Information systems. Computer sciences. Issues of information security
Информационные системы. Информатика. Проблемы информационной безопасности

UDC 004.02+004.4+004.891+004.91
https://doi.org/10.32362/2500-316X-2022-10-3-7-23

RESEARCH ARTICLE

Pedagogical design of a digital teaching assistant in massive professional training for the digital economy

Elena G. Andrianova @**,**
Liliya A. Demidova,
Petr N. Sovetov

MIREA – Russian Technological University, Moscow, 119454 Russia
@ *Corresponding author, e-mail: andrianova@mirea.ru*

Abstract

Objectives. The active digitalization of the Russian economy has resulted in a shortage of IT personnel; this is particularly true of software developers. Thus, the Russian university education is faced with the task of undertaking the large-scale professional training of such specialists. The purpose of the present work was to support the large-scale ("massive") professional training of programmers through the creation and implementation of Digital Teaching Assistant (DTA) computer system, allowing teachers working under stressful conditions to concentrate on functions that require a creative approach, namely, drawing up and discussing nontrivial programming tasks.

Methods. Pedagogical methods for the personification of learning processes were employed. The general approach was based on satisfying the constraints for creating programming task generators. Tasks were generated using methods for generating random programs and data based on probabilistic context-sensitive grammars, along with translation methods using an abstract syntax tree. The declarative representation of the task generator was performed using functional programming methods, allowing the creation of a domain-specific language using combinators. The solutions were checked using automated testing methods.

Results. The developed structure of the proposed DTA system was presented. Considering the automatic generation of programming tasks, classes of practical tasks that reflected the modern specifics of software development were identified along with examples of their generation. A diagram of the programming task generator was provided along with a description of the procedure for automatically checking the solutions of the tasks using a set of program tests generated by the task generator. The presented procedure for comprehensive assessment of a student's solution included verification of the correctness of the result and plagiarism checks in the case of tasks created manually by the teacher; this also involved validation for compliance with coding style standards, along with metrics for assessing program complexity, etc. The means for recording of statistics of academic achievement of students was characterized along with the interface of interaction between students and teachers.

Conclusions. The experience of introducing a DTA into the learning process of teaching programming in Python confirmed the possibility of personifying the learning process in the form of individual learning paths.

Keywords: massive programming learning, Digital Teaching Assistant, task generation, individual learning path, student motivation

• **Submitted:** 29.12.2021 • **Revised:** 03.03.2022 • **Accepted:** 11.05.2022

For citation: Andrianova E.G., Demidova L.A., Sovetov P.N. Pedagogical design of a digital teaching assistant in massive professional training for the digital economy. *Russ. Technol. J.* 2022;10(3):7–23. https://doi.org/10.32362/2500-316X-2022-10-3-7-23

Financial disclosure: The authors have no a financial or property interest in any material or method mentioned.

The authors declare no conflicts of interest.

НАУЧНАЯ СТАТЬЯ

«Цифровой ассистент преподавателя» в массовом профессиональном обучении для цифровой экономики

Е.Г. Андрианова [@], **Л.А. Демидова**, **П.Н. Советов**

МИРЭА – Российский технологический университет, Москва, 119454 Россия
[@] *Автор для переписки, e-mail: andrianova@mirea.ru*

Резюме

Цели. Активная цифровизация российской экономики вызывает дефицит ИТ-кадров и, в первую очередь, дефицит разработчиков программного обеспечения. Для российского университетского образования актуальной является задача массовой профессиональной подготовки таких специалистов. Цель работы – повышение качества массовой профессиональной подготовки программистов путем создания, внедрения и развития функциональности компьютерной системы «Цифровой ассистент преподавателя» (ЦАП). Эта система позволяет преподавателю в условиях массового обучения сконцентрироваться на функциях, требующих творческого подхода – составлении и обсуждении нетривиальных задач по программированию.

Методы. Использованы педагогические методы персонификации учебного процесса. Общий подход основан на удовлетворении ограничениям для создания генераторов задач по программированию. При генерации задач применены методы порождения случайных программ и данных на основе вероятностных контекстно-зависимых грамматик, а также методы трансляции с использованием дерева абстрактного синтаксиса. Для декларативного представления генератора задач применены методы функционального программирования, позволяющие создать предметно-ориентированный язык с помощью комбинаторов. Для проверки решений использованы методы автоматического тестирования.

Результаты. Разработана структура системы ЦАП. Рассмотрена автоматическая генерация задач по программированию, выделены классы практических задач, отражающие современную специфику разработки программного обеспечения, приведены примеры их генерации. Приведена схема генератора задач по программированию. Описана процедура автоматической проверки решения задач, осуществляемая с помощью набора программных тестов, сформированного генератором задач. Приведена процедура комплексной оценки решения обучающегося, включающая проверку корректности результата и проверку на плагиат решений в случае задач, созданных преподавателем вручную; соответствие стандарту стиля написания программы, метрикам оценки сложности программы и т.д. Рассмотрены ведение статистики успеваемости обучающихся и интерфейс взаимодействия обучающихся и преподавателей.

Выводы. Опыт внедрения ЦАП в учебный процесс курса «Программирование на языке Python» подтвердил возможность обеспечения персонификации учебного процесса для обучающихся в виде индивидуальных образовательных траекторий.

Ключевые слова: массовое обучение программированию, цифровой ассистент преподавателя, генерация заданий, индивидуальная траектория обучения, мотивация обучающихся

• **Поступила:** 29.12.2021 • **Доработана:** 03.03.2022 • **Принята к опубликованию:** 11.05.2022

Для цитирования: Андрианова Е.Г., Демидова Л.А., Советов П.Н. «Цифровой ассистент преподавателя» в массовом профессиональном обучении для цифровой экономики. *Russ. Technol. J*. 2022;10(3):7–23. https://doi.org/10.32362/2500-316X-2022-10-3-7-23

Прозрачность финансовой деятельности: Авторы не имеют финансовой заинтересованности в представленных материалах или методах.

Авторы заявляют об отсутствии конфликта интересов.

INTRODUCTION

The digitalization of the economy, which has the potential to radically transform all fields of human activity, generates an urgent need for a large number of professionals with IT or digital competences. Thus, a significant challenge facing learning providers concerns how to provide massive professional training of personnel for the digital economy, in the first place, software developers. In the medium term, the Russian labor market anticipates an increase in the demand for software developers in such promising areas as artificial intelligence, big data analytics, robotics, virtual reality, and the Internet-of-things[1] [1, 2].

Technologies of massive training of software developers require the active implementation of intelligent and IT technologies in the organization of learning process and use of electronic educational resources to support convenient and rapid communications between participants in learning process and apply advanced pedagogical innovations and practices.

However, there is a concern that the digitalization of the learning process for massive training of software developers might imply the unification of approaches to all students. On the contrary, it is necessary to provide a student with wide opportunities to create and support personification of learning, first of all, by creating individual learning paths, which imply the active individual work of the student. According to Yaroslavtseva, in making digital technologies available for massive use, which structures the social system and social relationships, electronic culture implies the formation of new views of the world and methods of perceiving it. The human in digital space thus becomes a continuously developing "open anthroposystem," an interactive system, which "rapidly expands its capabilities and creates the corresponding parameters of the future" [3]. Meanwhile, some pedagogical experts note a decrease in the motivation of students for individual work. There is even an opinion that this is due to educational labor now having the "traits of forced labor" [4]. Thus, existing standards and training materials are failing to provide students with personally significant learning results but are instead perceived as motivated by the requirements imposed by the education system. Students do not feel as of their actions are independent; rather, they are forced to conform to the learning objectives, contents and methods of an external control system. "Motives for learning are replaced by motives for responsibility, social need and enforcement: 'you should,' 'you must'" [5].

