PROGRAMMING ASSIGNMENTS AUTOMATIC GRADING: REVIEW OF TOOLS AND IMPLEMENTATIONS

Julio C. Caiza, Jose M. Del Alamo

Universidad Politécnica de Madrid (SPAIN)
j.caiza@alumnos.upm.es, jmdela@dit.upm.es

Abstract

Automatic grading of programming assignments is an important topic in academic research. It aims at improving the level of feedback given to students and optimizing the professor time. Several researches have reported the development of software tools to support this process. Then, it is helpful to get a quickly and good sight about their key features. This paper reviews an ample set of tools for automatic grading of programming assignments. They are divided in those most important mature tools, which have remarkable features; and those built recently, with new features. The review includes the definition and description of key features e.g. supported languages, used technology, infrastructure, etc. The two kinds of tools allow making a temporal comparative analysis. This analysis shows good improvements in this research field, these include security, more language support, plagiarism detection, etc. On the other hand, the lack of a grading model for assignments is identified as an important gap in the reviewed tools. Thus, a characterization of evaluation metrics to grade programming assignments is provided as first step to get a model. Finally new paths in this research field are proposed.

Keywords: Automatic Grading, Programming Assignments, Assessment.

1 INTRODUCTION

The first reference about programming automatic grading comes from 1965 [1]. It has been almost fifty years since it started and the number of students who requires of programming skills is growing. It is not only about computer science or information technology degrees. It includes students of many engineering degrees as well. Nowadays, almost every engineering program includes at least a basic programming course.

Another point to consider is the difficulty of getting programming skills by students. The main path to improve this has been the increment of solved programming exercises. This has to be accompanied with a good feedback. The feedback would be provided by a professor or a teaching assistant (teaching staff). Considering the number of engineering students and a good set of programming assignments, a manual assessment turns into a difficult or even an impossible task. The problem for the teaching staff is the excessive and maybe repetitive workload.

Several researches have reported the development of software tools to automate the process. These tools would give a feedback to orientate the students' learning, and will liberate teaching staff to do more productive work, giving focused help for instance. Almost every tool supports these goals and additionally tries to offer new features based on solve new gaps. These new gaps, among others, refers to plagiarism detection, secure test environment, controlled resource use, the diversity of criteria for grading [2] and the definition of pedagogical models [3].

There has been a good research in the field but now the problem is that there are many tools. If there is an institution which needs to implement a tool of this type or wants to develop a new tool it would be necessary to get a quickly and good sight about the state of the art. A tools' review will be helpful to find important features of already built tools. In addition this kind of work will give new ideas to improve or to build a new tool, which could be used broadly.

This paper reviews a set of mature and recent tools for automatic grading of programming assignments getting and showing key information. The next sections include the revision of related work to get information about the evolution in this research field. Next, a set of important tools is described, taking into account their key features. A comparison and an analysis will be shown to establish the current situation in this field. Then a grading metrics characterization is proposed. The last section includes conclusions and future work.

2 RELATED WORK

Many tools for automatic assessment of programming exercises have been built since the first appearance. A tools' review has been done before. It is necessary to know which the main conclusions of these works were. They will be useful to know if actual tools have already filled all the gaps.

To see an evolution, it is necessary to take a temporal perspective. Douce et al. in [4] make a good and quickly characterization of these tools evolution until 2005. It identifies three generations of tools. The first one refers to times when working on operating systems and programming languages was necessary, and the assessment was only made considering a right or a wrong answer. The second generation refers to work with tools, which came with the operating system, to build new tools. C and Java languages were mostly used in development. The third generation is just around the time that this work was done. The main improvements in the reviewed tools are the orientation of using web-based technologies. It reports an increment in support for more programming languages as well.

Douce et al. [4] gave the next steps for automatic assessment of programming assignments. In [5] Ihantola et al. made a work covering tools developed since 2005. Taking these two works, it is possible to contrast them to show the improvement in some issues. These issues can be classified as technical, pedagogical and for a system adoption.

Regarding technical issues, Douce et al. indicated some research paths in [4], which included grading of GUI (Graphical User Interface) programs; meta-testing which refers to qualify applied tests; use and configuration of safe systems to test the programming assignments, the idea is to protect the host system form intentional or unintentional malicious code; integration of systems to avoid overwhelm the user, usually the idea would be integrate the tool with an LMS (Learning Management System), it can be reached using web-services; and support for web programming grading, it was because universities started to teach web programming and then grade this was necessary.

