Automated Testing of Case Studies in Programming Courses

M. Novák*, M. Biňas*
* Department of Computers and Informatics, Technical University of Košice, Slovakia
marek.novak@tuke.sk, miroslav.binas@tuke.sk

Abstract — **The majority of Learning management systems (LMS) provide the functionality only for evaluation of built-in tests activities without any support for evaluation of case studies, which are frequently used in programming courses. These have to be checked manually by an instructor, what is time consuming especially in case if courses are attended with a high number of students. In this paper we propose an automated submission process decomposed in several phases. We provide a detailed description of particular phases with selected tools as jPlag and PMD. The aim of our work is to combine existing tools into one coherent system connected with LMS (Moodle) and IDEs and not to go a way of implementing a new standalone validation and assignment system. We present also some partial results and experiences using jPlag and PMD tools on our courses – Programming, Object oriented programming and Technologies Java.**

I. INTRODUCTION

One of the key parts of teaching is the evaluation process and examination. Its aim is to objectively assess students' knowledge that they gained by attending a course. Test preparation requires a plenty of time, however the evaluation may be even more time consuming.

Automated test evaluation is supported by the majority of currently used Learning management systems (LMS) as Moodle, Sakai, OLAT, etc. [1]. However, the problem occurs when we would like to evaluate student's skills by implementing a program or working on some case study. LMS systems provide mainly the environment to upload documents relevant for a case study to the system, but any ability to evaluate at least some formal aspects of the student's work is missing. These tasks have to be done manually by the lecturer.

This paper is dedicated to the problem of automated evaluation of case studies in programming courses. Related work in automation of programming assignment submission and evaluation is mainly oriented in developing new or enhancing existing tools with a specific purpose as MARMOSET [2] or ProgTEST [3]. Rather than going this way, we decided to combine existing tools into one functional system with added value. In this paper we propose an architecture overview of a web service in the scope of existing environment of LMS and IDEs already used in our courses with some partial tests of submission process sub-steps.

The paper is organized as follows. In Section II we provide an overview of submission process decomposition into sub-processes and in Section III we describe these sub-processes in more detail with deeper focus on used tools as *jPlag* and *PMD*. In Section IV we propose an architecture overview of a future system connected to an existing infrastructure. The Section V describes some experimental results acquired during two years running courses *Programming*, *Object oriented programming* and *Technologies Java*.

Although we target on the area of case studies from programming courses, concepts presented in this paper may be used also in non-programming areas.

II. SUBMISSION PROCESS DECOMPOSITION

Testing of a programming case study involves verification if the program solves given problem and fulfills required criteria and measurement the quality of a program. Assignment testing may be divided into two types: static analysis and dynamic testing referring to whether a program needs to be executed while it is being assessed [4]. Dynamic testing is furthermore divided on a white-box testing, when the assessment process is done by looking into source code structure, or black-box testing, when the assessment is done by functional behavior. In addition there are evaluated characteristics as correctness, complexity, reliability or style of a program [5]. Our work focuses on both static and dynamic testing in separate steps of testing procedure.

The submission process can be decomposed on several sub-processes, where each of them is used for evaluation of specific part of submitted case study. The results of some sub-processes may be binary (the requirements were full-filled or not), other may provide score in percentage.

We have decomposed submission process into following steps:

- Evaluate a file structure of an uploaded bundle.
- Verify the originality of a solution.
- Compile a program.
- Verify the programming principles (OOP).
- Check the correctness of functional parts based on unit tests.
- Check the proper functionality of the program based on given inputs and outputs.
- Create a report based on results of previous steps with the final score.

The composition and a sequence of steps of the evaluation process are illustrated in the Fig. 1. The sub-processes 2 and 3 require a parameter – threshold in percentage, above which the program is considered a plagiary and threshold defining how many rules of a particular programming methodology, e.g. object oriented principles (OOP), may fail and the assignment is still accepted. Other sub-processes provide binary results meaning that assignment is either accepted or rejected. In the second case a failure report with a detailed description is sent back to a student. Naturally, a student has to correct indicated failures and upload the assignment once again.

