Evaluation of Student Programming Assignments in Online Environments

Mario Đanić, Danijel Radošević, Tihomir Orehovački
Faculty of Organization and Informatics
University of Zagreb
Pavlinska 2, 42000 Varaždin, Croatia
{mario.djanic, danijel.radosevic, tihomir.orehovacki}@foi.hr

Abstract. *This paper concerns the development of a Web-based learning programming interface that would assist students in acquiring programming skills. The current programming tools that are commonly used in application development do not prevent plagiarism and rote learning of previously written programs. An example of such a Web-based learning programming interface is presented and discussed in this paper. It can be adapted to meet learning needs and also provide additional features like the archiving of students' solutions and their comparison for plagiarism detection.*

Keywords. online programming, Web interface, cloud-based approach

1 Introduction

Programming has often been treated as a form of an obscure art. Therefore programming instructors quite often make a mistake of assuming a certain level of knowledge from the audience whose primary motivation and interests do not lie in programming. The frustration of such an audience is further reinforced by using tools that were not built specifically for learning but rather promote values and processes that seem cryptic.

Until recent years, it was almost unimaginable to develop applications using a Web programming interface. This can be attributed to the lack of appropriate development tools in form of Web applications, or their poor performance that would result in reduced interactivity and usability of the end-user interface.

The above-mentioned shortcomings can be overcome by Web programming interfaces. According to Synodinos [16], the advantages of such interfaces in program development include:

- Ease of use – the editor is easily accessible and usually not intimidating (with lots of tools);
- Real-time collaboration – written code is simply accessible to colleagues, so collaborative editing is possible;
- Extensibility and self-hostedness – the programmer does not need to worry about installation of necessary libraries and other resources;
- Universal accessibility – the code editor should work from anywhere and from any device that can use a Web browser.

The most advanced recent Web IDE systems are based on cloud-computing systems. According to Mirzaei [10], cloud computing is a style of computing that allows users to access technology-enabled services from the Internet without knowledge or control over the technology infrastructure that supports them. By using services offered by cloud computing, one is essentially outsourcing the IT infrastructure through an Infrastructure-as-a-service (IaaS) type of scenario [18].

There are three types of services that Microsoft offers in form of cloud systems [7]:

- Infrastructure as a Service (IaaS) – on-demand computing and storage to host, scale, and manage applications and services;
- Platform as a Service (PaaS) – consists of an operating system, a relational database, and Web-based services;
- Software as a Service (SaaS) – subscription-based online services, on-demand applications and hosted services.

Cloud systems typically include some kind of Virtualization Management, that is, the technology that abstracts the coupling between the hardware and the operating system. Virtualization enables the mapping of single physical resources to multiple logical resources or partitions [13]. Cloud-based Web IDE systems use a Web interface, which enables

developers to use cloud resources in a desktop-like way.

Drawing on the potential of Web programming interfaces, this paper focuses on a particular problem concerning the grading of student work. The specific questions it addresses are how to objectively grade a student assignment or perform a systematic source code plagiarism analysis.

Possible advantages of Web IDEs for educational purposes are:

- All students' solutions could be collected in the same base. Accordingly, archiving of students' solutions and their mutual comparison for plagiarism detection becomes much easier;
- Feature enabling the generation of programming tasks variations. Students can receive different tasks, but all them will be generated from the same templates, as described in [14];
- Feature enabling peer assessment among students.

In the following sections, we shall investigate several source code evaluation systems to see how they approach problems we identified in the process of learning programming. Our solution to these problems is also proposed.

2 Related work

Source code evaluation, or rather, the evaluation of the final application derived from it, is certainly not a novel concept. Different attempts at solving it have been made, with a varying degree of success. The purpose and usage of such systems vary widely, depending on the location and the goals their authors had in mind when designing them. However, our focus is on source code evaluation related to education and extracurricular activities related to that evaluation. Therefore in this paper we will distinguish between three types of source code evaluation systems: automated evaluators, online compilers and plagiarism detectors.