Ihantola et al. in [5] and Romli et al. in [6] had reported improvements in systems integration with LMS and in security for the host system. Then, issues like grading of GUI programs, meta-testing, and support for web programming stayed waiting for more research.

Regarding pedagogical issues, the reported works lack a common grading model. Every institution and even every teacher has their own way to establish a grade. So a reference model could be helpful. In reviews did in 2010, the correctness is reported as the main metric to grade. Some works started to use a static and a dynamic analysis as well, but in general, every work proposes its own set of metrics to grade. As a result, at that time, there was not a common approach yet; maybe the first step to build a model could be the metrics' characterization.

About feedback, there are some implications: quickly feedback could trigger trial-error practices, how much useful is the automatic feedback, and which is the adequate quantity of feedback. Some works try to provide flexibility to feedback and allow manual and automatic solutions [7].

The implementation of plagiarism detectors has been considered by some tools. Usually they are an additional module in the tool but not affect the grading process. Although it is clear that plagiarism detection would imply in a sanction.

With regard to systems' adoption, both works [5] and [6] showed that a big number of tools had been built but they are not broadly used. It is because every tool has been done considering specific requirements. An important way to increase the adoption was to work on open source projects. Some projects have done this and its acceptance has grown [7] but a definitive broadly used tool had not been reached. In [6] is proposed the building of a flexible and parameterizable system and it seemed a good path to reach the goal.

3 TOOLS' REVIEW

The common goal to build or to use these kinds of tools is to improve programming skills in the students, paying special attention in beginner students. The skills will be improved through solving many programming exercises. The students can go on the problems as quickly as they get good feedback. It would help them to understand their mistakes and to improve their skills. Additionally students get a real benefit, which is to get a fair grade not dependent on personal considerations of the academic staff [2].

Considering the quantity of students in a regular class of engineering and a big number of programming exercises, it is not viable a manual grading. Then the idea is not overwhelm the academic staff either, so another goal is to optimize the time of academic staff. The saved time could be used in more productive process like planning and designing the lectures or just giving more personal attention in focused problems.

As the research in this field increases, new goals are proposed. Thus, in [8], [9], [10] and [11] an extra goal is to get the integration with a LMS to improve the performance of the programming assignments assessment process. In [12] is proposed the use of services to reach this goal. In [13] one goal is to collect detailed information to research deeply the students' skill improvement process. More recently, Allevato and Edwars [14] have as a goal to get the interest of students using the popularity of smartphones and mobile applications.

3.1 Analyzed Key Features

The next key features have been defined considering they are important in a deployment case:

- Supported programming languages. It is a very important feature when it is considered to make a quickly implementation. It could define the use or not of a tool.
- Programming language used to develop the tool. This feature has great relevance when there is a set of policies respect of the software used in an institution. In the case of customization or maintenance, it would be a valuable feature to choose a tool.
- Logical architecture. It is an important feature when a modification of the tool is being considered. This architecture will show the modularity, scalability and flexibility level. It could show how the different modules work and how the system could connect with other systems.
- Deployment architecture. It shows how the hardware over which the tool works is. It is helpful to know if a current environment will support the implementation of a tool. In the worst case it will indicate the resources needed and therefore will help to determine the cost of an implementation.
- Work mode indicates if the tool can work alone, for clear implementations; or if the tool can work as plugin, when it is supposed to work with another system, an LMS for instance.
- Evaluation metrics. It displays how the tool can establish a grade. It considers which metrics are being considered inside the grading process. Even, how the grade calculation is done.
- Technologies used by the tool. It is helpful when it is considered to deploy or to build a new tool. In the first case it would help to establish compatibility between the tool and a legacy system. It is useful for future maintenance as well. In the second case it is very helpful to know which technology (standards, protocols, libraries, etc.), could help to face a requirement.

