Informatics in Education, 2010, Vol. 9, No. 1, 141–158
© 2010 *Institute of Mathematics and Informatics, Vilnius*

Improving the Evaluation Model for the Lithuanian Informatics Olympiads

Jūratė SKŪPIENĖ
Institute of Mathematics and Informatics
Akademijos 4, LT-08663 Vilnius, Lithuania
e-mail: jurate@ktl.mii.lt

Received: August 2009

Abstract. The Lithuanian Informatics Olympiads (LitIO) is a problem solving programming contest for students in secondary education. The work of the student to be evaluated is an algorithm designed by the student and implemented as a working program. The current evaluation process involves both automated (for correctness and performance of programs with the given input data) and manual (for programming style, written motivation of an algorithm) grading. However, it is based on tradition and has not been scientifically discussed and motivated. To create an improved and motivated evaluation model, we put together a questionnaire and asked a group of foreign and Lithuanian experts having experience in various informatics contests to respond. We identified two basic directions in the suggested evaluation models and made a choice based on the goals of LitIO. While designing the model in the paper, we reflected on the suggestions and opinions of the experts as much as possible, even if they were not included into the proposed model. The paper presents the final outcome of this work, the proposed evaluation model for the Lithuanian Informatics Olympiads.

Keywords: informatics olympiads, programming contests, algorithms, evaluation, grading.

1. Introduction: Structure of LitIO

The Lithuanian Olympiads in Informatics is an individual competition on algorithmics, open for every interested student in secondary education. The competition follows the model of the International Olympiads in Informatics (IOI), the most famous international competition in programming (algorithmics) for individual contestants in secondary education from various invited countries. There are many other competitions for secondary school students (and for others) which follow a similar model as the IOI (IOI; Poranen, 2009; ACM-ICPC; TopCoder; BOI'2009; CEOI'2009).

The Lithuanian Olympiads in Informatics (Dagienė, 2007, 2004) consist of three rounds and focus on algorithmic problem solving. The contestants get two or three algorithmic tasks in each round. They have to design an algorithm which would solve the given task, write a short verbal motivation for the algorithm, and implement[1] the algorithm in one of the allowed programming languages, currently Pascal, C and C++.

[1]There have been other types of tasks in LitIO which do not require submission of an implementation (e.g. theoretical tasks), but they are very rare in LitIO and will not be part of this investigation.

The solutions (programs together with verbal descriptions in the form of comments) have to be submitted through a web-interface of a *Contest Management System* (CMS), which provides various services needed to manage the contest (Mareš, 2007). Lithuania started using a CMS in its Informatics Olympiads in 2003 (Skūpienė, 2004). When a contestant submits a solution, the program is compiled and executed with some small sample tests. However, other bigger tests may also be included in the acceptance test, if the task designers can motivate that. All submission errors are immediately reported back to the contestant. The contestant has the possibility to work on his program, to improve it and resubmit it again. There is no penalty for resubmission. Thus it is assured that the submitted programs compile and satisfy input/output formatting requirements. Executing a program with sample tests helps to achieve this. The grading procedure consists both of a human part and black-box grading parts. The verbal algorithm description and programming style are graded by human evaluators, whereas each submitted program is executed with various input data by the grader in CMS. Typically, the points, awarded for each part, are summed to get the final grade (Dagienė, 2007).

The number of contestants in the final round varies from 50 (on-site contest) to 300 (on-line contest). On average there are five to eight jury members available for task design and grading. They agree to spend up to three to five working days for preparing one task for senior students and up to two to three days for preparing one task for junior students. Those who have more time available are preparing more than one task.

The scores have to be produced either in four-five hours (on-site contest) or in two weeks (on-line contest). However, no member of the jury will agree to spend more than a couple of full working days on grading for one contest.

The Lithuanian Informatics Olympiad in many aspects follows the International Olympiad in Informatics. At the same time it inherits evaluation problems which have become a topic of discussion in the IOI community as well as in scientific papers in recent years (Cormack, 2006, Forišek, 2006; Verhoeff, 2006). One of the main questions is to which extent black-box grading results satisfy the expectations of the jury. The nature of black-box grading suggests no knowledge of internal logic and structure of the program. Another question (specific to the Lithuanian Olympiads) is evaluating the verbal descriptions of algorithms and relating the evaluation of programming style to the overall correctness of a solution (typically to the results of black-box testing). For example, a program which does not try to solve the task should not be awarded points for programming style (Skūpienė, 2006).

2. The Concept of Quality

There exist different perspectives on evaluation in informatics contests. In Cormack (2006), achievement is chosen as the central perspective and the evaluation in the contests is discussed from this perspective. Our perspective on the evaluation in this paper will be the quality of the solution. Before going any further we will discuss the concept of quality in general.

Hoyer (2001) indicates that quality is mainly defined in one of two ways. One of them is conformance to a specification, i.e., the product is of a good quality if its measurable characteristics satisfy specified characteristics. The second understanding is meeting the customer's needs. In this case, quality is the capability to satisfy the needs of the client and it is not related to any measurable characteristics.

Berander (2005) gives an overview of different understandings of quality. P.B. Crosby indicates that quality is neither luxury nor elegance, but just strict conformance to requirements (Crosby, 1979). In his view the standard for quality must be very precise and "close enough" is not acceptable. Following Deming (1990) quality is conformance to user requirements, but this becomes more difficult when there is a need to describe the user requirements using measurable characteristics. He claims that the quality can be described only in terms of the agent, i.e., the concrete judge of the quality. According to A.V. Feigenbaum, quality can be determined only when the customer is using the product. However, the needs of the customer are changing and therefore the concept of quality is also changing. K. Ishikawa also supports the idea that quality is the ability to conform to the requirements of the customer. At the same time he emphasizes that international standards (ISO, IEEE) are not perfect and are not flexible enough to keep in pace with constantly changing consumer requirements. W.A. Shewart describes two views on quality. On the one hand, quality is the objective reality independent of the existence of designers and users. On the other hand, it is a subjective perspective dependent upon the feelings and senses as a result of the objective reality.