Various concepts of learning approaches aimed at increasing the motivation of students have been proposed. For example, the heuristic learning concept, which is based on an exteriorization model, is aimed at the self-actualization of a student [6, 7]. Conversely, the developmental learning concept, which is based on an interiorization model, is targeted at the development of a student [6, 7]. Whether proceeding according to the heuristic or developmental learning approaches, the purpose of the teacher lies not so much in the development of course content (e.g., lectures and practical exercises) as the search and monitoring of available educational resources, which can comprise materials for the learning content of the course. By taking this approach, it is possible to support a student in creating an individual learning path. However, it should be noted that the important issue of providing support of the motivation of a student to software development competence cannot be reduced only to the material aspect.

When carrying out training of software developers, technical universities using a classical form of programming training have long competed in the provision of high-quality massive open online course at leading companies and educational institutions and with specialized online educational platforms, such as Stepik [8] and JetBrains Academy [9], which provide tools for the efficient learning of programming languages that are currently in demand by employers in software development.

The increased use of online technologies in the learning processes as part of recent epidemic control measures resulted in blended learning becoming a standard part of the learning process at universities. This has changed the duties of the teacher by supplementing them with a tutoring function, which obviously increases the scope of the teacher's creative functions related to

[1] Rynok truda v Rossii (The labor market in Russia). *Tadviser. Gosudarstvo. Biznes. Tekhnologii* (in Russ.). https://clck.ru/ZD5AU. Accessed November 28, 2021.

the tutorship due to the necessity to take into account the individual traits of a student in practice-oriented training in programming. Due to the resultant significant increase in the teacher's workload, it has become necessary to automate the routine part of the teacher's duties by creating and using a Digital Teaching Assistant (DTA). Comprising a kind of intelligent learning system, a DTA can take on functions of delivery and creation of practical programming tasks, generation of tasks and checking of results, monitoring of the attendance and work of students, digital footprint acquisition, etc.

Thus, the development of a DTA, as well as its realization and implementation in teaching a training course in programming at a technical university, becomes an urgent problem.

1. PEDAGOGICAL DESIGN AND MAIN ELEMENTS OF COMPUTERIZATION OF PROGRAMMING TRAINING

The state of the art in the educational system of programming training provides a teacher with significant opportunities for setting learning objectives and undertaking pedagogical design of a training course, which determines means for achieving the set objectives. Pedagogical design is a systematized approach to creating learning solutions, which uses pedagogical principles and theories to ensure high quality of learning [10].

In the context of digitalization of education, pedagogical design represents an efficient tool for developing learning content aimed at ensuring a reasonable combination of online and offline learning (blended learning), creating an intelligent environment within the programming training course and supporting the realization of an individual learning path.

The main principles of pedagogical design are to provide a clear explanation of the goals and objectives of learning, attraction and retention of student's attention, awaken interest in topic being learned or learning methods applied, ensure rapid testing of obtained knowledge in practice, receive feedback from students, deliver assessments (including self-assessment) of academic achievement and general assessment of efficiency of training course, and help to student in preserving and correctly using obtained knowledge [10].

By using pedagogical design principles in developing the content of a training course, it becomes possible to determine optimal conditions for achieving learning goals: model of interactive interaction of participants in the learning process, forms of representation of the learning content of the training course and methods of increasing the motivation of students for independent work on the materials of the course (cognitive function). For example, using crowdsourcing methods identifies learning needs and deficits that are significant for students trained in this programming training course.

The pedagogical design of the programming training course encapsulates not only the format of representation of the learning content, but also methodological guidelines for students and teachers, tasks and exercises on the developed content, as well as test assignments and student self-assessment forms. Test assignments, their performance, and the subsequent self-assessment of academic achievement in the course by students can partially be based on gamification. Student self-assessment of competence can correspond to events in an electronic system, e.g., such as evaluation of a performed task on a pass/no pass or letter-grade basis, and so on. For the content of the training course to be of high quality, motivating stimuli should necessarily be included, e.g., as creative tasks, tasks for determining the problematic area of the course, etc.

The pedagogical design of a practice-oriented programming training course has the following mandatory elements:

- interactive materials available online (a student can run examples directly in a study guide, can use elements of a built-in graphical interface to update the contents of graphs, etc.);
- presentation of the materials in a personalized form: as a dependency graph (knowledge graph);
- online system for reception and verification of tasks;
- automatic adaptation of the complexity of tasks and theoretical materials to the level of a student;
- automatic task generation;
- automatic generation of hints and comments on the solutions of the tasks.

2. CONTEMPORARY PROGRAMMING LEARNING TOOLS

A promising trend in the pedagogical design of a university course of massive training in programming consists in the use of learning tools and blended learning (a combination of offline and online learning). The asynchrony of the online part of the learning process provides a student with the space to select the material and time to learn it. Of particular relevance here is the flipped classroom model based on online self-directed online learning of students in the main theoretical knowledge in the course. In this case, classroom lectures become more interactive to include discussions of the topics self-learned by students; such lectures widely feature additional materials, as well as involving collaborative solution of tasks.

For students to better retain the training material, a conventional 1½-hour-long lecture is more and more often represented online as a set of short (5–15 min) videos [11]. Using virtual reality technologies in learning

process becomes increasingly popular [11]. For prompt assessment of academic achievement, feedback with students can be organized directly during the online lecture as brief test questions and test tasks. A special role in ensuring the deep retention of the lecture material by students can be played by such interactive elements of the lecture as executable code fragments and various apparent dynamic models, which are controlled by a graphical interface (explorable explanation) [12]. In particular, the Online Python Tutor [13] visualizes details of the execution of a Python program in a dynamic mode.

In organizing a massive programming training course, it is important to personalize learning by creating an individual learning path to adapt the learning process to an individual student's needs. For lectures, the material of the course can be represented as a knowledge graph with the freedom of choice between topic nodes taking into account dependency edges [14].

During performing practical exercises and under quantitative constraints on instructional personnel, an intelligent tutoring system (ITS) can be employed to obtain feedback from students, transfer an individual instruction or provide a response to the actions of students on a real-time basis without teacher intervention. ITS are computer systems that combine a knowledge domain model with a student model and a teacher model. Such integrated systems often use machine learning methods and have a complex design [15].

The complexity of the design of an ITS as a single system integrating all the situations and scenarios of interaction led to propagation of computer tools, which solve individual problems related to an ITS. In particular, they comprise automated grading systems [16], data-driven hint generation systems [17], as well as automatic generators of exercises of a given complexity [18] and systems for checking solutions for plagiarism [19].

A disadvantage of such systems is that they focus only on a student's individual work along with a limited, formally defined set of tasks. In this context, specialized forums such as Piazza[2], which may include rewards for students who help their colleagues, can be useful. An important role is also played by using learning projects developed collaboratively by students (project-based learning) and using technologies to provide the gamification of learning processes [20].

3. DIGITAL TEACHING ASSISTANT

3.1. DTA structure

In university massive programming training courses, much of the teaching workload is borne by teacher assistants, who create programming tasks and check the correctness of their solution by students. In this context, a new approach is proposed for organizing the learning process of training of software developers based on a DTA computer system. DTA helps teachers working under massive learning conditions to concentrate on functions that require a creative approach, in particular, creating and discussing nontrivial programming tasks and also developing the DTA functionality.

The main DTA functions (Fig. 1) are:

- generation of individual tasks;
- check of tasks with informative answer for a student;
- recording student academic achievement statistics;
- interface of interaction between learning process participants.

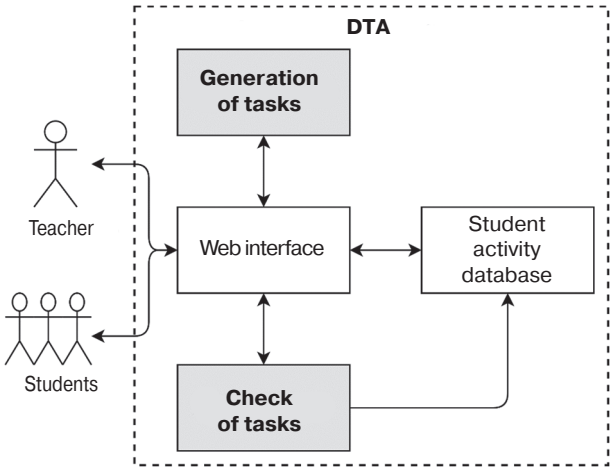

Fig. 1. DTA structure

Unlike an ITS [21], a DTA is not intended to completely replace a teacher or resemble a "black box" that automatically select tasks during learning or evaluates solutions according to parameters hidden from a student. The model of a tutor used in ITS is replaced in DTA by real teacher's settings.