3.2 Tools

3.2.1 CourseMarker

A tool developed in the Nottingham University to avoid the particular criteria of teaching staff. The main advantages are considered being scalability, maintainability, and security [2]. The supported programming languages for grading are Java and C++ and it has been built using Java. Its architecture shows 7 subsystems: login, it controls all the authentication process; submission, it receives the different submissions precisely; course, it stores information about the process; marking, it has in charge the grading process, and the storing of the submitted files and marks gotten; auditing, it has as responsibility to log all actions; and a subsystem to control the communication among the others.

As metrics to establish a grade it considers typography (indentations, comments, etc.), functionality through test cases, programming structures use, and verification in the design and relations among the objects.

It works with technologies like Java RMI (Remote Method Invocation) for communication among the subsystems, regular expressions to verify results and DATsys [15] to verify objects design.

Additional important features include: the capacity to work with feedback levels, the orchestration among subsystems is defined by a configuration file, feedback and grades can be customized, there is

plagiarism detector when grading, submissions number and CPU quantity are configurable, and finally there are security considerations which are detection of malicious code and execution in a sandbox environment.

3.2.2 Marmoset

It has been built in the University of Maryland. Its main goal is to collect information about development process to improve the student skills [13]. Its main advantages are to make a complete snapshot about the student's progress, so the student development can be analyzed in detail; the use of different types of test cases (student, public, release, secret); and a personal support through comments' threads on the code.

Originally the paper reported grading of code written in Java, C, Ruby and Caml Objective. Now, the official web page[1] informs that it works with all different programming languages. The architecture includes: a J2EE (Java Enterprise Edition) webserver, a SQL database, and one or more build servers. These last are used in a safe and lonely environment to prevent effects of possible malicious code. The disposition of build servers helps to provide scalability and security. The metrics to establish a grade include dynamic and static analysis. The dynamic analysis is done through test cases.

3.2.3 WebCat

The main features are the extensibility because of its plugins-based architecture and a grading method based on how well the students grade their own code [7]. The architecture design provides a set of important features: security, it is provided through means like authentication, erroneous or dangerous code detection; portability, because it has been built as a Java servlet; extensibility and flexibility, it is inherent to the architecture; and support for manual grading as well, it is because the academic staff can check students' submissions and enter comments, suggestions, and grade modifications. The official wiki[2] affirms that it is the only tool which integrates all these features.

The tool supports Java, C++, Scheme, Prolog, Standard ML, and Pascal, but it offers flexibility to support any programming language. The grade is based on code correctness (how many tests are passed), test completeness (which parts of the code are actually executed), and a test validity (test accurate-consistent with the assignment). Additionally plugins can provide more metrics for evaluations (static analysis for instance). Additional features include: there are a lot of plugins for Eclipse and Visual Studio .NET IDEs, and it has a GNU/GPL license.

3.2.4 Grading Tool by Magdeburg University

It has a really interesting goal which is providing a tool which is not forced to work with a given LMS, but avoiding the use of two systems independently [12]. It can be reached using services. It shows a configurable focus. Then there are selectable components like the compiler, the language interpreter, the grading method, and the data set. The submissions' number, and time features are configurable as well.

The tool uses dynamic tests, compilers and interpreters to establish the grade. The supported languages are Haskell, Scheme, Erlang, Prolog, Python, and Java. The architecture is very interesting. It considers three servers: the front-end, it will be an LMS system; the spooler server, it controls the request, the submissions queues and the back-end calls; and the back-end servers, which are the modules to evaluate a programming language. To communicate the servers, XML-RPC (Remote Procedure Calls) has been used.

3.2.5 JavaBrat

It is a tool reported in [8], and built as a master thesis in San José State University. It gives support for two programming languages, Java and Scala. It uses Java to develop the grader software and PHP to build a plugin for Moodle. The design includes three important modules: a Moodle server with a plugin; a module which contains the graders depending on language and a repository of problems; and the last module is Javabrat which has a set of services to call graders and problems.

Although it can works as a Moodle's plugin, this tool can work alone through a web interface developed as part of the project. This web interface was developed using JSF (Java Server Faces) 2.0. The services are implemented using JAX-RS.

---

[1] http://marmoset.cs.umd.edu/

[2] http://wiki.web-cat.org/WCWiki/WhatIsWebCat

The work was centered in develop the web interface and the problems' repository. Then the grading process is not very complex and is based on correctness, which is determined by test cases. It is a semi automatic tool because it is necessary a revision of the report generated when the grading process is done.

