PROGRAMMING EXERCISES EVALUATION SYSTEMS: AN INTEROPERABILITY SURVEY

Ricardo Queirós[1] and José Paulo Leal[2]
[1]*CRACS & INESC-Porto LA & DI-ESEIG/IPP, Porto, Portugal*
[2]*CRACS & INESC-Porto LA, Faculty of Sciences, University of Porto, Portugal*
ricardo.queiros@eu.ipp.pt, zp@dcc.fc.up.pt

Keywords: Learning Objects, Standards, Interoperability, Programming Exercises Evaluation.

Abstract: Learning computer programming requires solving programming exercises. In computer programming courses teachers need to assess and give feedback to a large number of exercises. These tasks are time consuming and error-prone since there are many aspects relating to good programming that should be considered. In this context automatic assessment tools can play an important role helping teachers in grading tasks as well to assist students with automatic feedback. In spite of its usefulness, these tools lack integration mechanisms with other eLearning systems such as Learning Management Systems, Learning Objects Repositories or Integrated Development Environments. In this paper we provide a survey on programming evaluation systems. The survey gathers information on interoperability features of these systems, categorizing and comparing them regarding content and communication standardization. This work may prove useful to instructors and computer science educators when they have to choose an assessment system to be integrated in their e-Learning environment.

1 INTRODUCTION

One of the main goals in computer programming courses is to develop students' understanding of the programming principles. The understanding of programming concepts is closely related with the practice on solving programming exercises. Due to increasing class sizes, the practice of programming exercises assessment leads to extensive workload to instructors. Apart from being time-consuming, manual assessment hinders the consistency and accuracy of assessment results as well as it allows "unintended biases and a diverse standard of marking schemes" (Romli et al., 2010). Therefore, automatic assessment of programming exercises has become an important method for grading students' programming exercises as well as giving feedback on the quality of their solutions. In this paper we survey Programming Evaluation Systems (PES) focusing on their interoperability features.

Nowadays there are a large number of PES referenced in several surveys (Romli et al., 2010) found in literature. The majority of the surveys enumerates and compares the same set of features such as how the analysis of the code is made, how the tests are defined or how grades are calculated. These surveys seldom address the PES interoperability features, although they generally agree on the importance of the subject, due to the comparatively small number of systems that implement them. This lack of interoperability is felt at content and communication levels. Both levels rely on the existence of specifications that uniformly describe the content of programming exercises and the way they should be shared among the systems that are typically coupled with PES. Examples of these systems are Learning Management Systems (LMS), Contest Management Systems (CMS), Evaluation Engines (EE), Learning Objects Repositories (LOR) and Integrated Development Environments (IDE).

The main goal of this paper is to gather information on the interoperability features of the existent PES and to compare them regarding a set of predefined criteria such as content specification and standard interaction with other tools.

The intended benefit of this survey is twofold: 1) to fill the gap on PES interoperability features found in most surveys; 2) to help instructors, educational practitioners and developers when they have to choose a PES to integrate in their e-Learning environments.

The remainder of this paper is organized as follows: Section 2 summarizes recent PES surveys. The following section presents our survey focused on interoperability, organized in three facets: programming exercises, users and assignment results. Then we discuss the survey results and pointed some recommendations based on the result data. Finally, we present our view on the future trends on PES interoperability and open challenges for research on this subject.

2 RELATED WORK

2.1 Evolution of assessment tools

In recent years, programming courses in secondary schools and universities are characterized by extensive curricula and large classes. In this context, the assessment of programming assignments poses significant demands on the instructor's time and other resources (Douce et al., 2005). This demand stimulated the development of automated learning and assessment systems in many universities (Ala-Mutka, 2005). These systems assess programming exercises and assignments submitted by students, and provide evaluation data and feedback. They present a wide variety of features, such as programming language support, evaluation type, feedback, interoperability, learning context, security and plagiarism.

Early systems (Reek, 1989), (Jackson & Usher, 1997), (Mansouri et al., 1998) and (Saikkonnen et al., 2001) assess exercises and assignments in a single programming language respectively, Pascal, ADA, Prolog and Scheme. With the advent of the Internet and the increase of platforms heterogeneity, web interfaces began to play an important role in the dissemination of several systems (Pisan et al, 2003), (Juedes, 2003), (Leal, 2003) and (Blumenstein et al, 2004). The last two were among the first PES to support multiple programming languages**,** such as Java, C++ and the C.