3.2. Automatic generation of programming tasks

The automatic generation of programming tasks provides each of students with an individual set of tasks. This inherently laborious function is difficult for teachers to perform alone, especially if the number of students is large. Automatic task generation also partially solves the problem of plagiarism of solutions.

The following requirements are imposed on a task generator;

- support of the main topics of the course and language constructs;
- interest and practical utility of tasks;

[2] https://piazza.com/. Accessed January 13, 2022.

- clarity and univocality of the text of a task statement and its resemblance to a human-composed text;
- identical complexity of all the generated variants of a task at fixed settings of the task generator;
- absence of obvious cliché solutions to different variants of a task;
- possibility of automatic check of solution.

A DTA contains a large number of task generators $G = \{g_1, \ldots, g_n\}$, each generating variants of tasks of its own type. The inputs of the task generator function $g_i(h, s)$ consist in the hash value h of the individual data of a student and parameter s of complexity or size of the task.

Approaches to automatic generation remain to be developed for all the types of programming tasks. The types of tasks for which generators have already been created are discussed below.

In a course of programming in a certain language, one can identify the following classes of practical tasks reflecting the modern specificity of software development:

- creation of a new program;
- analysis of the existing program;
- creation of unit tests for the existing program.

Among the tasks involved in creating a new program, there are generators of tasks of such types as [18]:

- translation of a certain representation into a program code;
- data format conversion.

In tasks of the former type, the input representation can be a mathematical formula. Thereby, a student learns a practically important skill to translate a mathematical language into a programming language. Another example of a task of the former type is to translate a program from one programming language into another. The input representation can also be graphic notation. Examples of tasks using graphic notation including converting a finite-state machine graph or a UML diagram into a program code.

Tasks to convert input data format into output data format can use both text and binary data formats.

In the general form, generators of tasks of the considered types can be created using approaches from the field of combinatorial problems of artificial intelligence. In particular, a promising approach is to use the constraint satisfaction method [22]. Using this approach, a set of variables and a set of their values describe a space of generated programs or data formats, and a set of constraints enables one to filter off only the results that correspond to given characteristics for a chosen type of tasks.

Random configuration p (program or data format description) from a space of configurations P is defined as

$$p \in \mathrm{P} \wedge \phi_c(p) \wedge \phi_t(p).$$

Here, the specification predicate ϕ_c determines the conditions of choosing p, while the predicate ϕ_t characterizes the possibility to create for p a set of nondegenerate (i.e., with different values within given limits) program tests.

Random generations can be generated using probabilistic grammars, in which each rule is supplemented with the probability of using it in the process of generation of the result. The same approach is used in fuzzing [23].

Of special interest are probabilistic context-sensitive grammars, which not only enable a syntactically correct result to be achieved in the event of obtaining a random result, but also carry out a number of semantic checks. An example of such a check is correct use of a previously defined variable in its scope.

In generating a task, it is insufficient to formulate its statement. It is also necessary to obtain a set of program tests, against which the correctness of students' solutions is evaluated. The step of test generation is part of the total process generating the task. If the quality of the obtained tests is unsatisfactory, the statement of the task should be changed.

Along with simple unit tests, which contain set of pairs $\{(x_1, y_1), \ldots, (x_n, y_n)\}$, where x_i is the vector of input values and y_i is the computation result, tests with monitoring the state for responding systems can be used. The input data in this case are a trace of calls (methods, messages) and the responses of the system to each of the calls.

Among typical programming tasks for analyzing the correctness of the existing programs, generators that allow the creation of variations using a set of program templates could be of interest, as well as those for introducing an error to a program code, whose correction is delegated to a student.

A class of tasks related to the creation of program tests is characterized by using mutation testing [24] to determine the quality of program tests composed by students for an automatically generated program.

3.3. Realization of task generators

Figure 2 presents a general flowchart of a programming task generator. The statement of a task is formulated based on the hash code of information on a student and the task complexity parameter, along with program tests for automatic check of solutions.

The possible realization of the random configuration generator described below forms a random program or a random data format. The configuration is further subjected to a sequence of simplifications of its structure. In the case of a generated program, they can consist of conventional compiler-side optimizations, such as constant folding and simplification of expressions.

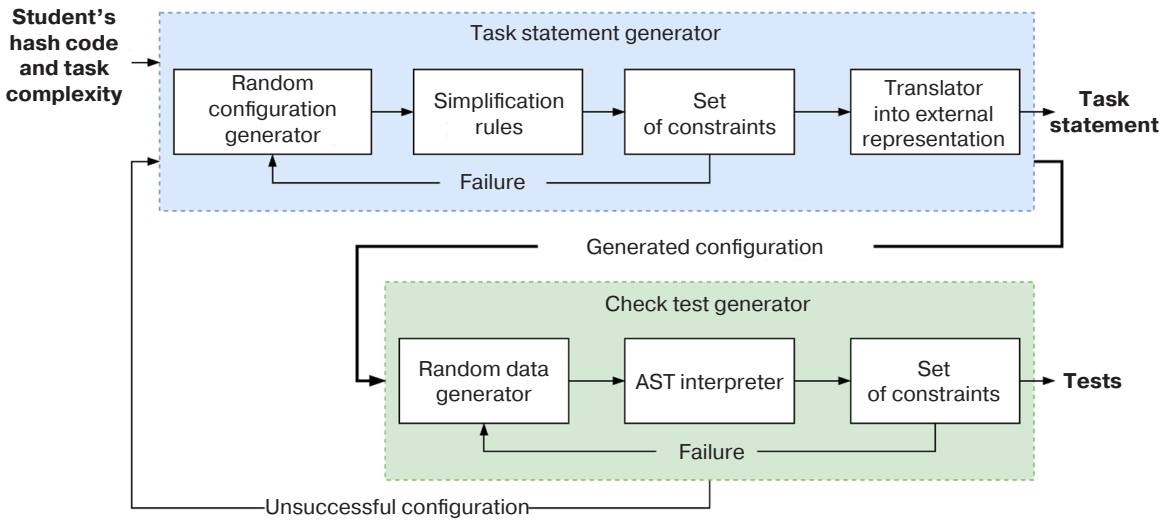

Fig. 2. Flowchart of programming task generator

The characteristics of the configuration are evaluated using a set of constraints according to which the uniform complexity of the generated tasks is maintained. The well-generated configuration is translated into an external representation, which comprises the task statement for a student.

The check test generator creates random data according to the type of a task being formed. The previously obtained configuration is evaluated using a set of tests. In the case of a program configuration, its internal representation is interpreted as an abstract syntax tree (AST) using random data as program inputs. In the case of a data format configuration, the corresponding structures are filled with random values. If no set of tests with given characteristics is successfully formed (e.g., after making a given number of attempts), then the return to the generation of a new configuration is performed.

To realize a random configuration generator, it is proposed to use a special grammar-describing language—configuration generation rules. This language is an extension of probabilistic context-sensitive grammars and is a set of combinators in the spirit of functional programming.

Combinators are higher-order functions used to create embedded domain-specific languages (eDSL). In particular, numerous realizations of parser combinators are known [25]. In the case under consideration, the combinator after computation of the arguments returns the lexical closure *f*:

$$f: c \to c'.$$

This uses a computation model in which combinators receive context c and return a new version of it. The context $c = \{(a_1, v_1), \ldots, (a_n, v_n)\}$ comprises a set consisting of n pairs of the attribute–value form. The context always contains the following attributes: computation result a_{val} and Boolean error value a_{err}. If an error emerges during the computation, a local rollback occurs from combinators: this is a return to one of the calling combinators, which processes this error.

