International Journal of Engineering Education Vol. 30, No. 3, pp. 644–653, 2014
Printed in Great Britain

0949-149X/91 $3.00+0.00
© 2014 TEMPUS Publications.

Programming Assignments in Virtual Learning Environments: Developments and Opportunities for Engineering Education*

VALENTINA DAGIENE and BRONIUS SKUPAS
Vilnius University Institute of Mathematics and Informatics, 4 Akademijos Street, Vilnius 08663, Lithuania.
E-mail: valentina.dagiene@mii.vu.lt, bronius.skupas@mii.vu.lt

EUGENIJUS KURILOVAS
Vilnius University Institute of Mathematics and Informatics, 4 Akademijos Street, Vilnius 08663, Lithuania.
Vilnius Gediminas Technical University, Sauletekio Ave. 11, 10223 Vilnius, Lithuania. E-mail: eugenijus.kurilovas@itc.smm.lt

The aim of the paper is to present observations on automatic and semi-automatic assessment for programming assignments used in different e-learning contexts. Teaching of programming is an important part of different Informatics Engineering, Computer Science or Informatics, Computing, Information Technology and Communication courses in Universities and high schools. Students taking these courses have to demonstrate competences in problem solving and programming by creating working programs. Checking program validity is usually based on testing a program on diverse test cases. Testing for batch-type problems involves creating a set of input data cases, running a program submitted by a contestant with those input cases, analysing obtained outputs, etc. Assessment of programming assignments is as complex as testing of software systems. A lot of automatic assessment systems for programming assignments have been created to support teachers in submission assessment. However the problem of balance between the quality and the speed of assessment for programming assignments is important. Authors conducted the research on the possibilities of advanced semi-automatic approach in assessment, which can be used as compromise between manual and automatic assessment. A semi-automatic testing environment for evaluating programming assignments is developed, and the practical use of this system in Lithuania's optional programming maturity examination is presented. Presented research is useful for evaluating results of engineering education in general, and informatics/computer engineering education particularly.

Keywords: engineering education; programming assignments; computer program assessment; automatic and semi-automatic assessment; personalised feedback; virtual learning environment

1. Introduction

The aim of the paper is to present observations on automatic and semi-automatic assessment for programming assignments. Problem solving is an important part of most modern Informatics related studies (including Software Engineering, Computer Science, Computing, Information Technologies, Communication Sciences, etc.). Is problem solving in Informatics different from that of other science? Yes, it is. The main difference is that the student is designing an algorithm (program) following the exact requirements—rules of the game.

One of the main goals in teaching computer programming is to develop students' understanding of the programming principles. The understanding of programming concepts is closely related with the practice on solving programming exercises.

Programming assignments are typical to use as an indicator of problem solving competencies. However, programming is not an easy job: it requires much effort and specific skills. Programming is a creative process that encourages thinking and the integration of knowledge from various fields.

The teaching of programming influences the development of thinking (computational, algorithmical, logical. operational, etc.). Capability of constructing programs (algorithms) helps a student to conceive technical performance of a computer, forms conceptual understanding of information technologies. Constructing an algorithm (program) is a discovery for a student, training of his/her creative capabilities.

The problem is that it is not an easy task to evaluate a program created by a student. In computer science educational community, the widely spread manner of checking program validity is based on testing a program on diverse test cases. Testing for batch-type problems involves creating a set of input data cases, running a program submitted by a contestant with those input cases, analysing obtained outputs, etc.

There are two major approaches in evaluating programming assignments: automatic and manual evaluation. 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.

* Accepted 23 July 2013.

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" [1].

The automatic evaluation is a method in which a software system provides for the teacher student's work testing and grading results and facilitates the feedback process. The automatic approach tends to be very fast but still have problems with quality of evaluation. Manual evaluation on the contrary is fully performed by human evaluator and it is quite slow and requires a lot of time.

We faced a real life problem in assessment of maturity programming exam in Lithuania. It faces with problem of assessment for thousands of student solutions to programming assignments. Discussion on the maturity exam in Informatics has been presented in [2–4]. Assessment of student submissions is complicated as high percentage of solutions has different type of problems— they are failing to compile or to output correct answers. However results of this exam have high importance for student future as students are entering Universities by order provided by exam results.

We analysed automatic and manual assessment approaches and performed constructive research on mix of these approaches—*semi-automatic assessment*.

The rest of the paper is organised as follows: methodology of the research is described in Section 2, presentation and discussion are presented in Section 3, and conclusion—in Section 4.

2. Methodology of the research

We selected constructive research to use as a method for research. Kasanen et al. [5] propose six phases of the constructive approach: (1) find a practically relevant problem, which also has research potential; (2) obtain a general and comprehensive understanding of the topic; (3) innovate, i.e., construct a solution idea; (4) demonstrate that the solution works; (5) show the theoretical connections and the research contribution of the solution concept; (6) examine the scope of applicability of the solution.