The standard way of evaluating a program is to compile it and then execute it with a set of test cases comprising input and output files. The submitted program is accepted if compiles without errors and the output of each execution is what is expected. This evaluation strategy has been shown to bring undesirable pedagogical issues such as student frustration and confusion (Tang et al., 2009a, 2010). Jackson and Usher (1997), Saikkonen et al (2001), Pisan et al (2003), Juedes (2003), Blumenstein et al (2004) and Mandal et al. (2006) test not only the behaviour of single programs but also analyse the structure of source code. This approach guarantees that the program was written in a particular way, following a particular algorithm or used certain data structures. To assess the correctness of student submissions Edwards (2006) use also unit tests defined by teachers. Another important issue is the non-determinism of the program outputs where different correct (or acceptable) solutions to the same programming exercise may not always produce exactly the same output (Tang et al., 2009b). Leal (2003) deals with non-determinism using dynamic correctors invoked after each test case execution. For instance, if the solution is a set of values that can be presented in any order then a dynamic corrector can be used to reduce the output to a normal form.

Depending of the learning context (competitive or curricular) the systems may provide feedback to help students to solve a particular exercise. The feedback generation relies on static and dynamic program analyses (Ala-Mutka, 2005). The development of PES with high quality feedback (e.g. compilation errors, execution errors, execution tests) show good results (Malmi et al., 2005; Higgins et al., 2005) and along with visual, incremental and personalized feedback should shape the future regarding this topic. (Striewe, 2010).

The PES interoperability is also an important issue to address. An evaluator should be able to participate in learning scenarios where teachers can create exercises, store them in a repository and reference them in a LMS and where students can solve exercises and submit to PES who delivers an evaluation report back to students. Luck and Joy (1999), Benford et al (1993) were early systems that try to address this issue allowing the integration with course management systems. Nowadays with the advent of Service Oriented Architectures (SOA) the trend is service orientation rather than component-based systems. An evaluator system as a service will automate the existent business logic in distributed e-Learning scenarios allowing more flexibility in the comprised workflows and keeping the systems simple and easy maintainable. Leal et al. (2010) specified a service for programming exercises evaluation in a well-known e-Learning framework called the E-Framework. This work was used in Edujudge project (Verdu et al., 2011) with promising results.

Luck and Joy (1999) analysed security issues on PES covering robust environments, privacy, and data integrity. Security can be handled from ad-hoc

solutions to solutions based on Virtual Machines (VM) in order to execute the programs on a safe and controlled environment. Other concerning is the increase of plagiarism (Engels, 2007) and (Cheang, 2003). Luck and Joy (1999) and Blumenstein et al (2004) analyse the integration of plagiarism services in the assessment workflow.

Regarding the learning context, PES can be used in two contexts: curricular and competitive learning. In the former, teachers use practical classes, assignments and examinations to evaluate students' evolution. The latter relies on the competitiveness of students to increase their programming skills mostly in computer programming contests. In this last context, automated judge systems (or online judges) are used to run programming contests and to practice for such contests. These systems include automatic evaluators and many of these systems organize their own contests, such as, Mooshak (Leal, 2003), UVA-OJ (University of Valladolid Online Judge), SPOJ (Sphere Online Judge), DOMJudge and others.

2.2 Recent surveys

In the last decade several surveys appeared reporting PES features and trends.

Douce et al. (2005) review the history of the field from 1960s characterized by a number of projects that automatically assess student programming exercises using a test-based approach. Three generations of PES were identified: the first-generation was represented by several initiatives to automate testing, however their usability was confined to their particular computing laboratories. The second generation was characterized by command-line-based PES. The third generation made use of web-based technologies to leverage the use of PES worldwide and provide additional support for educators in the form of assessment management and reporting facilities. The paper also mentions four development directions in this field: evaluation of GUI programs, meta-testing (evaluation of the students' tests), service orientation adoption and use of interoperability standards.

Kirsti AlaMutka (2005) organizes PES features according to whether they need execution of the program (dynamic analysis) and/or can be evaluated from the program code (static analysis). In one hand, dynamic analysis is often used to assess functionality, efficiency, and testing skills. In other hand, static analysis is used to provide feedback from style, programming errors and software metrics. The authors conclude that automated systems approach should always be pedagogically justified and state that systems are in-house built and no common standards or interfaces exist.