3.2.6 AutoLEP

A tool developed in Harbin Institute of Technology and which is presented in [16]. It has as main feature the combination between static and dynamic analysis to give a grade. The dynamic analysis refers to evaluate correctness using test cases. The static analysis doesn't need to compile or execute the code. It is just about to make a syntactic and semantic analysis and it is reported as main difference with previous works.

The architecture includes: the client, a computer used by a student, it does the static analysis and can provide of a quickly feedback; a testing server which has to do the dynamic analysis; and a main server which has to control the information of the other components to establish a grade.

3.2.7 Petcha

A tool developed in University of Porto. Its main goal is the building of an automatic assistant to teach programming [11]. An important feature is the coordination among existing tools like IDEs (Integrated Development Environment), LMSs and even automatic graders. It supports the programming languages that IDEs do. The tested IDEs are Eclipse and Visual Studio.

Its architecture is defined as modules for every connected tool. Then, there is a module for the LMS, the IDE, the exercises repository, and for the assessment engine. It relies on some technologies to guarantee interoperability: IMS Common Cartridge as format to build packages with resources and metadata, IMS Digital Repositories Interoperability and bLTI (Basic Learning Tools Interoperability). Additionally it used JAWS (Java Web Start) to build the client interface and it is working with MOOSHAK [17] as assessment engine.

3.2.8 JAssess

It has been built by researchers in two universities in Malaysia, University of Technology and Tun Hussien Onn University [10]. Their goal is to have only one interface to access the assessment process. JAssess is presented as an integrated tool with Moodle.

Their architecture shows the next modules: Moodle server, MySQL Server, JAssess, and JAssesMoodle to communicate Moodle and JAsses.

About supported languages it only supports Java, and precisely it is the language used to build the tool. Then it used libraries as Java File, Java Unzip, Java Runtime, Java Compiler and Java Reflection. About the metrics considered to grade, it is a weakness for the tool because it only depends on compilation. The evaluation process is not completely automatic.

3.2.9 RoboLIFT

The main approach is to get interest of students in programming using the popularity of mobile applications and smartphones [14]. The increasing market of android smartphones and applications makes increase the interest of students. This knowledge will be helpful when they will finish their studies as well. The tool supports grading of Android applications.

The tool is based on WebCat [7], so the architecture will be the same with an additional variation. The variation is the use of Robolectric[3] , which is software to accelerate the grading process. The tool uses the development tools for Eclipse provided by Google.

The unit testing is considered as metric to grade. The tests are of two sorts, public and private tests. The first one kind is known by the students and the second type is only used in the definitive submission.

3.2.10 Virtual Programming Lab

A tool built in Las Palmas University [9]. The goals of the project includes to provide the students with many programming assignments, and to support the managing and grading process. It can be gotten through the integration with a LMS system. The tool supports many programming languages including

[3] http://pivotal.github.com/robolectric/index.html

Ada, C, C++, C#, FORTRAN, Haskell, Java, Octave, Pascal, Perl, PHP, Prolog, Python, Ruby, Scheme, SQL, and VHDL.

The architecture includes three modules: a plugin for Moodle, which allows the integration with the submission and grades modules in Moodle; a code editor based on browser, which allows coding without the necessity of an installed compiler; and a jail server, which hosts the environment where the assignment will be evaluated. To develop this tool they have worked with PHP to build the Moodle plugin. To implement the jail server, C language has been used. Every language has an associated shell script for evaluation as well. The communication between Moodle and jail servers is done with XML RPC. The jail server gives the services through a Linux program called Xinetd. In addition the jail server implements a safe environment with the Chroot Linux program.

For grading it consider the correctness, evaluated through test cases by default. The test cases are specified in an own and easy syntax. The default scripts which evaluate the programs can be changed to improve the evaluation method.

Additionally, this tool has some interesting features like: it is built under GNU/GPL license, it allows automatic and semiautomatic process, it includes a plagiarism control tool, and it provides configurable features for every assignment.

3.2.11 Moodle extension by Slovak University of Technology Bratislava

It is presented in [18]. Its main goal is managing and modeling digital systems using HDL (Hardware Description Language). Then the work reports managing features like assignments managing, and user type definitions. The only language supported is VHDL.

The tool evaluates a submission based on: compilation and syntactic analysis, functionality doing comparisons with a model, and then through a stage to detect plagiarism.