In practice the steps of our research have been in other sequence — the process was both iterative and recursive. Main reason for this was that research was started seven years ago and some of solutions have been innovated for several times. The national exam rules and requirements have been changed also.

We selected problem to create high quality and efficient assessment method for programming assignments. For step (2), we performed analysis of publications and surveys, and also analysis of publically available tools. Some of tools were used for competitions in lessons. Other were analysed by use in their websites. Some of tools were analysed by documentation, as they were not publically available.

Innovative idea generated in step (3) was based on possibility to include interactivity to semi-automatic assessment for programming assignments. We made assumption that iterative process of fixing errors and solution testing by black box testing can help to evaluator to decide level of achievements of student.

For step (4), we developed a software system for semi-automatic assessment of programming assignments, which was used for experimental maturity exam assessment in Lithuanian secondary schools in 2006. Testing of innovative idea in real world was successful. The system for semi-automatic assessment of programming assignments was used in assessment of national-wide programming exam in Lithuania. It was improved several times through years and it is under constant development.

For step (5), we performed quantitative analysis of assessment results. Use of the system was analysed by qualitative analysis of specific set of student submissions and their errors.

In step (6), we made assumption that improved method for semi-automatic assessment can be applied for other assessment systems for programming assignments. Analysis of required changes to include new approach into Edujudge plug-in (see: http://uva.onlinejudge.org) for Moodle virtual learning environment (VLE) was performed.

3. Presentation and discussion

Automated evaluation of programs is useful for faster evaluation of tasks of maturity examination in programming. However, unpreparedness of the candidates to strictly specified automated evaluation causes some problems. Therefore, using semi-automated evaluation cannot be avoided in this situation.

Semi-automated evaluation raises high demands to the quality of criteria and the common attitude of the evaluators in the evaluation situations. The quality of semi-automated evaluation is increasing when big amount of submissions are evaluated. However, it is difficult to force group of evaluators to very similar thinking in algorithm analysis.

More precise evaluation of the students' program codes can be achieved by combining automated testing and manual evaluation of programs into semi-automated evaluation. It poses higher requirements to the quality of criteria and the attitudes of evaluators at the evaluation situations. The criteria should be precisely developed together with the exam tasks, tested, and after examination, reasonably discussed by the evaluators. The quality of the semi-automated evaluation grows recognisably when number of evaluated programs increasing.

A system for semi-automatic assessment of programming assignments has been developed and used in assessment of national-wide programming exam in Lithuania. It can be integrated in VLE and used as a formal grading method or as self-evaluation tools for the students.

3.1 Observations in publications and tools analysis

The *research questions* addressed were as follows: What kind of improvements to automatic or semi-automatic assessment for programming assignments can help to improve quality of assessment for not finished programs? How to make automatic or manual assessments more efficient?

Several high quality surveys of automatic testing systems have been published since 2005 (Table 1). We have found analysis of most important trends in development of assessment tools for programming assignments. However, our specific issue with assessment of the quality of not finished programs was not addressed there.

There are two main areas of use for these systems: curricular (e.g. practical classes, assignments and exams) and competitive (e.g. corporative contests like Google Code Jam or learning society organised contests like International Olympiad in Informatics or ACM-ICPC International Collegiate Programming Contest). Our analysis of requirements for systems has found that exam requirements are somehow different, but major problems in quality of assessment are the same: automated assessment is usually based on back-box testing, which is good for speed. However, black-box testing has some set of concerns, most important is formulated by Dijkstra in 1972: "program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence" [11].

3.2 Alternative evaluation and semi-automatic testing

Our research was directed to maturity exam, but most research results can be applied to other systems. It is quite clear that automatic testing of student's codes is not enough for the maturity exam. Problem is that most students are not at high level in programming. Quite a lot of submitted codes are not functional even with data set provided in task description.

Researchers found [12, 13] that combination of two totally different approaches (automatic and manual) can give semi-automatic assessment, which combines talents of human and machine. As stated by Ahoniemi and Reinikainen [14], semi-automatic assessment can generate much higher quality feedback.

The typical semi-automatic system include automatic evaluation component, which provide data about student's program passed tests. These results are visualised for human evaluator, who is responsible for filling evaluation form, confirming final grade and providing feedback for student. Our experience in teaching of programming shows that in stress (exam, important test) most students failing to provide properly functioning program. About 80% of student submitted programs are failing to compile. However, this does not mean that students are incapable to write good programs.