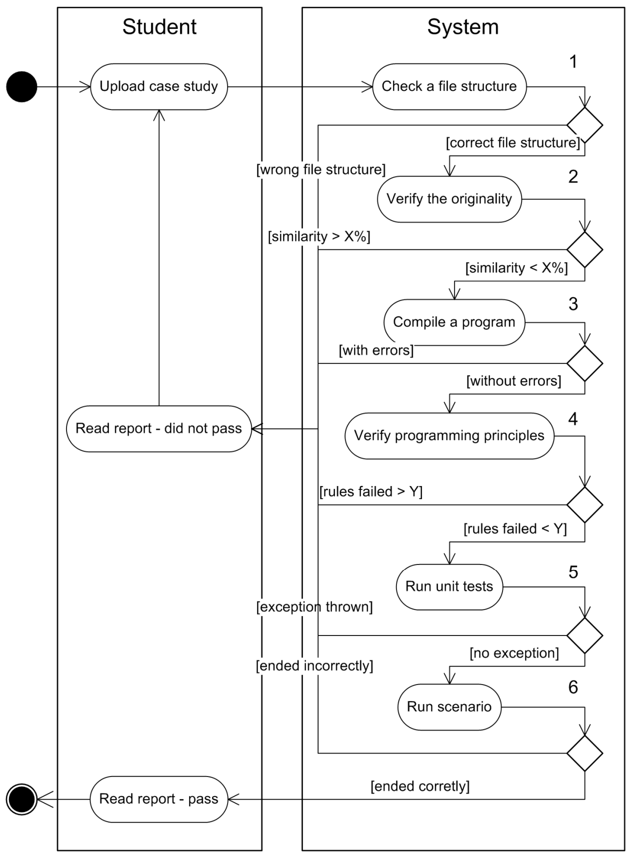

Figure 1. Activity diagram of a decomposed submission process

The submission process may be completely automated, or may work as a sieve, filtering students, who just try to submit an assignment of someone else. In later case, the student will be allowed for a personal defense only when passing all submission sub-processes. The continuous monitoring of student's progress may be also an asset, forcing the student to work gradually, not at once just before the final examination.

III. PHASES OF A SUBMISSION PROCESS

A. *File Structure Evaluation*

Finalized case study on a programming course usually comprises of various data files. There are not only source files, but also documentation (may be generated from a source files during checking phase), executable, external libraries, readme file or other files requested by the instructor. On our courses we prefer Game Based Learning (GBL) [6] where students are implementing a game during a semester. Therefore our students are requested to upload, besides previously mentioned files, also a user guide and a scenario how to complete the game successfully.

Checking the formal aspects of an uploaded assignment is definitely time saving and easy to be implemented either as a web service or as a plug-in in LMS. During this step, the uploaded bundle is unpacked and all required files are verified. If something is missing a student is asked to upload a bundle with right structure.

B. *Verifying the Originality*

To find plagiaries is labor demanding tasks, especially in courses attended for a high number of students. Finding a duplicate manually is usually an accident when student forgets to change a name in comments, or other document enclosed with an assignment or some weird bug is common for more than one assignment.

There exist several tools for automated plagiary detection. Usually their compare all pairs of assignments with various software metrics or compare structure of programs.

jPlag is one of such tools comparing the programs based on tokens coverage. Firstly it parses the input source code and converts it to a stream of tokens. After that the tokens are compared in pairs for determining the similarity of each pair. The similarity value is counted as the percentage of the token streams that can be covered [7].

jPlag tool is accessible as a web service, which may be important especially when it is aimed to be included into a larger system.

C. *Verifying Programming Principles*

There are available plenty of tools for static code analysis. From tools dedicated to Java programming language there could be mentioned open source tools as PMD [8] and Checkstyle [9] or commercial tools as JTest [10]. All these tools are aimed at scanning source code and report problems like inappropriate expressions (many nested if statements or loops), duplicated code, wrong variable naming, dead code as unused local variables, parameters and private methods, tight coupling or low cohesion.