To summarize we can say that there is no absolute definition of quality; it is a constructed and changing concept.

The same holds for the quality of a solution in a contest. The term *solution* often is used in a broader sense. It might include possible solution, expected solution or theoretical solution. Therefore we will talk about *submission* (and *quality of a submission*) which consists of an algorithm implementation in one of the allowed algorithmic languages and/or other material required by the task, *submitted for evaluation.* I.e., *submission* is the material presented for the examination of the jury.

3. Approach to Creating an Improved Evaluation Model

As we concluded in the preceding section, quality is either conformance to very concrete specifications or conformance to the needs of the users. However, in this case, there is only one acting subject. It is the scientific committee who determines the specifications and at the same time is the only user of the submitted solutions[2]. Therefore the scientific committee has the final word in determining the quality of the submission and should take into account the contest goals and resource limitations, the problem solving part of the contest and the software quality models (Skūpienė, 2009b).

[2]According to the LitIO regulations there is only one body (the National Scientific Committee) which performs the functions of the scientific committee (it is responsible for preparing a contest syllabus, tasks and tests) and the jury (it is responsible for the evaluation).

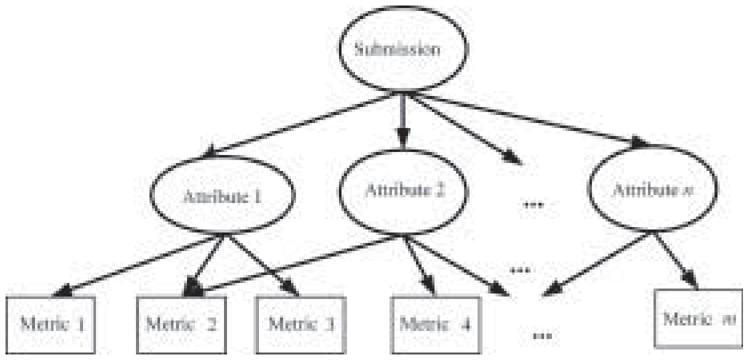

Fig. 1. The framework of evaluation model.

To discuss this we invited a group of ten experts to participate. By *an expert* we understand a person having at least several years experience of working in informatics contests either as a member of the scientific committee or as a jury member. We made an exception and invited as an expert one person who had one year experience of being a member as scientific committee. However, he had been the participant and the winner of many national and international Olympiads before he joined the NSC of LitIO. Eight out of ten experts were involved in the contests for more than ten years. Even though the object of discussion is evaluation in LitIO, in order to discuss it from a broader perspective, the group of experts consists half of Lithuanian and half of international members, all having experience in various national, regional and international contests. As the experts were located remotely, there was no interaction between them as a group during the work.

The experts were provided all the relevant information about the structure, scope and available resources for LitIO. Creating an evaluation model consists of identifying aspects that need to be measured and defining metrics that measure each aspect. The following tree model was used as the basis of the questionnaire distributed to the experts.

We asked the experts to answer the following questions:

1. *What attributes of a submission are most relevant for determining a score and can be objectively measured? You can restrict yourself to what you consider the five most relevant attributes.*
2. *What metrics would you suggest to measure these attributes (more than one metrics could be used to measure each attribute). Define each metric as precisely as you can.*
3. *How would you suggest to implement each metric (taking into account the resources and limitations described below). Metrics can be implemented by a manual measurement procedure, or by an automated procedure, or by a combination. Describe each procedure as precisely as you can.*
4. *How would you suggest to integrate the separate metrics to get one score (for one submission).*[3]

[3]Score aggregation is not discussed in this paper and the responses of the experts to this question will be used in further research.

The concept of *submission* was also included into the evaluation model, i.e., the experts could suggest to alter it. The second level is the *attribute* level. The third level contains *metrics* for measuring these attributes. This model was developed on the basis of the *Goals, Questions, Metrics* (GQM) approach which is a mechanism for defining and interpreting measurable software (Basili, 1994).

3.1. *Defining the Concept of a Submission*

Currently the submission consists of a verbal description of an algorithm and the source code of the implemented algorithm. It should be noted that the algorithm presented in the verbal description does not have to match the implemented algorithm. Therefore, the verbal description is graded independently of the implementation. It is graded even if the implementation is not submitted.

The experts came with two types of suggestions which could be classified into two totally opposite categories. One group of experts suggested to shrink a submission to the source code only, while the other group was suggesting to expand it by including test data as a new element of a submission.

The verbal description of an algorithm was the first submission element to be discussed.

The opinions expressed by the experts were totally opposite. Some of them suggested removing the verbal description from the submission, motivating that *the main skill of a contestant is whether he can make work his program or not; any other efforts outside this (e.g. verbal description with impressive ideas) are of a little help if the program does not work*. Others were more lenient, but strongly emphasized that these are of secondary importance and *whatever is included into the evaluation model, functional correctness and efficiency are the main attributes to be evaluated, ... verbal algorithm description is for those who didn't have enough time to implement their solution, but not for those who have failed to implement it. The contestants should submit either verbal description or the implementation, but not both.* In the responses of some experts it was mentioned that it is most likely that the verbal description of an algorithm is used for identifying heuristics. This helps to achieve that submissions with heuristic algorithms will not get a full score. The experts suggest other ways to try to avoid this, rather than through a verbal description.

