PROGRAM ANALYSIS AND EVALUATION USING QUIMERA

Daniela Fonte[1], Ismael Vilas Boas[1], Daniela da Cruz[1], Alda Lopes Gancarski[2] and Pedro Rangel Henriques[1]

[1]*Department of Informatics, University of Minho, Braga, Portugal*

[2]*Institut Telecom, Telecom SudParis, CNRS UMR Samovar, 9 rue Charles Fourrier, 91011 Evry, France*

{*danielafonte,danieladacruz,prh*}*@di.uminho.pt, ismael.vb@gmail.com,alda.gancarski@it-sudparis.eu*

Keywords: Automatic Grading System, Contest Management, Program Evaluation, Automatic Judgement.

Abstract: During last years, a new challenge rose up inside the programming communities: the *programming contests*. Programming contests can vary slightly in the rules but all of them are intended to assess the competitor skills concerning the ability to solve problems using a computer. These contests raise up three kind of challenges: to create a nice problem statement (for the members of the scientific committee); to solve the problem in a good way (for the programmers); to find a fair way to assess the results (for the judges). This paper presents a web-based application, QUIMERA intended to be a full programming-contest management system, as well as an automatic judge. Besides the traditional dynamic approach for program evaluation, QUIMERA still provides static analysis of the program for a more fine assessment of solutions. Static analysis takes profit from the technology developed for compilers and language-based tools and is supported by source code analysis and software metrics.

1 INTRODUCTION

An important aspect when learning programming languages is the ability to solve practical problems. This ability can be easily stimulated with *competition*, like the one present in a *programming contest*. According to experts on the field, the challenge associated with competitive environments provides a meaningful way to learn and easily acquire practical skills.

In a typical programming contest, competitors participate in teams to solve a set of problems. For each problem, the team submits the source code of the program developed to solve the problem. Many well known programming contests in the world — like ACM-ICPC[1] or IOI[2] — are based on the automatic grading of the solutions proposed. It means that the submitted code will be immediately evaluated by an *automatic grading system*. The evaluation normally involves tasks like running the program over a set of predefined tests (actually a set of input data vectors), and comparing each result (the actual output produced by the submitted code) against the expected output value. Time and memory space consumptions are usually measured during execution and are taken into account for this *dynamic grading approach*. This process is typically complemented by the action of a human judge, who takes the final grade decision according to the specific rules for each contest (Leal 2003).

The research on programs that are capable to automatically grade source code is not a recent topic. In 1965, Forsythe and Wirth (Forsythe & Wirth 1965) introduced a system that follows the fundamental principle of the modern grading systems, validating the submitted solutions with a set of tests (pairs of input-output vectors). With the evolution of computers, grading systems increased in complexity, diversifying the tests made to the subject programs and introducing tools for monitoring the grading process (Leal & Moreira 1999).

Nowadays grading systems can be distinguished according to the type of the source code verification. This verification can be done employing two different techniques: static and dynamic. The second one focuses on the execution of the program against a set of predefined tests (the traditional one, already described above). The first one, more recent, takes profit from the technology developed for compilers and language-based tools and is supported by source code analysis and software metrics. Our proposal is

[1]International Collegiate Programming Contest http://cm.baylor.edu/welcome.icpc

[2]International Olympiad in Informatics http://www.ioinformatics.org/index.shtml

to combine both approaches and provide a system in which the final grade is a combination of the information obtained from both analyzers.

So, in this paper we present a web-based application, named QUIMERA, that extends the traditional dynamic grading systems with static analysis aimed at the automatic evaluation and ranking of programming exercises in competitive learning or programming contest environments. QUIMERA provides a full contest management system, as well as an automatic judgement procedure.

The paper is organized in 6 sections. Besides the Introduction and Conclusion, Section 2 introduces the basic concepts and similar tools; Section 3 gives an overview of QUIMERA, presenting its architecture and main features; Section 4 provides more details about each feature, illustrating QUIMERA functionalities; and Section 5 presents the technical details underlying the system implementation.

2 RELATED WORK

As previously referred, the approaches to develop *Automatic grading systems* can be distinguished between *static* and *dynamic* (Danić, Radošević & Orehovački 2011).

Static approaches include systems that check the submitted program against a provided scheme, to find the degree of similarity relatively to a set of characteristics. In this category we have, for example, the Web-based Automatic Grader (Zamin, Mustapha, Sugathan, Mehat & Anuar 2006), which evaluates programming exercises written in Visual Basic, C or Java languages. One of its disadvantages is that it can not be used for testing the correctness of programs which contain input and output operations (Danić et al. 2011). Another important disadvantage is that better classification is assigned to solutions that are more similar to the provided scheme, penalizing different programming styles (Rahman, Nordin & Che 2008) that can be so good or even better.