Liang et al. (2009) details dynamic and static analysis methods of existing PES. The paper also enumerates several unsolved issues in this area such as security, algorithms for automatic generation of test data in dynamic analysis and low accuracy and precision of correctness in static analysis. Finally the authors claim as new directions in the PES development the content standardization.

Ihantola et al. (2010) gather information on PES from 2006 to 2010 and discuss their major features such as tests definition, resubmission policies and security features. The author expects new research to emerge from the following fields: integration of automatic assessment on LMS and automatic assessment of web applications.

Romli et al. (2010) enumerate approaches for automatic programming assessment, test data generation and integration of both. The authors conclude that there is a lack of use of existing test data generation techniques (commonly used to test software) in the scope of automatic programming assessment. The same survey made an exhaustive study on 41 assessment tools that appeared in the last 50 years focusing on the evaluation methods and test data generation techniques used. Dynamic analysis is the most used method to assess programs with 74% of the tools studied using it. This is explained since program correctness is the most important quality factor while evaluating a program. In dynamic analysis the test data assumes a relevant role. The process of creating tests can be labour demanding. Manual generation is time-consuming and error-prone and seldom covers the potential range of a program. In spite of these issues, the study shows that the most used method for feed the assessment systems with test data is through manual data entry. This is due to the complexity inherent to the automatic generation of test data.

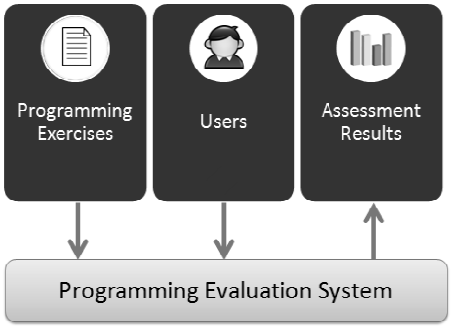
Beyond these facets, all above surveys stated the need for interoperability and security on PES. The former can be achieved by the creation and adoption of content and communication standards. The latter is a well-know issue that should not be overlooked and can be addressed by the use of secure environments (sandbox) to execute untested code and algorithms to filter out malicious code.

3 INTEROPERABILITY ANALYSIS

Based on the previous section, we conclude that interoperability is the main trend on PES. Moreover, this topic was never analysed in the above surveys.

Thus, we decided to survey existing PES regarding their interoperability features. Given the multiplicity of systems found we apply a multi-criteria approach for the selection of tools based on its effective use. The tools should be flexible enough to allow the **configuration of exercises** and the **management of users**. The former covers not only the selection of existing exercises on the evaluation tool but also the support for adding new exercises. The latter refers to the support of the tool for select users that will solve the exercises.

With this multi-criteria approach we selected 15 tools. After the selection of the tools, we applied an iterative process to identify which facets (current subsections) will be used to verify the interoperability maturity level of the selected tools. We began with an initial set of facets based on the issues and trends raised on the previous surveys in conjunction with our background in working with interoperability on automated assessment. Then, we read the published papers of the tools and consult their official websites and revised the facets. Figure 1 shows the selected facets.



PES.

These facets are also synchronized with the main objective of a typical automatic evaluation system - to evaluate a **user's** attempt to solve a **programming exercise** and produce an **assessment result**. Each facet includes three interoperability maturity levels:

Level 0 - manual configuration of data;
Level 1 - data import/export;
Level 2 - services invocation.

In order to belong to Level 0, the evaluation tool must support the configuration of data by allowing either the selection of existing data or the addition of new data. In the Level 1, the evaluation tool must also support the import/export of data from/to other sources. In the last level, the evaluation tool should also support the communication with other tools through the invocation of web services.

In the next subsections we detail the three facets and for each facet we present the respective interoperability maturity levels of the selected PES.

3.1 Programming Exercises

Nowadays we can find a large number of programming exercises. Despite their number, these exercises exist only in PES silos and seldom include mechanisms to share the exercises among researchers and instructors in an effective manner. Moreover, each of these systems dictates the persistent format of an exercise that may not be interoperable with other automatic evaluation systems. This is a significant barrier in the creation and sharing of programming exercises and can only be addressed through the standardization of exercise content and its storage on public repositories.

