Informatics in Education, 2010, Vol. 9, No. 2, 229–237

© 2010 *Institute of Mathematics and Informatics, Vilnius*

Feedback Improvement in Automatic Program Evaluation Systems

Bronius SKŪPAS
Institute of Mathematics and Informatics
Akademijos 4, LT-08663 Vilnius, Lithuania
e-mail: bskupas@ktl.mii.lt

Received: January 2010

Abstract. Automatic program evaluation is a way to assess source program files. These techniques are used in learning management environments, programming exams and contest systems. However, use of automated program evaluation encounters problems: some evaluations are not clear for the students and the system messages do not show reasons for lost points. The author proposes several ideas for possible improvements in black box testing, which can lead into better service for the users of automatic evaluation systems.

Keywords: automatic program evaluation, feedback.

Why We Need Automatic Program Evaluation in Contests and Teaching

Nowadays it is common to have programming classes in secondary schools and universities. The assessment of programming assignments places significant demands on the instructor's time and other resources (Douce *et al.*, 2005). This demand has led to the creation of automatic evaluation systems.

Students could be given different programming assignments, which are assessed using automatic program evaluation systems. Such systems provide a possibility for the student to submit solution, test it with some set of tests, and receive some feedback. These systems also provide feedback for the teachers and evaluators.

In general the systems provide feedback generated by a wide range of static and dynamic program analyses (Ala-Mutka, 2005). Development of automatic program evaluation systems with high quality feedback show good results (Malmi *et al.*, 2005; Higgins *et al.*, 2005).

Use of automatic evaluation systems in programming exams is not so popular. The main reason is the approach to marking the submissions during the exams. Some programming errors should not influence the final score heavily. This cannot be achieved using only a black box testing. A typical approach is to use semi-automatic systems for programming exam evaluation. Feedback from the automatic program evaluation part is helpful for the examiner in search for the errors (Skūpas and Dagienė, 2008). Analysis methods used in these systems are similar to methods used in systems developed for teaching.

Automatic evaluation systems are also typical for the programming contests. There they are used as a fair and efficient way of marking and distinguishing the solutions (Ribeiro and Guerreiro, 2009). Most contests (including IOI – International Olympiad in Informatics) use dynamic program analysis, black box testing approach. Feedback in these systems helps the contestants to examine program in real testing environment. However, feedback in most contest systems is rather poor and discussions about other possible grading methods are still open (Pohl, 2006; Forišek, 2006).

Typical Automatic Evaluation System

A typical approach for automatic evaluation systems is to provide interfaces for 3 classes of users: administrators, teachers/evaluators and students. Possibilities for the users are rather different in the systems. However, it is possible to find something in common in most of them. Fig. 1 presents the most typical use cases for automatic evaluation systems.

As is shown in the use case diagram, automatic evaluation systems include a lot of use cases which are responsible for task data handling, task preparation, user management etc. Most of the current systems are providing Web interfaces for the users. Popular platforms for this type of systems include Tomcat application server, Apache web server with PHP, and Python. MySQL database is the most popular for storing submissions.

Typical Evaluation Process Workflow

Most important part for feedback generation in automatic evaluation systems is the use case “Evaluate submission(s)”. This part usually is separated in a grading client process, which is usually running on separate computer(s). The grading clients are responsible for program evaluation and feedback generation.

Most grading clients’ workflow is similar in different automatic evaluation systems. It involves preparation of sandbox (safe running environment), running of compilers, preparation of input data, analysis of program provided output, and generation of feedback.

One of possible client workflows is presented in Fig. 2. OWINF grading system (Polley, 2006) has an advanced feature in test data organization: It is possible to define testcases – groups of tests (testruns). A testcase is graded considering results of all testruns.

Runcave – the name of sandbox used in this client. The sandbox is required to run programs in a safe environment. Usually in a POSIX compliant system, the programs are linked to static executables and run in chroot’ed environment with low privileges. Sandboxes usually are written in C and C++ languages (Mareš, 2007).