Dynamic approaches include systems that evaluate the submitted program by running it through a set of predefined tests. One example is Online Judge[3], implemented to help in the preparation of ACM-ICPC (Cheang, Kurnia, Lim & Oon 2003). Another example is Mooshak[4], a system originally developed for managing programming contests, and a reference tool for competitive learning (Leal 2003, Leal & Silva 2008). Generally these approaches use a *simple string comparison between the expected output and the output actually produced* to determine if both values are equal (the program submitted will be considered correct only if this condition is true); of course, this strict comparison can be a limitation — it is the main drawback of these systems.

There are also several other tools, used by instructors of some modern educational institutions, that facilitate the automatic grading of programming assignments (Patil 2010). Some of them are developed as web applications, where users can exercise their programming skills by coding a solution for the given problem — like Practice-It[5] or Better Programming[6]. One of the advantages of these systems is that the user gets instantaneous feedback about their answers, which help him to attempt the problem until he reaches the correct solution. Other example is WebCAT[7], a tool focused on test driven development which supports the grading of assignments where students are asked to submit their own test cases.

One major disadvantage of these traditional approaches is their incapability to analyze the way the source code is written. This is especially relevant in educational environments, where the instructor wants to teach programming styles or good-practices for a specific paradigm/language. This feature would be crucial to detect situations where the submitted solution does not comply with the exercise rules. As an example consider a typical C programming exercise that asks the student to implement a Graph using *adjacency lists* to print the shortest-path between two given nodes. The referred grading systems will consider completely correct a solution implemented with an *adjacency matrix* if the final output is equal to the expected one; however, that solution is not acceptable because it does not satisfies all the assignment requirements. Or even more dramatic, if the user computes by hand the shortest-path and the submitted program only prints it, the solution will again be accepted because the evaluation system can not detected such erroneous situation.

This means that traditional dynamic grading systems leave aside one important aspect when assessing programming skills: the source code quality. However, other tools like static code analyzers (see for example Frama-C [8] or Sonar[9]), are able to identify the structure and extract complementary information from the source code in order to understand the way

[3] http://uva.onlinejudge.org

[4] http://mooshak.dcc.fc.up.pt/

[5] http://webster.cs.washington.edu:8080/practiceit/

[6] http://www.betterprogrammer.com

[7] http://web-cat.cs.vt.edu/

[8] http://frama-c.com/what_is.html

[9] http://www.sonarsource.org/features/

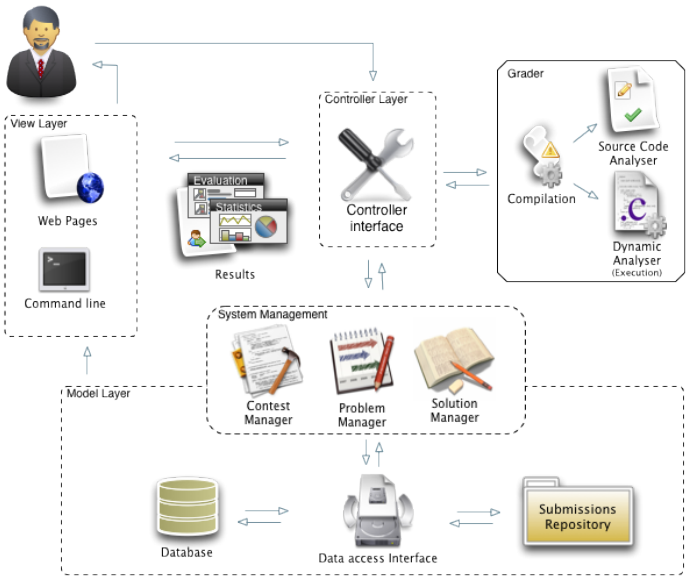

Figure 1: QUIMERA architecture modules view

the program is written and discuss its quality in terms of software metrics that can be easily computed. They can be invoked after compilation and do not need any execution to produce grading data about the program under evaluation. Thus, if combined with traditional grading systems, the static analyzers can provide an accurate notion of the source code quality.

This assumption led to the construction of systems like CourseMaker[10] or Boss[11], that improve the dynamic testing mechanism by calculating metrics and performing style or quality analysis. Providing this immediate feedback to users (students/competitors or instructors/judges) is obviously a relevant extra-value. A very recent system, AutoLEP (Tiantian, Xiaohong, Peijun, Yuying & Kuanquan 2009, Wang, Su, Ma, Wang & Wang 2011), improves traditional grading mechanisms by combining source code static analysis, similarity degree and dynamic testing. Summing up, it evaluates the program construction and how close the source code is from the correct solution.