Based on these facts, we specialised the abstract maturity levels with the following:

Level 0 - manual configuration of exercises;
Level 1 - import/export of exercises;
Level 2 - integration with repository services.

In the Level 0, the evaluation tool should support the selection of exercises and the addition of new exercises. In this level, the tool relies on ad-hoc or internal formats to describe exercises data.

In the Level 1, the evaluation tool should also provide mechanisms to import/export exercises from/to other sources. In this level, the tool must explicitly support an exercise format. There are few exercise formats. Typically an exercise format can be obtained by modelling a programming exercise into a Learning Object (LO) definition. This definition describes an exercise as a learning package composed by a set of resources (e.g. exercise descriptions, test cases, solution files) and a manifest that describes the package and its resources in terms of its contents, classifications, lifecycle and several other relevant properties.

In the Level 2, the evaluation tool should also support the communication with other tools, typically LOR, through web services. A LOR is a system used to store and share learning objects. The repository should support simple and advanced queries to retrieve LO and export them to other systems through a set of web service flavours (e.g. SOAP, REST). In this communication, a service broker (e.g. exercise format conversion service) can be used when the evaluator does not support the format of the exercises stored in the repository.

Based on these levels we have prepared the following table that enumerates for each tool the maturity level regarding the management of programming exercises.

Table 1: Programming exercise facet (P-partial and F-full)

Systems	Level 0	Level 1	Level 2
AutoGrader	F	-	-
BOSS2	F	-	-
CourseMaker	F	-	-
CTPracticals	F	-	-
DOMJudge	F	-	-
EduComponents	F	-	-
GAME	F	-	-
HUSTOJ	F	P	-
Moe	F	P	-
Mooshak	F	F	F
Peach3	F	P	-
Submit!	F	-	-
USACO	F	-	-
Verkkoke	F	F	-
Web-CAT	F	F	P

According to the Table 1, all systems support the configuration of exercises. However, only six tools provide a way to export exercises and only three support bidirectional transfers with other sources. These systems often use exercises formats. HUST Online Judge uses FPS (FreeProblemSet) as an XML format for transporting exercises information between Online Judges. Peach3 system uses PEF (Peach Exchange Format) as a programming task package containing all task-related information and serving as a unit for storage and communication. Verkkoke system relies on SCORM packages to wrap all the exercise data.

The second level of interoperability is only achieved by Mooshak and partially by Web-CAT. Mooshak is a system for managing programming contests on the Web. The last version (1.6a2) supports the communication with repositories complying with the IMS DRI specification using a broker service responsible for the conversion between formats. Web-CAT is an automatic grading system using student-written tests. This system can communicate with other repositories, such as CollabX, through specific plug-ins. Unlike Mooshak, the interaction with repositories is not standard-based.

Based on these facts we can conclude that most systems use internal and proprietary formats. Those who adhere to explicitly formats do not reach a consensus to use a single format. This noncompliance to a single format leads to the standard fragmentation for describing exercise content. One solution to address this issue is instead of creating new formats we should start looking for broker services responsible for the conversion between formats.

Other issue is the relation with repositories of learning objects. The majority of PES store the exercises inside their systems hindering the proliferation and sharing of exercises. In order to communicate with repositories the evaluation systems must follow communication standards (e.g. IMS DRI) rather than ad-hoc implementations.

3.2 Users

In order to select and solve exercises users must be authenticated in the evaluation system and have authorization to submit their solutions. The users' facet also specialises the abstract maturity levels with the following:

> **Level 0** - manual configuration of users;
> **Level 1** - import/export of users;
> **Level 2** - integration with user directories services to provide authentication and academic management systems (AMS) to provide authorization.

In the Level 0, the evaluation tool should support the configuration of user's data.

In the Level 1, the evaluation tool should also provide mechanisms to import/export users from/to other sources. In this level, the tool can export a list of users based on standard formats. As far as we know, there are few standards that formalize users' data and how data is sent. Two know-standards are the IMS Learner Information Services (IMS LIS) and the IMS Learner Information (IMS LIP). The former is the definition of how systems manage the exchange of information that describes people, groups, memberships, courses and outcomes within the context of learning. The IMS LIS is focused on the connection between an LMS and an AMS. The latter addresses the interoperability of internet-based learner information systems with LMSs. It describes mainly the characteristics of a learner.

