A study on implementation and usage of web based programming assesment system: Code

Tomche Delev[1] and Dejan Gjorgjevikj[1]

Faculty of Computer Science and Engineering
{tomche.delev,dejan.gjorgjevikj}@finki.ukim.mk

Abstract. *E-Lab* is a system developed at Faculty of Computer Science and Engineering for solving and auto-grading programming problems from introduction to programming courses. The main goal is to simplify and improve the organization and the process of solving programming problems from large group of students in dedicated computer labs using centralized server. All the work from the students is done in a web browser using a web-based code editor and everything is stored, compiled and executed on the server. The system keeps records of all problem attempts from identified students which are used as attendance records. All the problems and solutions are under version control system (Git). The platform supports different types of problems in several programming languages (C, C++, Java) and it's designed to be easily extended.

Keywords: Online submission, programming languages, automated assessment

1 Introduction

In last three years, the trend of new enrolled students in CS is showing constant increase. This trand directly effects the large number of students in introductionary programming courses. The data at the Faculty of Computer Science and Engineering (FCSE) shows that in 2012 and 2013 the number of enrolled students in the introductionary programming course Structured Programming were 900 and 1029 respectivly.

One step in better maneging the learning process of large groups of students was development and implementation of web based system for automatic assessment of programming problems then called E-Lab [5] and now renamed to Code. The initial idea of the system was to help tutors and instructors in identified difficulties they have trying to assess all of the students' solutions. Later on the system was also used in practical exams in courses that involve programming assignments. The timed and informative feedback to students and automatic assessment is top priority of the system.

Application of automatic assessment in programming assignments is suggested long time ago [9]. In the context of very large group of students and the new MOOCs it may be the only solution to provide effective feedback and grading. Speed, availability, consistency and objectivty of assessment are some

advantages mentioned in [3], and [16] are showing that automatically generated grades are highly correlated with instructor-assigned grades.

In this paper we present the experience and initial results from implementing the system Code in two programming coureses taught at FCSE. We study the data generaged by the usage of the system, and try to identify patterns of usage that can reveal some potential new features, but also problems with our system. We investigate the results from plagiarism detection system and show results from qualitative evaluation on the system from representative group of end users.

2 Related Work

The work on automatic assessment can be broadly categorized in research on systems and tools and research on new methods and difficulties of novice programmers. Examples of recent systems are eGrader [14] graph-based grading system for Java introductionary programming courses, CAT-SOOP [7] a tool for automatic collection and assessment of homework exercises and WeScheme [17] that is a system similar to Code [5] in using the web browser as coding environment. In their work [10] review most of the recent system. They discuss the major features of these systems such the ways of defining test by teachers, resubmission policies, security issues and concluded that too many systems are devolped, mainly because most of the systems are closed and collaboration is missing.

There are also studies on different aproaches and learning methods that can be helpful in designing, implementing or improvment of automatic assessment systems. One such study is on the difficulties of novice programmers [12], where by surveying more than 500 students and teachers, autors provide information of the dificulties experienced and percieved when learning or teaching programming. One interesting conclusion they present is that students overestimate their understanding, while the teachers think that the course contents are more difficult for the students than the students themselves. Students usually get the right perception lately during the exam sessions.

Other interesting subject in research are studies on student programming bugs and most occured syntax errors. A one year empirical study of student programming bugs is done in [4], where authors conclude that approximately 22% of the problems are due to problem solving skills, while the remaining problems involve a combination of logic and syntax problems. Also there is a study on the most common syntax errors [6], where results are showing that many of these errors are consuming large amount of student time, and even students with higher abilities are not solving them more quickly. There are also studies that investigate the dynamics and process of solving programming problems in novice programmers. An analysis of patterns of debugging is done in [1], and in [8] they try to reveal the process of solving programming problems which is mostly invisible to the teachers. Using analysis of interaction traces they investigate how students solve Parson's [13] programming problems.

3 Methodology

In this paper we analyze and study the data generated from students using the web-based system for automatic assessment of programming problems Code at the Faculty of computer science and engineering in Skopje. This system is in use from September 2012 and it is integral part in eight courses that involve some kind of programming assignments in programming languages such as C, C++ and Java. More than 2000 students are working on total 1296 problems, organized in 367 problem sets, from which 165 (45%) are exams. Students can work on the system directly using the web-based code editor or they can use any IDE, and then paste the code to run and test. By observing students in lab and examp sessions, they mostly use the web-based editor in introductionary programming courses or when making small changes in code, while in more advanced courses they usually use IDEs such as Eclipse, NetBeans or Code::Blocks.