The basis of the considered combinator language are combinators realizing operations known from the theory of formal languages: concatenation (*seq*) and union (*alt*). The basic combinator *seq* computes a sequence of its combinator arguments, thus updating the context. If an error emerges, there is a return from the combinator with restoration of the state of the context at the beginning of the execution of *seq*. The basic combinator *alt* computes, among its arguments, a randomly chosen combinator. For each of the arguments, the probability that it has been selected can be given. The combinator *check* checks the argument expression for truth using the current context as an argument. It the expression is false, then the unit value of a_{err} in the context indicates an error. The combinator *find* repeats the execution of the argument combinator until the computation result is good.

A variant of a combinatorial language developed in Python is considered. Using combinators, a random polynomial generator can be developed as shown in Fig. 3. This generator is realized according to three rules. The *coeff* rule in Fig. 3a determines the value of the coefficient. A choice is made between a random value in a given range and zero with the probability given in *prob*. The *term* rule forms an AST representation for an expression of the form kx^n. The *main* rule forms an AST representation of a polynomial to the third degree. The combinator *let* is used to name the results, while *ref* provides a means for their subsequent reference. Using the combinator *check*, a constraint is imposed that the generated polynomial should have at least one nonzero coefficient at x.

```
coeff = alt(randint(-100, 100), 0, prob=[0.3, 0.7])

term = λ k, n: mul(k, power('x', n))

main = find(seq(
    let(a=coeff), let(b=coeff),
    let(c=coeff), let(d=coeff),
    add(
        term(ref('a'), 3),
        term(ref('b'), 2),
        term(ref('c'), 1),
        term(ref('d'), 0),
    ),
    check(λ c: c.a != 0 or c.b != 0 or c.c != 0)
))
```

(a)

$$86x^3 + 20x^2 + -27$$
$$-85x$$
$$-98x$$
$$86x^3 + -47x^2 + 51x$$
$$40x + 44$$
$$93x$$
$$-63x + -66$$
$$-62x^3 + -17x$$
$$-97x^2$$
$$94x^2$$

(b)

Fig. 3. Random polynomial generator:
(a) realization in eDSL and (b) examples of the results of generation in the LaTeX format

The search for the correct result is made by the combinator *find*. Thus, a simple declarative constraint programming system is realized. This approach is much less productive than, e.g., SMT solvers and specialized solvers for constraint programming tasks. Nevertheless, the considered declarative language allows constraints of arbitrary complexity to be used. Moreover, the generation rate typically is not the most important characteristic of creating learning tasks. Finally, SMT solvers and other third-party tools can be used together with the considered combinatorial language.

Figure 4 presents a more extended example of the realization of generation of random functions in C++ for array handling. Here, a task in the spirit of finite-impulse-response filters is imitated.

In the generated random functions, the data types and the variable names are consistent. A cycle can be realized using both *for* and *while*, whose realization is similar to that of *for*. The realization of the combinator *expr* is not shown because it presents nothing new in comparison with the previous example of polynomial generation.

```
args = shuffle_list(
    decl_ptr(ref('ty'), ref('buf1')), decl_ptr(ref('ty'), ref('buf2')),
    decl('int', ref('size'))
)
for_array = λ *body: for_loop(
    eq(decl('int', ref('idx')), ref('mtap')), lt(ref('idx'), ref('size')),
    postop('++', ref('idx')),  block(*body)
)
memset = call('memset',
    lst(ref('buf2'), 0, mul(call('sizeof', ref('ty')), ref('size')))
)
loop_body = stmt(eq(load(ref('buf2'), ref('idx')), expr))
main = seq(
    let(ty=one_of('int float double')),
    let(name=one_of('data buf arr x y')),
    let(buf1=to(λ c: c.name + '1')), let(buf2=to(λ c: c.name + '2'))
    let(size=to(λ c: c.buf1 + '_size')),
    let(idx=one_of('i j k n')),
    let(taps=to(λ c: sorted(sample(range(1, 5), randint(1, 3))))),
    let(mtap=to(λ c: max(c.taps))),
    function(
        one_of('calculate process compute perform'),
        'void', args, block(stmt(memset),
            alt(for_array(loop_body), while_array(loop_body))
        )
    )
)
```

Fig. 4. Realization of a generator of random functions of array handling

Figure 5 gives some examples of the generated random functions.

```
void process(int x_size, double *y, double *x) {
  memset(y, 0, sizeof(double) * x_size);
  for (int i = 4; i < x_size; i++) {
    y[i] = 8 * x[i] + 2 * x[i - 1] + -4 * x[i - 3] + -5 * x[i - 4];
  }
}
void perform(float *arr1, int arr1_size, float *arr2) {
  memset(arr2, 0, sizeof(float) * arr1_size);
  int k = 2;
  while (k < arr1_size) {
    arr2[k] = 5 * arr1[k] + 8 * arr1[k - 2];
    k++;
  }
}
void compute(int y1_size, float *y2, float *y1) {
  memset(y2, 0, sizeof(float) * y1_size);
  for (int k = 3; k < y1_size; k++) {
    y2[k] = -6 * y1[k] + -7 * y1[k - 3];
  }
}
```

Fig. 5. Examples of the generated random functions of array handling in C++

3.4. Automatic check of tasks

The correctness of the obtained solution was checked using a set of program tests, which is formed by a task generator. For testing, a "sandbox", representing a medium of safe execution of program code, is realized.

It is also expedient to make a comprehensive evaluation of a student's solution without restricting only to checking the correctness of the result. Elements of such evaluation made automatically can include:

- check of solutions for plagiarism in the case of tasks created manually by teachers;
- check for compliance with the program writing style standard;
- analysis of the metrics for assessing the program complexity;
- evaluation of the correctness of the result, the computational complexity of the program, and the memory size used by the program.

The first three items in the list can be realized using statistical analysis tools, while the last item, requires the use of program profiling tools.

In the case of an incorrect solution, the answer of DTA should include at least a massage of the compiler and an indication of the nonexecuted test.

A more informative answer includes an indication of a typical error made by a student. Such a mechanism can be realized by seeking templates in the AST of the incorrect program. Rules of the template-message form are added to DTA by the teacher. There are also approaches that enable the generation, based on an incorrect solution, of automatic hints directing the student toward the correct result [26].

3.5. Recording of statistics of academic achievement of students

By storing solutions received from students in a DTA database, the teacher can dispense with the requirement to manually fill out a gradebook. Furthermore, by comprehensively evaluating solutions using the DTA, it is possible to abandon simplified formulas for calculating the final grade or ready sets of tasks. As a result, the teacher can satisfy him- or herself with indicating the minimum necessary number of solved tasks for each of the topics of the course. At the same time, a DTA allows a student to carry out an unlimited number, complexity and type of training tasks.

3.6. Interface of interaction between students and teachers

For a DTA, it is preferable to use a web interface that provides access to the system from any device with an Internet connection.

From a student's standpoint, the interface should allow the student to choose the types and parameters of tasks on the current topic of the course. By providing the results of the integrated assessment together with the history of the previous results, the student is empowered to evaluate his or her progress.

From a teacher's standpoint, an important role is played by the analysis of students' activity data. Using the DTA interface, the teacher can promptly assess the situation and make the necessary decisions. This concerns, e.g., a situation in which a task turned out to be too easy or too difficult to solve for the majority of students. Here, data analysis can be used to identify underachievers or predict a student's final grade.

4. EXPERIENCE OF DEVELOPMENT OF PYTHON PROGRAMMING TRAINIUNG COURSE

The Python programming training course at the MIREA – Russian Technological University, Moscow, Russia, is organized as a blended learning environment. Figure 6 presents online tools used in this course.