Most of automatic program evaluation systems provide several feedback possibilities for submissions. Typical feedback messages include the following:

- program compilation error,
- program run time error X (program finished with signal X) in testrun Y,
- program run time limit exceeded in testrun Y,

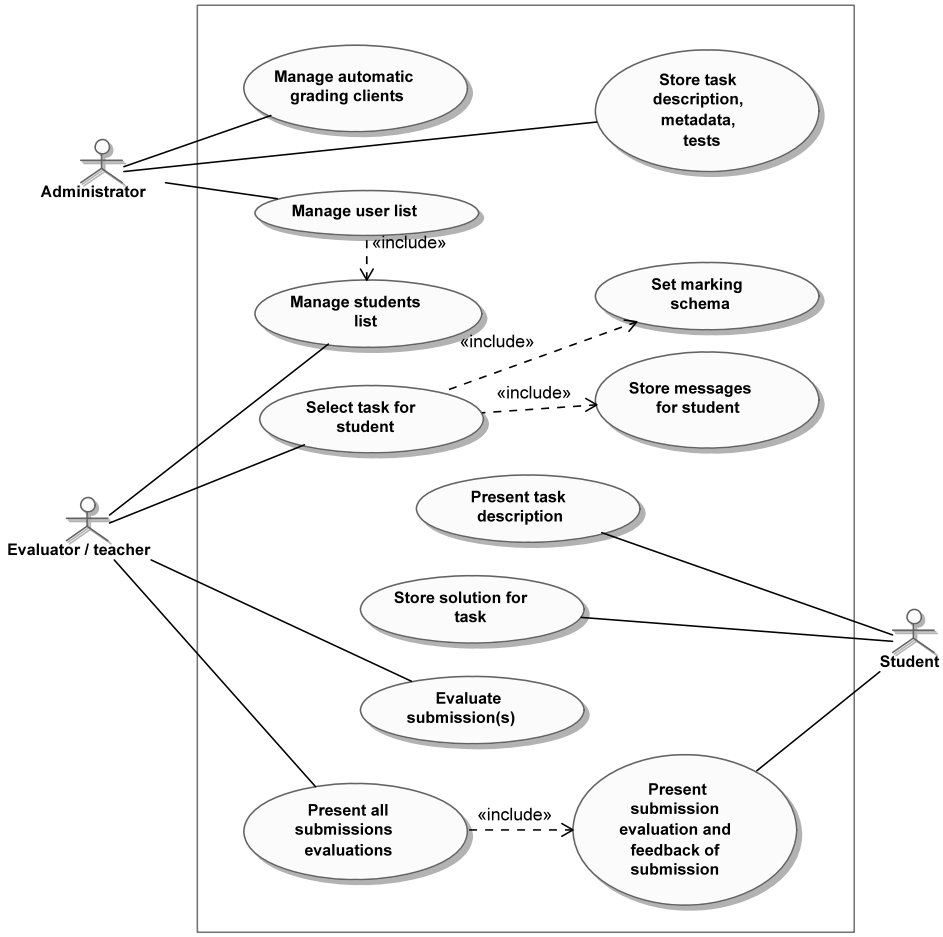

Fig. 1. Use case diagram for typical automatic evaluation system.

- program provided wrong answer in testrun Y,
- program provided correct answer in testrun Y.

Feedback is accumulated through all the grading process and sent back to the evaluation system after grading of all testcases have been completed. Grading is performed by the **grader (evaluator) process**, which generates most important feedback messages about output correctness.

Grader (Evaluator) Process

Operations performed in the grader process depend on the type of the task. For example OWINF can handle three (typical for IOI) task types (Polley, 2006; Mareš, 2007):

