NESCUME – A SYSTEM FOR MANAGING STUDENT ASSIGNMENTS

Vlado Glavinić, Marko Čupić and Stjepan Groš

Faculty of Electrical Engineering and Computing,
Department of Electronics, Microelectronics, Computer and Intelligent Systems,
University of Zagreb, Zagreb, Croatia

{vlado.glavinic|marko.cupic|stjepan.gros}@fer.hr

ABSTRACT

This paper describes Nescume – a Web enabled package for managing student (lab) assignments. The package has been designed targeting a number of requirements met in teaching software-based courses. The paper discusses objectives for Nescume development, presents its capabilities such as source code comparison for cheating detection, testing of developed program and source code archiving, and shows its architecture. Furthermore, the paper explains how the system can be used to improve both the course as well as students quality. Integration with the course management software is also considered, as illustrated by the WODLS package.

KEYWORDS

testing of student work, source code comparison, Web-enabled laboratory management, plagiarism

1. INTRODUCTION

In the last decade programming permeated into quite a number of university courses, what is particularly true for the area of computer science. Following the progress in computing power, some other more classical areas evolved as well, like formal modeling and verification for both software and hardware. For the latter, the use of hardware description languages (HDLs) additionally supports circuit/systems simulation. On a more abstract level this reduces to (i) writing a program, be it either production software or specification, and (ii) testing. Programs are typically tested for errors, specifications are formally verified (e.g. using model checkers) and hardware is typically simulated to assess its correctness, time delays, performance etc. In many courses students are confronted with those tasks through a course's laboratory, thus inducing the creation of a system for supporting automated verification of student solutions.

With the popularization of the Internet, students can now much easier share their knowledge but are also increasingly tempted to cheat by distributing solutions to assigned problems. The "management of cheating" is defined as a three-stage process that consists of (i) cheating pre-emption, (ii) cheating detection and (iii) response to cheating [1]. We believe that software for cheating detection can be used in the first two stages, since (i) its mere existence discourages students to even try to cheat, and (ii) it is supposed to effectively detect those who cheated anyway. For these and other reasons to be explained later, we have developed Nescume (NEtwork based Software acCUMulation and Evaluation) – a system for managing student assignment, which is one building block of our comprehensive effort to ensure efficient and objective student treatment within our courses. As an aside, let us mention another system – WODLS (Web Oriented Distance Learning System) – we had previously built with the same objectives, which has been used for some time already for testing student knowledge [2], [3].

2. MOTIVATION

Nescume has been designed by having the having in mind the following goals, which are characteristic of the educational process:

- *Archiving of student assignment solutions* – the system should archive all the submitted work and track all the necessary data about it (author, lab exercise, course, academic year, etc.).

- *Cheating elimination* – the system should either prevent or detect the majority of attempts to cheat, and specifically to plagiarize [4], [5]. Cheating is threatened, at least partially, by having students know that their work will be archived (what generates a psychological barrier) and by using source code comparison methods to detect suspiciously similar programs.

- *Testing student solutions for correctness* – the system must provide easily configurable means for assessing student solutions, either formally or for some typical errors. The tests can be prepared beforehand, as part of a course lab.

- *Suggesting quality improvements* – the system has to provide means to analyze student solutions and provide them with quality improvement suggestions (e.g. by critiquing [6]). E.g. if a student submits a C source code, an analysis could be performed to see if insecure functions are used instead of secure ones (`strcpy/gets` vs. `strncpy/fgets`). Coding style can also be assessed. This can encourage students to write secure and clean code, and on the long run cultivate quality programmers.

- *Ensuring archive format restrictions* – the system should enable upload of document or/and (various formats of) archives, and their content verification.

3. KNOWLEDGE ASSESSMENT BY INTERPERSONAL DISCUSSION

To assess a student understanding of a software lab exercise and especially the originality of her/his solution, she/he is typically questioned, which is usually performed in a three-step process:

1. Ask the student to demonstrate her/his program operation under typical error-disclosing conditions.

2. Ask the student what the program is supposed to do, and how it would do it under some hypothetical conditions.

3. Ask the student to orally comment the source code submitted, and to identify the parts of code responsible for some behavior observed during its run.

Step 1 is usually performed by instructing the student how to set up the testing environment, and by subsequently observing and comparing the program operation with the expected behavior. The examiner usually checks both the program outputs and time constraints (if the program includes time dependent operations). Because of the examiner's time limitations, all scenarios cannot be tested, which is the reason why the next step is so important.