We have decided for PMD because changeable rule sets, possibility to overwrite descriptive messages for violations and accessible plug-ins for Netbeans and Eclipse IDE.

D. *Running Unit Tests*

Object-oriented paradigm is widely used in introductory or basic programming courses. OOP may seem more appealing to students, but it sometimes orient students more on technological aspects. Thus the students have problems with designing a program or finding bugs in their application [11]. To solve this problem many universities introduced testing concepts in their programming courses [12].

As part of automated testing there may be realized unit tests prepared by students and also unit tests prepared by a lecturer. To support this approach, the students should be provided a library with interfaces and a set of empty unit test. This will navigate students which values to test and therefore help them to prepare unit test properly

E. *Running scenarios*

Depending on type of a case study, students may prepare a scenario to prove the right functionality of a program. This type of testing is more visible for programming assignment, where students implement a game. For instance an adventure game has a scenario how to successfully accomplish it (go west, take sword, go north, kill a dragon).

Students may prepare an input scenario, which is bundled with a program and this way uploaded for testing. The automated testing system runs the scenario on program and evaluates its functionality.

Obviously this type of testing is adaptable only for programs accepting text input and explicit sequence of steps known before the program execution. Testing of a successful scenario may be implemented also through unit tests in some programs.

IV. An Architecture Overview

Rather than developing a new system from scratch, that will cover all phases of the submission process, we decided to interconnect existing systems and applications into one collaborating unit with added value. When designing the architecture of a submission system we came out from IDEs and information systems already used on our courses as Netbeans, Eclipse, LMS Moodle and student information system Mais.

The main evaluation component of a system, which glues other existing systems, is a remote web service *LANESS (LAzy and NEgligent Student Search)*. It accepts the assignment from a student, executes all validation steps and provides a result to the student through both an IDE and LMS Moodle.

The proposed system is not finished yet and we are working on the implementation. From the architecture point of view and internal processing of LANESS is decomposed into two parts:

- uploading the assignment from IDE to a web service and validation steps execution
- writing the results into LMS or other IS

A. Uploading an assignment to the Web service

The simplified architecture of an uploading process is depicted in Fig. 2. A student uses an IDE to work on an assignment (e.g. Netbeans or Eclipse – they provide the environment for various programming languages as Java, C, C++ etc.). When the program is finished, the student uploads the assignment to the web service for validation. This may be done only once during the semester, when the assignment is finished or more times after some milestones, depending on a validation mechanism. The web service executes all validation steps (file structure evaluation, originality verification, programming principles evaluation, etc.) and prepares a result report that is sent back to the student.

To provide a more user friendly environment, the uploading functionality is accessible directly from an IDE as an ANT task. This way a student is released from tasks such as compressing all files into one ZIP file, uploading the ZIP file through some web interface and downloading the result. All is done automatically through an ANT script that is executable directly from an IDE by one click. The student should take care only of a file structure, since all types of documents are required to be on a predefined place.

B. Sending the Result into LMS

When the assignment is validated by the LANESS service the result is sent back to a student providing a comprehensive feedback. The feedback helps a student to fix problems and fulfill all requirements to have the application accepted. The information about student progress is important also for a lecturer. For this purpose LANESS sends the shortened report to LMS.