In the Level 2, the evaluation tool should also support the communication with other tools to provide authentication and authorization facilities. User authentication is based on directory services such as LDAP or Active Directory. User authorization relies on AMS that manages academic processes such as the enrolment of students in courses, the management of grades or the payment of fees. They are the best candidates to offer

authorization services since they store information about courses and students enrolled in them. The communication with AMS is not standardized. This fact burdens the integration of AMS with evaluation systems that must resort to ad- hoc solutions.

Table 2 shows the maturity level of automatic evaluation tools regarding the users' facet.

Table 2: Users facet (P-partial and F-full)

Systems	Level 0	Level 1	Level 2
AutoGrader	F	F	P
BOSS2	F	-	-
CourseMaker	F		
CTPracticals	F	F	P
DOMJudge	F	F	P
EduComponents	F	F	P
GAME	F	-	-
HUSTOJ	F	-	-
Moe	F	-	-
Mooshak	F	F	P
Peach3	F	-	-
Submit!	F	-	-
USACO	F	F	-
Verkkoke	F	F	-
Web-CAT	F	F	-

According to the Table 2, all systems support the manual configuration of users for a specific assignment or course. More than a half of the systems studied allow the import/export of users in non-standard formats. However only five partially support the communication with authentication services (mostly with LDAP). We can conclude that AMS are still immature in terms of standard communication with other systems since we do not found any system interacting with it. AutoGrader, CTPraticals, EduComponents and Verkokke beneficiate from the fact that they are integrated with LMS thus taking advantage of its authorization facilities.

3.3 Assessment results

After the student's submission the evaluation system assesses the program and returns an evaluation result. The assessment results facet also specialises the abstract maturity levels with the following:

Level 0 - visualization of evaluation results;
Level 1 - export of assessment results;
Level 2 - integration with LMS.

In the Level 0, the evaluation tool should support the visualization of the assessment results. The result data is essential for the success of an assignment and can include feedback and grades. This information should be present to the user on the evaluation tool graphical interface.

In the Level 1, the evaluation tool should also export evaluation reports to other sources. As far as we know, there are few standards that formalize evaluation results. A formalization of an evaluation report can be found in the Evaluation service (Leal et al., 2010) - a contribution for the E-Framework. An implementation of this service evaluates an attempt to solve a programming exercise and produces a detailed report. This evaluation report includes information to support exercise assessment, grading and/or ranking by client systems. The report itself is not an assessment, does not include a grade and does not compare students.

In the Level 2, the evaluation tool should also communicate with other tools. A typical scenario is the evaluation tool sends the grades to the LMS grade book. A common interoperability standard that is increasingly supported by major LMS vendors is the IMS Learning Tools Interoperability (IMS LTI) specification. It provides a uniform standards-based extension point in LMS allowing remote tools and content to be integrated into LMSs. Currently, only a subset (IMS Basic LTI) of this specification is implemented by the major LMS. This subset exposes a unidirectional link between the LMS and the application. For instance, there is no provision for accessing run-time services in the LMS and only one security policy is supported.

Table 3 shows the maturity level of PES regarding the assessment results facet.

Table 3: Assessment results facet (P-partial and F-full)

Systems	Level 0	Level 1	Level 2
AutoGrader	F	F	P
BOSS2	F	-	-
CourseMaker	F	-	-
CTPracticals	F	F	P
DOMJudge	F	F	-
EduComponents	F	F	P
GAME	F	-	-
HUSTOJ	F	-	-
Moe	F	-	-
Mooshak	F	F	-
Peach3	F	F	-
Submit!	F	-	-
USACO	F	-	-
Verkkoke	F	F	P
Web-CAT	F	F	-

Table 3 shows that all systems present the evaluation results to users and the majority allows its

exportation in non-standard formats. Regarding the communication with other systems, four systems support the communication with LMS by providing the evaluation results on the LMS grade book. AutoGrader, CTPraticals and EduComponents are integrated with specific LMS, respectively, CascadeLMS, Moodle and Plone. Verkkoke is the only that do not depends on a specific LMS and can be integrated on any LMS that supports the SCORM specification.

4 SYNTHESIS & CONCLUSIONS

In this section we start by synthesizing the interoperability facets of the PES included on the above survey. Figure 2 depicts the percentage of interoperability maturity of each PES.