As a consequence of the research done, QUIMERA aims at evaluating and ranking automatically programming exercises in competitive learning or programming contest environments, by combining a very complete static source code analysis with dynamic analysis. Thus, QUIMERA is a system capable of ensuring the grading of the submitted solution based not only in its capability of producing the expected output, but also considering the source code quality and accuracy. Our system guarantees that different programming styles will not be penalized if they produce the correct output and satisfy all the requirements; this can be an advantage on learning environments, because the student is stimulated to find different correct solutions. Moreover, this does not restrict the contest to problems that have only one correct solution, which can represent a significant reduction in time and effort for larger problems.

Besides encouraging competitive learning by providing immediate feedback to the user, our system completes its static analysis with a plagiarism detection tool, in order to prevent fraud among submitted solutions (a common issue in learning environments). To the best of our knowledge, this feature is not provided by the cited tools.

QUIMERA also allies a simple and intuitive user interface with several graphics exhibiting various statistics concerning the contest flow. These statistic graphics are useful for the competitors, as they illustrate their personal evaluation and performance, and are also useful from the judges, as they provide overviews of each competitor performance, of the current contest, and of all the problems/assignments proposed.

3 QUIMERA OVERVIEW

QUIMERA is a web-based application which provides a full management system for programming contests, as well as a semi-automatic feature for assessing programs. It allows to create and manage contests, or programming exercises. For a new contest, QUIMERA permits to add problems (statements/requirements + input-output vectors for testing), register competitors and associate them to groups. Moreover, QUIMERA offers various facilities to follow up contests, allowing to monitor the overall process and

[10] http://www.coursemaker.co.uk/whatis.html

[11] http://www.dcs.warwick.ac.uk/boss/about.php

Figure 2: QUIMERA Main Page

different activities involved.

To support all the functionalities announced, the application is complex and its success was a consequence of the architecture drawn and the use of recent and powerful technologies. This section is devoted to QUIMERA architecture and a general description of the application. Implementation details are postponed until Section 5.

3.1 Architecture

Basically, QUIMERA architecture is organized in three levels, as can be seen in Figure 1. This layered structure follows the Model-View-Controller[12] (MVC) pattern, in order to create a clear separation among the different aspects of the application (business logic, user interface, and orchestration) while providing a loose coupling between these elements.

The Model layer represents the core of QUIMERA system, this is, it models all data and basic operations realizing the underlying business logic. As described in Figure 1, this layer has two main components, one for Data Storage and another for System Management. In our case we split the Data Store into a traditional relational Database—to store general data concerned with contests, users, history, etc. — and a Submission Repository—to archive submissions (problem statements, competitor solutions and the testing set). A Data Access Interface (*objectification component*[13]) creates a map between entities in the databases and objects in the program allowing a smooth interface between Data Storage and System Management components. The System Management sub-layer concentrates all the tasks concerned with users and contest creation, follow-up and monitoring. It is composed of three modules:

- *Contest Manager*, responsible for the basic contest management tasks (like create or edit a contest);
- *Problem Manager*, responsible for the problem/assignment creation (submission of a new statement or a set of the input-output vectors);
- *Solution Manager*, responsible for the submission of the answers (or solutions) developed by the competitors.

The View layer, which renders the data model into a web page suitable for interaction with the user, is based on a set of Twig templates to generate the different user profiles. This layer also enables a command line interface which allows to perform the essential system administration tasks.

Finally, the Controller layer reacts to the user actions and, as appropriate, changes the view interface or communicates with the Model layer to produce the answers required. It can be compared to the *Maestro* that drives the orchestra.

Figure 1 also shows that in our architecture the Grader was put aside in an independent component.

[12] http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html

[13] As will be seen later, Doctrine2 tool was used to implement this module.

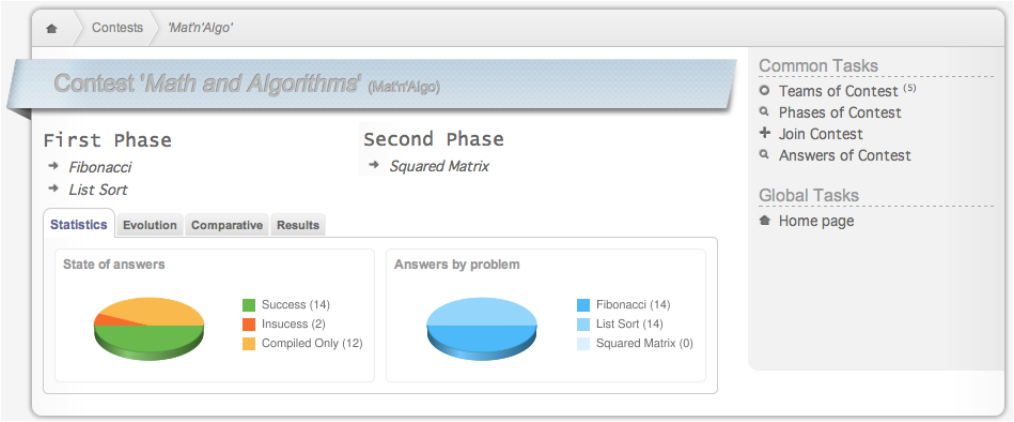

Figure 3: QUIMERA Competitor Interface — Contest Main Page

This decision aims at system flexibility, allowing for easy extensions to support other source languages, or even for the incorporation of different static or dynamic analyzers. This module is incharge of all the assessing and grading tasks. Besides the Compiler, it is composed of three modules:

