SOURCE CODE VALIDATION AND PLAGIARISM DETECTION
Technology-rich Course Experiences

Ivana Bosnić, Branko Mihaljević, Marin Orlić and Mario Žagar
Faculty of Electrical Engineering and Computing, University of Zagreb, Unska 3, Zagreb, Croatia

Keywords: e-Learning, Programming Assignments, Validation, Plagiarism.

Abstract: Learning various programming languages in a short amount of time is a challenging task. To help students tackle several programming languages during the course of a semester, while reducing the teaching assistants' support efforts, a system named ORVViS was implemented and integrated with Moodle Learning Management System. ORVViS is used to assist students validate assignment solutions, and also to check for source code plagiarism. This paper presents the course Open computing, our motivation, system use cases, as well as our results and experiences. These observations helped us improve the assignments to better suit our teaching goals and help students learn the course concepts more quickly.

1 INTRODUCTION

Higher education computer science courses vary greatly in the breadth and depth of syllabi, from very focused courses to the general ones with a broad set of topics. Deciding how to test students' knowledge in broader courses can be a challenge, especially if the course has practical programming assignments.

The work described in this paper focuses on helping students with exercises based on various programming, scripting and markup languages in a course *Open computing*. In the course, students are given an overview of a wide range of concepts and recent technologies. As the course is quite broad and attempts to teach a lot of concepts, there is no time to thoroughly describe the programming languages and frameworks used, nor is that the course objective. However, students need to acquaint themselves with the languages, solve programming tasks based on their prior programming skills and quickly adopt new knowledge.

During the process, students encounter different types of problems, from environment setup and configuration issues to common problems associated with learning a new programming language like Java, PHP, or a descriptive language like XML or HTML. Solutions for common problems, although usually quite simple in essence, are either obtained from the provided code snippets, answers on the course forum, or by trial-and-error.

To help students overcome most common problems, automated validation of student assignments was introduced, using a system called ORVViS[1]. It is integrated with an existing LMS (Learning Management System) where students submit their assignment solutions. It checks basic code validity using validators for various languages required by the assigmnent, tests general exercise requirements and existence of all parts of solution, and finds the similarities between the submissions, which helps reduce plagiarism.

The paper is organized as follows. After the introduction and related work presented in section 2, section 3 describes the course where this system is used. System use cases are described in section 4. Experiences and results are outlined in section 5.

2 RELATED WORK

Most of modern LMSs (e.g. Moodle, WebCT, Blackboard) offer functionalities such as course management and organization, content repositories, student management and knowledge assessment, but are limited in some specific areas. One of those areas is automatic validation of solutions in courses with programming exercises, where the code, submitted by the student, is validated and the results are reported to the students as well as to the teachers.

[1] ORVViS is an acronym of "Open Computing - Validation, Verification and Simulation" in Croatian

A number of automated assessment approaches for programming assignments (Ala-Mutka 2005; Ihantola et al. 2010) can be used in computer science courses. Some of more popular free/open source automated assessment systems are CourseMarker (Higgins et al. 2003), BOSS (Joy et al. 2005), and Web-CAT (Edwards & Perez-Quinones 2008). Modern automated assessment systems provide some level of integration with an LMS. Such functionalities could be developed as extensions or as individual systems integrated with LMSs.

Since our Faculty (FER) already extensively uses a combination of FER e-Campus CMS and Moodle (Tomić et al. 2006), compatibility with Moodle was our goal. So far only a few similar systems have been developed, validating VHDL, Matlab, SQL, assembly languages and other programming languages (C, Java etc.) (Ihantola et al. 2010). As those products do not cover a required set of programming, scripting and markup languages we use, and cannot be easily adapted, we found them not to be a solution to our problem.

Student cheating was always a problem on computing courses (Wagner 2004) and is becoming easier with widespread communication devices and software, which allow simple sharing of solutions and code in a matter of seconds.

Another problem is the lack of ethics, since a large number of students do not perceive some actions (e.g. unsolicited collaboration on assignments, public posting of solutions, reuse of past year solutions) as serious offences ((Dick et al. 2001; Sheard & Dick 2003; Cosma & Joy 2006), even if these are explicitly prohibited by the Student code of conduct. Detection of similarities between student assignment solutions can be used to prevent cheating (Dick et al. 2003). The presence of such a system, publicly announced, discourages cheating. The presentation of validation results is a reminder that the solutions are validated and similarities are going to be detected.