4 COMPARISON AND ANALYSIS

The key features of each tool can be used to identify the real improvements since the last reviews [5] [6] were carried out.

To analyze the improvements through the time, two tables with tool's features are shown. The Table 1 join tools built a few years ago, previous to the work presented by Ihantola [5] which have been updated continuously. Precisely by their maturity, they count with really good features and in some cases with a broad use.

The second table joins more recent tools, which have not been broadly used but that present new features and propose new research lines even.

Firstly it is necessary to take into account which where the pending issues reported until 2010. They were mentioned in an earlier section; technical issues which include lack of a GUI grading tool, meta-testing, and support for web programming; pedagogical issues including lack of a model to grade, trial-error practices, adequate quantity of feedback, and plagiarism; and adoption issues.

As it can be seen most of these issues have been solved. Thus, RoboLIFT has the feature of grade GUI applications because this uses LIFT, a library included in the WebCat project to grade GUIs. Web programming languages have been considered as well, VPL can grade PHP programs for instance. Trial-error practices has not been reported as a serious problem, it is because a quickly solution is the submissions' limitation. Technically it would not be difficult to implement. The amount of feedback has been considered in tools like CourseMarker which offers the possibility to configure the level of detailed feedback to give the student.

Plagiarism has been seen as an important module inside an automatic assessment tool. Then, some projects have considered it already.

The adoption of a tool depends on some features which include: how long the tool has been tested, if the tool has been developed as open source, how much flexible, scalable, and configurable a tool is. For example VPL have been adopted by many institutions as it can be seen in its web page[4].

[4] http://vpl.dis.ulpgc.es/

Table 1. Mature tools.

Tool's name	Main Features	Supported Languages	Work Mode	Grading Metrics
CourseMarker	Scalability, maintainability. Security, configurability. Plagiarism detection. Work with levels of feedback.	Java, C++.	Standalone	Typography. Functionality. Structures use. Objects design. Objects relations.
Marmoset	Detailed information. Language independence. Security and scalability for evaluation module. Apache 2.0 license	Any language.	Standalone	Dynamic and static analysis.
WebCat	Extensibility and flexibility based on plugins. Access security. Portability. Semi and automatic process. GNU GPL license.	Java, C++, Scheme, Prolog, Standard ML, and Pascal. Flexibility for any language.	Standalone	Code correctness. Completeness. Test validity. Extensible by plugins.
Virtual Programming Lab	Moodle integration. Customizable grading mode. GNU GPL license. Plagiarism detection. Configurable activities. Jail environment.	Ada, C, C++, C#, Haskell, FORTRAN, Java,Octave, Pascal,PHP, Prolog, SQL, Ruby,Python, Scheme,Vhdl.	Moodle plugin	Correctness based on test cases. Open for new methods.
Grading Tool (Magdeburg University)	Use of services. Configurable evaluation process.	Haskell, Scheme, Erlang, Prolog, Python, Java	LMS extension.	Compilation. Execution. Dynamic tests.

Following the temporal comparison, almost all issues reported in the last review have been covered, but for meta-testing. However, it does not imply that everything is worked out, as we will discuss later on the conclusions.

Table 2. Recently developed tools.

Tool's name	Main Features	Supported Languages	Work Mode	Grading Metrics
JavaBrat	Use of services. LMS integration.	Java, Scala	Moodle plugin. Standalone.	Correctness.
AutoLEP	Static and dynamic analysis to grade.		Standalone	Static analysis. Dynamic analysis.
Petcha	Coordination among existing programming-support tools. Use of technology for interoperability.	Languages supported by Eclipse and Visual Studio	Standalone	Based on test cases.
JAssess	Moodle integration.	Java	Moodle plugin	Compilation
RoboLIFT	Grading mobile applications. GUI grading.	Java	Standalone	Unit testing (public and private)
Moodle ext. (Slovak University of Technology)	Oriented for digital systems. Plagiarism detection.	Vhdl.	Moodle plugin	Compilation. Syntactic analysis. Functionality by comparison.

There are a good number of tools for automatic assessment in programming. Then, does it make sense to continue building new ones? Usually the main reason to build a new tool is that the existing ones do not fulfil our requirements. If this is the case then it may be a good idea to get the tool and extend it through a plugin.