Several our group researches demonstrated that incorporating semi-automatic assessment could help achieve higher quality in assessment for not functioning programs while providing acceptable speed. We made hypothesis that most important part of these results are based on interactive approach which we included into system developed for national matura exam in programming of

Table 1. High quality surveys of automatic testing systems

Author	Methodology	Trend
Douce et al. [6]	Details features of systems organised in 3 generations.	Evaluation of GUI programs, meta-testing (evaluation of the students' tests), Service Oriented Architectures and use of interoperability standards.
Ala-Mutka [7]	Organises systems in dynamic and static evaluators.	Content and communication standardisation.
Liang et al. [8]	Details dynamic and static analysis methods of systems.	Security, algorithms for automatic generation of test data and content standardisation.
Ihantola et al. [9]	Discuss features of 2006–2010 systems (e.g. tests definition, resubmission policies and security).	Integration with VLE and assessment of web applications.
Romli et al. [1]	Evaluate approaches for test data generation.	Test data generation techniques, interoperability and security.
Queirós et al. [10]	Analyse interoperability features of systems.	Integration with VLE, export and import features of systems.

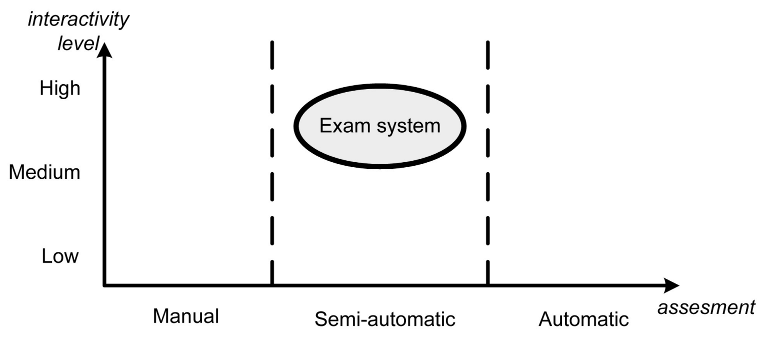

Fig. 1. Different approaches in assessment for programming assignments.

Lithuania. This approach is providing ability for evaluator to retest student program after modification while tracking the changes. Such system can be classified to filled area in Fig. 1. Main idea of improvement to typical semi-automatic assessment can be presented like diagram in Fig. 2.

Evaluators' team is trying to evaluate solutions positively. This means that students getting points for their shown effort, e.g. correct input/output routines can be assessed with several points. Also some points can be received for dividing program to subroutines, for using complex data structures like array or record, for writing nice comments, for good programming style, etc. These criteria can be easily evaluated by a person, but computer evaluation is not so obvious. This is the reason for manual evaluation of solutions.

Several years' observation of evaluation process of the national exams showed that evaluators need an interactive semi-automatic evaluating environment, as some solutions have only some small syntax problems, like semicolon missing.

Automatic evaluation in most cases is one of the most objective ones as it is highly related to strict programmed rules [15, 16]. However, discussions about maturity exam in programming evaluation raised the hypothesis that evaluation in this exam cannot be limited to automated testing. Such hypothesis is based on opinion that students should get score for their effort in not working programs. We also tried to identify typical pitfalls in automatic assessment which can be overcome by using semi-automatic evaluation. Some typical human-evaluator errors were identified in the research process.

During semi-automated evaluation, the evaluator can use the data obtained during static and dynamic analysis, modify the submitted program and re-run automated evaluation. At this stage, the evaluator has to decide how significant were the mistakes of the candidates, which important parts of the assignment he/she solved, which programming constructions the candidates mastered. Many problems arise at this point due to splitting opinions

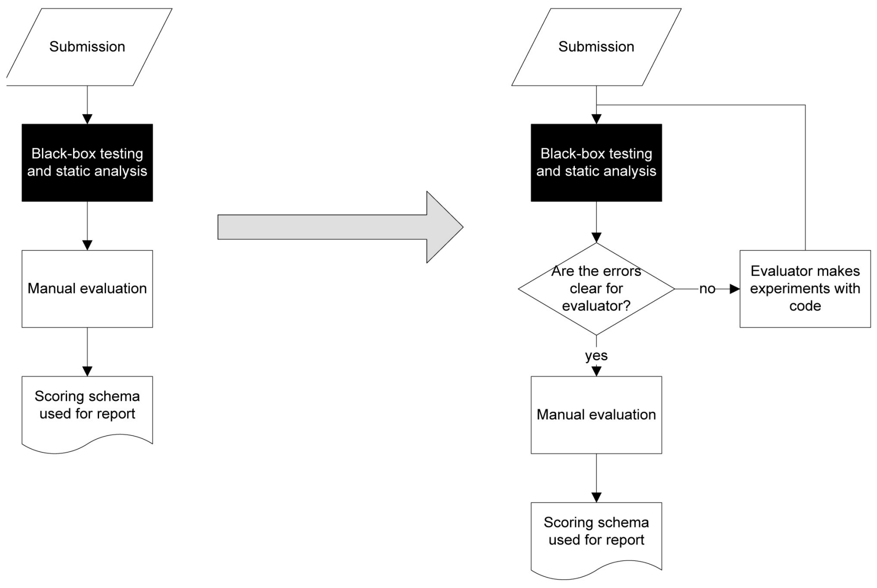
Submission
Black-box testing and static analysis
Manual evaluation
Scoring schema used for report
Submission
Black-box testing and static analysis
Are the errors clear for evaluator?
no
Evaluator makes experiments with code
yes
Manual evaluation
Scoring schema used for report

Fig. 2. Improvement to semi-automatic assessment for programming assignments process.

of the evaluators, different level of their experience, etc.