- *Source Code Analyzer*, responsible for assessing the source code through the analysis of its static properties and the evaluation of a set of metrics;
- *Dynamic Analyzer*, responsible for the execution of the compiled code and the verification of the output produced for each input data vector;
- *Grader*, responsible for grading the submitted program (the competitor answer), according to the information delivered by both Analyzers; this module also computes and delivers various statistics about the answer and competitor behavior.

The design approach above described has proven to be very effective in what concerns the maintainability effort and the application flexibility.

3.2 Actors and Roles

QUIMERA provides different user interfaces targeted to the profiles that reflect the roles of the different contest participants, as descried bellow.

- Administrator – this view has two modes: *Administration* and *General*. The first one is used to access the back-end area to setup the system data and make it operational. The second one allows to participate in a contest as a general user.
- Teacher – this view allows a user to publish problems in a contest, add a set of tests.
- Competitor – used for the player associate himself to a Contest and a Team and compete.
- Judge – used to act as a decision maker, giving feedback about the team submissions.
- Guest – this mode allows any user (not involved in any of the previous roles) to follow the progress of a contest in an anonymous mode (without being registered).

For security purposes and to ensure that only authorized users access the system, all the user profiles are accessible by authentication (except for the Guest mode). For this, it is mandatory that users register themselves on the system. The Administrator may posteriorly define new permissions for Teachers and Judges in each contest.

4 USING QUIMERA

QUIMERA user interface offers a simple and intuitive way to set up and manage contests, complemented with an extended feature for assessing and grading programming exercises. Figure 2 shows the main page of QUIMERA user interface, which follows the depicted structure: it is composed of a Main Menu, where the user can find the principal navigation options; a Secondary Menu that lists the options related to the actual page content; the Sitemap, which shows the user current location on the application and can be used to navigate through the website levels too; and the Content area, where the information is displayed.

QUIMERA offers full support for evaluating source code written in C language, since it is widely used both in academic and industrial environments. In what follows we briefly present how to use QUIMERA and its major features.

4.1 Contest and Problem Management

The setting-up process for a new contest is performed by the system Administrator. Besides the contest name and duration, it associates to each contest a group of Teachers and a group of Judges.

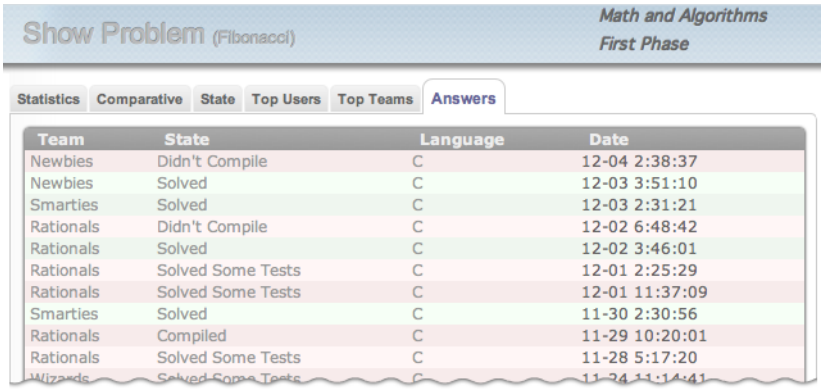

Figure 4: QUIMERA Teacher Interface — List of submitted solutions

After a contest creation, the associated Teachers can use QUIMERA front-end to publish new problems, distributed by phases. Associated to each problem description, it is required to define the maximum execution time allowed and to provide a set of input values and the respective expected outputs for testing the solutions submitted by each competitor team. When submitting a new test, Teachers can define if it will be publicly available as example for competitors, or if it will be hidden and only used for grading purposes.

In their turn, associated Judges can consult the current submissions and the evaluation statistics. They can also follow the contest flow by accessing the contest statistics, and it will be asked to them to assign a final grade to each submission.

To participate in a contest, Competitors need to register in the system and to be associated to the intended contest. This association can be done directly by the Administrator, by the Teachers or by the Competitor itself. For this purpose and after choosing the desired contest in the *Active Contest* list (which can be accessed through the Secondary Menu of QUIMERA Main Page depicted in Figure 2), the user should select the option "Join Contest" on the Secondary Menu of the Contest Preview page (as depicted in Figure 3).