Automated Evaluators. This type of evaluation systems can be further decomposed into dynamic and static approaches. *Static approach* includes systems that compare students' submission with model programs and determines the degree of their similarity. Through comparison of students' solutions with answer schemes provided by the teacher, the Web-based Automatic Grader System (WAGS) [20] evaluates programming exercises written in Visual Basic, C, or Java language. Thruong et al. [17] introduce a component for analysis of students' Java and C# programs as a part of the Environment for Learning Programming (ELP) system. Their framework uses structural similarity analysis to convert students' source codes into pseudo-code abstracts and compare them with teachers' solution models. The main disadvantage of the static approach is that it cannot be used for testing the correctness of programs which contain input and output operations.

Dynamic approach encompasses systems that evaluate student programs by running students' submissions through a set of testing data. Online Judge [2] was implemented with the aim to help students in preparation for the ACM Intercollegiate Programming Contest. By using simple string matching, it compares submitted programs with pre-defined solutions and thus checks their correctness and efficiency. Homework Generation and Grading (HoGG) system [11] consists of three parts. The framework, implemented in the Perl language, manages the entire grading process and communicates with other two parts (Driver and Evaluator) via MySQL database. In the first step of the grading process, the framework runs a Java compiler which compiles the source and saves executable code into a database. After successful compiling, the framework runs the Driver, which loads executable code together with student program methods and places the results of the run back to the database. In the final step, the framework invokes the Evaluator, which conducts bit-vector-based evaluation of the run output. The drawback of the dynamic approach is that it does not evaluate the style that is used in problem solving, which is particularly important for novice programmers.

AutoLEP [19] combines static analysis and dynamic testing, thus enabling the evaluation of both testing results and program constructs. The evaluation of submitted programs consists of two main steps. Firstly, the system checks the syntactic correctness of source code. If the code does not contain errors, the system checks the semantic similarity of the student's submission with each model program.

Online Compilers. Malinowski and Wilamowski [9] introduced the Intranet Compilers package. Its user interface allows the compilation of source code written in several different programming languages (C, C++, Fortran, Java, or Pascal) using one of the listed compilers (GNU, Borland, Microsoft, Sun). The basic element of the package is a PERL script that manages the compiling process and provides feedback in form of a report which includes error-related messages. Systems/(C, C++, ASM) [4] is a compiler aimed for developing mainframe applications in C, C++, or IBM assembler language. In its Web-based form, it allows users to compile up to 200 lines of source code. After submitting the code, it provides the generated assembly language source together with error-related messages. The disadvantage of both Intranet and Systems compilers is that they are meant only for compiling source code which was previously written in a text editor.

Hazel developed Codepad [6], which allows compiling of code written in thirteen different languages. In addition to a compiler, it serves as an interpreter and a collaboration tool. Namely, users have the option to run and download source code, or

share it in chat, e-mail and a social network via a short URL. However, there are two considerable limitations related to it. Firstly, it works only with output streams while input ones are not supported. Secondly, the processes of compiling and running the source code are lengthy. WebIDE [5] is a system that includes compiling, source code coloring, testing and debugging as well as data entry functionalities. At the top of the interface several buttons (comment, if, switch, while, load data, print) are located that serve to speed up the writing of source code. However, IDE does not possess features that would enable it to appropriately exploit code templates. Moreover, the input functionality seems to be broken as any attempt to use even the simplest input function results in the termination of the program due to the time limit (1 second). Ace editor [1] is a successor of the Mozilla Skywriter (Bespin) Project that serves as the primary editor for Cloud9 IDE. It represents an extension of functionalities and quality in use of all the aforementioned online compilers and thus enables coding in the cloud.