It is highly important to prepare properly for this evaluation stage, i.e. thoroughly prepare and specify the evaluation scheme. However, it is not easy to prepare for the alternative evaluation because:

The criteria have to:

- be clear, unambiguously understandable to the evaluators,
- exactly match possible acceptable solutions,
- award the points to the candidates for the knowledge and skills they demonstrated.

The points for the criteria should:

- correspond to the matrix of the examination,
- be proportional to weight of knowledge.

The use of small dichotomic (yes/no) criteria increases objectivity [14]. So, usually criteria are corrected and divided into smaller ones (Table 2) after several packages of work of evaluators. After this, evaluated codes must be re-graded with modified criteria. The biggest problem of such evaluation is that it requires a lot of human working hours.

The researcher's task is to probe the data in a way that helps to identify the crucial components that can be used to explain the nature of the evaluation process of the exam. In our research, we are focussing on a mixed approach of widely spread 'quantitative' and 'quality' research. We are starting with categorising the data, looking for recurrent issues, capturing ideas, and then we are making interpretation of the findings and alternative explanations that is more common to qualitative research. As

Table 2. Programming task evaluation criteria after reviewing of evaluators

First task: evaluation criteria	**Points**	**Comments**
All tests.	20	Full points if the program provides correct outputs to all tests.
Correct reading from file:	4	
1. Preparing file for reading, closing after the reading is finished.	(1)	
2. Number of students.	(1)	
3. The first loop for reading the number of students.	(1)	
4. Reading the pieces brought by the students (only part of data are read correctly).	(1)	
Calculation of paws, knights, rooks and bishops (it might be either in the function or in other place of the program, e.g. in the input).	4	
5. Initial values of sums.	(1)	
6. The loop.	(1)	
7. Cumulating the sum (for small mistakes –1 point).	(2)	
The function, which calculates the number of chess sets that can be collected from the pieces brought by the students is created (search for the smallest number among the calculated pieces):	5	Evaluated only if the program scores no points for the tests. (Alternative evaluation criteria)
8. Initial value of minimum (e.g. min : = 100 – 1 point).	(2)	
9. Search loop, conditional sentences.	(2)	
10. Assigning new value to the minimum.	(1)	
The result is outputted correctly:	2	
11. File is prepared for printing, closing it after printing is finished.	(1)	
12. Outputting the result.	(1)	
13. Correct calculation of sets for each different piece (there are some mistakes, but it is correct in general – 1 point).	2	
Other functions, procedures (if there are ones) and the main program are correct:	3	
14. A user declared one-dimensional array data type. If the array is not used in the calculations or is used unreasonably, then 0 points.	(1)	
15. No loop and other auxiliary global variables.	(1)	
16. No syntax errors.	(1)	
The data type of one-dimensional array and the corresponding variables are declared correctly.	1	Always evaluated manually
The function which performs the indicated calculations is crated and the call to the function is correct.	2	
Meaningful names of the variables. Program parts are commented, spelling is correct. (If at least two items are followed – 1 point).	1	
Programming style is consistent, no statements for working with the screen (if at least one item incorrect – 0 points).	1	
Total	25	

we know, quantitative approaches tending to shape their data more consciously and more explicitly in earlier stages of the process and focus more on number compared with qualitative approaches.

The analysis of research data in relation to both quantitative and qualitative methods tends to follow a process involving some stages, e.g. data preparation, initial exploration of the data, analysis of the data, representation and validation of the data.

3.3 Data preparation and analysis procedure

The first stage in the analysis of quantitative data is to organise the data in a way that makes them more easily understood. There were investigated the programs designed during the maturity examination in programming in 2010. All data of exam evaluation (including programs, criteria's, evaluators, every evaluator's scores for program) was put into MySQL database. Specific web-based tool for easier analysis was built using PHP language.

After initial browsing of programs and evaluations one interesting feature was found: a lot of programs have rather high evaluation in alternative evaluation. This can occur when program is really close to good one. It was difficult to select which score is best showing this phenomenon. Decision was made that all points for alternative evaluation should show something important.

For the case study, there were selected programs that failed the tests but scored full points from at least one evaluator in the alternative evaluation. Quantitative data about the number of analysed programs are presented in Table 3.

The programs were chosen for the case study because presumably they were close to the working ones. Otherwise, they would not have scored full points in the alternative evaluation. Such programs comprise 5% of the submitted programs and that confirms the hypothesis that evaluation in the maturity exam in programming cannot be limited to automated testing.