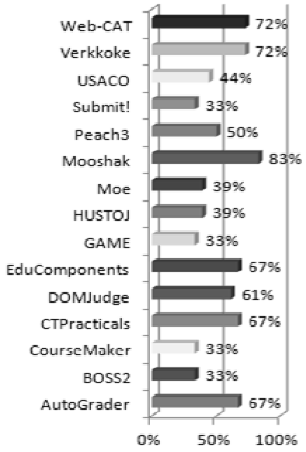


Fig. 2 Interoperability maturity percentage level of PES

We can conclude that a half of the systems studied did not reach 50% of the maturity rate. This illustrates that there are a lot to do in this field regarding the integration of PES with other systems.

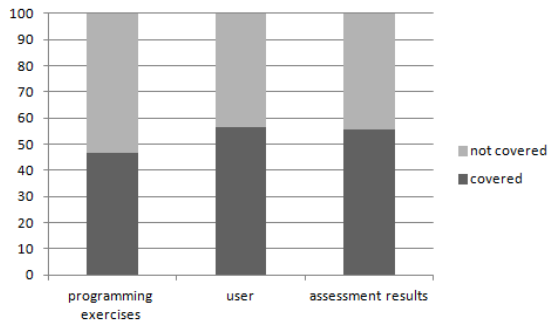


Fig. 3 Coverage of interoperability features

Figure 3 depicts the coverage of interoperability features of the PES studied organized by facet. The major conclusion to take is that there is no specific trend on interoperability facets since the distribution of interoperability coverage is equitably distributed among the three facets.

In this paper we present an interoperability survey on PES. Based on a multi-criteria approach we select 15 tools and organized the survey based on three interoperability facets: programming exercises, users and assessment results. For each facet we characterised each PES based on its interoperability level. Based on this study we detect two issues that can hinder the proliferation of PES: the lack of content standards for describing programming exercises and to communicate with other e-Learning systems.

This work fills the gap existent in most surveys since all of them point to interoperability as an issue for PES use and a trend for PES development but never explained in detail what are the paths to follow in order to achieve interoperability on this domain. The results achieved on this survey may also prove useful to instructors and computer science educators when they have to choose an assessment system to be integrated in their e-Learning environment.

REFERENCES

Ala-Mutka, K., 2005. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83-102.

Blumenstein M., Green S., Nguyen A. & Muthukkumarasamy V. 2004. An experimental analysis of GAME: a generic automated marking environment. *In Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, Leeds, United Kingdom, pp 67-71

Cheang B., Kurnia A., Lim A. & Oon W.C. 2003. On automated grading of programming assignments in an academic institution. *In Computer Education*, vol. 41, pp. 121–131.

Douce C., Livingstone D., Orwell J., 2005. Automatic test-based assessment of programming: a review. *In JERIC - Journal of Educational Resources in Computing*, 5(3):4.

Edwards S. H. &Pugh W. 2006. Toward a common automated grading platform. *In SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium*

on Computer science education, (New York, NY, USA), ACM.

Engels S., Lakshmanan V. & Craig M. 2007. Plagiarism detection using featurebased neural networks. *In SIGCSE*, pp. 34–38, 2007

Higgins, C.A., Gray, G., Symeonidis, P., Tsintsifas, A. 2005. Automated assessment and experiences of teaching programming. *In Journal on Educational Resources in Computing (JERIC)*, 5(3).

Ihantola, P., Ahoniemi, T., Karavirta, V. And Seppälä, O., 2010. Review of recent systems for automatic assessment of programming assignments. *In Koli Calling '10 Proceedings of the 10th Koli Calling International Conference on Computing Education Research*. ACM.

Jackson, D. & Usher, M. 1997. Grading student programming using ASSYST. *In Proceedings of 28th ACM SIGCSE Tech. Symposium on Computer Science Education*, San Jose, California, USA, pp 335-339.

Jena, S. 2008. Authoring and Sharing of Programming Exercises. *In Master's Projects*. Paper 19. http://scholarworks.sjsu.edu/etd_projects/19

Juedes, D.W. 2003. Experiences in Web-Based Grading. *In 33rd ASEE/IEEE Frontiers in Education Conference* November 5–8, 2003, Boulder, CO

Leal, J.P. & Silva F. 2003. Mooshak: a Web-based multi-site programming contest system. *In Software Practice & Experience, Volume 33 , Issue 6*. Pages: 567 - 581, 2003, ISSN:0038-0644

Leal, J.P., Queirós, R. & Ferreira D. 2010. Specifying a programming exercises evaluation service on the e-Framework. I*n Xiangfeng Luo, Marc Spaniol, Lizhe Wang, Qing Li, Wolfgang Nejdl and Wu Zhang (Eds), Advances in Web-Based Learning - ICWL 2010 - 9th Internation Conference*, Shanghai, China, December, 2010, LNCS 6483, pp. 141-150, ISBN 978-3-642-17406-3

Liang, Y., Liu, Q., Xu, J. and Wang, D. 2009. The recent development of automated programming assessment. *In CISE - Computational Intelligence and Software Engineering*, 1-5.