The other experts suggested to include an implementation description or the reasoning for the design. Currently the required algorithm description may have no connection with the implementation. The reasoning for the design might come in the form of a separate text or comments and *ideally, once the design decisions have been established, the program code is a straightforward derivative (synthesis).*

None of the experts clearly supported the current practice where the verbal algorithm description may have no connection with the implementation. There had been lengthy discussions in the scientific committee of LitIO whether it should be required to describe the implemented algorithm or just any algorithm which solves the given problem. The decision not to connect the description and the implementation was motivated by the

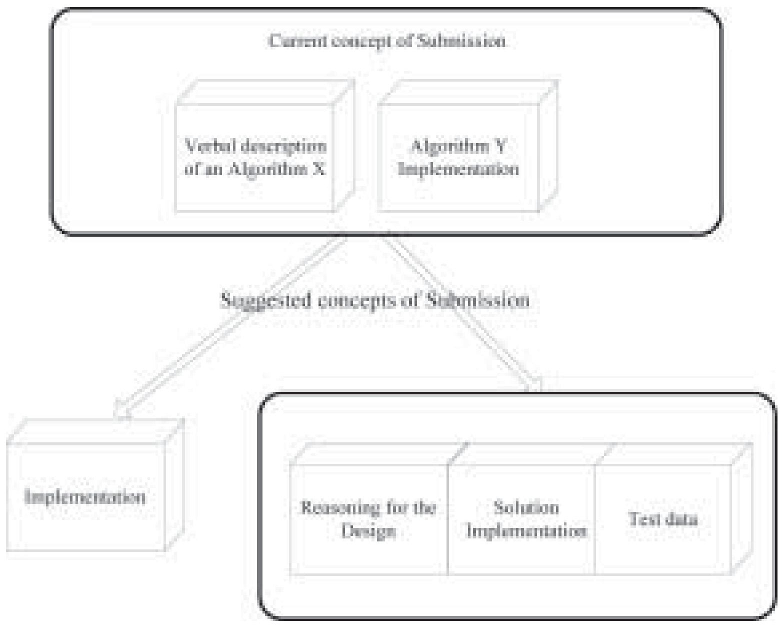
Current concept of Submission
Verbal description of an Algorithm X
Algorithm Y Implementation
Suggested concepts of Submission
Implementation
Reasoning for the Design
Solution Implementation
Test data

Fig. 2. Suggestions from the experts to modify the current submission concept.

simpler evaluation process. Another reason was the possibility for the contestants to come up with a better solution, once they implemented their solution and realized that it was not good enough.

One more suggestion found in the responses of several experts was to include the test data, preferably with motivation. Each test should consist of an input file and the corresponding output file. One of the experts commented that *when equal programs have been developed by two contestants (or companies), I would have more trust in the program that was systematically tested over one that was not.* One suggestion was to include test data in the form of a challenge phase like in the TopCoder contest (TopCoder; Skiena, 2003). I.e., the participants would have to provide test data intended to break other submissions (prepared by the jury or those of other contestants). Different approaches of the experts represent the existing variety of views in the community of informatics contests (Cormack, 2006; Verhoeff, 2006; Pohl, 2008).

To choose a particular submission model, we decided to focus on the goals of LitIO as the key factor. One very important goal in LitIO is an educational goal, i.e., to disseminate good programming practice. Even though we agree with the point of one of the experts that *contest as educational event… sounds strange;… you learn lot of things before or after, but not during the contest*, it must be taken into account that there are few qualified teachers in Lithuania, who have experience in applying and teaching good programming practices and who would know how to write a program in conformance with academic standards. Therefore the two major events for high-school students, the maturity exams in programming (Blonskis, 2006) and the Lithuanian Olympiads in Informatics serve as guidelines and kind of “reference” for the teachers.

The proposed submission model consists of:

- *Reasoning for the design* which replaces the verbal description of an algorithm;
- *Solution implementation* presented as source code in one of the allowed programming languages; it should be based on *reasoning for the design,* if such is provided;
- *Set of test cases* with motivation.

Long experience of the author working in LitIO showed that it took many years for the contestants of LitIO and their coaches to get used to providing material other than the source code (i.e., writing verbal descriptions of an algorithm). In order to avoid reluctance towards the new element of submission, i.e., a set of test cases, it would make sense to look for different options to implement the new concept of submission (for example to arrange it as some kind of challenge phase rather than simply ask for a set of test cases for checking submissions).

There is one more practical motivation for that. Two attributes might be checked for each separate test case (input data and the corresponding output). One of them is validity: is it a valid input file and the correct valid output to this input. Another attribute is whether the test is really assessing the feature it claims to. Checking the latter might be complicated or even require writing several different checkers. For example if it is claimed that the test assesses performance of a solution when the input graph satisfies specific conditions, then the jury might need to code a checker to verify whether the graph modelled in the input file really has those specific properties.

Another practical issue is related to the contest structure. Flexibility has always been present in LitIO to reasonably distribute the efforts of the contestants and the jury. Sometimes two tasks instead of three are given in an exam session, for some tasks it is not required to present verbal descriptions of algorithm (those tasks are chosen very carefully) and for some tasks programming style is not graded, as it has been noticed that programming style of submissions is not influenced by the concrete grading scheme (i.e., whether the points are awarded for style or not). Complementing a submission with test cases would require even more flexibility as generating test cases might require a lot of extra time depending upon the task.

3.2. *Submission Attributes*

The current grading model foresees three measurable attributes of a submission. They are the quality of a verbal description of an algorithm (sub-attributes: correctness and efficiency of the described solution and quality of the description), performance of already compiled program code (sub-attributes: functional correctness and time and memory efficiency) and programming style (sub-attributes: consistency, clear layout, use of proper names, suitable explicit substructures, absence of magic numbers, proper comments).