In step 2 the student understanding of the problem assigned is assessed. A certain typical situation is specified, and the student is then asked to explain how her/his program would react under these setting (without consulting the source code), what can be further experimentally

verified. If the program performs differently then expected (or even worse, differently than the student claimed it would), this can be further analyzed in step 3.

In the final step 3, an attempt is made to discover those students who learned the program functionality and took someone other's solution, effectively being ignorant on the role of particular code portions. E.g. the function `recvfrom` applied on sockets blocks the program execution until delivering data. A part of the assigned problem was to implement program blocking for a predefined amount of time. The solution is illustrated in Figure 1, where the function `select` is used to implement the limited time blocking. The student was asked where in the code the blocking occurs (while testing showed that the time-limited blocking was indeed implemented as in Figure 1)? The student reported that `recvfrom` blocks execution. She/he could, however, not explain why the observed program behavior was behaviourC, which was a clear indication that she/he did not understand what `select` was used for, and that he did not write this program himself.

```
select(socket, time_constraint, …)
If notTimeout() Then
   Recvfrom(socket) //to read a packet
   If packetReceivedSuccessfully Then
      behaviourA // process data
   Else
      behaviourB // report error
   End If
Else
   behaviourC // timeout occurred
End If
```

Figure 1. Pseudo-code for limited time socket operation

Using the described methodology, a rather thoroughly examination can be performed. The drawbacks are, however, twofold: the time needed to examine one student is unacceptably large (about 30 minutes) and objectivity is compromised if there is more than one assistant (meaning different criteria, longer or shorter questioning, etc).

4. DETECTION OF PLAGIARISM

In recent years plagiarism (considered as a form of cheating [7]) has become more common among students. Some studies [8], [9], [10] came to distressing conclusions. E.g. the following misdeeds are by students perceived as not too serious offences [9]: (i) collaboration on assignments meant to be completed individually, (ii) posting to Internet newsgroups for assistance, or (iii) submitting a friend's assignment from past running of the subject. To prevent cheating [11], [7], steps are to be undertaken to discover the student involved [12], which usually means that sources submitted by students must be compared.

There exist already a number of freely available source comparison programs like Comparator-2.5 [13], CtCompare-1.3 [14], Sherlock [15] and Moss [16]. Table 1 summarizes plagiarism detection methods used by these programs. However, some of the above programs are erroneous, and some cannot be integrated into a student management system. Also, some of the programs are language dependent (e.g. having a C pre-processor).

Table 1. Plagiarism detection software and their methods

Program	Plagiarism detection method
Comparator	Computation of hashes of overlapping n-lines long shreads
CtCompare	Token sequence matching
Sherlock	Calculates digital signatures from word series
MOSS	Calculates k-gram fingerprints [17]
Nescume SC proto1	Heuristic token stream matching with certainty factors with language dependant preprocessing

There are also some other questions to be addressed like (i) comparison flexibility (e.g. comparing program sources vs. HDL specifications) and (ii) source code transformations (e.g. it is not obvious how to classify syntactically different, but necessarily semantically identical solutions to particular assignments). Regarding the latter, it should be noted that an assignment could be successfully programmed by different source codes, provided it behaves in the expected manner. This consequently defines two extremes for the "plagiarism detector" functionality: find syntactically identical sources (what any savvy cheater would avoid), and find semantically identical sources (what is otherwise expected as the correct solution). Just as humans would, the solution should obviously be for the "plagiarism detector" to search for patterns and structures in sources, what is implemented in the Nescume prototype.

5. NESCUME ARCHITECTURE AND IMPLEMENTATION

5.1. Architecture

Nescume is composed of a number of modules (see Figure 2.): the controller module is the core of the system; the database module makes a persistence layer for used data and communicates with the database proper, the program repository module manages student programs, the testing module performs program testing, the similarity checking module performs source comparison, and the two user interface modules enables system accessibilty over the Internet (Web module) and over command-line (Command-Line module). Such architecture resulted from the following two requirements: support of pluggable program test implementations, and pluggable plagiarism detection solutions.