Plagiarism Detectors. JPlag [12] allows comparison of source codes written in Java, Scheme, C or C++ programming languages. In the first step, it transforms every program into a string of canonical tokens. Thereafter, using the Greedy String Tiling algorithm, it conducts pairwise comparison of token strings and thus tries to determine identical items. The percentage of token strings that can be covered is the extent of similarity between evaluated programs. Winnowing [15] is a fingerprint-based pair-wise algorithm that decomposes each program into k-grams (contiguous substrings of length k). Subsequently, it hashes each k-gram and selects a small subset of these hash values that become the program's fingerprint. In the final step, the algorithm tries to find substring matches between sets of evaluated programs. RoboProf [3] is a learning environment that allows for the detection of plagiarism during automatic grading of student programs. Before compiling, a Java applet (which runs on a student's computer) adds a watermark to the code. A watermark is a binary number composed of a year, student id, exercise id, and checksum. If two or more students submit a solution with the same watermark, RoboProf identifies them as plagiaries. The main shortcoming of this system is that, while it compares watermarks, it does not consider parts of the programming code, which makes it vulnerable to bypassing. Konecki et al. [8] developed a prototype system that compares source codes written in C and C++ programming languages. Prior to comparison, the source codes need to be simplified by applying several filters, including: placing each instruction and brackets in a separate line; omitting input / output instructions, arguments of all functions, brackets without body, declarations, blank spaces and comments, and unifying names of functions and variables. As a result, the system reports the number of similar and different lines and the percentage of similarity between the evaluated codes. Given that the prototype is developed in Visual Basic, for the purpose of our research it would be necessary to adapt both the algorithm and the filters to the Web environment.

While the end goal of the aforementioned types of systems is very similar, ways to achieving it and the features that will be included in the process can vary to a greater or lesser degree. As discussed above, all the three types of evaluation systems have their advantages and disadvantages, but none of them seems fully adequate for their specific purpose: evaluation of student programming assignments. Moreover, we feel that the current solutions are simply not suited for an enjoyable programming environment, and would like to stimulate students by introducing them to a fundamentally different, yet an enjoyable way to learn. The main objective of our research is to develop a platform that, as a complementary solution, would combine the advantages of the three types of evaluation systems.

The following section contains the introduction to basic concepts we created to facilitate more productive learning and automatic source code evaluation, that would lead to reduced teachers' workload and make the whole grading process more objective.

3 Proposed solution concept

Drawing on various shortcomings we identified in the methods and tools that are currently used for teaching and learning programming, we designed a solution aimed not only to foster student productivity and interest in the subject, but also to help instructors and various other persons involved in teaching programming to identify and objectively evaluate student assignments.

Strictly speaking, the solution we designed is composed of two parts: Web IDE and the system daemon component. A clear distinction between the two components allows for a cleaner architecture leading to flexibility in terms of user-friendliness and features.

The Web IDE component is the user-facing component in our architecture. Therefore one of the primary requirements for it is to be user-friendly and to expose a minimum of the needed features without the unnecessary interface clutter.

On the other hand, the system daemon component is used to facilitate communication between Web IDE and system components, such as the code compiler. This approach is in contrast to direct compiler access, and was selected due to the increased security and flexibility it provides, despite the additional complexity in terms of development and deployment it can cause.

In order to effectively differentiate between various implementations of our proposed solutions,

we developed a set of guidelines concerning the implementation of such a system in cases where new systems are developed. Our concept is shown in Figure 1.