3.1 Data recorded

When students are using the Code system for solving the programming problems, the system is storing most of the data generated in the process. Among the data collected by the system are the time when problem is opened, and records for each student submission (attempt to solve the problem). In order to test the correctness of their solution, students have two options, to *Run* or to *Submit* the solution. When *Run* is executed the student code is saved, compiled, executed and if no syntax errors are present, tested using dynamic analysis on a sample test case. If there errors are present in the compilation, the messages from the compiler are returned as an output of the execution, and if not, the result is shown next to the expected sample output, so easy comparisson on the outputs can be performed. When saving the solution, if the content of the code is different from the previous solution, it is stored as a new version of the solution, keeping the old one. The system have implemented version history of the solutions, so students can revert back to any previous version of their solution. This can be very usefull specially to begginers who have not heard or tryied any version control system. When users *Submit* their solution, additionally to the steps performed when running, the system saves a problem attempt record with time of the attempt and result from testing the result on all the test cases of the given problem. The result of the testing is the number of test cases passed, and if all test cases passed, the attempt is marked as correct. Students can do unlimited submissions and create as many problem attempts records. Even when the result from submission is success, they still have the option to resubmit their solutions, so as a result we can have multiple correct problem attempts by problem. In more than two years of active usage of the system it has recorded more than 750,000 problem attempts and more than 1,000,000 versions of solutions. An initial step in studying and analysing this part of data is presented in this paper.

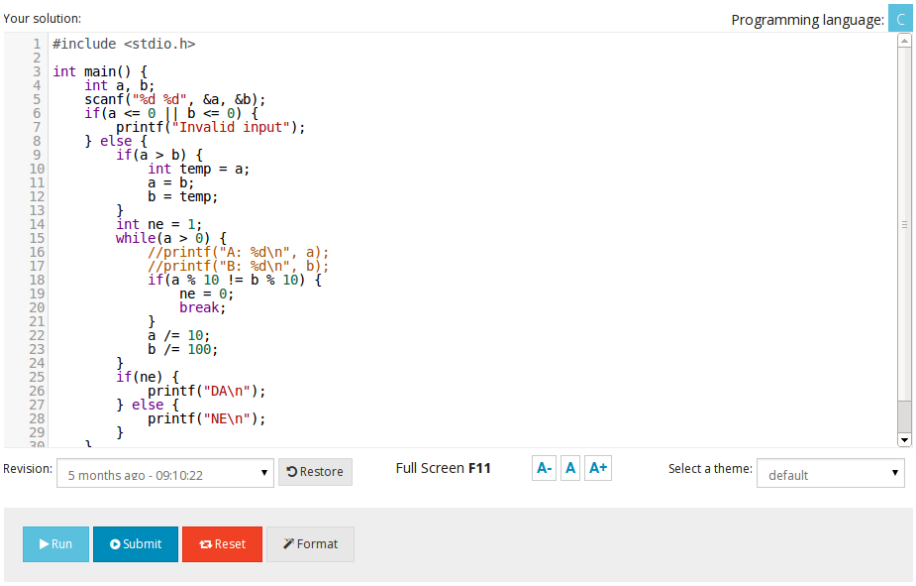

Fig. 1. The web-based code editor in Code.

3.2 The context

From all the courses that are using Code, in this paper we report here on data collected from the winter semester (September - December) 2012 of the following two: Structed Programming (SP) and Advanced Programming (AP). Structured Programming is a first year introductionary course thought in C, and Advanced Programming is a seccond year more advanced elective course thought in Java. 1029 enrolled the course Structred Programming and each student had to attend at least 80% of 9 lab sessions, and had oportunity to take two midterm exams and one final exam. Completing the lab sessions they could earn a total of 10% credit, and from solving the problems on the midterms or exam session thay could earn a total of 70% credit towards their final grade in the course. The students in this introductionary course are with different level of motivation and there are significant number of students that are enrolling the courses seccond time or third time. Advanced Programming was enrolled by 149 students, and the settings of the course are simillar to those in Structured Programming. It should be noted that students in Advanced Programming were already familiar with the system, by working on it in two previous courses from first year, and they are more motivated because they chose the course by their own will.

We chose this courses because the system is in use second year, in both lab programming assignments and exam programming assignments.

3.3 General problems success rate

todo...