Moreover, the user must associate himself to a Team, as contests are organized in Teams and not in individual competitors. The same Team is allowed (and is encourage) to compete in several contests, however in the same competition, a user can only be assigned to one Team.

4.2 Answer Submission

To submit a solution[14], a Competitor starts selecting the respective contest page. It is then shown the problem statement and some examples of the desired input and the correspondent expected output. After selecting the option for submitting an answer, the user uploads the respective file. If the upload process is unsuccessful, the system informs the user to try again. If successful, QUIMERA compiles the submitted (uploaded) program and notifies the user about this task.

In case of compilation errors, QUIMERA reports the errors found and provides some tips aiming at guiding the Competitor to a correct them and resubmit the answer. If no compilation errors are detected, the system goes to the next step, the solution assessment process, as described in Sections 4.3 and 4.4.

4.3 Dynamic Analysis

QUIMERA dynamic analysis follows the traditional approach executing the compiled submission with a set of predefined tests. As referred, each test is composed of an input data vector and the correspondent output. For each test, the compiled answer is invoked (loaded and executed) receiving the data vector as an argument; the output produced, as a result of this run, is compared with the expected output. If they are strictly the same values, this test is OK. The submission under assessment is accepted only if all the tests pass.

After the execution, the user can consult the percentage of tests successfully passed, as we can see in Figure 6. The user only has access to the public tests, never knowing the other tests in the set, neither how many they are. This is useful to avoid situations of *trial and error* for *test set guessing*.

As the execution of an external program is a critical task for any computer environment (malicious code can damage the system), QUIMERA only compiles and executes the solution compiled if it complies with strong constraints (consensual safety limits are defined). With the help of the static analysis, system calls and suspicious instructions are blocked. By default the execution timeout is 3 seconds, but this can be personalized at the problem creation moment.

In a typical competitive environment, a Team can submit several solutions before one is accepted. Each

[14]In this context, the words *solution*, *answer*, or *submission* are alternative names to designate the program developed by the Competitor aimed at solving the problem (the programming challenge) stated by the Teacher.

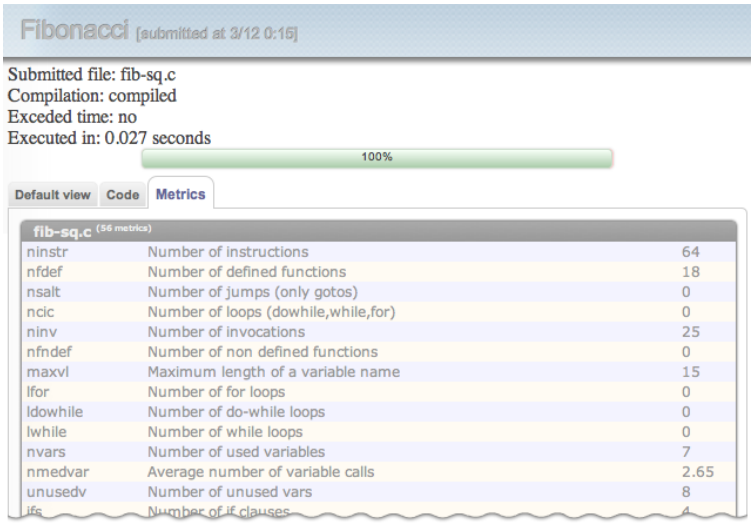

Figure 5: QUIMERA Teacher Interface — Assessment data for a submitted solution

submission is ranked based on the following levels: *Didn't Compile*; *Compile* (when it fails all tests); *Timeout*; *Passed some tests*; *Solved*, as depicted in Figure 4. For final grading purposes, QUIMERA considers the best ranked answer. This allows the submission of several answers, developed by different members of one Team, in order to get a better score. We believe this (not conventional) approach stimulates the competition and privileges the competitive learning, allowing also to improve the code quality.

4.4 Static Analysis

QUIMERA produces a complete report about the quality of a submitted solution (as depicted in Figure 5), through the direct source code analysis based on a set of predefined metrics. Each metric represents a measurement that contributes for the quality estimation, working like an indicator for Teachers and Judges.

Currently, QUIMERA measures 56 different metrics, grouped by five classes: *Size*, *Consistence*, *Legibility*, *Complexity* and *Originality*.

Size metrics are based on the lines code number and are used to compare the size of a submitted solution against the average size of all the submissions of the problem. They represent a good indicator of solutions that are unusually small or too big, comparatively to other solutions.

Consistence metrics work as an indicator of the probability of execution errors occurrence due the use of Dynamic Structures and Memory Management. In a learning environment, a student can easily make mistakes when implementing thise structures. We assume that an overuse of thise structures will increase the probability of execution exceptions and may affect the answer final score. For this we measure markers like the percent of pointers use, the variance between the allocated and released memory or the number of returns in a function.