Since most of computer science assignments use scripting, markup and programming languages, it is necessary that the anti-plagiarism solution can analyze source code written in such languages to detect similarities. Most of the code comparison tools process only code written in the most common languages (Java, C, C++, and C#). In addition, the tool should be easily configurable to support other languages, and also to compare code structure with string tokenizing and similar techniques.

A number of free/open source comparison tools can be used to detect plagiarism (Goel & Rao 2005) (Lancaster & Culwin 2004), including Sherlock[2], BOSS – Sherlock[3], CtCompare[4], JPlag[5], Plaggie[6], MOSS[7], PMD CPD[8], and Comparator[9], but only some of them could be easily integrated with ORVViS and Moodle. Two tools - both named Sherlock - a simple application with a command-line interface, and another, a standalone Java application that supports languages within the C syntax family, but can also detect similarities in other markup and scripting languages, were selected as an appropriate mix of usability and features.

3 COURSE OVERVIEW AND MOTIVATION

The *Open computing* course is taught to 50 - 100 third-year students of computing at our faculty. It gives a broad overview of various aspects of openness in hardware, software and user experience, with an emphasis on standards, their purpose, utilization, and means of establishing them in the world of distributed and interactive information services. It also includes topics like concepts of open systems, open technologies and their importance, as well as the nature of open culture and open licenses.

The course is designed as a blended e-learning course. Bi-weekly assignments are the main course activity, described in detail in the following chapter.

3.1 Course Assignments

The objective of assignments is to illustrate the presented concepts in exercises focused on practical use of open internet protocols and web technologies. In this way the students get a short hands-on experience by integrating various open technologies.

There are six assignments in the course. Each assignment builds on previous ones, until students complete a simple, but fully functional, web site/application with search capabilities. Although students share the same assignment topic, there are eight topic instances, e.g., a DVD store, a library, a document management system or a phone book.

The assignment descriptions include detailed

[2] Sherlock - sydney.edu.au/engineering/it/~scilect/sherlock
[3] BOSS - www.dcs.warwick.ac.uk/boss
[4] CtCompare - minnie.tuhs.org/Programs/Ctcompare
[5] JPlag - www.ipd.uni-karlsruhe.de/jplag
[6] Plaggie - www.cs.hut.fi/Software/Plaggie
[7] MOSS - theory.stanford.edu/~aiken/moss
[8] PMD CPD - pmd.sourceforge.net
[9] Comparator - www.catb.org/~esr/comparator

instructions, assignment set-up and example code snippets. Students work individually and may iteratively upload their code to Moodle before a final solution. They present solutions to the assistants to discuss their approach, code, and functionality.

Here are the short assignment descriptions:

1. HTML and CSS – in the first assignment the students should create a skeleton of two web pages using standardized and validated HTML code with CSS design;
2. XML, XSL and DTD or XSD – the second assignment demonstrates structuring a data model in an XML file, using Document Type Definition or XML Schema Definition for data validation, and transforming XML data to XHTML with XSL;
3. PHP and DOM – in the third assignment HTML mock-up previously created is now extended with logic and search functionality of a simple web application, implemented in PHP using DOM to read XML data;
4. Java and XML – in the fourth assignment students create a Java application that parses data from a text file and creates a structured XML data file using SAX or DOM;
5. Java Servlets – in the fifth assignment, the Java application is converted to a Java Servlet and deployed on an application server to produce XML input data for PHP;
6. JavaScript and AJAX – the final assignment integrates all parts of the application, and enhances the client-side web page with detailed information fetched from the application server using AJAX, presented in JavaScript tooltips.

Over the course of the years, we have identified some issues related to student work:

- lack of student experience in object-oriented programming, especially in languages like Java, PHP and JavaScript;
- dispersion of information on fast-changing technologies on the web;
- lack of detailed step-by-step instructions required to configure the work environment;
- inexperienced students lacking time to finish the assignment.

In addition to these issues, the teaching staff has worked on reducing the efforts to answer relatively simple recurring (beginners') questions, not allowing or preventing invalid submission solution files, ensuring the submitted solutions follow some basic structure, requirements and standards, as well as reducing plagiarism. As Moodle lacks tools to tackle these problems, resolving assignment validation and plagiarism issues were our major motivators in development of the ORVViS system.

4 ORVVIS SYSTEM OVERVIEW

ORVViS is an assignment validation system integrated with Moodle, designed to help the students resolve problems with their programming assignments. ORVViS provides separate validation for each technology required by the assignment.