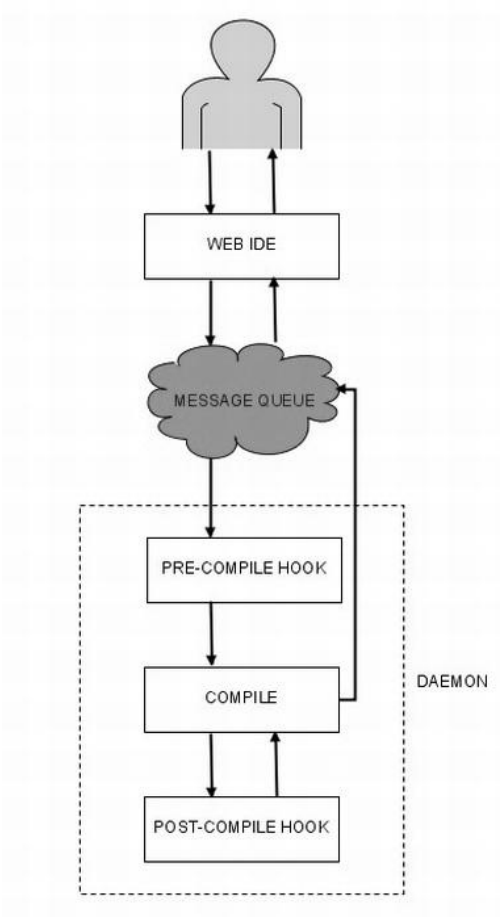

Figure 1: Concept of Web IDE

Our concept defines the necessary features set and the usability guidelines that are designed to create a de-facto standard in the implementation of similar systems aimed to aid young students in learning programming. The feature set and other guidelines to be followed are:

- Code editor with syntax-highlighting;
- Easy sign-up process;
- Code storage for easy re-editing;
- System daemon for compiling running as user;
- Facilitator between the daemon and the Web IDE;
- Helpful programming materials.

The extension of the set of guidelines aimed for convenience and further assistance to both students and instructors in the process of learning and teaching programming includes:

- No-sign up process (university or school-wide LDAP or similar);
- Public code and assignments storage for peer review and studying;
- In-code help system:
- Hook system integrated in the daemon;
- Code plagiarism detection;
- Automatic code evaluation.

The daemon is the most interesting component of the system since it has the potential to substantially improve the way programming is currently perceived both by students and instructors.

In our implementation, Web IDE communicates with a message queue, whereby it passes all the tasks that need to be executed, mainly passing the message to the lower components of the system. Once the message containing the task reaches the message queue, the daemon takes it out of the queue and executes the required tasks.

The daemon also has a built-in hook system, separated into two clearly separated categories: pre-compile and post-compile hooks. The built-in hook system allows for further system customizability and flexibility without changing the underlying code and risking system instability and problems.

The pre-compile hook could be used to verify various segments, the most important of which, in our case, is code analysis to determine the areas of program execution where input is needed. In contrast, the post-compile hook can be used for reporting compile success to the upper components, code plagiarism detection or automatic code evaluation.

In our implementation, hooks are designed in a way to allow virtually unlimited extensibility and usability, so the uses mentioned here are certainly not exhaustive. In other words, it is possible to extend the system the features needed for a particular use.

Considering various changes that this model and concept bring to the programming class, the next section focuses on presenting their possible uses in educational and other institutions for which teaching and learning programming is essential.

4. Example of a Web IDE

Our prototype is only the initial step in implementing the concept presented in this paper. It uses a *g++* compiler installed on a Linux Web server. Although some of its features are still rudimentary the IDE enables writing C++ programs, their execution and debugging. The online C++ IDE[1] was therefore developed as an example of a learning-oriented Web IDE. As shown in Figure 2, there are four sections in the Web form:

- Source code. Contains C++ source code to be compiled. Code should be copied from the clipboard.
- Compiler output. Contains compiler error report after compiling (Figure 3).

[1] The example is available at:
http://arka.foi.hr/~darados/online_ide/handler.cgi

- Program input. Because of the restrictions of the Web user interface, the textual input has to be specified separately. Each *cin* instruction reads a piece of program input (separated from others by blank or newline).
- Program output. The program output after execution.