3.4 Analysis of interactivity influence to semi-automatic evaluation

The research highlighted certain attitudes of the evaluators: the scores of alternative evaluation were not identical, i.e. if one evaluator assigned

Table 3. The number of submitted and analysed programs

Tasks	**Programs submitted in exam**	**Analyzed programs**
1	1224	34
2	1019	62
Total	2243	96

full points, the other one might have not done so. We analysed the criteria, marking which resulted in most disagreements (i.e. different scores for the same program) among the evaluators. There was identified that either the criterion was applied not precisely enough, or its formulation was too vague to apply in unambiguously for some programs.

National maturity programming exam in Lithuania consists of several parts; one of them is based on programming practice. Students must write two batch style programs. These programs are assessed using common grading schema which is made of three steps. The first step is automatic assessment based on black box testing. The third step is manual assessment on criteria like programming style, comments, use of known algorithms and data structures. The second step involving semi-automatic assessment is most important to our research. Semi-automatic assessment in exam takes place only in case when student submitted program fails to provide correct answer to at least one test. In this case, all points for automatic testing are rejected but teacher gains possibility to write points for other student achievements during semi-automatic assessment process.

Assessment of student submission is done by two independent evaluators, and in case of difference in points higher than 2, submission is evaluated by third evaluator. In all exam sessions, it was observed, that no more than 20% of submissions get third evaluation, which is performed by the best experts and assumed as finale grade.

An example of such a criterion could be the use of global variables. Programs with a globally declared array were interpreted differently by the different evaluators. Another example could be the need to initialise a global variable for storing the sum. Some evaluators assumed that there was no need to initialise it because Free Pascal performs initialisation automatically.

During the case study, there were analysed typical mistakes and the reasons why the programs failed during automated testing.

One of common situations was that the students used different dialects of Pascal. The evaluation system uses Turbo Pascal 7.0 dialect which standard de facto for teaching and for various programming contests. Free Pascal has many different language extensions which allow using name of function as a parameter, allow using empty parenthesis when declaring functions, etc. The evaluators modified the sources as little as possible and the programs passed the tests during retesting. Different evaluators interpreted the feature of the program which required modification in different ways depending upon their experience.

Another group of mistakes are programs with

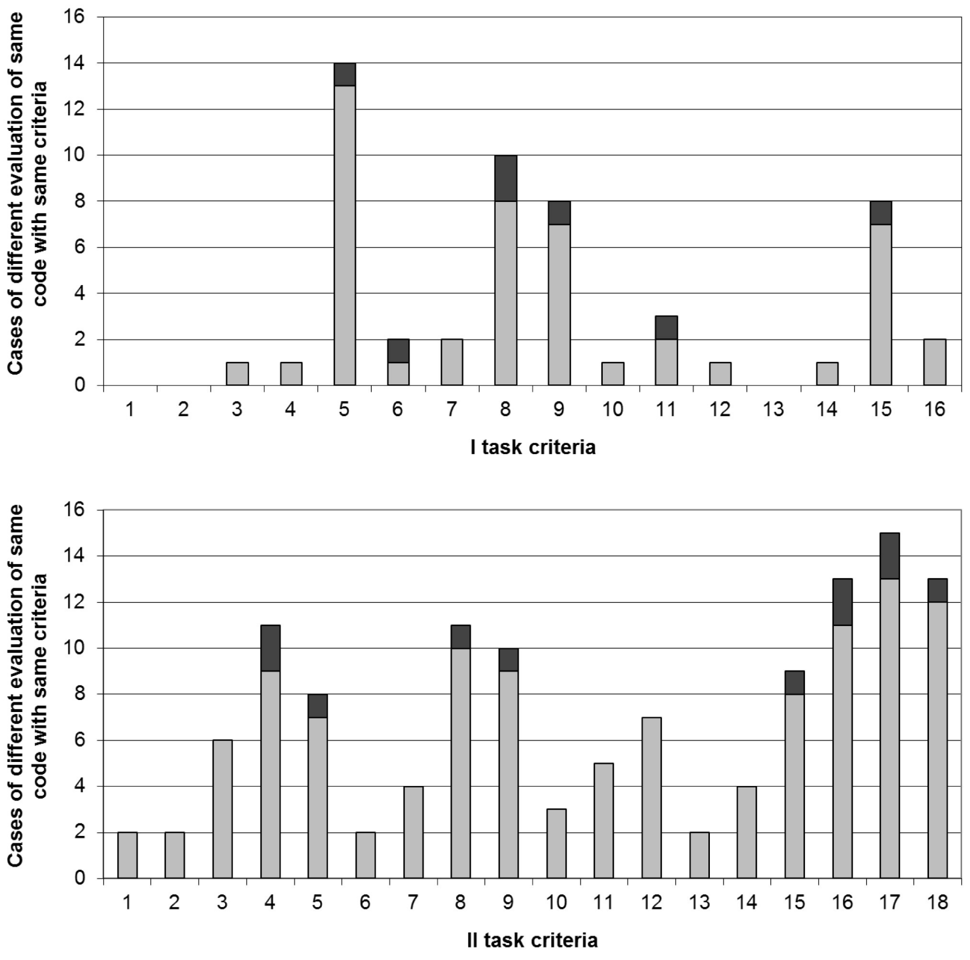

Fig. 3. The level of disagreement between the evaluators varied depending upon the criteria (cases between second and third evaluator are marked in darker colour).

errors which occurred due to carelessness and rush: misspelled file names, important source fragments are commented, the variables are left uninitialised. We may guess that it is hard to avoid such mistakes due to the exam stress and the limited time.