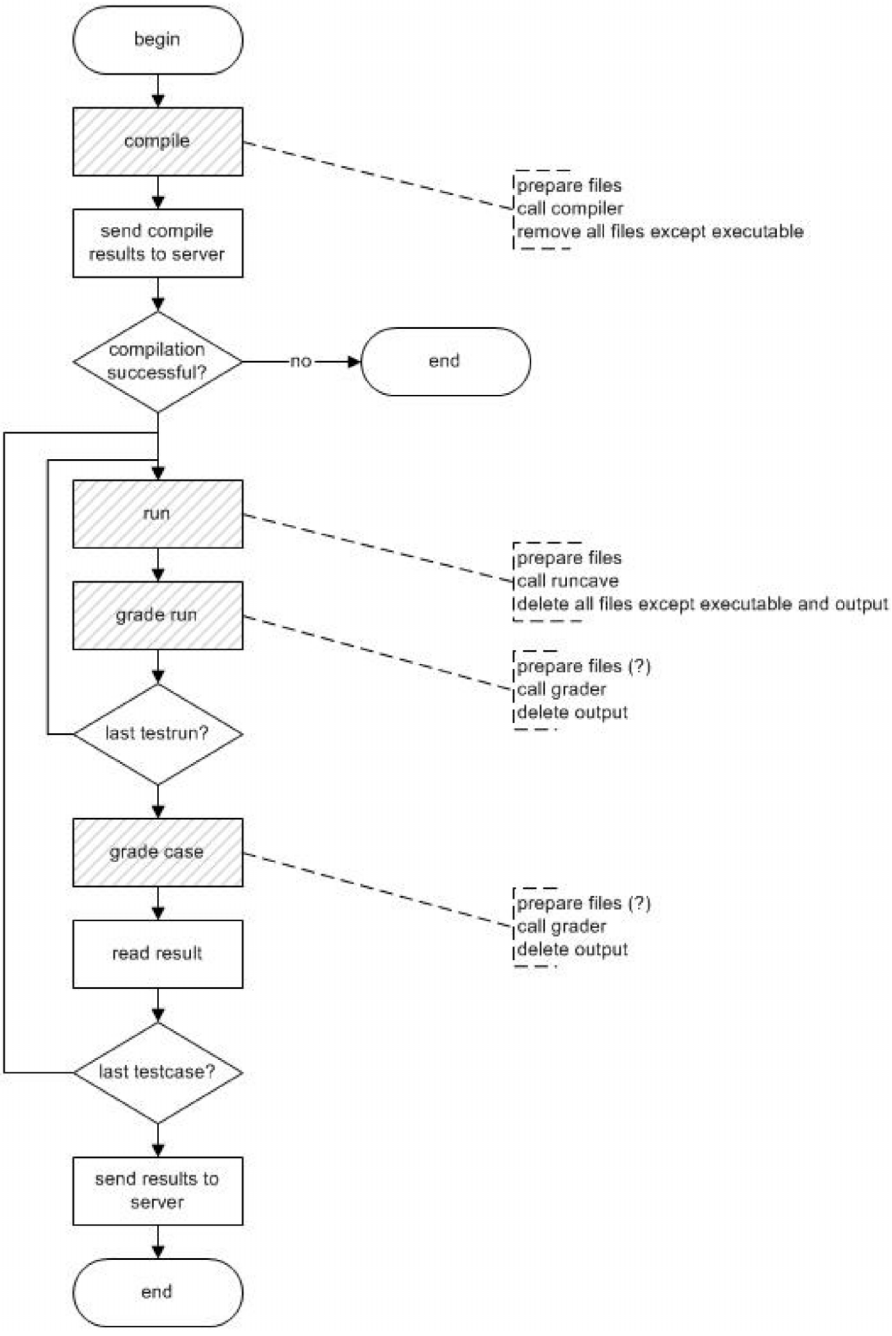

Fig. 2. OWINF automatic grading system client workflow (Polley, 2006).

- *Batch tasks (input/output tasks):* programs which transform an input file into an output file.
- *Reactive tasks (interactive tasks)*: programs which call library functions of a provided library.
- *Output only tasks (open-data tasks)*: the submission consists only of an output data file.

Batch task type is most popular in automatic evaluation systems. The main reason for this can be:

- Easy task presentation on paper, no need of specific libraries (opposite to reactive tasks). Tasks were prepared in a long period of time in different textbooks and transferred to automatic evaluation systems.
- Students are forced to write programs (opposite to output only tasks).