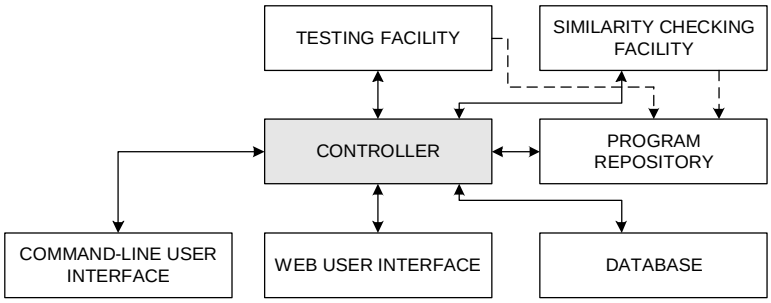

Figure 2. Architecture of Nescume

5.2. Model of Lab Exercises

The Laboratory consists of one or more Laboratory Exercises. Each Laboratory Exercise is subdivided into one or more Tasks, while each Task contains one or more Jobs. A Job is an elementary unit of student work. In each Laboratory Exercise a student can be assigned one or more Tasks to solve, what includes solving all of a Task's Jobs. A Job is solved when its solution is both uploaded onto a server and locked so that the student cannot further change it. E.g. for a Lab Exercise "Communication in the Data Link Layer" the first Task could be "ARP protocol" that in turn consists of two jobs: "ARP listener" (retrieving and printing ARP packets from the network), and "ARP query" (sending ARP queries and printing responses). A second Task could be "RARP protocol" with a similar job structure as before.

5.3. Implementation Details

Nescume has been built having in mind portability and open source tools. It is implemented in Java, on top of AppFuse framework [18], as a Web application. It is currently running under the Linux operating system in Apache Tomcat [19], and uses the open source relational database MySql [20].

6. How Nescume Enhances Course Execution

Using Nescume as a part of lab exercises can contribute to efficiency. When a student completes her/his work, she/he uploads it onto the server, which in turn automatically starts the execution of related tests, and informs the student on the results. A typical test is the compilability test – the feasibility for a student source to be compiled into an executable program on an independent system. If the source passes it, a series of program correctness tests can be performed automatically, thus eliminating the need for the assistant to instruct each student how to set up the test environment and demonstrate the source correctness. This practically means that there is no need to perform step 1 of the previously described testing procedure. If some test fails, the student can analyze (using test description and results) her/his source and fix it. Nescume supports description of tests and their interdependencies, and can execute tests in correct order. The system is also extensible, allowing new tests to be added easily. For a given Task (or Job), tests to be performed are described in appropriate test-descriptors, which are XML files containing all of the information needed to execute the requested tests (see Figure 3). For tests which are inherently insecure (such as compiling and running programs requiring administrative permissions), an isolated test environment can be easily built by using some PC emulation software like e.g. VMWare [21]. Having Nescume track all test executions, if something goes wrong, the author of the offending program can then be easily found.

```
<tests>
 <prepare name="prep" jobs="1" />
 <compile name="compilation1"
  dependsOn="prep" compiler="gcc"
  options="-Wall -pedantic"
  outputName="arp.exe" sources="1"
  compiledId="1" />
 <checkQuality name="qualityVerification"
  dependsOn="compilation1"
  program="splint" options=""
  sources="1" />
 <test name="test1"
  dependsOn="compilation1" />
</tests>
```

Figure 3. An example of multiple test definition with interdependency specification

Nescume is compliant with all of the requests stated in section 2, and includes an early implementation of source code similarity checking for plagiarism detection. Although this implementation isn't an actual threat to cheaters (yet), the fact that it exists has proven as a rather strong cheating deterrent.

7. Conclusion

Nescume is a system intended to ensure efficient and objective examination of student lab work. It is already successfully used at the Faculty of Electrical Engineering and Computing. Combined with WODLS's capabilities to examine student knowledge through quizzes, further objectivization of the examination process can be achieved (this is especially true for step 2 of the testing procedure). It has potential for wide usage, because it can support a variety of courses. E.g. instead of checking C program correctness, either formal checks of uploaded model descriptions or simulation of uploaded hardware design could be performed.

Nescume must be further improved e.g. by devising a better source comparison algorithm, whose performance should be tested against non-C-type sources. We plan to perform this on a batch of VHDL sources to be generated in a digital design lab.

Acknowledgements

This work has been carried out within project 0036033 *Semantic Web as Information Infrastructure Enabler*, funded by the Ministry of Science, Education and Sport of the Republic of Croatia.