The instructional personnel of the course comprise 4 lecturers and 13 tutors of practical exercises. Since more than 1500 students participated in the training, a Telegram channel was created for their prompt notification of important events during the course. In terms of a duplex communication channel, the MantisBT bug tracker was used. This was already known to the

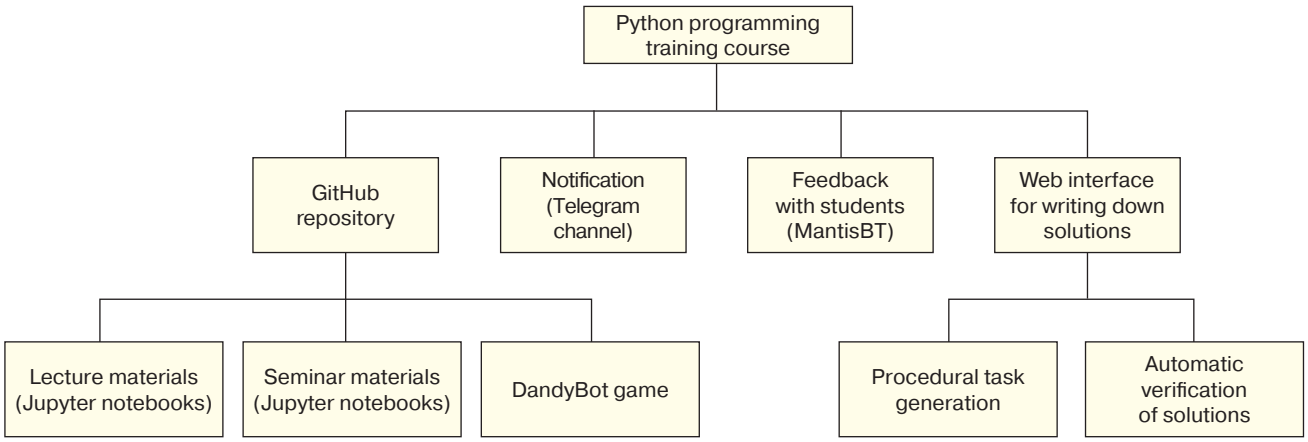

Fig. 6. Structure of the inline part of the Python programming training course

students from a configuration management course and was used as a replacement of conventional interaction tools (electronic mail and forum).

The basic part of the course, which includes issues of using the main constructs of the language and introduces students to the object-oriented programming style accepted in Python, was presented in the form of lectures. The final lectures consider a number of special issues, in particular:

- test automation: test coverage metrics, mutation testing, design by contract, property- and model-based testing;
- functional programming: higher-order functions, closures, decorators, generators, immutable data types, functools and itertools modules, and NumPy library.

The main materials of the course are stored in a separate GitHub repository. The students are rewarded for errors found in the materials. Messages about the found errors are received as pull requests.

The use of Jupyter notebooks increased the interactivity of the online lectures delivered in a webinar format. During the lecture, the students directly experiment with brief proposed programs; responding inter alia to the students' questions, the lecturer can modify and run examples of programs in cells of a Jupyter notebook. The representation of lecture material in the conventional slide form supplemented with the interactivity provided by cells is performed using the RISE extension for Jupyter notebooks.

The classroom lectures include an interactive theoretical part, which allows:

- assessment of the retention of the lecture material by the students online and, if necessary, timely addressing of gaps in the students' knowledge;
- introduce the students to the subject domain of the tasks considered in the exercise.

In seminar classes, tasks of several types are used:

- procedure-generated tasks;
- tasks from Jupyter notebooks;
- tasks related to DandyBot game;
- individual and group creative projects on tutors' topics.

Among the main problems involved in the design of the considered course were the creation and organization of rapid checks of practical tasks under massive learning conditions given a shortage of tutors. It was decided to use an approach based on procedural generation: an individual set of tasks was automatically formed for each of the students [18]. As well as solving the problem of plagiarism among the students, this approach also reduced the tutors' workload due to the automatic checking of solutions.

For example, variants of the following tasks are generated automatically:

1) conversion of mathematical notation into code: arithmetic expression;
2) conversion of mathematical notation into code: branching;
3) conversion of mathematical notation into code: cycles;
4) conversion of mathematical notation into code: recursion;
5) conversion of solution tree into code;
6) realization of bit field swapping in a bit word;
7) tabular data format conversion (Fig. 7);
8) binary file format parsing (Fig. 8);
9) realization of finite-state machine by object-oriented programming features.

For a given generated task, the degree of complexity can be set along with the number of formed tests to allow a student to check the correctness of his or her solution.

An important part is played by the informativeness of messages received by the students from the system submitting task solutions. For some types of tasks, there is a support of rules according to which a detailed error message is generated and accompanied by a reference to methods for obtaining the correct solution using the template found in the solution code. The pattern-matching search is performed at the level of the representation of a student's program as an AST. In the bit field swapping

task, such rules allow an incorrect solution approach to be identified (the presence of a number-to-string conversion function) as well as hints given concerning the correct direction (using bitwise operations).

The exam in the Python programming training course also uses procedure-generated tasks. Figure 9 presents the web interface via which the students send their solutions of tasks of this type.

For the majority of the seminar class, practical tasks from Jupyter notebooks are discussed and solved. Some of such tasks are considered "at the blackboard": the tutor invites someone of the students to take a seat at a computer connected to the room's projector. Using such a format, other students can keep track of the solution and the tutor's comments. Better-prepared students during the classroom learning solve tasks by themselves and send the solutions to the tutor for checking through the bug tracking system. Such a workflow allows the tutor to actively participate in the learning without continuously moving over the room from computer to computer and, thereby, to respect COVID-19 restrictions more rigorously. A sudden transition to offline learning using such a workflow of practical exercise is not a serious problem because the students send their task solutions using the same bug tracking system.

Task 3. Realize a tabular data conversion function. The input and output tables are given in row-by-row form. Filled cells have a row data type. Empty cells have the None value.

Make the following conversions of the input tables:
– remove empty columns;
– divide column 1 by divider "!";
– convert cell contents by examples;
– transpose the table.

Examples of table conversions:

1. Initial table:

		+78694947943!03-08-2003	No
		+79774177489!27-03-2002	Yes
		+77571568485!05-12-2001	Yes
		+73113372701!21-07-2003	No

Conversion result:

8694947943	9774177489	7571568485	3113372701
false	true	true	false
03-08-03	02-03-27	01-12-05	03-07-21

2. Initial table:

		+76058136232!11-04-2001	No
		+72178434337!02-10-2000	No
		+76826857881!25-11-2003	Yes

Conversion result:

6058136232	2178434337	6826857881
false	false	true
01-04-11	00-10-02	03-11-25

Fig. 7. Example of an automatically generated task of tabular data conversion

Task 3.1. Realize binary file format parsing (in the spirit of the WAD format of the Doom game or the graphic PNG format). The data begin with the signature 0 × 42, 0 × 58, 0 × 59, 0 × 9a followed by the structure A. Byte order: big-endian. The addresses are expressed as the offsets from the data origin. In the solution, it is allowed using the struct module.

