Model of the Task Formal Description for the Programming Skills Evaluation in the System of Distance Learning Programming

A.YU. Uzharinskiy, A.V. Koskin, N.F. Nasyrov

Abstract: ***In this article the authors consider the construction of a task model in creating a system for assessing programming skills of students studying IT technologies. Here is proposed the architecture of the considered system. The authors also described the structure of the task model.***

Keywords: ***modeling, task model, information system, architecture, programming skills assessment.***

INTRODUCTION.

The recent growth and development of the IT technology market and the gradual transition to a digital economy leads to an increase in the demand for highly qualified specialists in the field of information technology development. Along with the high proliferation of information technologies the requirements for their quality and reliability are increasing. So the requirements to the qualification level of bachelors and masters studying IT technologies are increasing accordingly. However nowadays there are no mechanisms for an objective assessment of the skills and qualifications of specialists that affects largely the final result of training.

According to the requirements of the current federal state educational standards, the goal of professional education at universities is to form a set of general and professional competences for graduates. However the standards do not reveal mechanisms for assessing the level of formation of these competencies. Any competence includes three aspects: knowledge, skill and possession. Existing formal mechanisms of assessment of students often allow to assess only the level of knowledge. At the same time there are no objective formalized methods and procedures for the assessment of students' practical skills. But the level of specialists' practical skills is first of all important for IT companies. In this connection, the task of developing an automated system for evaluating programming skills of IT students is becoming relevant. [2, 6, 7].

The construction of such system requires solving a number of problems which are described in the article “Development of an automated environment for evaluating programming skills of students in IT specialties” [1]. One of the key problems is to develop the model of task solving which is aimed to formalize the process of evaluating the programming skills that are demonstrated by the students when they perform it. The difficulty of this problem lies in the fact that there are no formal criteria and measurable metrics that make it possible to assess unambiguously the quality of the obtained solution and the level of skills involved in this. The process of task solving is largely creative and difficult to formalize. The creation of the described model will allow to assess objectively the current level of programming skills and modify the further training plan in order to improve the final practical skills of university graduates [4, 8].

Next we consider the general architecture of the automated system for assessing the programming skills of students in IT specialties and determine the place of the task solving model in this system.

1. Architecture of the system for evaluating the programming skills of students in IT specialties.

The system for assessing the programming skills of students in IT specialties consists of three components (Figure 1): the environment for the task solving, the system for assessment of programming skills and the application of a teacher.

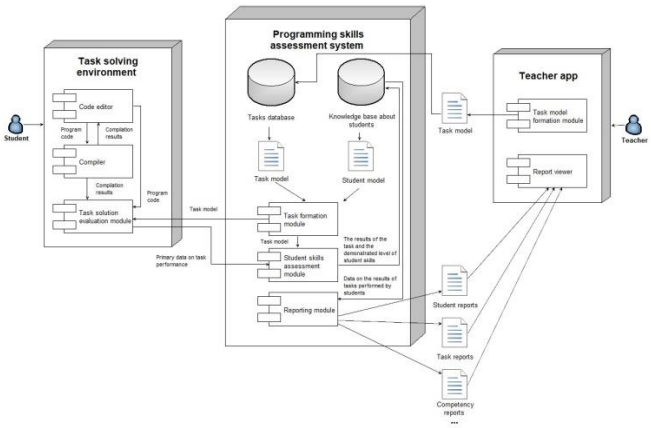

Figure 1 - Architecture of the system for the assessment of the programming skills of students in IT specialties

The central component in the given architecture is the system for evaluating programming skills. At the heart of this system are two databases. The task database stores information about the tasks that students have to solve. All

Revised Manuscript Received on April 12, 2019.