The level of disagreement between the evaluators varied depending upon the criteria (Fig. 3). Besides the criteria that were already mentioned, the evaluators had different attitudes at initialisation, calculation and return of function parameters passed by reference. This shows that manual evaluation of these criteria causes problems for the evaluators.

Purely technical criteria can also be distinguished in the histogram, e. g. reading data from a file. Apparently they caused no doubts for the evaluators because the programs did function properly and it was easy to estimate the value of the mistake.

It can also be concluded that there were less differences between the third and the second evaluation than between the first and the second. This can be explained by the rising qualifications and experience of the evaluators in terms of the task.

Typical marking scheme for programming assignment involves 20 points for automatic testing (all or nothing scoring schema), 20 points for semi-automatic assessment (only in case when automatic testing provides 0 points), and 5 points for manual assessment. In Fig. 4, histogram demonstrates influence of semi-automatic evaluation to final score. It is clear that students after semi-automatic assessment are spread in histogram in better way.

In Fig. 4, points (0–25) are presented in horizontal axis and student count in vertical. Histogram in black present results without semi-automatic assessment, and grey histogram present final results for assignment.

We have also made experimental evaluation of the same student submissions. Evaluators had to evaluate 50 programs for the second time after two months. They had no possibility to use semi-automatic exam system and worked in manual manner. This experimental evaluation took at least twice more time. This proves that interactive semi-automatic assessment is more efficient than manual.

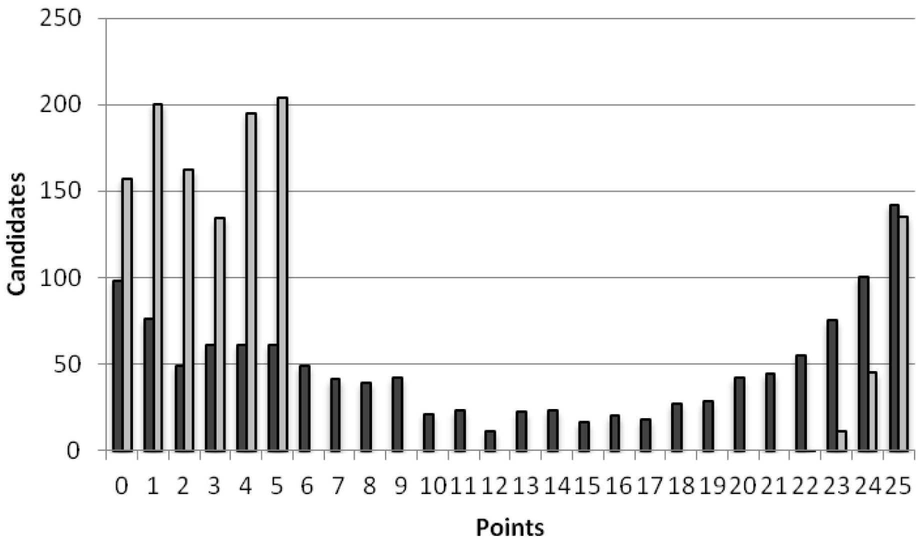

Fig. 4. The histogram of one assignment in matura exam.

Actually, this result was such as expected. We also checked difference in points in both evaluations. 5% of submissions had bigger than 2 points difference of evaluation.

Qualitative analysis of student submissions with maximum points reached in semi-automatic evaluation demonstrated that most students had almost finished program, but failed in syntax errors, like unfinished comment, missing semicolon, not proper I/O file name, etc. However, these programs are evaluated in 0 points on most automatic assessment systems. This demonstrates higher quality of exam evaluation to failing submissions.

Our goal is not only to improve the evaluation process of the maturity exam but also to enhance the students' motivation while choosing the programming module. Studies on teaching programming have indicated various difficulties faced by the students when learning to program. In our opinion, this should not be an excuse for not teaching programming at lower levels of education, e.g. at secondary schools. On the contrary, we feel that the well-organised and well-structured maturity exam should serve as a motivating factor for introducing programming at an early stage. Personalised learning style based learning programming with individual assessment also should be taken into account [17].