Structure A:

1	Structure array B, size 4
2	int16
3	int16
4	int16
5	Structure C
6	Size (uint32) and address (uint32) of the double array
7	uint32
8	Structure D

Structure B:

1	uint32
2	double
3	Size (uint32) and address (uint32) of the char array

Structure C:

1	Int16 array, size 5
2	uint64
3	int16
4	int8

Structure D:

1	int16
2	int8
3	int8
4	uint64
5	Size (uint32) and address (uint32) of the uint16 array

Fig. 8. Example of an automatically generated task of binary file format parsing (test data are not shown)

Exam in the Python programming training course

Group B1 Variant 25

Solution:

```
import struct

def f31(x):
    res = {}
    res['A1'] = struct.unpack('f', x[3:7:])[0]
    res['A2'] = struct.unpack('q', x[7:15:])[0]
    res['A3'] = {}
    addr = x[15] + x[16] * 16 + x[17] * 16 * 16 + x[18] * 16 * 16 * 16
    res['A3']['B1'] = struct.unpack('q', x[addr : addr + 8])[0]
    res['A3']['B2'] = struct.unpack('H', x[addr + 8 : addr+ 10])[0]
    res['A3']['B3'] = struct.unpack('b', x[addr + 10 : addr + 11])[0]
    res['A4'] = struct.unpack('h', x[19:21:])[0]
    res['A5'] = {}
    res['A5']['C1'] = ''
    size = x[21] + x[22] * 16 + x[23] * 16 * 16 + x[24] * 16 * 16 * 16
    addr = x[25] + x[26] * 16 + x[27] * 16 * 16 + x[28] * 16 * 16 * 16
    for i in range(addr, addr + size):
        res['A5']['C1'] += chr(x[i])
    res['A5']['C2'] = [] # = ''
    size = x[29] + x[30] * 16
    addr = x[31] + x[32] * 16
    for i in range(size):
        res['A5']['C2'].append({})
        res['A5']['C2'][i]['D1'] = struct.unpack('H', x[addr + 4 * i : addr + 2 + 4 * i :])[0]
        res['A5']['C2'][i]['D2'] = struct.unpack('H', x[addr + 2 + 4 * i : addr + 4 + 4 * i :])
```

Send

Table of results

Error log

Fig. 9. Web interface for receiving the solutions of automatically generated tasks

Practical tasks from Jupyter notebooks, which belong to different subject domains, are developed so that they are of interest for students and in such a way as to permit a solution to be obtained within a single practical exercise. The topics of the practical tasks are as follows: random digital economy report generator; Burrows–Wheeler transform; ASCII banner generator; Schelling's model of segregation; graph imaging by physical modeling; stack language interpreter; hierarchical clustering algorithm; formal verification of computer game puzzles; undo/redo mechanism in graphic editor; SQL-like query language; Julia fractal; Floyd–Steinberg algorithm.

A special type of practical tasks is constituted by tasks of simple data analysis based on information collected by a web tool for receiving the students' solutions of procedure-generated tasks. Here, in the reflection mode, the students learn their own activity in the course. Figures 10 and 11 show the results of such an analysis.

Additional tasks for practical exercises are provided via the DandyBot game, which was developed especially for the considered course. It belongs to the genre of games for programmers in which, for success in the game, it is necessary to write a program code. The DandyBot game, an example of a level of which is shown in Fig. 12, is presented in the style of Roguelike games to allow the "intelligence" of player characters to be programmed in Python.

At the first levels, the players should write the simplest programs; therefore, even the students with minimum programming experience can be involved in writing a code in Python in a playful way.

The DandyBot game used by the students is a semi-product presented in the form of a GitHub repository requiring numerous improvements. This stimulates the students to develop the program code of the DandyBot project as well as to create additional levels.

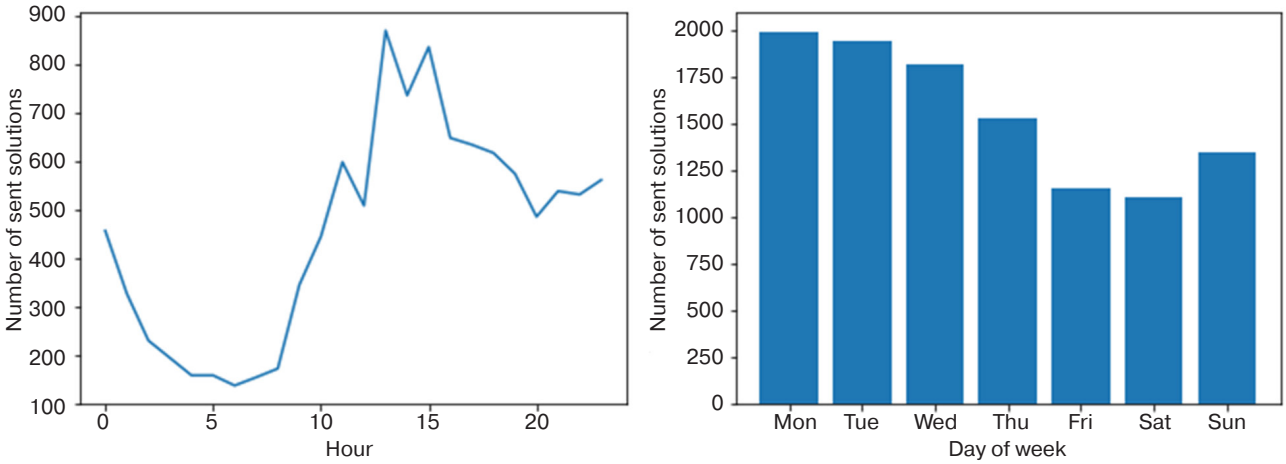

Fig. 10. Students' activity over the day and the days of the week

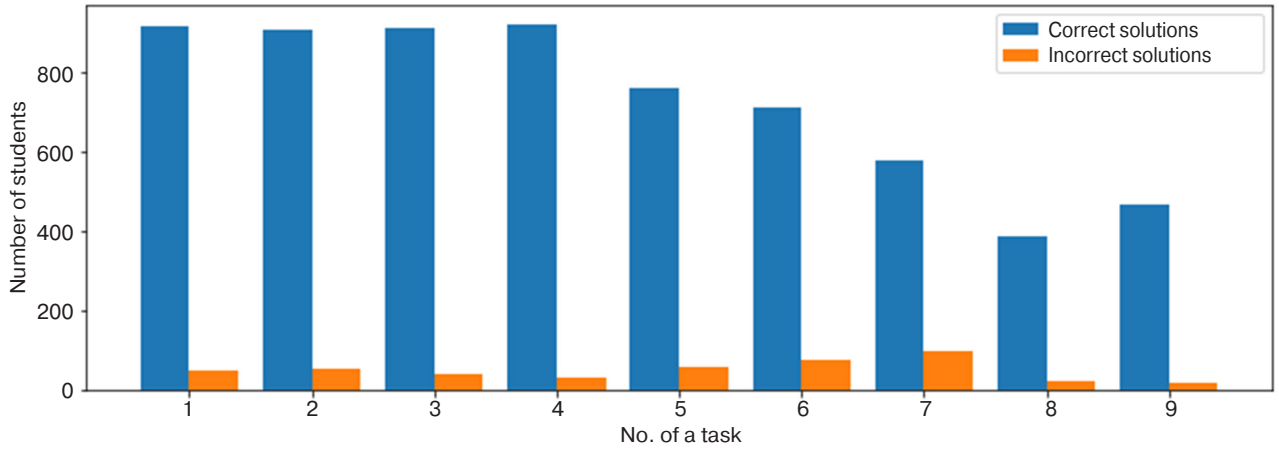

Fig. 11. Statistics of solutions of procedure-generated tasks that were sent by the students