A.Yu. UZHARINSKIY, Federal State Budgetary Educational Institution of Higher Education «Oryol State University named after I.S. Turgenev», Candidate of Technical Sciences, Associate Professor at the Department of Software Engineering, Mobile: 89102645676 (E-mail: udjal89@mail.ru)

A.V. KOSKIN, Federal State Budgetary Educational Institution of Higher Education «Oryol State University named after I.S. Turgenev», Doctor of Technical Sciences, Professor at the Department of the Information Systems, Tel.: 8 (4862) 419815 (E-mail: koskin@ostu.ru)

N.F. NASYROV, St. Petersburg state budget professional educational organization "Radio engineering College" (E-mail: pasdel@mail.ru)

tasks are classified and ordered according to different criteria: programming language, applied technologies, complexity, discipline and generated competencies. A task model is formed on the basis of the information from this database. The model contains the problem statement, the restrictions imposed on the task solving, the data necessary to verify the correctness of the task solving and the accompanying information necessary to assess the skills having demonstrated in solving the task.

The knowledge base about students contains information about students. This database accumulates information on tasks which a particular student was performing, the results of these tasks solving, the number and types of made errors. Based on the information from this base a student model is formed which characterizes his/her skills, the dynamics of the formation of competencies and the overall rating.

The task formation module is intended for the automatic selection of tasks for students in order to develop certain skills. The student model and task model come at the input of this module. In accordance with the specified criteria the module compares these models in order to select a task aimed at the development of the least developed competences of a student. As a result, one task is selected from the task database which is sent to the task solving environment for a specific student.

After the task is completed by the students, the primary data on the results of the task solving enter the module of assessing the students' skills. The primary data include information on the time spent on the decision, the number of errors and their types made during the solution process, the quality of the source code and others. On the basis of these data information about the learner's skills is corrected and its model is changed. The results are stored in the knowledge base.

The reporting module is necessary for analyzing information about students and visualizing learning dynamics. It allows you to provide the results of solving tasks in various sections: by student, by assignment, by competence, etc. This information is needed to assess learning outcomes and adjust the learning process.

The training application is an independent component of the system and is an environment for solving tasks. This environment includes three main modules: a code editor module, a compilation module and the module of the task solving evaluation. The code editor module is an interactive program code editor with syntax highlighting of the selected programming language and verification of semantics. The student writes the program code in this editor and starts the solution process. The code editor checks the program for syntax and semantic errors. If errors are found, the student is notified of their presence. The information on found errors is transmitted to the task solving evaluation module. The code editor can work in two modes. In the learning mode the editor itself prompts students how to solve the task and explains the proposed solutions. In the control mode only syntax highlighting occurs but error checking is performed when the program starts. If there are no errors, the code is passed to the compilation and solution module. The compilation module compiles the program. If errors are found during the compilation process, information about them is transmitted to the code editor and the module for evaluating the results of solving the task. After successful compilation of the program, the executable file is transferred to the task solving evaluation module. Here, on the basis of the task solving model the executable file and the resulting source code are tested. First, a set of tests is performed and the results are compared to verify the correctness of the solution. Then the source code of the program is analyzed for the presence of the required algorithm fragments and quality. If the task solving evaluation module decides that the task is solved correctly and then all the data collected during the task solving process is sent to the programming skills assessment system in the students skills assessment module. In this module on the basis of the obtained data the indicators characterizing the level of students' skills are adjusted and the results are stored in the knowledge base of the students. In this case, students receive information about the assessment of his skills in the process of solving the task on the screen.

It is used a report generation module to view statistics and learning outcomes. This module takes information from the students' knowledge base and allows you to build reports in various sections. This module also allows you to view information on the dynamics of students' competencies development in the learning process and adjust the learning model. The system provides for the formation of reports on an individual student, on a task, on the selected competence, etc.

The training application is an independent client, launched by the trainer to view the learning outcomes and replenish the database of tasks. The main components of this application are the task model formation module and the report viewer module. The former is designed to create new tasks and add them to the task database. The latter provides visualization and processing of statistical and reporting data on learning outcomes received from the server.