Table 1 shows important information which supports the reuse of tools; this is based on existing tool features like extensibility, flexibility and configurability. The information in Table 2 shows that Java is the most common supported language by recent tools. This language has been already supported by older tools and this cannot be sufficient reason to build a new tool. If new support for a given language is necessary, it can be done through adding a new submodule or plugin to extensible tools.

An important fact is the use of LMSs in most universities. The ideal thing would be to seamlessly use the automatic tool within the LMS. Some recent tools are considering the integration with an LMS but they do not provide features like scalability, flexibility, maintainability as the older ones. Maybe the next step to evolve with automatic tools is to add the LMS integration to the features of the more mature tools. Probably it could be reached by a redesign of the architecture of a mature tool. It means, the architecture could consider flexibility for the tool's front-end. This flexibility can be reached through services. The front-end could be a module in the LMS or a module developed in any technology for user interfaces.

Finally, it can be seen that metrics to grade are not normalized. Every tool has considered its own set. It is a real problem which has come about since the automatic tools have appeared. This problem will be treated in the next section.

5 GRADING METRICS ISSUES

The lack of a common model to grade is still an important problem. First, every institution and even every teacher has his own critter to grade an assignment. Additionally, as Rodriguez in [19] says, it is necessary to recognize that some metrics cannot be measured. The creativity or the right sense of a comment cannot be determined by an automatic tool. In spite of these facts, and taking into account the importance of defining a set of metrics, a characterization of metrics is proposed below as a first step to get a grading model.

The importance of a characterization can be inferred by seeing the Tables 1 and 2. Every tool has its own set of metrics to establish a grade, e.g. one tool just includes the compilation, and another considers extensibility of metrics through plugins. Additionally, some tools refer to the same metric by different names.

Looking at all the metrics expressed in the previous tables as well as criteria from the software engineering discipline; Table 3 shows a characterization for grading metrics.

Table 3. Grading metrics characterization

<table>
<tr><th colspan="2"></th><th>Metric</th></tr>
<tr><td colspan="2" rowspan="2">Execution</td><td>Compilation</td></tr>
<tr><td>Execution</td></tr>
<tr><td colspan="2">Functional Testing</td><td>Functionality (system or method level)</td></tr>
<tr><td rowspan="6">Non functional Testing</td><td>Specific requirements</td><td>Specific requirement for an exercise</td></tr>
<tr><td rowspan="3">Maintainability</td><td>Design</td></tr>
<tr><td>Style</td></tr>
<tr><td>Complexity</td></tr>
<tr><td rowspan="3">Efficiency</td><td>Use of physical resources</td></tr>
<tr><td>Execution times</td></tr>
<tr><td>Processes number</td></tr>
<tr><td></td><td>File or code size</td></tr>
</table>

Every metric could have an associated tool which evaluates it. For compilation, a language compiler; for execution, a language interpreter; for functionality, it can be used a program based on test cases (JUnit in Java for instance); for specific requirement, it will be necessary a particular program; for

design, style and complexity, an external program will be needed (Checkstyle for style in Java for instance); for the last four metrics, it could be useful shell script programs.

Some tools offer the possibility of support any grading metric through the building of plugins. But it would be better to consider a complete evaluation process. This evaluation process would have as feature a high level of configurability to support any metric. The goal is not see just a metric; it is to consider the whole grading process.

6 CONCLUSIONS AND FUTURE WORK

This work has reviewed the state of the art of automatic tools for programming assignments assessment. A set of requirements and key features for these tools has been described, and a comparison and analysis of existing tools has been carried out. As a result, a clear snapshot of their current status was provided, showing the lack of a common grading model as the major issue detected. A grading metrics characterization has been proposed as a first step to get a grading model.

The future research lines point towards the consolidation in one tool of several features like LMS integration (without a dependency on a given one), flexibility, scalability, maintainability, portability, and security; and the establishment of a configurable evaluation process where the metrics can be selected as the teacher needs.

7 ACKNOWLEDGMENT

The first author would like to extend thanks to *Secretaría Nacional de Educación Superior, Ciencia, Tecnología e Innovación* of the Ecuadorian Government, which provided him with financial support during his master studies.