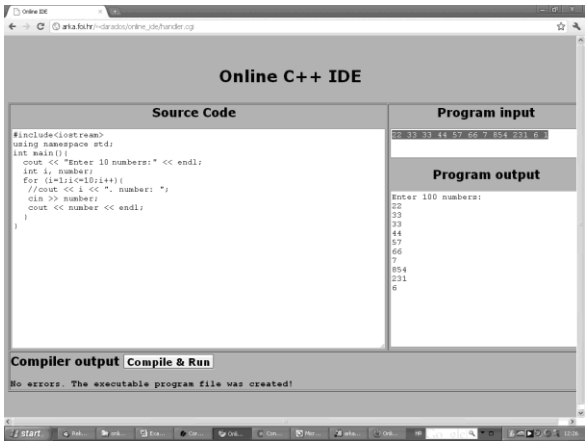

Figure 2: Online C++ IDE with program output

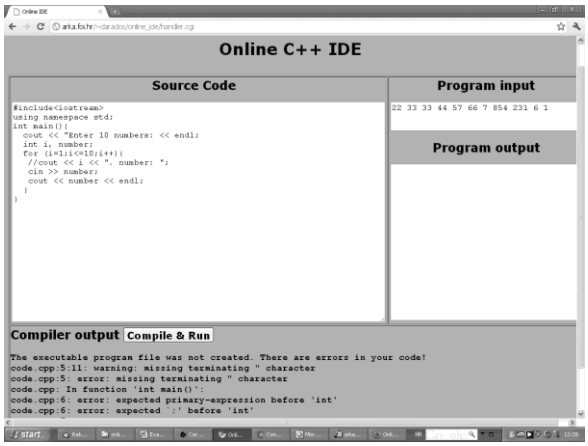

Figure 3: Compiler report

As a result, this Web IDE is very easy to use for the purpose of verifying program correctness. To ensure its correct operation, some adaptations of the IDE to the Web interface were needed. Most of the adjustments are the related program input and program output. For the purpose of achieving program input, the system code is changed internally, that is, in a way invisible to the programmer, as follows:

- The *#include "file_reading.cpp"* is added at the beginning of the C++ code with the purpose of redefining the standard *cin* object. Next, *cin* reads the textual file containing the data from the *Program input* field. The library "*file_reading.cpp*":

```
#include <fstream>
using namespace std;
class cfile:public fstream{
  public:
  cfile(){
    open("input.txt",ios::in);
  }
  ~cfile(){
    close();
  }
};
cfile cin;   //new cin object
```

As shown, the new *cin* object is from class *cfile* that inherits *fstream*. The constructor opens the file containing program input for reading and the destructor closes that file.

- All occurrences of the *cin* in program code (also invisible to the programmer) are replaced by *::cin* to specify the global scope of the program. Otherwise, *cin* would be reported as undeclared.

Once it is started, program output is redirected to the 'output.txt' textual file, and loaded to the *Program output* field in the Web form. The system needs to scale to a reasonable number of simultaneous compile and capturing processes. Further research is therefore needed to analyze the possibilities of passing the correct input and capturing the output.

5 Conclusion

Until recent years, developing applications using a Web programming interface was hardly feasible. This can be explained by the lack of appropriate development tools in form of Web applications and reduced interactivity of Web interfaces.

The above-mentioned shortcomings can be overcome by Web programming interfaces. Even small-scale Web IDE systems, such as the one developed within our research, offer certain advantages to the programmer:

- There is no need for opening program projects. It is sufficient to simply copy the program code into a textual field in the Web form.
- Programmer does not need to enter all the program input via the keyboard for each testing, so this IDE can be used for testing purposes.

Furthermore, after additional improvements, the Web IDE system could be used for educational purposes. These improvements include:

- User authentication;
- Repository of students' program solutions;
- Comparing solutions for plagiarism analysis purposes;
- User interface interactivity enhancements, e.g. by using Ajax technology.

References

[1] Ajax.org: **Ace Cloud9 Editor**, 2010, available at http://ace.ajax.org/, Accessed: 10th May 2011.

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

[3] Daly C, Horgan J: **A Technique for Detecting Plagiarism in Computer Code**, The Computer Journal, 48(6), 2005, pp. 662-666.

[4] Dignus: **Systems compilers**, 2009, URL: http://www.dignus.com/products.shtml, Accessed: 10th May 2011.

