive learning have been reported [22, 30], overall these sites do not provide much feedback and do not provide help to instructors.

As Jutge.org, the URI Online Judge [3] is another judge that has been designed with an educational aim as it provides an Academic Module that enables instructors to manage disciplines and lists of exercises on specific programming topics. It contains about 1,400 problems but only supports four PLs. Its feedback is binary and, strangely, does not take public test cases into account. It also does not respect privacy, as any user can see the activity of other users.

Reference [19] provides an extensive survey on existing automated assessment systems for education and identifies their main features. According to that taxonomy, Jutge.org correction is by output comparison (dynamic) and syntactic analysis (static) while other systems do mostly dynamic correction only.

AI-supported systems. As already pointed out, Jutge.org has not been designed to be a programming tutor, but a powerful tool assisting human tutors. Nevertheless, the feedback provided by Jutge.org (see Sections 3

[2] Hack.pledge, Khan Academy, edX, Coursera, MIT OpenCourseWare or Stanford Online...

[3] Code, Code Avengers, CodingBat, Codecademy, Code School, Free Code Camp or Udemy...

[4] CodeChef, Sphere, Timus, TopCoder, URI or UVa...

[5] PHP-ITS, Ask-Elle...

[6] Robocode, Battlecode, AI Challenge...

and 4) supports self-learning of programming up to a certain extent. In contrast, various feedback-based AI-supported tutoring approaches exist [4,15,24,32]. Most of them use AI techniques to give examples, provide simulations, promote incremental or stage by stage programming and facilitate dialogues between students and human tutors. Undoubtedly, Jutge.org and any automated educational system would benefit from such features (which for sure would provide more insight and support to students in a difficult discipline such as programming). However, the already available AI-supported systems have some drawbacks. Indeed, many such existing systems are language-specific and only few programming languages are covered; although many involve sets of exercises (or problems), those sets are small and usually not easy to enlarge; some others are highly dependent on a human tutor's expertise and feed-back, and mostly support only individual learning (not peer or collaborative).

Programming games platforms. Concerning the tournament activity for the EDA course, let us note that its development was inspired by the ICPC-Challenges. To our knowledge, with the exception of dedicated platforms, no other online judge integrates programming games. Wider implications of gamification in learning can be found, for instance, in [5, 18]. Historically, programming games seem to go back to 1961, when the Darwin game was created [1].

Circuits. With respect to circuits, [9] describes a system where students can submit their design with LogicFlash [10], a graphical tool for schematic capture. Since [9] uses explicit testing, it was only feasible for combinational circuits and very small arithmetic/sequential circuits. In constrast, Jutge.org can handle circuits of medium complexity [11]. A project with a similar setting (using formal verification as a back-end) but different methodology (rely on a graphical language instead of Verilog, a widely used HDL in industry) has been presented recently [2].

7 Conclusion

Jutge.org is an online programming judge built with an educational aim. For the past ten years, this tool has played an essential role in improving the teaching in UPC's programming and algorithms courses.

Since designing algorithms and programming them requires a set of cognitive processes that naturally develop through practice, our judge helps students acquire the required skills by progressively solving problems and learning from their mistakes, without fully placing the burden of correction on their instructors. Featuring a rich and well organized problem repository, Jutge.org provides students with instant feedback, and supplies instructors with all necessary tools to manage their courses and assist their students. Additionally, Jutge.org is free, easy-to-use, mature, reliable, robust, and long-lasting.

The different experiences with Jutge.org that we have reported include several presence-based courses that involved thousands of students of our Computer Science and Mathematics Schools. For these courses, the judge has been a catalyst to modernize their contents and presentation, emphasizing the learning-by-doing approach. In contrast with the old courses where programming was vastly relegated to pseudo-code written on paper, in the current courses using the judge, with practical exams in front of the computer, the student has a more active role and truly internalizes that programming is a serious matter. Indeed, most students consider the judge a great help and would disfavor not using it.

Jutge.org not only relies on well-known techniques such as test-driven evaluation, code metrics and gamification but also integrates research and innovation: The introduction of formal verification methods and static code analysis provides a novel paradigm that contributes to broaden the use of such tools in e-learning. Similarly, new proposals such as data-mining incorrect solutions so as to distill the most relevant test cases to improve feedback emanate from this project.

Further development for Jutge.org is under work: On the one hand, an inspector for C++ code is being implemented (as C++ is the most used PL among our users). The goal is to provide similar features to the Haskell inspector, and to extend them to detect common pitfalls that novice programmers often succumb to (e.g., the inspector could report that some `while` loop would be written more naturally with a `for` loop or vice-versa).

References

[1] $\aleph_0$. Computer recreations: Darwin. *Software-Practice and Experience*, 2:93–96, 1972.

[2] D. Bañeres, R. Clarisó, J. Jorba, and M. Serra. Experiences in digital circuit design courses: A self-study platform for learning support. *IEEE Transactions on Learning Technologies*, 7(4):360–374, 2014.

[3] J. L. Bez, N. A. Tonin, and P. R. Rodegheri. URI online judge academic. In *2014 9th Int. Conf. on Computer Science Education*, pages 149–152, 2014.