The tool consists of two main units: validation core and Moodle LMS integration module. In order to seamlessly introduce ORVViS to students, validation was integrated with LMS using Moodle APIs to fetch student submissions. It works in the background, with web-based administrative interface for managing the submissions. To use ORVViS, students do not need to make any additional actions other than submitting their work on Moodle.

ORVViS was designed as a modular system, and new plugins can be developed based on future course needs. At this moment, available validation plugins, developed using freely available validators (such as HTML Tidy[10], Cssutils[11], DOM, Matra[12], Lint[13], and PMD[14]) can be used to validate HTML, CSS, XML, XSL, DTD, PHP, Java and JavaScript.

4.1 Use Cases

Assignment Setup

To use the system, the assignment should be created in Moodle. The teacher then creates a new task in ORVViS, and configures it with the Moodle assignment ID, submission start and finish dates (can be different from the ones in Moodle), Moodle server URL, file names (or file extensions) expected in submissions, and the associated validation plugin.

Validation

Students use ORVViS transparently on each submission to Moodle. Using the Moodle API, ORVViS will check the file structure of all submitted files, and run the validation using the associated plugins. After the validation is complete, a detailed report will be sent to the student's e-mail address, with the validation results and possible errors. Students can upload their solutions iteratively, which triggers the validation and mails the report on each submission. Students should finalize the submission once it is complete.

When the assignment deadline is reached,

[10] HTML Tidy Library Project - tidy.sourceforge.net

[11] CSS Parser and Library for Python - cthedot.de/cssutils

[12] XML DTD Parser Utility - matra.sourceforge.net

[13] JavaScript Lint - www.javascriptlint.com

[14] PMD - pmd.sourceforge.net

ORVViS creates a cumulative report and sends it to the course administrators. This report contains an overview of the received submissions (number of submitted solutions, number of successfully validated solutions, etc.), as well as a collection of submitted files prepared for further analysis.

Plagiarism

After the deadline, ORVViS compares all submitted files using an external plagiarism detection tool Sherlock[15] to check for similarities between submissions over a similarity threshold. The final assignment report sent to the teaching assistants includes warnings of submissions crossing the threshold. Sherlock program used here has a reduced feature set, and this is only the first step of plagiarism detection. ORVViS prepares the submitted files for analysis with another plagiarism detection tool Sherlock from the BOSS package[16], with graphical view of similarities.

Depending on threshold level setup, it can report false positive results, especially for assignments where code templates were given. As a final step, the course staff manually reviews all of the detected submission pairs to confirm the similarity.

5 RESULTS AND EXPERIENCES

We group our experiences around two topics: plagiarism detection and assignment validation.

5.1 Plagiarism Detection

In the pilot period (academic year 2006/07), students were testing the system on some of the assignments, and helping to find bugs. In the academic year 2007/08, we started with the plagiarism component, motivated by a big issue of copying assignment solutions the year before. In that year, 113 students were enrolled in the course.

At the course start, students were informed that assignments have to be done independently, the solution files uploaded to the server will be compared for potential plagiarism, and every suspicious case will be thoroughly analyzed and submitted to the Faculty's ethical committee.

After analyzing the first assignment submission, we found two similar solutions and presented the graphical results of the comparison to the students. In the third assignment, another pair of similarities was observed, students were cautioned and this was reported to Faculty ethical committee. The solution pair detected in the first assignment was virtually identical (and thus easily detected). The similarity of the pair detected in the second assignment was above 80%, as demonstrated in the Figure 1. The detection process detected two false positives, our own submissions used to test the system.

The fourth, Java-related assignment, with a steeper learning curve, was an unpleasant surprise. ORVViS isolated 6 suspicious cases of plagiarism with 17 students involved. This assignment was harder than previous and more time consuming, but considering that the students were made aware of that beforehand, this had to be addressed again.

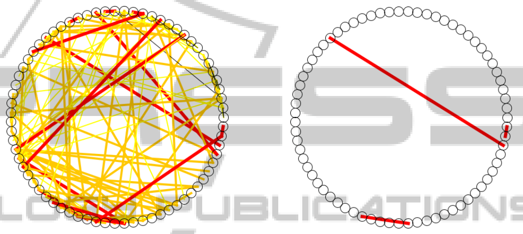

Figure 1: Graphical report on similarities – 0% and 80% similarity threshold.