Concerns of some of the experts about the elements of submission other than implementation were reflected in the suggestions about the attributes. Those who restricted a submission to source code only were against any attributes except for performance of already compiled code. They especially stressed programming style. *Once the style is bad enough, the contestant will leave a bug and will bear consequences. If the implementation is fully correct this means that the style was good enough.* Here is an opinion of

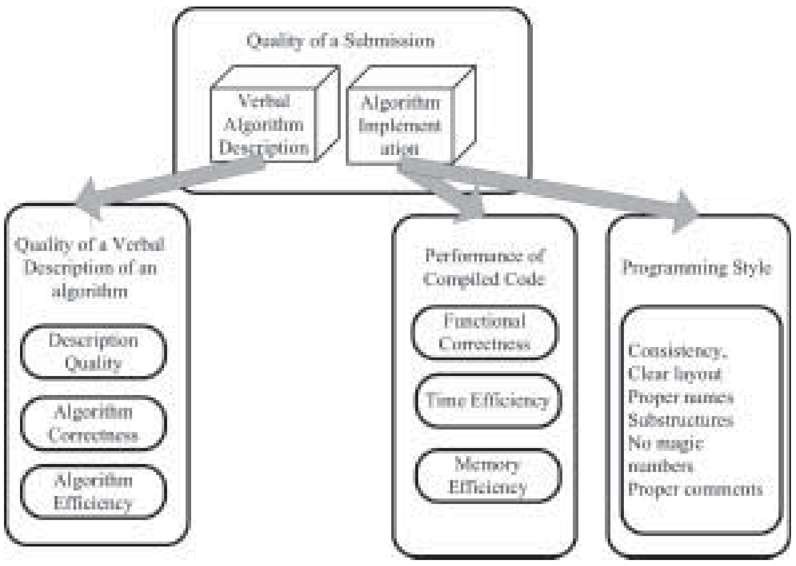

Fig. 3. Attribute level of current evaluation model; the sub-attributes of programming style are not separated because the score is not a direct combination of evaluation of each of the attributes.

another expert who refers to the research of Grigas (1995) who investigated relationship of programming style and IOI score in IOI'1994: *goto was used by the best and worst students therefore it is hard to say how particular programming construction influences achievement of the contestant... defining some formal criteria of what is a good style and what is not seems to be extremely hard; better style leads to better programs and therefore to better results*.

Different attitudes were expressed by other experts. There was proposed another interesting model consisting of five attributes[4]. The first attribute is *the quality of the solution idea.* The next three attributes refer to concrete parts of submission, i.e., *the quality of the description of the solution idea, correctness of implementation (with respect to solution idea), the quality of implementation (source code quality).* The last attribute, *conformance to the requirements of the task description,* refers to the entire submission.

This model we found interesting because it considers the submission as integral entity rather than a set of separate submission elements. From all the suggested models it seems to be the most educationally motivated. The quality of the solution idea is one concept, not divided between the reasoning for design and implementation. The implementation correctness is actually related to the implementation correctness but not to the combination of solution and implementation correctness. The model seems to separate problem solving and engineering, which is the ongoing problem of the current model (small implementation mistake resulting in the loss of many points). Despite its attraction, the most questionable in this model is its practical implementation within contest time pressure. For example the *independent grading of the quality of the solution idea can be done in*

[4]A similar model is applied in (Bundeswettbewerb Informatik).

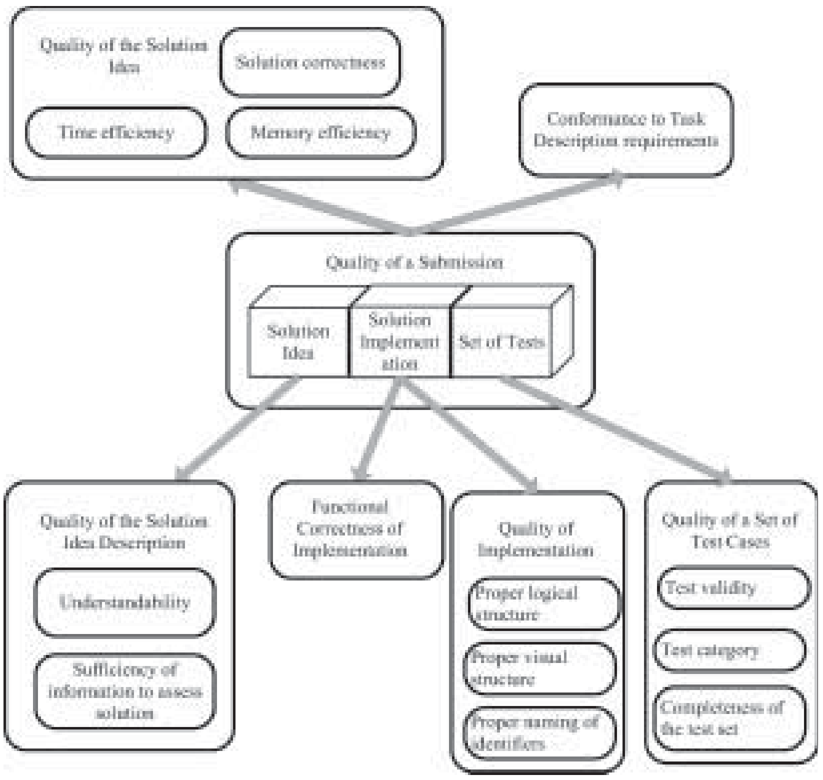
Quality of the Solution Idea
Solution correctness
Time efficiency
Memory efficiency
Conformance to Task Description requirements
Quality of a Submission
Solution Idea
Solution Implementation
Set of Tests
Quality of the Solution Idea Description
Understandability
Sufficiency of information to assess solution
Functional Correctness of Implementation
Quality of Implementation
Proper logical structure
Proper visual structure
Proper naming of identifiers
Quality of a Set of Test Cases
Test validity
Test category
Completeness of the test set

Fig. 4. Attribute level of evaluation model which separates problem solving part (the solution idea) from the engineering part complemented with attributes to evaluate the quality of test cases.

the manual way only… otherwise it will be mixed with implementation correctness. The model (the expert provided the whole model not just the attributes) seems to be a good choice for the maturity examination in programming or for smaller informatics contests than LitIO. This model might also be implemented for some tasks in the finals of LitIO where the number of submissions of one task might be rather low.