The system for evaluating the programming skills of IT students shows that the key element of this system is the task model. Next we consider the structure of this model in details.

2. Task Model Structure

The model of the task to be solved includes a number of components: the problem situation is Q, the constraint set is U, the model solution is O, the model for assessing the correctness and quality of the program code is M, the developed competencies are K.

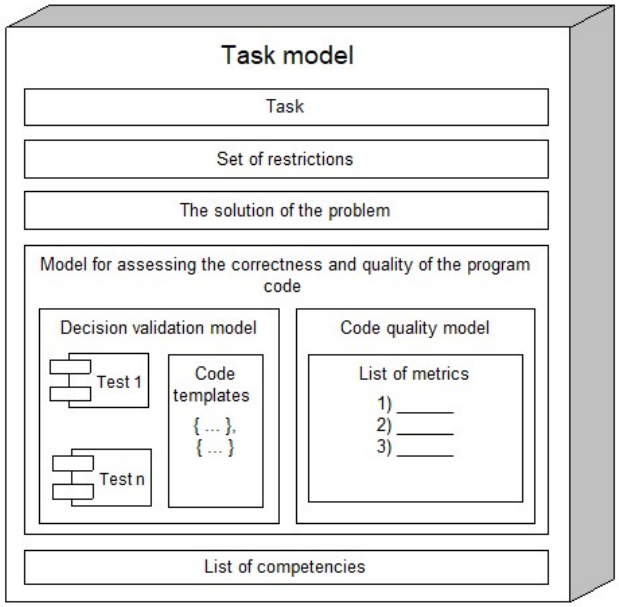

Figure 2 - The structure of the task model

Imagine the model of the task being solved in the form of some set T = <Q, U, O, M, K>.

The Q task condition includes a detailed description of what needs to be done and the expected results. This condition is provided to the student.

A programming task can be solved in an infinite number of different ways. However in the learning process it is necessary to use the required solution as the task is aimed just at its studying. In this regard each task contains a number of restrictions imposed on the solution which limit the set of solution options and simplify the task of checking the presented solution. These restrictions are described by the set $U=\{u_1, u_2, \ldots, u_n\}$ and are located in a block in the second section of the task model description. Restrictions are described in text form and are taken into account when assessing the correctness of the task solving.

The third component of the task model is the O task solution. The model task solution is used in the learning mode for prompting students and comparing the obtained solution with the sample.

The fourth component of the task solving model is the model for assessing the correctness and quality of the program code M. It is the most important component of the task model. It allows to evaluate automatically the obtained solution and the demonstrated skills. The task of assessing the quality of a program code is a difficult task since there are no quantitative assessment metrics. At the same time there are no clearly defined criteria for the quality of the program code and the reduced solution. This greatly complicates the process of automatic evaluation of the obtained solution. At present in the process of learning the quality of task solution is assessed by an expert (who is a trainer) on the basis of his/her experience and knowledge of the task to be solved. The second problem related to the evaluation of the quality of the program code is that in the learning process it is necessary to evaluate not only the correctness of the solution itself but also how the student obtained this solution, i.e. his/her actions in the process of obtaining a solution. To do this you need to use additional information on the obtained solution collected by the system.

The model for assessing the correctness and quality of the program code includes two components. The first component is an assessment of the correctness of the obtained solution - M_1. It is traditionally used the testing mechanism to evaluate the correctness of the solution in programming. The model includes a number of structural and modular tests $T = \{t_1, t_2, \ldots, t_n\}$ allowing to evaluate the correctness of the solution with various input data. Each test t_i in the model is described by four components — the internal state of the system and the conditions for running s_i, the input data — d_i, the expected results — r_i and restrictions on obtaining the result – u_i. The model must include tests that allow to check the nodal points of the program as well as the boundary values of the areas of allowable values. The number of tests depends on the task and is determined by the trainer independently when he/she forms the model of the task.