However batch task submissions raises problems in evaluation systems:

- Students must follow task description about strict data input and output formats.
- Program execution time is highly dependant on efficiency of data stream reading in programming languages and their compilers.

The grader process usually uses three input files and one output file as illustrated in Fig. 3. In simple unambiguous tasks usually it is enough to compare the student program generated output with the correct solution. However there are a lot of tasks where more than one correct solution is possible.

Another problem is related to strict answer format. Most of the students which use automatic evaluation systems are not top programmers, but students. This raises problem with errors in student programs related to line endings, extra spaces, blank lines in end of output, case of letters, etc. Adding or changing of these symbols in output files messes

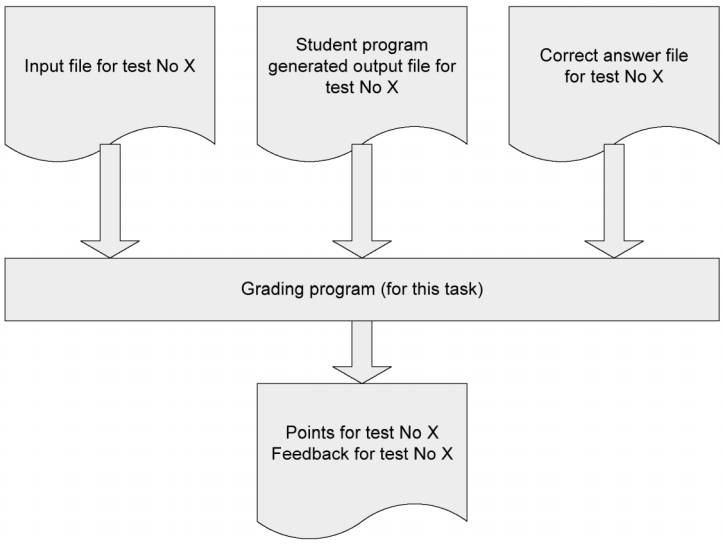

Fig. 3. Typical grading process has 3 input files and one output file.

up files comparison process and students can get 0 points for a rather good program. This can be solved by using simple output formats (Vasiga *et al.*, 2008).

The third problem is that the feedback of file comparison is not so evident if the output and the answer files are large.

All of the mentioned problems can be solved using a specific grading program for each task, which is not so easy. Martin Mareš (Mareš, 2009) proposes to use standard judges with different possible customizations on required comparisons. However this approach is not so comfortable for ambiguous tasks – these standard judges are not suitable for them.

In general, feedback of automated program evaluation in the contests is rather poor. Some contests (e.g., ACM ICPC) provide only general feedback for all the tests. One reason is that most of the contests are pointed to best programmers. Another point is that extra help provided from grading system will influence results of contest.

Grading tools, developed for a specific task can provide better feedback, but these are not so common, as they requires extra work. Scoring for partially correct outputs is not common also.

Automated program evaluation systems used for teaching faces another problem – authors of such systems try to have huge banks of tasks. Preparation of specific grader algorithms for every task is too expensive.

Suggestion for Better Feedback with Less Effort

It was considered that automated evaluation systems must have better feedback. Important suggestion is to provide new type feedback message – "output file format is not correct". This message must provide detailed analysis of provided output file and show wrong place. This can be reached using output file syntax analysis.

We already have practice to use this type of change: LOI (Lithuanian Olympiads in Informatics) automated evaluation system grading client (based on IOI'2002 system with several minor changes) was patched to support cascading graders. Typical use of these cascaded graders included two grading steps. The first step is responsible for student program output format checking. The second step is related to output correctness (Fig. 4).

Such a change required to have two graders per task. However it was easily solvable, as first grader was easily programmable with specific parser based library. Second grader was programmed without thinking about possible format problems so it was easier to create even for ambiguous tasks.

Two step grading worked slower, as it read output files twice. Bad output format files were rejected from second analysis and this saved part of the time. So required time for testing was not twice longer and it was not a big disadvantage.