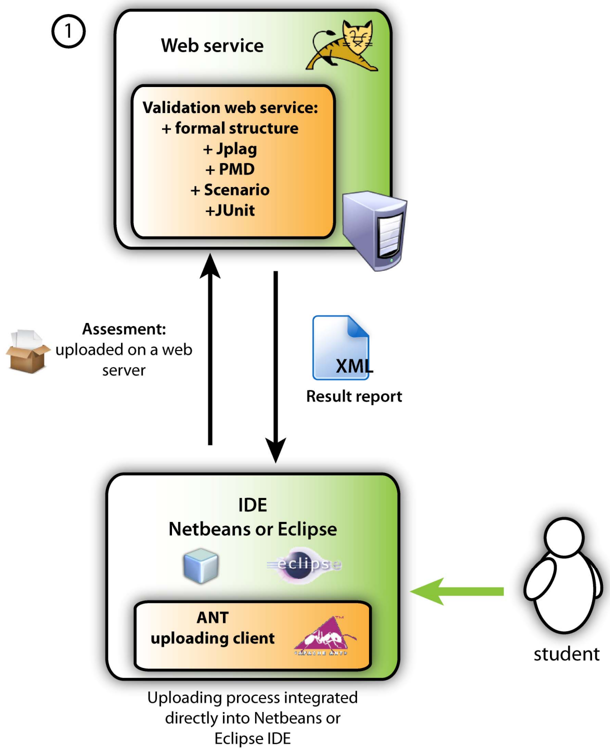

Figure 2. LANESS Architecture - uploading and evaluation part

C. Sending the Result into LMS

When the assignment is validated by the LANESS service the result is sent back to a student providing a comprehensive feedback. The feedback helps a student to fix problems and fulfill all requirements to have the application accepted. The information about student progress is important also for a lecturer. For this purpose LANESS sends the shortened report to LMS.

LMS systems are currently broadly used at the universities providing the environment with various features as sources management, submission assignment, grading, testing, calendar, etc. Since the grade or final evaluation on many courses comprises from more than one assignment (test/quiz + case study) and LMSs internally support test/quiz creation and evaluation, we want to integrate LANESS results directly into LMS (Fig.3). This will not force lecturers to manually transcribe the marks from external validation system into LMS.

LMS may be connected with a university IS (management of students, classes, schedules, etc.) where final marks will be sent when course is finished.

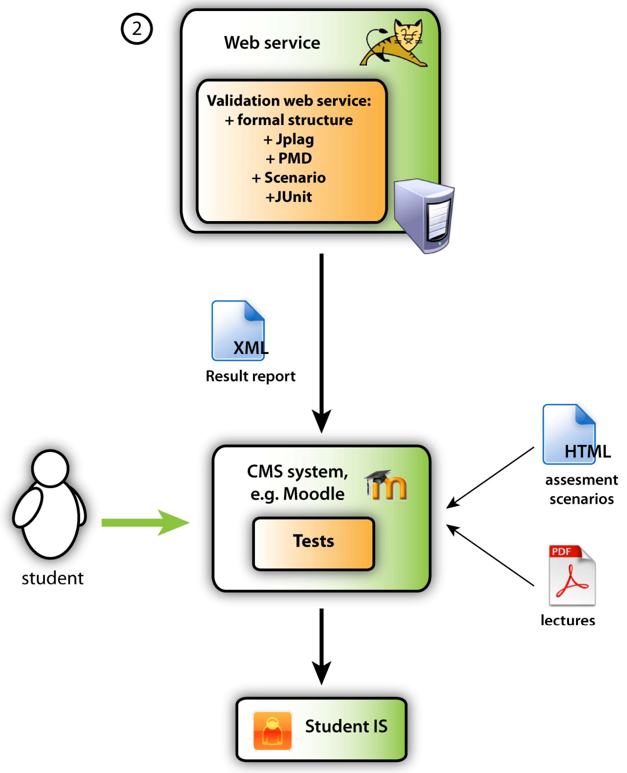

Figure 3. LANESS Architecture - writing results in LMS

V. Experimental Results and Future Work

We are trying to deploy this solution to the selected courses of undergraduate study programs at Faculty of Electrical Engineering and Informatics, Technical university of Košice, such as Programming, Object oriented programming (OOP) and partially at Technologies Java (TJava). These courses are one of the main courses of software engineering, where the number of attending students is between 200 and 400. The given numbers of students represent an interesting sample, where some of them are beginners in programming without any previous experiences, but some of them are advanced programmers with some habits. Obviously, such high numbers of students increase the probability of plagiarism, since manual checking all the assignments is a challenging task.

A. Case Studies and Scenarios

As was mentioned before, to increase students' motivation we prefer Game Based Learning [6], where students continuously work through semester on a game implementation following predefined scenarios. This type of education creates more space for students to be creative and to "play". Individual courses differ a little, for instance course OOP is denoted more to programming principles and technology is hidden as much as possible, whereas course TJava focuses more on special classes from Java SE and selected Java frameworks.