Legibility Metrics are associated to the easy of source code comprehension. Markers like *comment density*, *average lines per comment*, percent of *goto*, *break* and *continue* usage can influence the source code legibility and reuse. Therefor, they can be used as an indicator of competitor good or bad practices and influence the final grading. Other important marker associated with reuse and legibility is the density of duplicated source code found. If a competitor repeats several times the same piece of source code, instead of calling them in a function, it decreases both reuse and legibility.

Complexity calculation is done through a simple approach of the McCabe Cyclomatic Complexity[15], generally used to estimate a program complexity based on the amount of linearly independent paths through the source code, and computed based on its control flow graph. It is useful to estimate the complexity of each submitted solution: less complex solutions obtained better scores.

Originality is featured by an index which works like an indicator of plagiarism situations, calculated based on the similarities of the assessed solution and the other submitted solutions.

After the computation, metrics and other static assessment data are available through QUIMERA interfaces. An XML full report is also generated. The details behind the implementation of this analysis are introduced in Section 5.

[15] For more details, please consult `www.literateprogramming.com/mccabe.pdf`

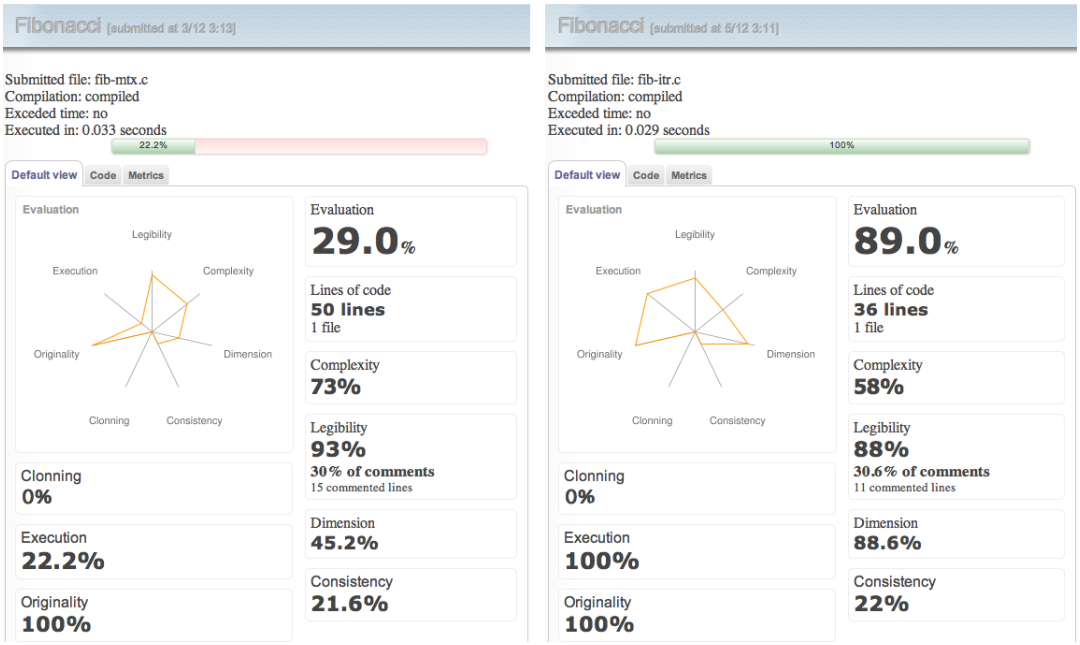

Figure 6: QUIMERA Common Interface — Final assessment comparison

4.5 Automatic Grading

After this assessment process, the Grader evaluates the results of the source code and dynamic analysis assessment, concerning a grading formula. This grading formula is composed of seven different assessment categories, which have associated different weights in the final grade. These categories are:

Execution — 75% of the final grade is related to the number of tests passed by the solution (on the dynamic analysis) and 5% of the final grade comes from the *Execution Time*;

Size — related to size metrics with 2% of final height;

Consistence — includes measurements related to the probability of execution errors and represents 5% of the final grade;

Legibility — associated to Legibility metrics, with a total weight of 2%;

Complexity — total impact of 6% in final grade;

Cloning — represents the total percent of duplicated code in the solution, with a weight of 5%.

Although these categories naturally emphasize on the execution, these weights also take into account the impact of the source code quality, which we believe that encourage Competitors to find improved solutions, to obtain better final grades.