These analyses were consistently conducted during the semester. From the next year onwards, regular ORVViS usage didn't find any cases of similarities higher than what was expected, given that code excerpts and examples were provided. Based on our experiences, we concluded the following:

- students should be explicitly informed in advance that such a system will be used;
- comparison technology and results should be shown to students, to persuade them that teachers use these regularly, instead of just having the possibility of checking (although demonstrating the system could lead to more "creative" ways to bypass it, we find this method to be more fair);
- perseverance in decisions is needed, as students wouldn't get used to completely independent work all at once, after the first recognition of plagiarism.

5.2 Assignment Validation

To perform assignment validation and plagiarism detection, exact assignment submission structure had to be enforced. During the first testing year, 2007/08, the system reported the following structure errors in first assignment submissions:

- 52% of solutions were incorrectly named;

[15] Sherlock - sydney.edu.au/engineering/it/~scilect/sherlock

[16] BOSS - www.dcs.warwick.ac.uk/boss

- 10% of submissions were packaged as RAR instead of ZIP archives;
- 25% of archives had incorrect file structure;
- 4% submissions had incorrectly named files in the archive.

After initial testing and improvements, we started using all ORVViS features in year 2009/10. Here we present statistics and observances from two years we used the system. In 2009/10, 70 students enrolled, and in 2010/11, 53 students enrolled.

As stated, students can upload the assignment solution more than once and receive a new validation report for each submission. We compared the total number of submissions per assignment to the number of final submissions, shown in Figure 2.

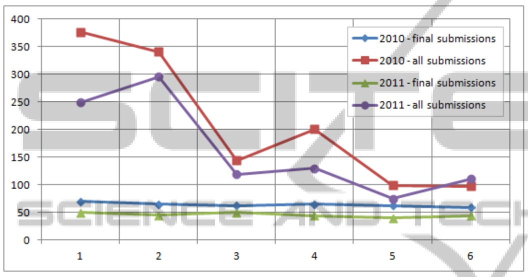

Figure 2: Number of submission instances per assignment.

The number of final submissions is always similar to the number of students in the course (the difference is due to some students' late upload). The total number of submissions shows the following:

- in the first two assignments, the students have a high number of submissions – around 5 per student. Students use the validation service a lot, to solve submission structure issues, and to check conformance to HTML&CSS or XML&DTD formats, which is a tedious process if done manually;
- the third assignment (PHP) is usually straightforward with one new technology to learn. By that time the students got used to the system, so the ratio of total/final submissions drops to ~2.4 per student;
- the fourth assignment introduces Java, which presents the students with a completely new environment, so there is an increase in the number of submissions;
- the final assignments become easier again, with the submission ratio of about 2.

There is a similarity between chart trends for 2010 and 2011. Teaching staff can focus on the issues demonstrated in these charts (e.g. the Java assignment), and work on resolving them.

The second set of charts (Figures 3 and 4) shows the validation results per assignment, for years 2010 and 2011, respectively. The lowest part of bars shows the percentage of fully correct submissions – the ones where all submitted files have passed the validation. The middle part shows the percentage of the solutions where at least one file was validated correctly. The upper part shows fully incorrect ones – where no files have passed the validation.

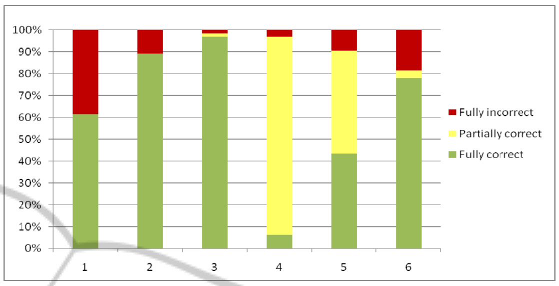

Figure 3: Level of validation correctness – year 2010.

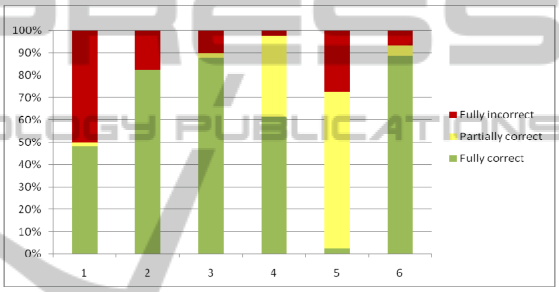

Figure 4: Level of validation correctness – year 2011.

The results included in Figures 3 and 4 can be compared to the chart of total/final submissions (Figure 2). In the first three assignments, the number of fully correct submissions increases, while the total number of submissions per student decreases. The fourth and fifth assignments (Java-related) show a big decrease in fully correct submissions, due to a steep learning curve and the environment setup.