What they have in common are scenarios, which define steps guiding the students to accomplish specific goals (e.g. to learn difference between static and non-static methods), sometimes very similar to a tutorial. Because of these scenarios, part of a source code is similar for one case study and the remaining part depends on an approach and creativity of a student.

B. Present Submission Procedure

Until the system and LANESS service will be ready for a practical usage, we try to check all case studies manually. Manual submission control conforms to that presented in the second section. A lecturer firstly checks formal aspects of an assignment as documentation (user guide, system manual or Javadoc, UML diagrams). Secondly he checks if a program compiles without errors and if source codes comply with programming principles of a particular language through PMD tool. If some of the formal aspects are not fulfilled, the student is not allowed to continue in submission and is asked to fix all deficiencies and come again.

Naturally, in the opposite case, when the case study contains all required parts, student is asked to introduce his solution. Discussion between the lecturer and the student about student's ideas and approaches is the most pleasant part of the submission, however many times it turns into proving that a case study was not worked out by the student, but by someone else. This proving, besides checking formal aspects, consumes a lot of time and strengths of the lecturer. Here the automated testing of case studies may significantly help the lecturer to focus on final phase of a submission – evaluate students own approach to a solved problem.

C. Programming Principles Verification with a PMD Code Analyzer

To verify object oriented principles in Java programs we use PMD tool. The tool is bundled into a startup template (Netbeans project) that students download on the first meeting. It is executable directly from Netbeans IDE as an ANT task. PDM supports integration with up to 15 different IDEs, however we encounter a problem, where command line version returns more violations than plug-in version on the same source code.

We have prepared our own ruleset and translated description messages, because students had a problem to understand several violations. As an example, there could be mentioned frequently reported violation *cyclomatic complexity*. Students complained that official PMD explanation is too complex for them.

The result, which is our success, was improved readability of the source code produced by students. They get used to naming conventions, avoiding overcomplicated expressions, 3 or more times embedded if statements and many more, what are usual errors made by novice programmers.

D. Originality Verification with jPlag Tool

At present, verification whether a case study is a plagiary or not is done largely by lecturer's intuition. We have used jPlag Web Start client to test the sets of student's programs, but just as a post-validation after we have collected all the assignments.

The tests were realized on a sample of 183 assignments from the subject OOP taught in a winter semester 2010/2011 and 122 assignments from the subject TJava taught in a summer semester 2010/2011. The results are shown in Fig. 4 and Fig. 5 as histograms of programs' coverage. The coverage or similarity between two programs is measured in percentage divided into intervals of a length of 10% and the Y-axis represents pairs of programs falling into a particular interval. The number of pairs is counted in (1), where n is number of realized comparisons (each program with all others).

$$(n \cdot (n-1)) / 2 \qquad (1)$$

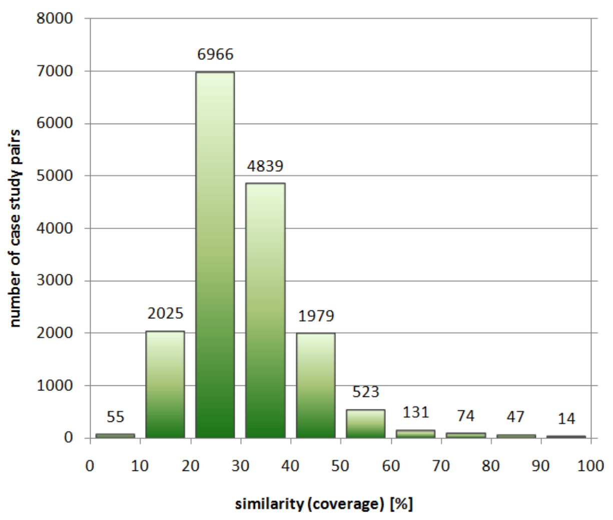

Figure 4. Histogram created by jPlag from assignments of a subject Object Oriented Programming

For instance, the algorithm of jPlag realized 16 653 comparisons among 183 programs from the OOP course. From the histogram in the Fig. 4 it is easy to see that similarity below 50% is a consequence of common scenarios and library provided to all students. Programs located in the interval of 50% - 70% coverage imply that students cooperated, but they added also their own implementation. Programs with the coverage above 80% are obviously plagiary. Counted from the jPlag report, 45 students (24,59% of all tested students) has the coverage of their programs higher than 80% and are considered to be cheating. 24,59% is rather a high number, which should be decreased in the future. The interval 70% - 80% should be considered manually by the lecturer, since there is a high probability that students cooperated, but not primarily with the aim to copy their programs.