Several experts suggested the same decomposition of the quality of the verbal description of an algorithm as currently used in LitIO, i.e., to emphasize the correctness and the efficiency of the described algorithm. However one of the experts suggested a different approach by putting emphasis on the design issues. The attribute *quality of a design reasoning* can be decomposed into *"story" organization* (i.e., appropriate separation of concerns, introducing appropriate concepts and notations), *effective reuse of prior knowledge* (e.g. standard algorithms and data structures) and *the level of formality and convincingness.* This decomposition seems to be more attractive as it puts emphasis on the design issues to reveal which is the main purpose of the written material rather than serving as double award or punishment for incorrect or inefficient solutions.

With shift of emphasis there still remains the question whether correctness and efficiency of design decisions should be evaluated or not. As this is the only place in the

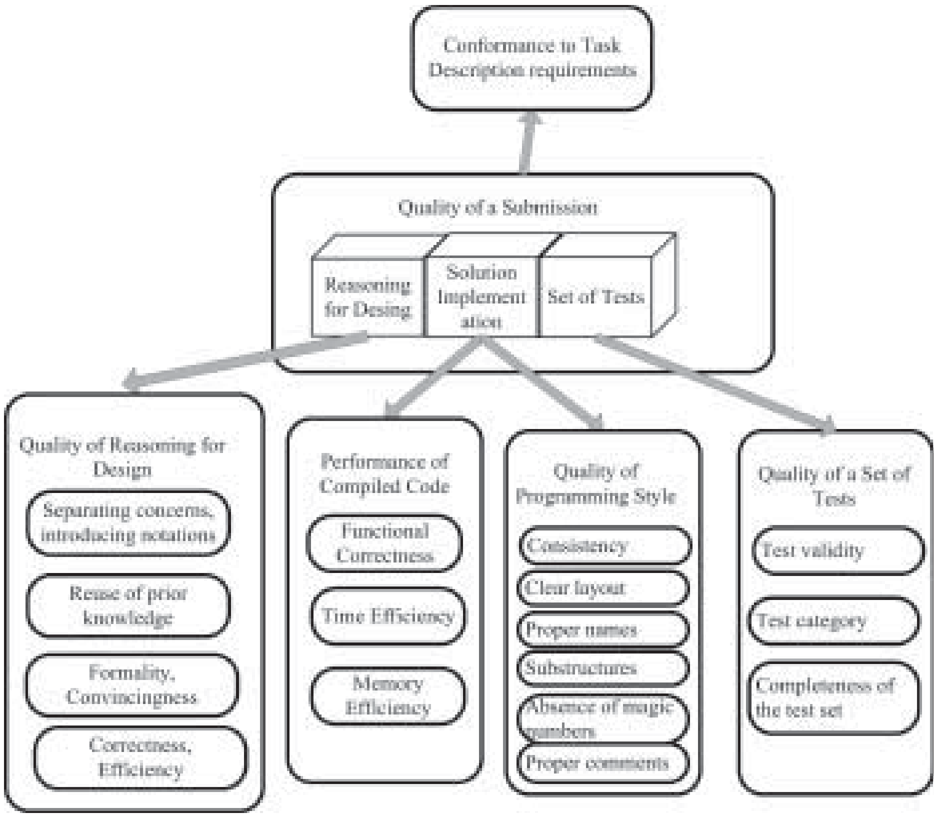

Fig. 5. The improved attribute level of evaluation model.

model where solution idea is evaluated explicitly (it is evaluated implicitly when testing the implementation), we decided to leave this as a sub-attribute.

Because tests were included into the submission, some attributes should reflect this. Three sub-attributes can be measured about a test set. They are test validity and belonging to some category (measured about each test separately) and the completeness of the whole test set.

The attribute *conformance to the requirements of the task description* was suggested by several experts. The attribute should be in the form of a checklist and also might act as the coordinator between other attributes. For example the item of this attribute might be the correspondence of the reasoning for design to the implementation.

None of the experts suggested *time spent on solving the task* as an attribute, which is common in ACM-ICPC style competitions where each minute from the contest starting time till the moment the submission is accepted is turned into one penalty point (ACM-ICPC) or if formulated in a more positive way – the participant gets a bonus for each minute from the submission time till the end of the contest (Myers, 1986).

4. The choice of Metrics for the Attributes

The proposed evaluation model suggests five attributes that can be measured to evaluate the quality of the submission. Some of those attributes require manual grading. All the experts provided suggestions regarding measuring performance of compiled code. About half of the experts provided suggestions for the metrics for measuring other attributes. We will review all the attributes one by one.

Quality of Reasoning for Design
Four sub-attributes were identified in the reasoning for design: *separating concerns and introducing notations, reuse of prior knowledge and formality and convincingness, correctness and efficiency.* All these sub-attributes require manual grading. A concrete grading scheme presenting a taxonomy of various approaches to solving the problem should be developed for each task. While it is accepted that tests are designed prior to the contest and not changed during evaluation, *the taxonomy for grading reasoning for design might have to be adapted during grading as blind spots might be discovered. This resembles IMO grading* (Verhoeff, 2002).

There were many suggestions to include clarity and understandability in the evaluation model. However clarity and understandability seemed to be more important in the previous model, where the only other sub-attributes were correctness and efficiency. In this model, clarity as separate metrics seems to be less important.

Performance of Compiled Code
Three sub-attributes were identified in the performance of the compiled code: functional correctness, time efficiency and memory efficiency. All the three sub-attributes are part

Table 1
Suggestions for metrics to measure quality of reasoning for design

Metrics	Human/ automated measuring	Scale	Comments
Level of clarity and understandability	Human	Ordinal scale	Applies to whole attribute; might influence other scores of this attribute
Level of story organization and concern separation	Human	Ordinal scale	Proper taxonomy should be developed for each task
Level of reuse of prior knowledge	Human	Ordinal scale	
Level of convincingness and formality	Human	Ordinal scale	
Level of correctness of design decisions	Human	Ordinal Scale	
Level of efficiency of design decisions	Human	Ordinal Scale	

of current evaluation model and the experts approved the current metrics.