However only testing is not enough to assess the correctness of the task solving [9]. It is necessary to evaluate how the obtained solution meets the requirements which are set for the program by the trainer. For example, if you need to implement a specific sorting algorithm, then you need to check that it is this algorithm that is implemented and not an alternative one. Or if you want to apply recursion to solve the task, you need to check that this technique is really used in solving the task. For this purpose we propose to include code patterns in the model - $B = \{b_1, b_2, \ldots, b_m\}$ and compare the presented code with these templates [5]. Patterns describe key fragments of the algorithm that must be present in the solution. If the proposed solution does not match the submitted pattern, then we can conclude that it is implemented the wrong algorithm. At the same time patterns should not limit the student's creative approach to the task solving and not impose strict restrictions.

The second component of the model for assessing the correctness and quality of program code is the model for assessing the quality of program code - M_2. In the process of assessing the quality of the obtained solution the program code is analyzed and it is necessary to determine whether the required technology is applied in this code, whether the specified algorithm is implemented, how efficient the obtained code is according to different criteria and whether there are possibilities for enhancement. In this case we propose to use the following evaluation methods. At first program code metrics are calculated - numerical indicators that evaluate the quality of the presented solution. In this model we suggest evaluating four main types of metrics [3, 9, 10]:

RESULTS & DISCUSSIONS

– software code complexity $V(G)$, i.e. how complex is the presented solution. To estimate this indicator we use

the McCabe metric of complexity:

$$V(G) = e - n + 2p$$

where e is the number of arcs, n is the number of vertices, p is the number of connected components of the control flow graph.

– software code connectivity - the degree of dependence of the program code components from each other. The main metrics in this group are the metrics of internal C_e and external C_a dependencies, the depth of inheritance is *DIT* and the average number of internal links per type is *H*:

$$C_e = \sum C_{in},$$

$$C_a = \sum C_{out},$$

$$DIT = \sum base(C),$$

$$H = \frac{R+1}{N},$$

where C_{in} is the number of internal links of the module, C_{out} is the number of external links of the module.

– program code structuredness, i.e. the degree of breaking the program code into functional blocks and the level of encapsulation. To estimate this indicator we use Hansen's metrics - *S* and Pivovarsky's ones - *N(G)*:

$$S = \{m - n + 2\},$$

$$N(G) = n^*(G) + \sum P_i,$$

where m is the number of arcs, n is the number of vertices, $n^*(G)$ is the modified cyclomatic complexity, P_i is the nesting depth of predicate vertex i.

– volumetric characteristics of the program code that shows the size of the program entities. The basic metrics in this group are the measures of width – *W* and depth – *D* code.

The proposed set of metrics will allow to assess the basic indicators of the obtained program quality and reduce them to some integral indicator. The advantage of using metrics is the ability to calculate them automatically for any program code. Some acceptable range of values is given for each metric in the program code quality assessment model.

The fifth component of the task solving model is the list of competencies $K = \{k_1, k_2, \ldots, k_l\}$. The list of developing competencies is needed to determine what competencies will be affected by the results of this task solving. Each competence contains some levels of indicators that correspond to a certain level of mastering the competence by students. The results of solving each task affect the dynamics of mastering the competence. Thus the formal task of the task solving model within the framework of the developing system can be described by the following sets:

$$T = <Q, U, O, M, K>,$$

$$U=\{u_1, u_2, \ldots, u_n\},$$

$$M = <M_1, M_2>,$$

$$M_1 = <T, B>,$$

$$T = \{t_1, t_2, \ldots, t_m\},$$

$$t_i = < s_i, r_i, d_i, u_i >,$$

$$B = \{b_1, b_2, \ldots, b_k\},$$

$$M_2 = \{V(G), C_e, C_a, DIT, H, S, N(G), W, D\},$$

$$K = \{k_1, k_2, \ldots, k_l\}$$

A number of factors were obtained on the basis of the proposed task model. They will be evaluated for each student. These basic factors are:

F1 – task solving time;

F2 – task complexity;

F3 – number of syntax errors;

F4 – number of errors at runtime;

F5 – total compilation attempts;

F6 – number of failed tests;

F7 – the ratio of the average time for program execution to the required one;

F8 – the ratio of the code lines number to the reference one;

F9 – the complexity of the software code relative to the reference value;

F10 – code structuredness;

F11 – code width;

F12 – code depth.