Low number of program pairs above 80% coverage (61) on the one hand and a high number of cheating students on the other hand is given by the fact, that usually one student is an author of a program and another student copies the program with only minor changes. If we consider 50 students, where half of students worked on the assignments on their own and the remaining 25 copied the program from somebody from the first group we get only 25 pairs. Naturally, this is true only when a condition that one original is associated with only one copy is maintained. However, the task to seek out who is an author and who is plagiarist lies on the shoulders of a lecturer.

The result from a scan of TJava assignments (Fig.5) shows that a bigger part of the source code was common for all students. At OOP students implemented a text game, while at TJava students implemented a Minesweeper game with GUI. The limit of plagiary coverage is therefore shifted to higher percentage, because source code generated by Netbeans IDE represents a notable part of source code of the whole program. From a distinct difference of coverage, visible in Fig.5, between the interval 60%-70% and 70%-80% we may conclude that programs with coverage above 70% may be considered a plagiary. Programs falling into low coverage intervals are individual case studies implemented by skilful students.

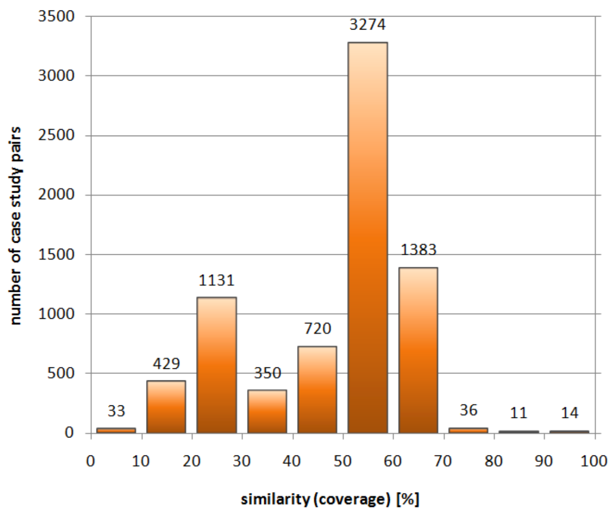

Figure 5. Histogram created by jPlag from assignments of a subject Technologies Java

E. Simple prove of the results

Because the proposed system is not fully functional and jPlag scan were realized after the end of a submission procedure, when the students were already graded, we have proved the results by questionnaires. The students were asked questions about their satisfaction with a course and a lecturer, how challenging was the case study, what they would change on a case study in the future, etc. The students whose programs were labeled as a plagiary were given a list of cooperating students (from the jPlag report) and should write if the list fits. They were assured, that their grades would be not degraded, if they own up.

From the questionnaires, we gained about 90% success in finding the plagiary. The result is thanks to truthful students (only when there is no threat of sanctions) quite high, but there should be considered some students who may not own up.

VI. CONCLUSION

High numbers of students attending courses, which are taught on more departments of a faculty in parallel, create better opportunities for lazy or worse students to cheat and copy programs from classmates. Manual control is time consuming and signs indicating a plagiary are easily overlooked by a lecturer.

This assumption is affirmed by the results from jPlag scans realized on assignments from OOP and TJava courses. We have verified the originality of a set of 183 OOP assignments, where 24,59% programs were marked as plagiary.

To speed up the submission process and to increase the number of detected plagiaries we propose a system for automated testing of case studies. The main part of the system is a web service LANESS responsible for particular testing steps as file structure verification, originality verification, control of programming principles, etc.