Based on this data and observances during the course, here are our experiences and lessons learned:

- ratio of correctness charts, combined with detailed error logs, can help the staff to analyze the particular assignments, and give students greater attention when needed;
- this system helps the staff to ensure the student did in fact write the solution, instead of just submitting something random, and discussing the fellow student's work;
- staff can view a detailed report on each submission before they meet, so they can help where mistakes were made;
- the number of forum messages related to simple problems regarding environment setup and configuration, has been reduced, which

helps both students and teaching staff focus on more relevant discussions;

- due to a number of technologies used in these assignments, a set of validators integrated with LMS makes it easier for students to check their assignments, instead of using validation services one by one;
- validations can help create a successful environment configuration (for instance, validation of XML configuration files for the application server);
- some validators have been set to be more sensitive and report more detailed warnings than a typical compiler would. This was effective in cases where students used newer compiler and runtime versions (e.g. Java) than those available on our servers, as additional warnings would give students a hint where to start looking for problems.

6 CONCLUSIONS

The experiences presented here give us a good starting point to continue using and upgrading ORVViS. The students are satisfied with provided help, while the staff can obtain relevant information on the students' behavior in solving the assignment problems. Whenever we have asked the students for some kind of help related to the system, such as testing, they did it enthusiastically, as they consider it to be beneficial and time-saving.

The downside is that students start to depend on it, and stop validating their solutions by themselves, which was observed in our initial experience report (Bosnic et al. 2010): the number of successful submissions dropped significantly after a few days the system was unavailable. Even with such systems in place, the students should be capable of creating valid solutions without help from the system.

It should be noted that ORVViS currently does not support grading nor checking the most of the assignment's complex semantic requirements (posed in a natural language), and currently focuses mainly on syntax. Such features are a well-worth asset and we plan to extend the system in the future. However, concerning the main course objective, the complexity of content taught, and focus on understanding the underlying open processes, we feel the need to discuss the solutions face-to-face.

Additional future work consists of integrating the staff / administrator interface with Moodle LMS as well. We expect that the additional APIs and plugin tools, available in new Moodle 2.x version, would simplify the task of integrating two systems.

ACKNOWLEDGEMENTS

This work is supported by the Croatian Ministry of Science, Education and Sport, under the research project ZP0361965 "Software Engineering in Ubiquitous Computing".

We would like to thank our former student Darko Ronić and other students under his supervision for their work on ORVViS application.

REFERENCES

Ala-Mutka, K.M., 2005. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education*, 15(2), p.83-102.

Auffarth, B. et al., 2008. System for Automated Assistance in Correction of Programming Exercises (SAC)

Bosnic, I., Orlic, M. & Zagar, M., 2010. Beyond LMS: Expanding Course Experience with Content Collaboration and Smart Assignment Feedback. *International Journal of Emerging Technologies in Learning iJET*, 5(4).

Cosma, G. & Joy, M., 2006. Source-code plagiarism: A UK academic perspective. *I Can*, (422), p.74.

Dick, M. et al., 2003. Addressing student cheating. *ACM SIGCSE Bulletin*, 35(2), p.172.

Edwards, S.H. & Perez-Quinones, M.A., 2008. Web-CAT: automatically grading programming assignments. *ITiCSE 08 Proceedings of the 13th annual conference on Innovation and technology in computer science education*, 3(3), p.60558-60558.

Goel, S. & Rao, D., 2005. Plagiarism and its Detection in Programming Languages. *Environment*.

Higgins, C. et al., 2003. The CourseMarker CBA system: Improvements over Ceilidh. *Education and Information Technologies*, 8(3), p.287–304.

Ihantola, P. et al., 2010. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research Koli Calling 10*. ACM Press, pp. 86-93.

Joy, M., Griffiths, N. & Boyatt, R., 2005. The BOSS online submission and assessment system. *Journal on Educational Resources in Computing*, 5(3), p.2.

Lancaster, T. & Culwin, F., 2004. A Comparison of Source Code Plagiarism Detection Engines. *Computer Science Education*, 14(2), p.101-112.

Sheard, J. & Dick, M., 2003. Influences on cheating practice of graduate students in IT courses: what are the factors? In *Proceedings of the 8th annual conference on Innovation and technology in computer science education*. ACM, p. 49.

Tomić, S. et al., 2006. Living The E-Campus Dream. In A. Szucs & I. Bo, eds. *Proceedings of the EDEN Conference*. Vienna, Austria: European Distance and E-Learning Network, pp. 644-650.

Wagner, N., 2004. Plagiarism by student programmers. San Antonio, TX, USA.