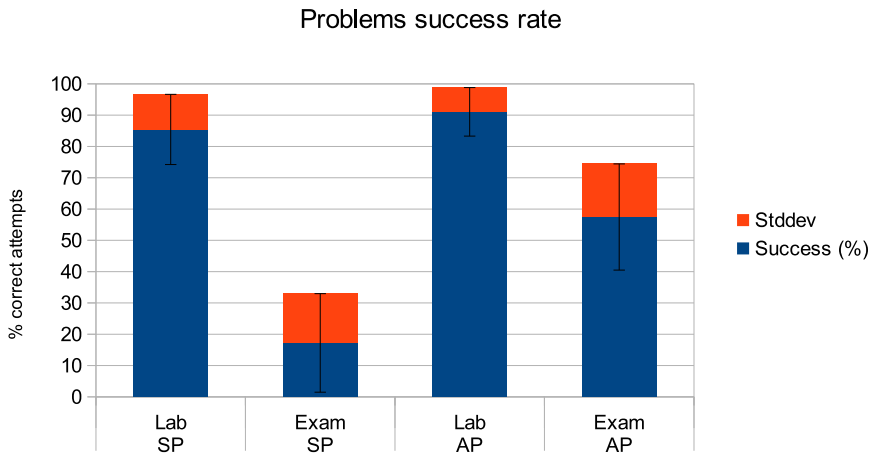

Fig. 2. Problems success rate

3.4 Compilation data

todo...

3.5 Evolution of correct solutions

todo...

4 Reports on plagiarism

Plagiarism in source code in programming assignments is a serious issue in most undergraduate courses that involve programming. In a study in 2004 at School of Computing at the National University of Singapore, 181 students admitted plagiarism [15], and the Centre for Academic Integrity (CAI) reported that 40% of 50,000 students at more than 60 universities admitted plagiarism [11]. Students involved in plagiarism learn a lot less than their honest coleagues, they harm the reputation of their own institutions, and reduce value of their own degrees.

Having in mind the importance of plagiarism detection, in our system we have integrated one of the most efficient systems for that purpose, MOSS (Measure of software similarity) [2]. Before starting the winter semester the students were clearly informed that their solutions in the system will be checked for plagiarism, and subsequently the involved parties will be sanctioned. The results presented on table 2 are showing evidence of plagiarism mostly in laboratory settings in SP course, where students have the freedom to share or use copied solutions. In plagiarism were involved the high achievers and low achievers, since latter

Table 1. Results on plagiarism detection using MOSS

Course	Settings	Average percentage match	Average lines matched	Potential plagiats*
SP	Lab	52.04%	16.37	3869
	Exam	22.74%	8.06	20
AP	Lab	10.08%	28.22	1
	Exam	10.26%	20.12	2

/home/git/code/plagiats/p1200/c/131078

```
#include <stdio.h>
#define MAX 100

int main (){
int mat[MAX][MAX],m,n,i,j,sum=0;
    scanf("%d%d",&n,&m);
    for(i=0;i<n;i++)
        for(j=0;j<m;j++)
    {
    scanf("%d",&mat[i][j]);
    }

    for(i=0;i<n;i++)
       for(j=0;j<m;j++)
    {
        if(i==0||j==(m-1)||j==0||i==(n-1))
            sum+=mat[i][j];
    }
    printf("%d\n",sum);
    return 0;


}
```

/home/git/code/plagiats/p1200/c/116038

```
#include <stdio.h>
#define MAX 100

int main(){
        int a[MAX][MAX],i,j,n,m,zbir=0;
    scanf("%d%d", &n,&m);
    for(i=0;i<n;i++)
        for(j=0;j<m;j++){
                scanf("%d", &a[i][j]);
        }

    for(i=0;i<n;i++)
        for(j=0;j<m;j++){
                if((j==0 || j==m-1) || (i==0 || i==n-1))
                        zbir+=a[i][j];
        }
        printf("%d\n", zbir);
}
```

Fig. 3. Plagiats example

mostly copied solutions from the former. When directly presented the evidence most of these students admitted the plagiarism and regreted the action.

On figure 3 we show example of the results from MOSS system, where plagiarism is detected. Lines collored green and red are matched and this clearly shows that these two solutions are plagiats.

While in all academic settings it is still very import to address plagiarism in every form, in a new learning model offered by MOOCs, where learning is no longer for credits and grades, most of the factors contributing to plagiarism may be eliminated. This is an important difference between MOOCs and traditional courses. In the latter, the need for plagiarism detection and action against plagiarism is paramount while in MOOCs it may serve no purpose.

5 Evaluation

Results from qualitative evaluation survey

Table 2. General usage questions

I used Code in?	1. Structured Programming (C)	40%
	2. Object Oriented Programming (C++/Java)	49%
	3. Algorithms and Data Structures (C/Java)	5%
	4. Advanced Programming (Java)	3%
	5. Advanced Algorithms (Java)	3%
Accessing Code from?	1. Faculty labs	38%
	2. From everywhere	62%
Do you want access from everywhere?	Yes	98%
	No	2%
How often you use Code?	1. I'm not using it	1%
	2. Once a week	42%
	3. 2-3 times a week	42%
	4. More than 3 times a week	15%

6 Conclusion

TODO...