On the other hand, improved process of semi-automatic evaluation has a high potential for VLEs. This approach can be easily implemented into many VLE systems like Moodle by using third party plug-ins like Edujudge or Virtual Programming Lab (see: http://vpl.dis.ulpgc.es/). It was found that most changes can be done by simple modifications [18–23]. Most of attractive features of new method like more precise feedback with "fixed" program for student and "the movie of program fixing" can be implemented without extraordinary effort.

4. Discussion

Constructed interactive semi-automatic method for programming assignments is suitable for specific task. However, it provides ideas for further developments.

One of possible use for such style assessment is high quality feedback generation for VLEs. We performed analysis on possibilities to use specified method in Edujudge plug-in for VLE Moodle. It was found that most changes can be done by simple modifications. The most complicated part is related to storage of teacher submitted fixed student solution.

In the future, we would like to put more attention to errors made in the exam by evaluators and to look for correlations with mistakes made by the students during the learning process. A thorough analysis on the students' program codes must be done and student survey after exam should be conducted. It could help to develop a better exam and assist the students in preparing for the exam.

Automated evaluation of programs is useful for faster evaluation of tasks of maturity examination in programming. However, unpreparedness of the candidates to strictly specified automated evaluation causes some problems. Therefore, using semi-automated evaluation in this situation cannot be avoided.

More precise evaluation of the students' program codes can be achieved by combining automated testing and manual evaluation of programs into semi-automated evaluation. It poses higher requirements to the quality of criteria and the attitudes of evaluators at the evaluation situations. The criteria should be precisely developed together with the exam tasks, tested, and after examination, reasonably discussed by the evaluators. The quality of the semi-automated evaluation grows recognisably when number of evaluated programs increasing.

Studies on teaching programming have indicated various difficulties faced by the students when learning to program. In our opinion, this should not be an excuse for not teaching programming at lower levels of education, e.g. at secondary schools. On the contrary, we feel that the well-organised and well-structured maturity exam should serve as a motivating factor for introducing programming at an early stage. Personalised learning style based learning programming with individual assessment also should be taken into account.

Further studies could also include qualitative analysis of the student knowledge in the preparation period before programming exam and the comparison with their achievement during the exam to verify hypothesis that psychological stress during the exam influences the results of the exam significantly.

5. Conclusions

Presented research on interactive semi-automatic method for programming assignments assessment is useful for evaluating results of informatics/computer engineering education. Combination of two totally different approaches (automatic and manual) can give semi-automatic assessment which combines talents of human and machine. Semi-automatic assessment can generate much higher quality feedback. Constructed interactive semi-automatic method for programming assignments assessment is effective and provides higher quality points to not functioning student submissions.

Semi-automated evaluation raises high demands to the quality of criteria and the common attitude of the evaluators in the evaluation situations. The quality of semi-automated evaluation is increasing when big amount of submissions are evaluated. However, it is difficult to force group of evaluators to very similar thinking in algorithm analysis.

The goal of the presented research is not only to improve the evaluation process of the maturity exam but also to enhance the students' motivation while choosing the programming module.