The presented factors can be considered as input parameters in artificial intelligence and machine learning systems to create an individual learning plan for students.

We believe that the proposed task model will allow to evaluate not only the correctness but also the quality of solutions as well as the level of skills demonstrated in the process of task solving. In contrast to the classical methods for assessing solutions which take into account only the correctness of the program this model allows to evaluate how the solution was obtained and what specific types of errors were made when it was received. It also allows to create a comprehensive view of the learner's skills reflected in the learner's competency model.

CONCLUSION.

The proposed architecture of the system for evaluating the programming skills of students in IT specialties will allow to collect objective information on the practical skills of students in the process of task solving and to build a learning model based on this information. The application of the task model described in the article will allow to evaluate not only the final result in the form of a finished program but also the quality of the program code and the programming skills shown in the process of task solving. The obtained set of indicators characterizing the students' skills in task solving can be used in creating and training a neural network that will replace the teacher in the process of distance learning and select tasks for students taking into account their individual characteristics.

REFERENCES

1. Uzharinsky, A.Yu. The development of an automated environment for assessing the programming skills of students of IT specialties: abstracts of the report / Materials of the XVI open All-Russian conference "Teaching Information Technologies in the Russian Federation" URL: http://www.it-education.ru/conf2018/thesis/2706/ (date of the application 23.08.2018)
2. Vlasov, V.V. Simulation of the learning process in educational systems using the macrosystem approach [Text] / V.V. Vlasov, A.V. Koskin, Information Systems and Technologies. – 2012. – №6 (74). – pp. 62-68.

3. Zvezdin, S.V. Problems of measuring the quality of software code // Bulletin of SUSU. Series: Computer technology, management, electronics. 2010. №2 (178). URL: https://cyberleninka.ru/article/n/problemy-izmereniya-kachestva-programmnogo-koda (date of the application 23.08.2018).

4. P. Blikstein Using learning analytics to assess students' behavior in open-ended programming tasks, proceedings of the 1st international conference on learning analytics and knowledge. ACM (2011) 110-116.
5. Mansoor, Usman, et al. "Multi-objective code-smells detection using good and bad design examples." Software Quality Journal 25.2 (2017): 529-552.
6. J. Sitthiworachart, M. Joy, Deepening computer programming skills by using web-based peer assessment. In Proceedings of the 4th Annual Conference of the LTSN Centre for Information and Computer Sciences (2003, August) 152-157.
7. T. Staubitz, H. Klement, J. Renz, R. Teusner, C. Meinel, (2015, December). Towards practical programming exercises and automated assessment in Massive Open Online Courses. In Teaching, Assessment, and Learning for Engineering (TALE), IEEE International Conference (2015) 23-30
8. T. Daradoumis, J. M. M. Puig, M. Arguedas, L. C. Liñan Analyzing students' perceptions to improve the design of an automated assessment tool in online distributed programming. Computers & Education, 128 (2019) 159-170.
9. S. Yaremchuck, V. Kharchenko, A. Gorbenko Search of Similar Programs Using Code Metrics and Big Data-Based Assessment of Software Reliability. In Applications of Big Data Analytics (2018). Springer, Cham. 185-211.
10. T. Hariprasad, G. Vidhyagaran, K. Seenu, C. Thirumalai Software complexity analysis using halstead metrics. In Trends in Electronics and Informatics (ICEI), 2017 International Conference (2017, May) 1109-1113