The architecture of a presented system, currently in the phase of development, links to existing systems as LMS Moodle and IDEs used at our courses. We want to integrate the service directly into IDEs to be accessible just by one click and the reports visible in the systems that are already used at the courses. Adding a new standalone system and therefore increasing the complexity of the submission process is not a way we would like to go.

We have realized tests with PMD and jPlag tools. Students used PMD tool with adapted ruleset during the implementation through the whole semester. jPlag was used to scan assignments from two different subjects – OOP and TJava.

Future work will focus on finalization of the proposed web service and plugins into LMS Moodle. A key aspect is scalability, to easily adjust the web service for a new case study or a new course. Another thing which shouldn't be neglected are descriptive reports for students. Feedback provided by a web service should help students to fix problems and successfully proceed the submission process, not to make it more complicated and stressful.

Acknowledgment

This work was supported by the Slovak Cultural and Educational Grant Agency of Ministry of Education of Slovak Republic (KEGA) under the contract No. 3/7245/09(60%). This work is also the result of the project implementation Development of the Center of Information and Communication Technologies for Knowledge Systems (project number: 26220120030) supported by the Research & Development Operational Program funded by the ERDF(40%).

References

[1] S. Kumar, A. K. Gankotiya, and K. Dutta, "A comparative study of moodle with other e-learning systems," *Electronics Computer Technology (ICECT)*, 2011 3rd International Conference on, Kanyakumari, April 2011, pp. 414 – 418, ISBN: 978-1-4244-8678-6.

[2] J. Spacco, D. Hovemeyer, W. Pugh, F. Emad, J. K. Hollingsworth, and N. Padua-Perez, "Experiences with marmoset: designing and using an advanced submission and testing system for programming courses," *In ITICSE'06: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, New York, NY, 2006, pp. 13–17.

[3] D. M. de Souza, J. C. Maldonado, and E. F. Barbosa, "ProgTest: An environment for the submission and evaluation of programming assignments based on testing activities," *Software Engineering Education and Training (CSEE&T)*, 2011 24th IEEE-CS Conference on, May 2011, pp. 1-10, ISBN: 978-1-4577-0349-2.

[4] Na Zhang, Xiaoan Bao,and Zuohua Ding, "Unit Testing: Static Analysis and Dynamic Analysis," *Computer Sciences and Convergence Information Technology, ICCIT '09*. Fourth International Conference on, Nov. 2009, pp. 232-237, ISBN: 978-1-4244-5244-6 .

[5] R. Romli, S. Sulaiman, and K. Z. Zamli, "Automatic Programming Assessment and Test Data Generation A review on its approaches," Information Technology (ITSim), 2010 International Symposium in, June 2010, pp. 1186-1192, ISBN: 978-1-4244-6715-0.

[6] M. Biňas, M. Novák, M. Michalko, and F. Jakab, "Ako motivovať študentov softvérového inžinierstva," *In: Social Networking: Proceedings of International Conferences ICT Bridges, Sunflower 2009, Silesian Moodle Moot 2009*, Nov. 2009, Ostrava, Czech Republic, pp. 199-204, ISBN 978-80-248-2117-7.

[7] L. Prechelt, G. Malpohl, and M. Phlippsen, "JPlag: Finding plagiarisms among a set of programs," Technical Report 2000-1, March 28, 2000.

[8] PMD tool home page, Accessed: 15.9.2011, Online: http://pmd.sourceforge.net/

[9] Checkstyle tool home page, Accessed: 15.9.2011, Online: http://checkstyle.sourceforge.net/

[10] JTest tool home page, Parasoft, Accessed: 15.9.2011, Online: http://www.parasoft.com/jsp/products/jtest.jsp/

[11] E. Lahtinen, K. Ala-Mutka, and H. J¨arvinen. "A study of the dificulties of novice programmers," *ITiCSE '05 Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, New York, NY, USA, 2005, pp. 14–18.

[12] S. H. Edwards, "Using software testing to move students from trial-and-error to reflection-in-action," *In 35th SIGCSE Technical Symposium on Computer Science Education*, Norfolk, Virginia, USA, 2004, pp. 26–30.