It is accepted to check functional correctness automatically using black-box testing strategy, even though this does not guarantee that all the faults will be discovered (Williams, 2006). Each submission is executed with each test input. The test is considered to be passed successfully if the program finishes its execution within given time and memory limits and provides correct output to the input. Time and memory efficiency is measured indirectly. Tests are designed in such a way that they would benchmark solutions with different time or memory efficiency, i.e., the solutions with a certain efficiency are expected to pass a certain subset of tests. However, if the program fails while executing some test, then in general without closer analysis it is not possible to determine whether it failed due to functional incorrectness or due to low time or memory efficiency. One of the experts wrote *if the program fails correctness tests, most of the time it will fail the efficiency tests*[5]*, not to mention the fact that in some cases the line between an incorrect solution and an inefficient solution is unclear.* It is possible to measure performance of the compiled code as an attribute; however it is not always possible to measure separately each sub-attribute.

Despite it, a majority of the experts suggest to measure two sub-attributes separately, i.e., to differentiate the tests into functional correctness and efficiency tests (both efficiency sub-attributes should be merged). Some required introducing small correctness tests. Those should test the very basic properties which the submission would have to solve correctly. I.e., in order to score any points for performance, the program *must* solve correctly some very basic case of the whole problem. There also were suggestions to measure the exact running time of a program with each test (i.e., use it as a metrics) and take it into account when calculating the score[6]. In our model we will distinguish correctness and efficiency tests in order to leave room for exploring various score aggregation models where the points for efficiency tests are related to the points for correctness tests.

There were suggestions to add one more metrics for functional correctness. The experts suggested to manually identify the programs which do not try to solve the task, but just output the same answer all the time (e.g. *no solution*) or just guess the answer (we do not refer to using randomization as part of solution strategy) and to assign zero score for performance. Indeed, if there are situations where for some reasons grouping is not used then this category of submissions may score an inadequate amount of points (Skūpienė, 2009a).

Measuring exact program execution time with the purpose to identify its complexity is a sensitive issue not just due to the choice of test data. Hardware issues, compiler options, differences between programming language influence program performance.

Different ideas about measuring efficiency were presented in Ribeiro (2009). They address the difficulties and concerns related to measuring efficiency. The paper suggests asking to submit functions (procedures) rather than programs and repeating the same function call several times to increase clock precision and thus decrease input size, which

[5]We haven't found corroborating empirical evidence in the literature.

[6]Exact program execution time is influenced by hardware issues, optimization level, compilers etc., and the expert had this in mind while suggesting this metrics and afterwards proposing score aggregation.

Table 2

Suggestions for metrics to measure Performance of the compiled code

Sub-attribute	Metrics	Human/ automated measuring	Scale	Comments
Functional Correctness	Does the program try to solve the task	Human	Binary: Yes/No	It makes sense to use the metrics only if the tests are not grouped
Functional correctness	Small correctness tests focusing on specific basic categories of input data	Automatic	Binary: Pass/Fail for each test	All these tests cases should be solved correctly in order to score any points for performance
Functional correctness	Correctness tests focusing on different categories of input data	Automatic	Binary: Pass/Fail for each test	Grouping test cases was implied by most experts
Time efficiency Memory efficiency	Efficiency tests sorted into groups to benchmark submissions of different efficiency	Automatic	Binary: Pass/Fail for each test	Grouping test cases was implied by most experts
Time efficiency Memory efficiency	Efficiency tests sorted into groups to benchmark submissions of different efficiency	Automatic	Ratio: Exact execution time for each test	Execution time might be used either as binary or as non-binary metrics

nowadays has become too large and started causing problems. Curve fitting analysis is proposed to be used to estimate program complexity rather than referring to the number of passed test cases. However the ideas need further investigation before being included into the model.

Quality of Programming Style

The quality of programming style has several sub-attributes (consistency, clear layout, proper naming, suitable substructures, absence of magic numbers and proper comments). The experts proposed three different metrics for evaluating quality of programming style and suggested to use both human and automated grading.

One metric assumed evaluating the quality of programming style as a whole, taking into account all the sub-attributes but not evaluating them separately. This should be human grading with the grading results presented on an ordinal scale. Another suggestion was to measure the sub-criteria separately, presenting the results on an Ordinal scale again and afterwards using each measure to aggregate into one score for quality of the programming style. Actually, there is no need to choose one approach. Both of them can be used depending upon available resources, as the first approach requires much less time, while the results based on the second approach would be much clearer to the contestants.

The third approach (suggested by several Lithuanian and foreign experts) included a combination of human and automated grading; however, it was emphasized that the possibilities of automated grading of programming style must be researched first and

Table 3

Suggestions for metrics to measure Quality of Programming Style

Metrics	Human/ automated measuring	Scale	Comments
Level of Quality of programming style	Human	Ordinal scale	This metrics should be used iff other metrics from this table are not used
Level of Consistency	Human	Ordinal scale	Possibilities for replacing/combining
Layout clarity	Human	Ordinal scale	with automated measuring
Level of proper naming	Human	Ordinal scale	might be investigated
Suitability of substructures	Human	Ordinal scale	
Absence of magic numbers	Human	Binary scale: Yes/No	
Suitability of comments	Human	Ordinal scale	

the proper tools developed. Such an approach is applied in the evaluation procedure of the Lithuanian maturity exam in programming, and special software was developed for that. The exam submissions are much simpler in complexity and and shorter in length than contest submissions and the only available compiler is FreePascal (Skūpas, 2008). However, none of the experts provided concrete metrics for performing this type of semi-automated grading.