REFERENCES

[1] Dick, M., Sheard, J., Bareiss, C., Carter, J., Joyce, D., Harding, T., Laxer, C. "Addressing Student Cheating: Definitions and Solutions", *ACM SIGCSE Bulletin*, Working group reports from ITiCSE on Innovation and technology in computer science education, Volume 35, Issue 2, June 2002, pp. 172-184.

[2] Glavinić, V, Čupić, M, Groš, S. "WODLS – A Web Oriented Distance Learning System", *Proc. 12th IEEE Mediterranean Electrotechnical Conference – MELECON 2004*, Vol. II, May 12-15, 2004, pp. 747-750.

[3] Groš, S. *Distance Learning System Based on Service Oriented Architecture*, MSc Thesis, Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, 2004.

[4] Harris, R. *Anti-Plagiarism Strategies for Research Papers*, http://www.virtualsalt.com/antiplag.htm, 2002.

[5] Clough, P. *Plagiarism in natural and programming languages: an overview of current tools and technologies*, Research Memoranda: CS-00-05, Department of Computer Science, University of Sheffield, UK, 2000. http://www.dcs.shef.ac.uk/~cloughie/plagiarism/

[6] Qiu, R, Riesbeck, C. K. "Making critiquing Practical: Incremental Development of Educational Critiquing System", in *Proc. Int'l Conf. on Intelligent User Interfaces – IUI'04*, Madeira, Funchal, Portugal, January 13-16, 2004, pp. 304-306.

[7] Harris, J. K. "Plagiarism in computer science courses", in *Proc. Conf. on Ethics in the computer age*, November 1994, pp. 133-135.

[8] Dick, M., Sheard, J., Markham, S. "Is it okay to cheat? – the views of postgraduate students", *ACM SIGCSE Bulletin, Proc. 6th Annual Conf. on Innovation and Technology in Computer Science Education*, Vol. 33, Issue 3, June 2001, pp. 61-64.

[9] Sheard, J., Dick, M., Markham, S., Macdonald, I., Walsh, M. "Cheating and plagiarism: perceptions and practices of first year IT students", *ACM SIGCSE Bulletin, Proc. 7th Annual Conf. on Innovation and Technology in Computer Science Education*, Vol. 34, Issue 3, June 2002, pp. 183-187.

[10] Sheard, J., Dick, M. "Influences on cheating practice of graduate students in IT courses: what are the factors?", *ACM SIGCSE Bulletin, Proc. 8th Annual Conf. on Innovation and Technology in Computer Science Education*, Vol. 35, Issue 3, June 2003, pp. 45-49.

[11] Kaczmarczyk, L. C. "Accreditation and student assessment in distance education: why we all need to pay attention", *ACM SIGCSE Bulletin, Proc. 6th Annual Conf. on Innovation and Technology in Computer Science Education*, Vol. 33, Issue 3, June 2001, pp. 113-116.

[12] Carter, J., Ala-Mutka, K., Fuller, U., Dick, M., English, J., Fone, W., Sheard, J. "How shall we assess this?", *Working group reports from ITiCSE on Innovation and technology in computer science education*, Thessaloniki, Greece, June 30 - July 02, 2003, published as *ACM SIGCSE Bulletin*, Vol. 35, Issue 4, June 2003, pp. 107-123.

[13] *Comparator* (Source comparison software), http://www.catb.org/~esr/comparator/

[14] *CtCompare* (Source comparison software), http://minnie.tuhs.org/Programs/Ctcompare/

[15] *Sherlock* (Source comparison software), http://www.cs.usyd.edu.au/~scilect/sherlock/

[16] *Moss* (Source comparison software), http://www.cs.berkeley.edu/~aiken/moss.html

[17] Schleimer, S., Wilkerson, D. S., Aiken, A. "Winnowing: local algorithms for document fingerprinting", *Proc. 2003 ACM SIGMOD Int'l Conf. on Management of Data*, San Diego, CA, June 2003, pp.76-85.

[18] *AppFuse Framework*, https://appfuse.dev.java.net/

[19] *Apache Jakarta Tomcat* (Servlet container), http://jakarta.apache.org/tomcat/

[20] *MySQL*, http://www.mysql.com/

[21] *VMWare* (Virtual computer emulator), http://www.vmware.com/