Table 3. System evaluation questions (1-5 grades)

Simple to use?	1 (0%)	2 (2%)	3 (10%)	4 (21%)	5 (67%)
Quality of problem view?	1 (0%)	2 (13%)	3 (6%)	4 (21%)	5 (60%)
Code editor functionality?	1 (4%)	2 (13%)	3 (13%)	4 (40%)	5 (31%)
Performance and speed?	1 (0%)	2 (6%)	3 (6%)	4 (31%)	5 (56%)
Do you think Code helps you in correctly solving the problem?	Yes (62%)			No (38%)	
When using Code?	First use IDE and then copy the solution (83%)		Use the web-based code editor (13%)		Other (4%)

Table 4. Learning programming

Where do you feel that you learn most (programming)?	1. Lectures	4%
	2. TA Exams lectures	17%
	3. Lab exercises	31%
	4. Individual learning	31%
	5. Solving problems in Code	5%
	6. Other	6%
What kind of materials helps you most in learning?	1. Books on subject	13%
	2. Lectures slides	15%
	3. Exercises questions and answers	4%
	4. Example problems with solutions	46%
	5. Interactive visualization of solutions	13%
	6. Other	10%
When solving problem on code I mostly need help in?	1. Understanding the problem and think of algorithm	21%
	2. Implementing (coding) my solution	27%
	3. Detecting and fixing errors in my solution	46%
	4. Example problems with solutions	46%
	5. Other	6%
What kind of help would be usefull to be implemented?	1. Automatically showing relevant materials with similar problems and solutions	63%
	2. Direct communication with tutors over chat	23%
	3. Other	7%

References

[1] Ahmadzadeh, M., Elliman, D., Higgins, C.: An analysis of patterns of debugging among novice computer science students. In: ACM SIGCSE Bulletin. vol. 37, pp. 84–88. ACM (2005)

[2] Aiken, A.: Measure of software similarity. URL http://www. cs. berkeley. edu/-aiken/moss. html (1994)

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

[4] Bryce, R.C., Cooley, A., Hansen, A., Hayrapetyan, N.: A one year empirical study of student programming bugs. In: Frontiers in Education Conference (FIE), 2010 IEEE. pp. F1G–1. IEEE (2010)

[5] Delev, T., Gjorgjevikj, D.: E-lab: Web based system for automatic assessment of programming problems (2012)

[6] Denny, P., Luxton-Reilly, A., Tempero, E.: All syntax errors are not equal. In: Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education. pp. 75–80. ACM (2012)

[7] Hartz, A.J.: CAT-SOOP: A tool for automatic collection and assessment of homework exercises. Ph.D. thesis, Massachusetts Institute of Technology (2012)

[8] Helminen, J., Ihantola, P., Karavirta, V., Malmi, L.: How do students solve parsons programming problems?: an analysis of interaction traces. In: Proceedings of the ninth annual international conference on International computing education research. pp. 119–126. ACM (2012)

[9] Hollingsworth, J.: Automatic graders for programming classes. Communications of the ACM 3(10), 528–529 (1960)

[10] Ihantola, P., Ahoniemi, T., Karavirta, V., Seppälä, O.: Review of recent systems for automatic assessment of programming assignments. In: Proceedings of the 10th Koli Calling International Conference on Computing Education Research. pp. 86–93. ACM (2010)

[11] Jocoy, C., DiBiase, D.: Plagiarism by adult learners online: A case study in detection and remediation. International Review of Research in Open & Distance Learning 7(1) (2006)

[12] Lahtinen, E., Ala-Mutka, K., Järvinen, H.M.: A study of the difficulties of novice programmers. In: ACM SIGCSE Bulletin. vol. 37, pp. 14–18. ACM (2005)

[13] Parsons, D., Haden, P.: Parson's programming puzzles: a fun and effective learning tool for first programming courses. In: Proceedings of the 8th Australasian Conference on Computing Education-Volume 52. pp. 157–163. Australian Computer Society, Inc. (2006)

[14] Shamsi, F.A., Elnagar, A.: An intelligent assessment tool for students' java submissions in introductory programming courses. Journal of Intelligent Learning Systems & Applications 4(1) (2012)

[15] Tsang, O.W., Choy, T.T.: a survey on awareness and attitudes towards plagiarism among computer science freshmen. Learning Issues 9, 3 (2005)
[16] Vujošević-Janičić, M., Nikolić, M., Tošić, D., Kuncak, V.: Software verification and graph similarity for automated evaluation of students assignments. Information and Software Technology 55(6), 1004–1016 (2013)
[17] Yoo, D., Schanzer, E., Krishnamurthi, S., Fisler, K.: Wescheme: the browser is your programming environment. In: Proceedings of the 16th annual joint conference on Innovation and technology in computer science education. pp. 163–167. ACM (2011)