Feedback transferring to students required deeper system analysis and it was not easy as we had to patch information transfer protocol and to secure that all other system components will "understand" new messages.

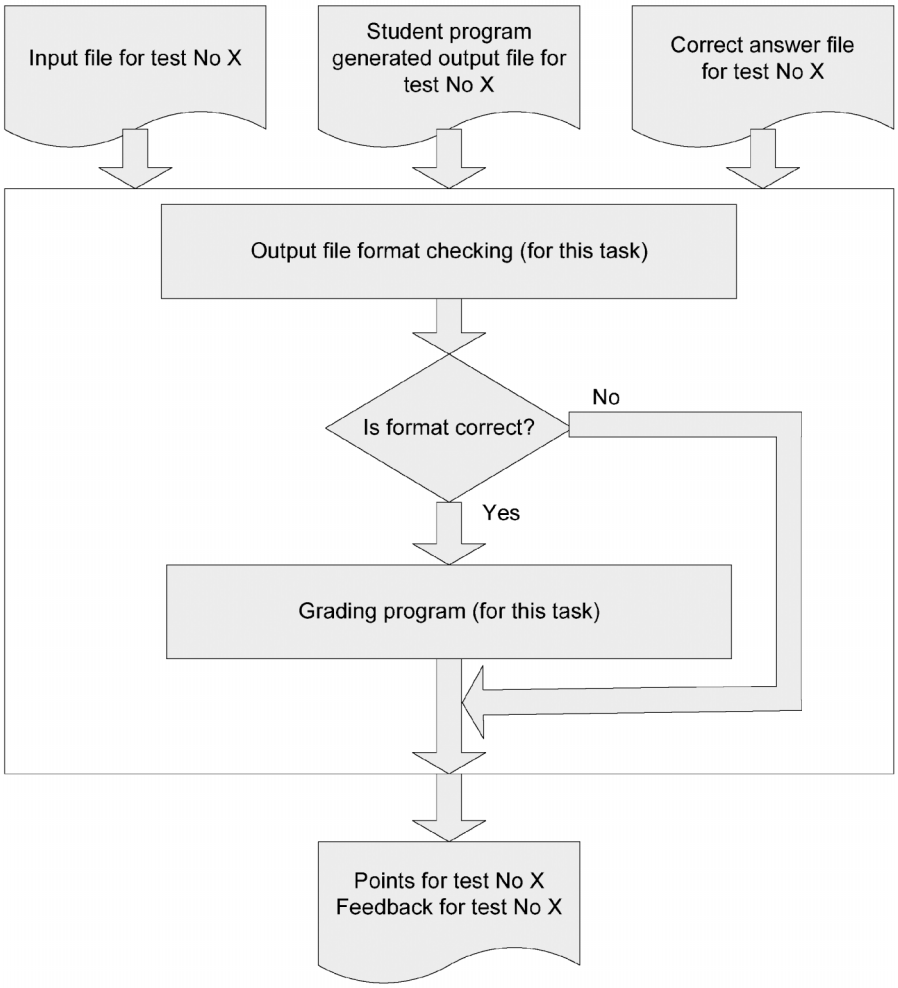

Fig. 4. Cascading grader structure used in Lithuanian Olympiads in Informatics.

Student Opinions about Feedback in Patched System

This approach was used for several years and in year 2009/2010 Lithuanian Olympiad moved to another evaluation system (OWINF). We used this technique with new system after easy grading script patching. However sending output format error feedback to students showed to be not so easy.

Because of this situation it was possible to discuss with a group of 5 students about advantages and disadvantages of the new system. Two students stated that they would like to have “old format error messages”, which are missing in the new system.

Students also mentioned new nice features of the new system like “all submissions testing”, which helps to analyze their performance in contest time, they really liked “after

a contest" mode, which presented a possibility to test other modified submissions after contest with all testcases.

Discussions with students demonstrate that they still need more feedback about their errors. They also would like to have more clear feedback about run-time errors. However the suggested solution does not help a lot in this place. Probably a patched Linux kernel and better sandbox are required for the grading client.