[4] C. Butz, S. Hua, and R. Maguire. A web-based intelligent tutoring system for computer programming. In *Conf. on Web Intelligence*, pages 159–165. IEEE, 2004.

[5] N. E. Cagiltay, E. Ozcelik, and N. Sahin Ozcelik. The effect of competition on learning in games. *Computers & Education*, 87(0):35–41, 2015.

[6] B. Cheang, A. Kurnia, A. Lim, and W. Oon. On automated grading of programming assignments in an academic institution. *Computers & Education*, 41(2):121–131, 2003.

[7] A. Cimatti et al. NuSMV2: An OpenSource Tool for Symbolic Model Checking. In *Proc. Int. Conf. on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*. Springer, 2002.

[8] M. Ruiz Costa-Jussà, L. Formiga, O. Torrillas, J. Petit, and J. A. Rodríguez. A MOOC on approaches to machine translation. *IRRODL*, 16(6), 2015.

[9] M. Damm, F. Bauer, and G. Zucker. Solving Digital Logic Assignments with Automatic Verification in SCORM Modules. In *Int. Conf. on Interactive Computer-Aided Learning*, pages 359–363, 2009.

[10] M. Damm, B. Klauer, and K. Waldschmidt. LogiFlash - A Flash-based Logic-Simulator for Educational Purposes. In *World Conference on Educational Multimedia, Hypermedia and Telecommunications*, pages 748–750, 2003.

[11] J. de San Pedro, J. Carmona, J. Cortadella, and J. Petit. Integrating formal verification in an on-line judge for e-learning digital circuit design. In *43rd ACM Technical Symposium on Computer Science Education*, pages 451–456. ACM, 2012.

[12] Delegació d'Alumnes de la FIB. Carta al respecte de l'avaluació de l'assignatura P1 a la FIB, 2008.

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

[14] A. Duch, J. Petit, E. Rodríguez-Carbonell, and S. Roura. Fun in CS2. In *5th Int. Conf. on Computer Supported Education*, pages 437–442. SCITEPRESS, 2013.

[15] A. Gerdes, B. Heeren, J. Jeuring, and L. van Binsbergen. Ask-elle: an adaptable programming tutor for Haskell giving automated feedback. *Journal of Artificial Intelligence in Education*, pages 1–36, 2016.

[16] O. Giménez, J. Petit, and S. Roura. Programació 1: A pure problem-oriented approach for a CS1 course. In *Informatics Education Europe IV*, pages 185–192, 2009.

[17] M.H. Halstead. *Elements of Software Science*. Elsevier Science, 1977.

[18] M. D. Hanus and J. Fox. Assessing the effects of gamification in the classroom. *Computers & Education*, 80:152–161, 2015.

[19] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä. Review of recent systems for automatic assessment of programming assignments. *10th Koli Calling Int. Conf. on Computing Education Research*, pages 86–93, 2010.

[20] D. Jackson and M. Usher. Grading student programs using ASSYST. *SIGCSE Bull.*, 29(1):335–339, 1997.

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

[22] A. Kosowski, M. Malafiejski, and T. Noinski. Application of an online judge & contester system in academic tuition. In *ICWL'07*, pages 343–354, 2007.

[23] A. Kurnia, A. Lim, and B. Cheang. Online judge. *Computers & Education*, pages 299–315, 2001.

[24] N. Le, S. Strickroth, S. Gross, and N. Pinkwart. A review of AI-supported tutoring approaches for learning programming. In *Advanced Computational Methods for Knowledge Engineering*, pages 267–279. Springer, 2013.

[25] A. Mani, D. Venkataramani, J. Petit, and S. Roura. Better feedback for educational online judges. In *6th Int. Conf. on Computer Supported Education*, pages 176–183. SCITEPRESS, 2014.

[26] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, 1976.

[27] N. Mohamed, R. Sulaiman, and W. Endut. The use of cyclomatic complexity metrics in programming performance's assessment. *Procedia - Social and Behavioral Sciences*, 90:497–503, 2013.

[28] A. Nguyen, C. Piech, J. Huang, and L. Guibas. Codewebs: scalable homework search for massive open online programming courses. *Procs. of the 23rd int. conf. on WWW*, pages 491–502, 2014.

[29] C. Pixley. A theory and implementation of sequential hardware equivalence. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(12):1469–1478, Dec 1992.

[30] M. A. Revilla, S. Manzoor, and R. Liu. Competitive learning in informatics: The UVa online judge experience. *Olympiads in Informatics*, 2:131–148, 2008.

[31] M. Shepperd. A critique of cyclomatic complexity as a software metric. *Softw. Eng. J.*, 3(2):30–36, 1988.

[32] R. Singh, S. Gulwani, and A. Solar-Lezama. Automated feedback generation for introductory programming assignments. *SIGPLAN Not.*, 48(6):15–26, 2013.

[33] S. Thompson. *Haskell: the craft of functional programming*. Addison-Wesley, third edition, 2011.

[34] Z. Vranesic and S. Brown. Use of HDLs in teaching of computer hardware courses. In *Workshop on Computer Architecture Education*, 2003.

[35] D. M. Zimmerman, J. R. Kiniry, and F. Fairmichael. Toward instant gradeification. In *24th IEEE-CS Conference on Software Engineering Education and Training*, pages 406–410, 2011.