After this grading process, it is possible to access the final report obtained from the solution assessment, which includes an overview of the applied metrics and its values, the number of tests passed and the final grading summary (depicted on both sides of Figure 6). For a better comprehension of this summary and the grading process, let us now introduce one example of a contest where Competitors are invited to solve several mathematical problems. The challenge is to present a solution to calculate the n^{th} fibonacci number, where n is a number asked to the user. In Figure 6, we are comparing two different solutions proposed by two Teams: the Rationals on the left-hand size, and Smarties on the right-hand size.

The first solution, assessed by the system with ”Solved Some Tests” (as we can see in Figure 4), only calculates correctly the first and second numbers of the sequence (0 and 1) and, consequently only passes in 22% of the tests (2 in 9). This leads to a final score on the *Execution* field of only 22.2%. In a closer look, we can conclude that this answer is well documented (it has 15 commented lines in 50 lines of code) and its *Legibility* has a final score of 93%. Its *Complexity* score of 73% is owed to its number of used variables and data structures. Its 50 lines of code exceed the average size of the contest submissions for this problem, which penalizes its final grade on the *Dimension* category (only 45.2%). In the *Consistency* category, its 21.6% are owed to the use of several returns through the code (assuming a maximum of two returns per function as a reasonable limit to the source code consistency) and also to the use of pointers.

The second one, assessed by the system with ”Solved” (as depicted in Figure 4), follows an iterative strategy to implement the Fibonacci recurrence. This answer is better documented (11 commented lines in 36 lines of code), but its *Legibility* has a lower final score (88%) once it did not used *defines*, as the other solution. Its *Complexity* score of 58% is owed to the implemented *loop*. Its 36 lines of code improve its final grade on the *Dimension* category to 88.6%. In

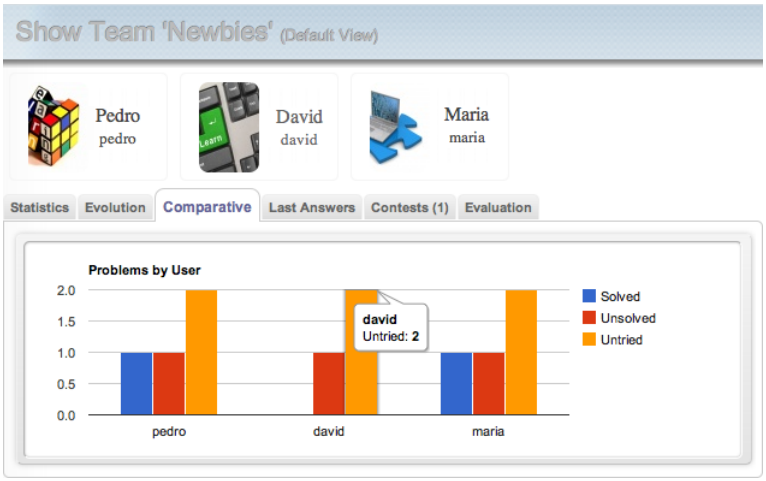

Figure 7: QUIMERA Teacher Interface — Team members grading

the *Consistency* category, its weak 22% are also owed to the use of several returns through the code.

Finally we can conclude that both solutions have not been plagiarized (100% original) and *Competitors* follow the good practice of no repeating their own code (0% of duplicated code). These assessments led to a final grade of 29% on the first case and a significant 89% on the second case. QUIMERA also completes this evaluation with radial charts to a quicker and easier comparison of the solution performance in the different grading categories.

4.6 Statistics

QUIMERA computes several statistics to enable the monitoring of the contests, which depends on the active interface and context. Statistics can be concerned with different perspectives like: all or a particular contest; all or a particular problem; all or a particular Team or Competitor (team member). Statistics are presented in the form of different type of graphics: piecharts (see Figure 2 and 3), columns charts (see Figure 7), tables (see Figure 4), line charts, radial charts (see Figure 6) and scatter charts.

In more detail, we can say that our system offers numerical data and listings concerned with: Competitors and Teams rankings; comparative graphics between problem submissions; the full list of submissions for each Competitor or Team; comparisons between Competitors inside a Team; comparisons between the Teams performance in a context; overviews for each contest, its phases and summary information about the current state of the problems and submissions, among many others.

This amount of statistical information makes QUIMERA a powerful tool for supporting the assessment and grading of programming exercises solutions, helping all kind of users, Administrator, Teachers, Judges and Competitors.

5 QUIMERA IMPLEMENTATION

QUIMERA system follows a typical web-application schema: a server-client framework connects users to a server, where submissions and system data are recorded. QUIMERA User Interface is rendered in HTML 5, CSS 3 and Javascript, optimized for the last generation browsers but also compatible with older browsers without loosing any features.

The system runs on any PHP5.3-enabled platform over HTTP and HTTPS protocols, and it is based on the Symfony2[16] framework. The essential application data is stored in a MySQL database, although the data layer of Symfony2 (managed by Doctrine2 technology) allows to use any other database engine. We also use the template manager Twig, embedded in this framework, to render HTML5 templates from the view layer.