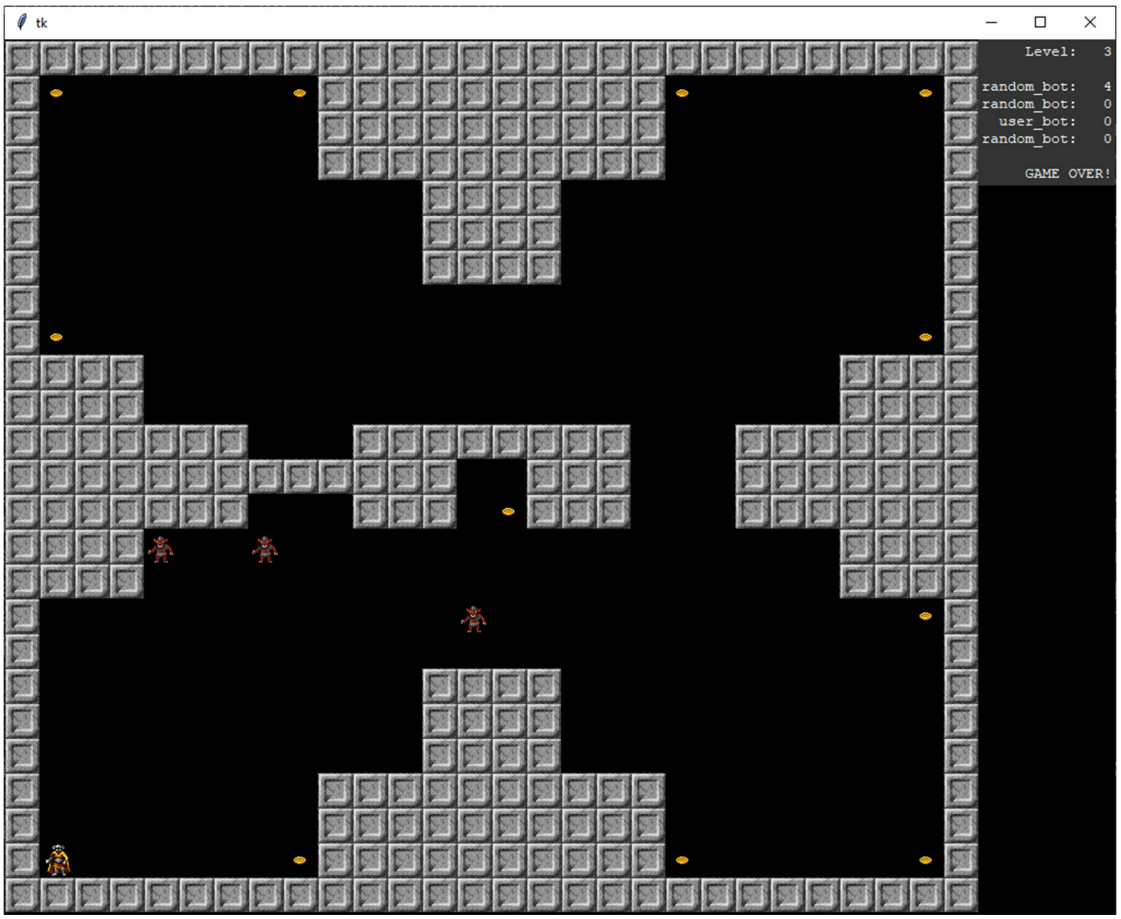

Fig. 12. A level of the DandyBot game with a player controlled by a program in Python language

CONCLUSIONS

Under present-day conditions (massive programming learning, transition from classroom learning to distant/blended learning and back within a semester, and competition with specialized courses and platforms of learning of programming languages), teachers at university programming teaching courses encounter a significant increase in the workload, which is mostly related to performing routine activities.

The developed DTA automates such procedures as generating individual tasks and checking their solutions, as well as providing an informative answer to the student and recording the academic achievement of students. This allows the teacher to devote more time and attention to functions requiring creative input—in particular, to creation and discussion of nontrivial programming tasks.

The experience of implementing the DTA in the learning process of the Python Programming course confirmed the possibility of personifying the learning process for students in the form of individual learning paths.

Authors' contributions:

E.G. Andrianova: The urgency of automating the process of massive programming training was substantiated. The personification methods for the massive programming training process were investigated and adapted. Types of practical tasks reflecting the contemporary specifics of software development were highlighted.

L.A. Demidova: A procedure for a comprehensive assessment of the student's solution was proposed. This procedure included checking the correctness of the result and checking for plagiarism of solutions in the case of tasks created by the teacher manually, as well as compliance of the program style with the standard and metrics for assessing the program complexity, etc. The maintenance of statistics of students' progress and the interface of interaction between students and teachers are described.

P.N. Sovetov: Algorithms satisfying the constraints for creating generators of programming tasks were developed, the Digital Teaching Assistant computer system architecture was built, and the implementation of this system was completed. A procedure for automatic verification of task solutions was developed and described. This procedure is realized using software tests generated by the task generator.

All authors took part in the development and implementation of the Digital Teaching Assistant computer system in the practice of the RTU MIREA educational process in the Programming in Python discipline.

REFERENCES

1. Gudov M.M., Ermakova E.R. Structural transformations of the Russian economy in the conditions of acceleration digitalization of industrial relations. *Teoreticheskaya i prikladnaya ekonomika = Theoretical and Applied Economics*. 2020;2:1–8 (in Russ.). https://doi.org/10.25136/2409-8647.2020.2.32625
2. Novikova E.S. Risks and perspectives of higher school transformation for the Russian economy in conditions of gobalization and digitalization. *Mezhdunarodnaya torgovlya i torgovaya politika = International Trade and Trade Policy*. 2021;7(4):147–162 (in Russ.). https://doi.org/10.21686/2410-7395-2021-3-147-162
3. Yaroslavtseva E.I. Humanitarian aspects of digital technologies. *Vestnik Rossiiskogo filosofskogo obshchestva = Russian Philosophical Society*. 2020;1–2(91–92): 248–251 (in Russ.). Available from URL: https://rfo1971.ru/wp-content/uploads/2020/03/09-03_248-251.pdf
4. Yaroslavtseva E.I. The potential of digital technologies and the problems of human creativity. *Voprosy Filosofii*. 2020;11:58–66 (in Russ.). https://doi.org/10.21146/0042-8744-2020-11-58-66.
5. Strokov A.A. Digitalization of education: problems and prospects. *Vestnik Mininskogo universiteta = Vestnik of Minin University*. 2020;8(2):15 (in Russ.). https://doi.org/10.26795/2307-1281-2020-8-2-15
6. Khutorskoi A.V. Pedagogical prerequisites for student self-realization in heuristic learning. *Vestnik Instituta Obrazovaniya Cheloveka*. 2020;1:1 (in Russ.). Available from URL: http://eidos-institute.ru/journal/2020/100/Eidos-Vestnik2020-101-Khutorskoy.pdf
7. Khutorskoi A.V. Interiorization and exteriorization – two approaches to human education. *Narodnoe obrazovanie*. 2021;1(1484):37–49 (in Russ.).
8. Khalyapina L., Kuznetsova O. Multimedia professional content foreign language competency formation in a digital educational system exemplified by STEPIK framework. *Lecture Notes in Networks and Systems*. 2020;131: 357–366. https://doi.org/10.1007/978-3-030-47415-7_38
9. Panova I.V., Kolivnyk A.A. An overview of the content of online courses on teaching the basics of programming in the Python language. In: *Sovremennye obrazovatel'nye Web-tekhnologii v realizatsii lichnostnogo potentsiala obuchayushchikhsya* (*Contemporary Educational Web-technologies in the Realization of the Personal Potential of Students*): Collection of research articles of international scientific and practical conference. Arzamas; 2020. P. 523–528 (in Russ).
10. Vorob'eva N.A., Oboeva S.V., Bernadiner M.I. Using pedagogical design technologies in the context of digitalization of education. *Vestnik Moskovskogo gorodskogo pedagogicheskogo universiteta. Seriya: Informatika i informatizatsiya obrazovaniya = The academic Journal of Moscow City University, series Informatics and Informatization of Education*. 2020;1(51):34–37 (in Russ).
11. Guo P.J., Kim J., Rubin R. How video production affects student engagement: An empirical study of MOOC videos. In: *Proceedings of the First ACM Conference on Learning @ Scale Conference*. 2014. P. 41–50. https://doi.org/10.1145/2556325.2566239