[5] Janković I, Gledec G: **WebIDE – online tool for remote teaching and programming** (in Croatian), Proceedings of the 33rd MIPRO International Convention on Computers in Education, 24th – 28th May, Opatija, Croatia, 2010, pp. 385-388.

[6] Hazel S: **Codepad**, URL: http://codepad.org/, Accessed: 10th May 2011.

[7] Hines P: **Cloud Services**, Microsoft Download Center, p. 1-16, URL: http://download.microsoft.com/documents/australia/health/MSFTCloudServicesBrochure.pdf, Accessed: 10th May 2011.

[8] Konecki M, Orehovački T, Lovrenčić L: **Detecting Computer Code Plagiarism in Higher Education**, Proceedings of the 31st International Conference on Information Technology Interfaces, 22nd – 25th June, Cavtat, Croatia, 2009, pp. 409-414.

[9] Malinowski A, Wilamowski BM: **Web-based C++ Compiler**, Proceedings of the ASEE 2000 Annual Conference, 18th – 21st June, St. Louis, MO, USA, 2000, p. 1-7, URL: http://www.eng.auburn.edu/~wilambm/pap/2000/ASEE2000_Complilers.pdf

[10] Mirzaei N: **Cloud Computing**, Pervasive Technology Institute Report, Community Grids Lab, Indiana University, 2008, p. 1-12, URL: http://grids.ucs.indiana.edu/ptliupages/publications/ReportNarimanMirzaeiJan09.pdf, Accessed: 10th May 2011.

[11] Morris DS: **Automatic grading of student's programming assignments: an interactive process and suite of programs**, Proceedings of the 33rd ASEE/IEEE Frontiers in Education Conference, 5th – 8th November, Boulder, CO, USA, 2003, p. 1-6.

[12] Prechelt L, Malpohl G, Philippsen M: **Finding Plagiarisms among a Set of Programs with JPlag**, Journal of Universal Computer Science, 8 (11), 2002, pp. 1016-1038.

[13] Rimal BP, Eunmi C, Lumb I: **A Taxonomy and Survey of Cloud Computing Systems**, Proceedings of the 5th International Joint Conference on INC, IMS and IDC, 25th – 27th August, Seoul, Korea, 2009, pp. 44-51.

[14] Radošević D, Orehovački T, Stapić Z: **Automatic On-line Generation of Student´s Exercises in Teaching Programming**, Proceedings of the 21st Central European Conference on Information and Intelligent Systems, 22nd – 24th September, Varaždin, Croatia, 2010, pp. 87-93.

[15] Schleimer S, Wilkerson DS, Aiken A: **Winnowing: Local Algorithms for Document Fingerprinting**, Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, 9th - 12th June, San Diego, CA, USA, 2003, pp. 76-85.

[16] Synodinos DG: **Web-based IDEs to become mainstream?** InfoQ.com, 2009, URL: http://www.infoq.com/news/2009/02/web-based-ide, Accessed: 10th May 2011.

[17] Truong N, Roe P, Bancroft P: **Automated Feedback for "Fill in the Gap" Programming Exercises**, Proceedings of the 7th Australasian conference on Computing education, Newcastle, NSW, Australia, 2005, pp. 117-126.

[18] Vaquero M, Rodero-Merino L, Caceres J, Lindner M: **A Break in the Clouds: Towards a Cloud Definition**, ACM SIGCOMM Computer Communication Review, 39(1), 2009, pp. 50-55.

[19] Wang T, Su X, Peijun M, Yuying W, Kuanquan W: **Ability-training-oriented automated assessment in introductory programming course**, Computers & Education, 56(1), 2011, pp. 220-226.

[20] Zamin N, Mustapha EE, Sugathan SK, Mehat M, Anuar E: **Development of a Web-based Automated Grading System for Programming Assignments using Static Analysis Approach**, Proceedings of the International Conference on Technology and Operations Management, 1st – 2nd December, Institute Technology Bandung, Indonesia, 2006.