References

1. R. Romli, S. Sulaiman and K. Zamli. Automatic programming assessment and test data generation a review on its approaches, *International Symposium in Information Technology* (ITSim 2010), **3**, 2010, pp. 1186–1192.
2. J. Blonskis and V. Dagienė, Evolution of informatics maturity exams and challenge for learning programming. *LNCS 4226*, 2006, pp. 220–229.
3. J. Blonskis and V. Dagienė, Analysis of students' developed programs at the maturity exams in information technologies, *LNCS 5090*, 2008, pp. 204–215.
4. D. Vitkutė-Adžgauskienė and A. Vidžiūnas, Problems in choosing tools and methods for teaching programming, *Informatics in Education*, **11**(2), 2012, pp. 271–282.
5. E. Kasanen, K., Lukka and A. Siitonen, The Constructive Approach in Management Accounting Research, *Journal of Management Accounting Research*, **5**(1), 1993, pp. 243–263.
6. C. Douce, D. Livingstone and J. Orwell, Automatic Test-Based Assessment of Programming: A Review, *ACM Journal of Educational Resources in Computing*, **5**, 2005, pp. 1–13.
7. K. M. Ala-Mutka, A survey of automated assessment approaches for programming assignments, *Computer Science Education*, **15**, 2005, pp. 83–102.
8. Y. Liang, Q. Liu, J. Xu and D. Wang, The Recent Development of Automated Programming Assessment, *International Conference on Computational Intelligence and Software Engineering* (CiSE 2009), 2009, pp. 1–5.
9. P. Ihantola, T. Ahoniemi, V. Karavirta and O. Seppälä, Review of recent systems for automatic assessment of programming assignments, *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (Koli Calling 2010), ACM, 2010, pp. 86–93.
10. R. Queirós and J. P. Leal, Programming Exercises Evaluation Systems—An Interoperability Survey, CSEDU, 1, 2012, pp. 83–90.
11. E. W. Dijkstra, The humble programmer, *Communications of the ACM*, ACM Turing Lecture, **15**(10), 1972, pp. 859–866. Accessed 29 December 2012. http://www.cs.utexas.edu/users/EWD/transcriptions/EWD03xx/EWD340.html.
12. D. Jackson, A semi-automated approach to online assessment. *Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '00)*. ACM, 2000, pp. 164–167.
13. M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting and T. Wilusz, A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *SIGCSE Bull.*, **33**(4) 2001, pp. 125–180.
14. T. Ahoniemi, T. Reinikainen. ALOHA – a grading tool for semi-automatic assessment of mass programming courses. *Proceedings of the 6th Baltic Sea Conference on Computing Education Research (Koli Calling'2006)*, 2006, pp.139–140.
15. B. Cheang, A. Kurnia, A. Lim and W. C. Oon, On automated grading of programming assignments in an academic institution, *Computers & Education*, **41**, 2003, pp. 121–131.
16. C. A. Higgins, G. Gray, P. Symeonidis and A. Tsintsifas. Automated assessment and experiences of teaching programming. *ACM Journal on Education Resource in Computing (JERIC)*, **5**(3), 2005, Article 5.
17. I. Beres, T. Maguar and M. Turcsanyi-Szabo. Towards a personalised, learning style based collaborative blended learning model with individual assessment, *Informatics in Education*, **11**(1), 2012, pp. 1–28.
18. E. Verdú, L. M. Regueras, M. J. Verdú, J. P. Leal, J. P. de Castro and R. Queirós, A distributed system for learning programming on-line, *Computers & Education*, **58**(1), 2012, pp. 1–10.
19. J. P. Leal and R. Queirós, Integrating the LMS in Service Oriented eLearning Systems, *International Journal of Knowledge Society Research*, **2**(2), 2011, pp. 1–12.
20. I. Novo-Corti, L. Varela-Candamio and M. Ramil-Díaz. Using Moodle Platform Online to Work out on Solving Multiple Options Questions on Microeconomics: Notes on Gender Differences, *International Journal of Knowledge Society Research*, **3**(2), 2012, pp. 65–74.
21. J. E. Labra Gayo, P. Ordóñez de Pablos and J. M. Cueva Lovelle, WESONET: Applying Semantic Web Technologies and Collaborative Tagging to Multimedia Web Information

Systems, *Computers in Human Behaviour*, **26**(2), 2010, pp. 205–209.

22. M. D. Lytras and P. Ordóñez de Pablos, Competencies and Human Resource Management: Implications for Organizational Competitive Advantage, Special Issue on Competencies Management: Integrating Semantic Web and Technology-Enhanced Learning Approaches for Effective Knowledge Management, *Journal of Knowledge Management*, **12**(6), 2008, pp. 48–55
23. M. Lytras, P. Ordóñez de Pablos, M. Mantziou and O. Mantziou, Information and communication technologies and challenges for the management of education: new managerial perspectives, *International Journal of Management in Education*, **1**(3), 2007, pp. 199–213.

Valentina Dagiene is professor at Vilnius University, Lithuania (MS in Applied Mathematics, PhD in Computer Science, Dr. Habil in Education). Her research interests focus on the computer science (informatics) education, teaching algorithms and programming, also localization of educational software. She has published over 200 scientific papers and methodological works, has written more than 50 textbooks in the field of Informatics and Information Technology for primary and secondary education. She works in various expert groups and work groups, organizing the Olympiads and contests. She is Editor of international journals "Informatics in Education" and "Olympiads in Informatics". She has also participated in several EU-funded R&D projects, as well as in a number of national research studies connected with technology and education.

Eugenijus Kurilovas is Head of International Networks Department of the Centre of Information Technologies in the Ministry of Education and Science of Lithuania, Associate Professor in Vilnius Gediminas Technical University, and Research Scientist in Vilnius University Institute of Mathematics and Informatics. He holds PhD in Informatics Engineering. His research interests focus on technology enhanced learning. He has published over 80 scientific papers, 2 monographs, and 4 chapters in scientific books. He is reviewer and member of 30 editorial boards and committees of international scientific journals (incl. 6 indexed/abstracted in ISI Web of Science) and conferences (incl. 7 indexed / abstracted in ISI Web of Science). He has also participated in about 30 large scale EU-funded R&D projects, as well as in several international research studies such as STEPS, SITES, and ICILS.

Bronius Skupas works at Vilnius University Institute of Mathematics and Informatics (Lithuania) and also senior teacher of Computer Science at Vilnius Lyceum. He holds PhD in Informatics Engineering. His research interests focus on teaching programming. He developed semi-automated assessment system for student programs. He is a leader of Lithuanian Olympiad in Informatics Technical Committee, member of Lithuanian Computer Society Council. He has published a number of scientific papers and participated in several EU-funded R&D projects.