Quality of a Set of Tests

The improved evaluation model suggests three sub-attributes to measure the quality of the submitted test set. The first sub-attribute is the validity of each test (a pair consisting of an input and the corresponding output). Test validity means that the input is a valid input according to task specifications and the output is a correct output to the given input.

The next sub-attribute is test category. Each test should be submitted with motivation, explaining what category of input it targets. The most sophisticated challenge seems to be verifying that the provided test really targets the category that it claims to be. Checking this automatically might require too many resources for some tasks.

The final sub-attribute is completeness. This attribute refers to the whole test set and it checks to which extent the submitted test set covers the domain.

Conformance to Task Description Requirements

Measuring conformance to task description requirements should be arranged in form of a checklist. The checklist might depend on the concrete task. In some cases it might be of secondary importance or not needed at all, but if the absence of some item on the check list makes it impossible or difficult to evaluate another item, then a low grade for this attribute will result in a low grade for the other attribute as well. One obvious item to be included in the check list is the correspondence of the reasoning for the design to the implementation. Another item to be included is the presence of motivation for test-cases.

The model presented in this paper does not include score aggregation, which is another important part of evaluation in contests. Score aggregation and validating the new

Table 4

Suggestions for metrics to measure Quality of a Set of Tests

Metrics	Human/ automated measuring	Scale	Comments
Test validity	Automated	Binary: Yes/No for each input/output pair	
Belonging to certain test category	Manual Automated	Binary: Yes/No for each test and accompanying motivation	Motivation is evaluated manually
Completeness of whole test set	Automated	Percentage of coverage statistics	Special plug-in for contest system might be needed

Table 5

Suggestions for metrics to measure Conformance to task description requirements

Metrics	Human/ automated measuring	Scale	Comments
A checklist based on the specification of the concrete task	Human Automated	Binary: Yes/No for each item of the checklist	The need for this attribute at all and its use in score decision procedure depends upon the concrete task

model by applying it in practice is left to future work. While working on this model, we tried to be reflective, and discuss and incorporate as many ideas and suggestions as possible that we found in the responses of the experts, even if they were not included into the proposed model. The model has room for flexibility and for tailoring through score aggregation methods.

5. Conclusions

The Lithuanian Informatics Olympiads were started twenty years ago and the evaluation model has not changed much since olympiads in the computer labs replaced pen and paper contests. The goal of this research was to reconsider the evaluation model. As quality of a solution is a constructed notion, we invited a group of Lithuanian and International experts to discuss this concept and suggest an improved evaluation model for the Lithuanian Informatics Olympiads. We provided the experts with a description of the Olympiad structure, scope and resources as well as a three level (submission, attributes, metrics) hierarchical evaluation model pattern to be discussed and filled.

The experts were representing two trends that can be distinguished in informatics contests. Some of them associate quality of submission with performance of implementation only, while the others think that a high-quality submission should have both reasoning for design and test data to check the implementation. The educational aspect of the Lithuanian Informatics Olympiad was the key factor in deciding in favour of the latter concept of quality of a solution. Two significant changes were introduced into the current model. The verbal algorithm description which was not related to implementation and emphasized correctness and efficiency was replaced by reasoning for the design, which should be related to implementation and shifted focus to design issues. Another new aspect was introducing a test set as part of a solution. As a result of this work, an improved evaluation model is proposed for the Lithuanian Informatics Olympiads. In the suggested model the concept of submission is altered. The attributes that need to be measured about the submission are identified, as well as metrics to measure each of the attributes. The next step would be to complete the model by creating a score aggregation algorithm and to propose adopting it in LitIO.

Finally, I would like to thank all the experts very much for their cooperation and useful ideas.

References

ACM-ICPC International Collegiate Programming Contest.
Accessed at: `http://cm.baylor.edu/welcome.icpc`

Basili, V., Caldeira, G., Rombach, H.D. (1994). The goal question metric approach. In: Marciniak, J. (Ed.), *Encyclopaedia of Software Engineering*. Wiley.

Berander, P., *et al.* (2005). *Software quality attributes and trade-offs*. In: Lundberg, L., Mattson, M., Wohlin, C. (Eds.), *BESQ – Bleckinge Engineering Software Qualities*. Blekinge Institute of Technology. Accessed at: `http://www.bth.se/tek/besq.nsf/pages/017bd879b7c9165dc12570680047aae2!OpenDocument`.

Blonskis, J., Dagienė, V. (2006). *Evolution of informatics maturity exams and challenge for learning programming*. In: Mittermeir, R.T. (Ed.), *Informatics Education – The Bridge between Using and Understanding Computers, Lect. Notes in Computer Science*, Vol. 4226, 220–229.

Bundeswettbewerb Informatik. Accessed at: `http://www.bwinf.de/`

(2009). *CEOI'2009 – The 16'th Central European Olympiad in Informatics.* Accessed at:
`http://www.ceoi2009.ro`

Cormack, G. (2006). *Random Factors in IOI 2005 test sase scoring*. *Informatics in Education*, 5(1), 5–14.

Crosby, P.B. (1979). *Quality is Free: The Art of Making Quality Certain*. New York, McGraw-Hill Education.

Dagienė, V., Skūpienė, J. (2007). Contests in programming: quarter century of Lithuanian experience. *Olympiads in Informatics: Country Experience and Developments*, 1, 37–49.

Dagienė, V., Skūpienė, J. (2004). Learning by competitions: olympiads in informatics as a tool for training high grade skills in programming. In: Boyle, T., Oriogun, P., Pakstas, A. (Eds.), *2nd International Conference on Information Technology: Research and Education*. London, London Metropolitan University, 79–83.

Deming, W.E. (1990). *Out ot the Crisis: Quality, Productivity and Competitive Position*. Cambridge Univ. Press.