Luck M. & Joy M. 1999. A secure on-line submission system. *In Software - Practice and Experience*, 29(8), pp721--740

Malmi, L., Karavirta, V., Korhonen, A., Nikander, J. 2005. Experiences on automatically assessed algorithm simulation exercises with different resubmission policies. *In Journal on Educational Resources in Computing (JERIC)*, 5(3).

Mandal C. , Sinha, V. L. & Reade C. M. P. 2004. A Web-Based Course Management Tool and Web Services. *In Electronic Journal of E-Learning*, Vol 2(1) paper no. 19

Mandal A.K., Mandal C. & Reade C.M.P. 2006. Architecture Of An Automatic Program Evaluation System. *In CSIE Proceedings*

Mansouri, F., Cleveland, A. Gibbon, Colin, A. Higgins. PRAM: prolog automatic marker. *In Proceedings of ITiCSE'1998.* pp.166~170

Pisan Y., Richards D., Sloane A., Koncek H. and Mitchell S. 2003. Submit! A Web-Based System for Automatic Program Critiquing. *In Proceedings of the Fifth Australasian Computing Education Conference (ACE 2003)*, Adelaide, Australia, Australian Computer Society, pp. 59-68.

Queirós R. & Leal J.P. 2011. A Survey on eLearning Content Standardization. *In 4th World Summit on the Knowledge Society*, Mykonos, Greece.

Reek, K. A. 1989.The TRY system or how to avoid testing student programs. *In Proceedings of SIGCSE*, pp 112-116.

Rehak, D. R., Mason, R. 2003. Keeping the learning in learning objects. *In Littlejohn, A. (Ed.) Reusing online resources: a sustainable approach to e-Learning.* Kogan Page, London, 2003. (pp.22-30).

Romli, R., Sulaiman, S. & Zamli, K.Z., 2010. Automatic programming assessment and test data generation a review on its approaches. *In Information Technology (ITSim), 2010 International Symposium in*. DOI: 10.1109/ITSIM.2010.5561488

Saikkonen R., Malmi L., Korhonen A. 2001. Fully automatic assessment of programming exercises. *In Proceedings of the 6th annual conference on Innovation and Technology in Computer Science Education (ITiCSE)*, pp. 133–136, 2001

Striewe M. & Goedicke M.2010. Visualizing Data Structures in an E-Learning System. *In Proceedings of the 2nd International Conference on Computer Supported Education (CSEDU)* 2010, Valencia, Spain, volume 1, pages 172-179, 2010

Tang, C. M., Yu, Y. T., & Poon, C. K. 2009a. Automated systems for testing student programs: Practical issues and requirements. *In Proceedings of the International Workshop on Strategies for Practical Integration of Emerging and Contemporary Technologies in Assessment and Learning*, pp. 132±136

Tang, C. M., Yu, Y. T., & Poon, C. K. 2009b. An approach towards automatic testing of student

programs using token patterns. *In Proceedings of the 17th International Conference on Computers in Education (ICCE 2009)*, pp. 188±190

Tang C.M., Yu Y. T., Poon C.K. 2010. A Review of the Strategies for Output Correctness Determination in Automated Assessment of Student Programs. *In Proceedings of Global Chinese Conference on Computers in Education*.

Trætteberg H., Aalberg T.2006 . JExercise: A specification-based and test-driven exercise support plugin for Eclipse. *In Proceedings of the 2006 OOPSLA Workshop on Eclipse Technology eXchange*, ETX 2006 (2006), 70-74.

Verdú E., Regueras L.M., Verdú M.J., Leal J.P., Castro J.P. & Queirós R. 2011. A Distributed System for Learning Programming On-line. *In Computers & Education Jornal, 2011*, ISSN 0360-1315