In Lithuanian Olympiad in Informatics it is typical to have several tasks with 10% points for programming style. Students were interested about more clear feedback concerning lost points for programming style.

Students were aware of new possibilities, but they still thought about possible improvements in the students interface. This demonstrates that we must pay more attention to our "customers" – the students and we must analyze all possible ways to improve feedback and provide more services for them.

Conclusions

Proposed two step grader showed to have more positive features than negative. It helped to prepare tasks for evaluation more quickly. It is easier to separate several parts of task preparation and to make technical work for administrator.

Olympiads showed these positive features of this suggestion:

- *More clear feedback.* Students can distinct programming errors from output format errors. A failing output format checker creates compiler style feedback for students with extract of bad output and an arrow, pointing to bad character.
- *Easy format checker programming.* Format checking is done using a specific parser based library. Format description is done with several short commands in a high level language.
- *Easy grader programming.* It can be done much easier as a programmer can be sure about data format in files. Less programming errors occurs during programming.
- *Approach is easily adoptable to different evaluation systems.* This approach was transferred to OWINF grading system, which was used for Lithuanian Olympiad in Informatics for the first time in 2009/2010.

Negative features:

- *The grading process is slower.* Each output file is read twice in the average.
- *The evaluation system administrator is forced to learn one more library or to write his own.*
- *Feedback from format checker can be misinterpreted by other system components.*

The success of this patching also gives an idea to use more than one cascading steps in other place of evaluation – in compilation. This can extend automatic grading systems to use static analysis tools, plagiarism detection systems, etc. These systems can provide an objective statistical programming style analysis, assure on the authenticity of programs.

References

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

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

Forišek, M. (2006). *On* the suitability of tasks for automated evaluation. *Informatics in Education*, 5(1), 63–76.

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

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

Mareš, M. (2007). Perspectives on grading systems. *Olympiads in Informatics*, 1, 124–130.

Mareš, M. (2009). Moe – design of a modular grading system. *Olympiads in Informatics*, 3, 60–66.

Pohl, W. (2006). Computer science contests for secondary school students: approaches for classification. *Informatics in Education*, 5(1), 125–132.

Polley, T. (2006). *OWINF Contest System*. Available at: `http://owinf.de/book/compiled/single-page-html/`.

Ribeiro, P., Guerreiro, P. (2009). Improving the automatic evaluation of problem solutions in programming contests. *Olympiads in Informatics*, 3, 132–143.

Skūpas, B., Dagienė, V. (2008). Is automatic evaluation useful for the maturity programming exam? In: *Proceedings of 8th International Conference on Computing Education Research*, 117–118. `https://www.it.uu.se/research/publications/reports/2009-004/2009-004.pdf`.

Vasiga, T., Cormack, G., Kemkes, G. (2008). What do olympiad tasks measure? *Olympiads in Informatics*, 2, 181–191.

B. Skūpas is a PhD student in the Informatics Methodology Department in the Institute of Mathematics and Informatics. He has been working in Vilnius Lyceum as teacher of informatics since 1995. Also he is a member of the Technical Committee of the National Olympiads in Informatics. His research interests include informatics and algorithmic contests, teaching informatics, computer science in secondary education, and automatic grading systems.

Automatinių vertinimo sistemų grįžtamojo ryšio tobulinimas

Bronius SKŪPAS

Automatinio vertinimo sistemos yra populiarus įrankis studentų ir mokinių parašytų programų vertinimui. Šis vertinimo metodas naudojamas vrtualiose mokymo terpėse, egzaminų vertinimui, programavimo varžybose. Tačiau taikant šias sistemas iškyla problemų – mokiniams nebūna aišku, kodėl jie prarado taškus taip pat iškyla problemų ieškant galimų klaidų. Autorius siūlo nesunkiai diegiamą patobulinimą, tinkamą daugeliui sistemų besiremiančių juodosios dėžės principu. Šis patobulinimas turėtų palengvinti užduočių parengimą automatinei vertinimo sistemai. Taip pat tai gali padėti mokiniams suprasti jų padarytas klaidas.