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

Grigas, G. (1995). Investigation of the relationship between program correctness and programming style. *Informatica*, 6(3), 265–276.

Hoyer, R.W., Hoyer, B.B.Y. (2001). What is quality? *Quality Progress*, 7, 52–62.

IOI – International Olympiad in Informatics. Accessed at: `http://ioinformatics.org/`

Lundberg, L., Mattson, M., Wohlin, C. (Eds.) (2005). *Software Quality Attributes and Trade-offs*. Blekinge Institute of Technology. Accessed at:
`http://www.bth.se/tek/besq.nsf/pages/017bd879b7c9165dc12570680047aae2!OpenDocument`

Mareš, M. (2007). Perspectives on grading systems. *Olympiads in Informatics: Country Experiences and Developments*, 1, 124–130.

Myers, D., Null, L. (1986). Design and implementation of a programming contest for high school students. In: *Proceedings of the Seventeenth SIGCSE Technical Symposium on Computer Science Education*, 307–312.

Pohl, W. (2008). Manual grading in an informatics contest. *Olympiads in Informatics*, 2, 122–130.

Poranen, T., Dagienė, V., Eldhuset, Å., Hyyrö, H., Kubica, M., Laaksonen, A., Opmanis, M., Pohl, W., Skūpienė, J., Söderhjelm, P., Truu, A. (2009). Baltic olympiads in informatics: Challenges for Training Together. *Olympiads in Informatics*, 3, 112–131.

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

Skienna, S., Revilla, M. (2003). *Programming Challenges – the Programming Contest Training Manual*. Springer-Verlag, New York.

Skūpas, B. (2008). Is automatic evaluation useful for the maturity programming exam. In: *Koli Calling 2008, Proceedings of 8th International Conference on Computing Education Research*, 117–1178.

Skūpienė, J. (2009a). Assessment of solutions of Lithuanian informatics olympiads from the point of view of software quality model. Accepted for publication in *Information Sciences*.

Skūpienė, J. (2009b). Credibility of automated assessment in Lithuanian informatics olympiads: one task analysis. Accepted for publication in *Pedagogika*.

Skūpienė, J. (2006). Programming style – part of grading scheme in informatics olympiads: Lithuanian experience. In: Information Technologies at School. Proceedings of the Second International Conference ”Informatics in Secondary Schools: Evolution and Perspectives”. Vilnius, 545–552.

Skūpienė, J. (2004). Automated testing in Lithuanian informatics olympiads. In: *Informacinės technologijos 2004, Konferencijos pranešimų medžiaga*, Kaunas, Technologija, 37–41.

TopCoder Algorithm Competition. Accessed at: `http://www.topcoder.com/tc`

Verhoeff, T. (2006). The IOI is (not) a science olympiad. *Informatics in Education*, 5(1), 147–158.

Verhoeff, T. (2002). *The 43rd International Mathematical Olympiad: A Reflective Report on IMO 2002*. Computing Science Report 02-11, Faculty of Mathematics and Computing Science, Eindhoven University of Technology. Accessed at:
`http://www.win.tue.nl/∼wstomv/publications/imo2002report.pdf`.

Williams, L. (2006) *Testing Overview and Black-Box Testing Techniques*. North Carolina State University Department of Computer Science. Available at:
`http://agile.csc.ncsu.edu/SEMaterials/BlackBox.pdf`

J. Skūpienė is a younger research fellow in the Informatics Methodology Department in the Institute of Mathematics and Informatics. She is a member of the Scientific Committee of National Olympiads in Informatics since 1994 and a Lithuanian team leader in International Olympiads in Informatics since 1996. For a few years she was director of studies of Young Programmers' School by Correspondence. Since 2004 she has been a coordinator of Informatics section in the National Academy of Students. She is author/co-author of three books on algorithmic problems for informatics contests. Her research interests include informatics and algorithmic contests, teaching informatics and computer science in secondary education.

Sprendimų vertinimo Lietuvos mokinių informatikos olimpiadose modelio tobulinimas

Jūratė SKŪPIENĖ

Lietuvos mokinių informatikos olimpiados yra mokiniams skirtos algoritmavimo ir programavimo varžybos. Olimpiadų dalyviai vertinimui pateikia algoritmus, sprendžiančius duotą uždavinį ir realizuotus veikiančiomis programomis viena iš nurodytų programavimo kalbų.

Vertinant darbus, vienus aspektus (programavimo stilių, sprendimo idėjos aprašymą) vertina vertintojų komanda. Tuo tarpu pateiktų programų teisingumas ir efektyvumas vertinamas jas automatiškai testuojant su iš anksto parinktais pradinių duomenų rinkiniais. Ši vertinimo schema paremta metodiniais samprotavimais ir ilgamete tradicija, tačiau nebuvo analizuota ir pagrįsta moksliniais metodais.

Siekdami pagerinti ir moksliškai pagrįsti olimpiadose naudojamą vertinimo schemą, sudarėme vertinimo modelio šabloną bei anketą ir pakvietėme dalyvauti grupę Lietuvos ir užsienio ekspertų, turinčių ilgametę patirtį organizuojant bei vertinant įvairaus lygmens algoritmavimo olimpiadose bei konkursuose.

Ekspertų siūlyti vertinimo modeliai buvo dviejų tipų ir remdamiesi Lietuvos informatikos olimpiadų edukaciniais tikslais, pasirinkome vieną šių krypčių. Konstruodami vertinimo modelį stengėmės kiek galima daugiau atspindėti ekspertų pateiktus siūlymus bei idėjas. Straipsnyje plačiai aptariama dauguma ekspertų siūlomų aspektų net jei jų ir neįtraukėme į vertinimo modelį. Svarbiausias šio straipsnio rezultatas – siūlomas patobulintas sprendimų vertinimo Lietuvos informatikos olimpiadose modelis.