To compile the submitted solutions, the system uses the GCC compiler. For the static analysis, we developed a plugin for Frama-C[17], a framework dedicated to the static analysis of source code written in C. This powerful platform works as a front-end for our plugin, once its kernel provides common functionalities, libraries and collaborative data structures, which simplifyies the development process and improves the final results. CIL[18] library provides methods to generate the Abstract Syntax Tree (AST) for the source code and to traverse the tree. Our plugin analyzes the AST for each submitted source file and evaluates a set of metrics (described in Subsection 4.4) with a single traversal, to obtain quantitative information about the quality of the submitted source code.

To detect cloning, we use regular expressions, string comparisons and sorting algorithms from PHP,

[16] http://symfony.com
[17] http://frama-c.com
[18] C Intermediate Language
http://cil.sourceforge.net/

to detect duplicated lines of code in the submission.

In order to detect plagiarism, we adopted the tool sherlock[19], which makes comparisons based on source code signatures, and compare them with related submitted code. This tool supports its comparison algorithm in a *Originality Index* that computes the ratio between similarities and differences detected in each pair of files. This plugin produces a final result that contemplates all these evaluation parameters as well as the final grade assessed by the system, as described in Subsection 4.5.

QUIMERA provides several statistical graphics related with a contest flow (as described in Subsection 4.6), rendered with Google Chart Tools[20].

6 CONCLUSION AND FUTURE WORK

Along this paper we have introduced QUIMERA, a web-based system that offers a simple and efficient way to create and manage online programming contests. QUIMERA underlying philosophy, its features and implementation were discussed.

After characterizing the area of programming challenges and their automatic assessment and grading, we reviewed the state-of-the-art looking for common approaches, applications and similar systems. The research done so far, summarized in the paper, led to the identification of a drawback in the available systems; it has inspired the proposal of an improved system and guided the design of its architecture.

In this direction, QUIMERA offers an automatic assessment and grading of programming exercises based on the information produced by their source code analysis, combined with the traditional information obtained from analysis of the output values generated for predefined input data values.

As the system is completely implemented for C programming language, we plan to make soon it available online for free use, with a complete user manual and usage examples.

After this first developing stage, we are willing to test the tool with real users in different scenarios, to study its usability and effectiveness. We will prepare experimentations properly designed to assess these two indicators.

As future work, we intend to increase QUIMERA flexibility enabling the development of new frontends to support other programming languages. To allow an easy support for new languages, we plan to modify the tool in order to accept *Language plugins*.

Another relevant improvement of the dynamic evaluation process, that will extend QUIMERA usability, is to implement a semantic evaluation of the output. This idea, that is a real challenge requiring much more research work, requires the definition of a metalanguage to describe the expected output for a given input data vector. Then, instead of doing a strict comparison between the outputs produced and expected, the evaluator checks the validity of the output content according to the formal description.

[19] http://sydney.edu.au/engineering/it/~scilect/sherlock/

[20] http://code.google.com/intl/en-US/apis/chart/

REFERENCES

Cheang, B., Kurnia, A., Lim, A. & Oon, W.-C. (2003). On automated grading of programming assignments in an academic institution, *Comput. Educ.* **41**: 121–131.

Danić, M., Radošević, D. & Orehovački, T. (2011). Evaluation of student programming assignments in online environments, CECiiS: Central European Conference on Information and Intelligent Systems.

Forsythe, G. E. & Wirth, N. (1965). Automatic grading programs, *Technical report*, Stanford University.

Leal, J. P. (2003). Managing programming contests with Mooshak, *Software—Practice & Experience* .

Leal, J. P. & Moreira, N. (1999). Automatic Grading of Programming Exercises, p. 383.

Leal, J. P. & Silva, F. (2008). Using Mooshak as a Competitive Learning Tool, *The 2008 Competitive Learning Symposium* .

Patil, A. (2010). *Automatic grading of programming assignments*, Master's projects, Department of Computer Science, San José State University.

Rahman, K., Nordin, M. & Che, W. (2008). Automated programming assessment using pseudocode comparison technique: Does it really work?

Tiantian, W., Xiaohong, S., Peijun, M., Yuying, W. & Kuanquan, W. (2009). Autolep: An automated learning and examination system for programming and its application in programming course, *First International Workshop on Education Technology and Computer Science*, USA.

Wang, T., Su, X., Ma, P., Wang, Y. & Wang, K. (2011). Ability-training-oriented automated assessment in introductory programming course, *Comput. Educ.* **56**: 220–226.

Zamin, N., Mustapha, E. E., Sugathan, S. K., Mehat, M. & Anuar, E. (2006). Development of a web-based automated grading system for programming assignments using static analysis approach.