In addition, this work has been partially supported by the Universidad Politécnica de Madrid, as part of the "*Ayudas a la Innovación Educativa y a la Mejora de la Calidad de la Enseñanza*" programme.

REFERENCES

[1] Forsythe, G. E., Wirth, N. (1965). Automatic Grading Programs. *Commun ACM,* vol. 8, pp. 275-278.

[2] Higgins, C. A., Gray, G., Symeonidis, P., Tsintsifas, A. (2005). Automated Assessment and Experiences of Teaching Programming. Journal on Educational Resources in Computing (JERIC), vol. 5, pp. 5.

[3] Choy, M., Lam, S., Poon, C., Wang, F., Yu, Y., Yuen, L. (2008). Design and Implementation of an Automated System for Assessment of Computer Programming Assignments. Advances in Web Based Learning–ICWL 2007, pp. 584-596.

[4] Douce, C., Livingstone, D., Orwell, J. (2005). Automatic Test-based Assessment of Programming: A Review. Journal on Educational Resources in Computing (JERIC), vol. 5, pp. 4.

[5] Ihantola, P., Ahoniemi, T.,Karavirta, V., Seppälä, O. (2010). Review of Recent Systems for Automatic Asessment of Programming Assignments. Proceedings of the 10th Koli Calling International Conference on Computing Education Research, pp. 86-93.

[6] Romli, R., Sulaiman, S., Zamli, K. Z. (2010). Automatic Programming Assessment and Test Data Generation a Review on its Approaches. Information Technology (ITSim), 2010 International Symposium, pp. 1186-1192.

[7] Edwards, S. H., Perez-Quinones, M. A. (2008). Web-CAT: AutomaticallyGgrading Programming Assignments. ACM SIGCSE Bulletin, vol. 40, pp. 328-328.

[8] Patil, A. (2010). Automatic Grading of Programming Assignments.

[9] Rodríguez-del-Pino, J. C., Rubio-Royo, E., Hernández-Figueroa, Z. J. (2012). A Virtual Programming Lab for Moodle with Automatic Assessment and Anti-plagiarism Features.

[10] Yusof, N., Zin, N.A.M, Adnan, N.S. (2012). Java Programming Assessment Tool for Assignment Module in Moodle E-learning System. Procedia-Social and Behavioral Sciences 56 (56), pp. 767-773.

[11] Queirós, R. A. P., Leal, J. P. (2012). PETCHA: A Programming Exercises Teaching Assistant. Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education, pp. 192-197.

[12] Amelung, M., Forbrig, P., Rösner, D. (2008). Towards Generic and Flexible Web Services for E-assessment. ACM SIGCSE Bulletin, pp. 219-224.

[13] Spacco, J., Hovemeyer, D., Pugh, W., Emad, F., Hollingsworth, J. K., Padua-Perez, N. (2006). Experiences with Marmoset: Designing and Using an Advanced Submission and Testing System for Programming Courses. ACM SIGCSE Bulletin, vol. 38, pp. 13-17.

[14] Allevato, A., Edwards, S.H. (2012). RoboLIFT: Engaging CS2 Students with Testable, Automatically Evaluated Android Applications. Proceedings of the 43rd ACM technical symposium on Computer Science Education, pp. 547-552.

[15] Higgins, C., Symeonidis, P., Tsintsifas, A. (2002). Diagram-based CBA Using DATsys and CourseMaster. Proceedings of Computers in Education International Conference, pp. 167-172.

[16] Wang, T., Su, X., Ma, P., Wang, Y., Wang, K. (2011). Ability-training-oriented Automated Assessment in Introductory Programming Course. Computer. Education, Elsevier, vol. 56, pp. 220-226.

[17] Leal, J., Silva, F. (2003). Mooshak: a Web-based Multi-site Programming Contest System. Software: Practice and Experience, vol. 33, pp. 567-581.

[18] Jelemenská, K. Čičák, (2012). Improved Assignments Management in MOODLE Environment. INTED2012 Proceedings, pp. 1809-1817.

[19] Rodríguez del Pino, J. C., Díaz Roca, M., Hernández Figueroa, Z., González Domínguez, J. D. (2007). Hacia la Evaluación Continua Automática de Prácticas de Programación. Actas De Las XIII Jornadas De Enseñanza Universitaria de la Informática, pp. 179-186.