СПИСОК ЛИТЕРАТУРЫ

1. Гудов М.М., Ермакова Э.Р. Структурные преобразования российской экономики в условиях форсированной цифровизации производственных отношений. *Теоретическая и прикладная экономика*. 2020;2:1–8. https://doi.org/10.25136/2409-8647.2020.2.32625
2. Новикова Е.С. Риски и перспективы трансформации высшей школы для российской экономики в условиях глобализации и цифровизации. *Международная торговля и торговая политика*. 2021;7(4):147–162. https://doi.org/10.21686/2410-7395-2021-3-147-162
3. Ярославцева Е.И. Гуманитарные аспекты цифровых технологий. *Вестник Российского философского общества*. 2020;1–2(91–92):248–251. URL: https://rfo1971.ru/wp-content/uploads/2020/03/09-03_248-251.pdf
4. Ярославцева Е.И. Потенциал цифровых технологий и проблемы творчества человека. *Вопросы философии*. 2020;11:58–66. https://doi.org/10.21146/0042-8744-2020-11-58-66
5. Строков А.А. Цифровизация образования: проблемы и перспективы. *Вестник Мининского университета*. 2020;8(2):15. https://doi.org/10.26795/2307-1281-2020-8-2-15
6. Хуторской А.В. Педагогические предпосылки самореализации ученика в эвристическом обучении. *Вестник Института образования человека*. 2020;1:1. URL: http://eidos-institute.ru/journal/2020/100/Eidos-Vestnik2020-101-Khutorskoy.pdf
7. Хуторской А.В. Интериоризация и экстериоризация – два подхода к образованию человека. *Народное образование*. 2021;1(1484):37–49.
8. Khalyapina L., Kuznetsova O. Multimedia professional content foreign language competency formation in a digital educational system exemplified by STEPIK framework. *Lecture Notes in Networks and Systems*. 2020;131:357–366. https://doi.org/10.1007/978-3-030-47415-7_38
9. Панова И.В., Коливнык А.А. Обзор содержания онлайн курсов по обучению основам программирования на языке Python. В: *Современные образовательные Web-технологии в реализации личностного потенциала обучающихся*. сборник статей участников Международной научно-практической конференции. Арзамас; 2020. С. 523–528.
10. Воробьева Н.А., Обоева С.В., Бернадинер М.И. Использование технологий педагогического дизайна в условиях цифровизации образования. *Вестник Московского городского педагогического университета. Серия: Информатика и информатизация образования*. 2020;1(51):34–37.
11. Guo P.J., Kim J., Rubin R. How video production affects student engagement: An empirical study of MOOC videos. In: *Proceedings of the First ACM Conference on Learning @ Scale Conference*. 2014. P. 41–50. https://doi.org/10.1145/2556325.2566239
12. Lau S., Guo P.J. Data theater: a live programming environment for prototyping data-driven explorable explanations. *Workshop on Live Programming* (*LIVE*). 2020. 6 p. URL: https://www.samlau.me/pubs/Data-Theater-prototyping-explorable-explanations_LIVE-2020.pdf

12. Lau S., Guo P.J. Data Theater: A live programming environment for prototyping data-driven explorable explanations. *Workshop on Live Programming* (*LIVE*). 2020. 6 p. Available from URL: https://www.samlau.me/pubs/Data-Theater-prototyping-explorable-explanations_LIVE-2020.pdf
13. Guo P.J. Online python tutor: embeddable web-based program visualization for cs education. In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. 2013. P. 579–584. https://doi.org/10.1145/2445196.2445368
14. Miller H., Willcox K., Huang L. Crosslinks: Improving course connectivity using online open educational resources. *The Bridge*. 2016;43(3):38–45. Available from URL: http://hdl.handle.net/1721.1/117022
15. Utterberg M.M., et al. Intelligent tutoring systems: Why teachers abandoned a technology aimed at automating teaching processes. In: *Proceedings of the 54th Hawaii International Conference on System Sciences*. 2021. P. 1538. Available from URL: http://hdl.handle.net/10125/70798
16. Sherman M., et al. Impact of auto-grading on an introductory computing course. *J. Comput. Sci. Coll.* 2013;28(6):69–75.
17. Rivers K., Koedinger K.R. Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. *Int. J. Artif. Intell. Educ*. 2017;27(1):37–64. https://doi.org/10.1007/s40593-015-0070-z
18. Sovietov P. Automatic generation of programming exercises. In: *2021 1st International Conference on Technology Enhanced Learning in Higher Education* (*TELE*). IEEE. 2021. P. 111–114. https://doi.org/10.1109/TELE52840.2021.9482762
19. Schleimer S., Wilkerson D.S., Aiken A. Winnowing: local algorithms for document fingerprinting. In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data* (*SIGMOD'03*). 2003. P. 76–85. https://doi.org/10.1145/872757.872770
20. Rogers M., et al. Exploring Personalization of gamification in an Introductory programming course. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (*SIGCSE'21*). 2021. P. 1121–1127. https://doi.org/10.1145/3408877.3432402
21. Putnam V., Conati C. Exploring the need for explainable artificial intelligence (XAI) in intelligent tutoring systems (ITS). *IUI Workshops*. 2019. V. 19. Available from URL: https://explainablesystems.comp.nus.edu.sg/2019/wp-content/uploads/2019/02/IUI19WS-ExSS2019-19.pdf
22. Shcherbina O.A. Constraint satisfaction and constraint programming. *Intellektual'nye sistemy = Intelligent Systems*. 2011;15(1–4):53–170 (in Russ.).
23. Wang J., Chen B., Wei L., Liu Y. Skyfire: Data-driven seed generation for fuzzing. In: *2017 IEEE Symposium on Security and Privacy* (*SP*). IEEE. 2017. P. 579–594. https://doi.org/10.1109/SP.2017.23
24. Papadakis M., et al. Mutation testing advances: an analysis and survey. *Adv. Comput*. 2019;112:275–378. https://doi.org/10.1016/bs.adcom.2018.03.015

13. Guo P.J. Online python tutor: embeddable web-based program visualization for cs education. In: *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*. 2013. P. 579–584. https://doi.org/10.1145/2445196.2445368
14. Miller H., Willcox K., Huang L. Crosslinks: Improving course connectivity using online open educational resources. *The Bridge*. 2016;43(3):38–45. URL: http://hdl.handle.net/1721.1/117022
15. Utterberg M.M., et al. Intelligent tutoring systems: Why teachers abandoned a technology aimed at automating teaching processes. In: *Proceedings of the 54th Hawaii International Conference on System Sciences*. 2021. P. 1538. URL: http://hdl.handle.net/10125/70798
16. Sherman M., et al. Impact of auto-grading on an introductory computing course. *J. Comput. Sci. Coll.* 2013;28(6):69–75.
17. Rivers K., Koedinger K.R. Data-driven hint generation in vast solution spaces: a self-improving python programming tutor. *Int. J. Artif. Intell. Educ*. 2017;27(1):37–64. https://doi.org/10.1007/s40593-015-0070-z
18. Sovietov P. Automatic generation of programming exercises. In: *2021 1st International Conference on Technology Enhanced Learning in Higher Education* (*TELE*). IEEE. 2021. P. 111–114. https://doi.org/10.1109/TELE52840.2021.9482762
19. Schleimer S., Wilkerson D.S., Aiken A. Winnowing: local algorithms for document fingerprinting. In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data* (*SIGMOD'03*). 2003. P. 76–85. https://doi.org/10.1145/872757.872770
20. Rogers M., et al. Exploring personalization of gamification in an introductory programming course. In: *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (*SIGCSE'21*). 2021. P. 1121–1127. https://doi.org/10.1145/3408877.3432402
21. Putnam V., Conati C. Exploring the need for explainable artificial intelligence (XAI) in intelligent tutoring systems (ITS). *IUI Workshops*. 2019. V. 19. URL: https://explainablesystems.comp.nus.edu.sg/2019/wp-content/uploads/2019/02/IUI19WS-ExSS2019-19.pdf
22. Щербина О.А. Удовлетворение ограничений и программирование в ограничениях. *Интеллектуальные системы*. 2011;15(1–4):53–170.
23. Wang J., Chen B., Wei L., Liu Y. Skyfire: Data-driven seed generation for fuzzing. In: *2017 IEEE Symposium on Security and Privacy* (*SP*). IEEE. 2017. P. 579–594. https://doi.org/10.1109/SP.2017.23
24. Papadakis M., et al. Mutation testing advances: an analysis and survey. *Adv. Comput*. 2019;112:275–378. https://doi.org/10.1016/bs.adcom.2018.03.015
25. Hutton G., Meijer E. *Monadic Parser Combinators*. Technical Report NOTTCS-TR-96-4. Department of Computer Science, University of Nottingham. 1996. 38 p. URL: https://www.cs.nott.ac.uk/~pszgmh/monparsing.pdf
26. Phothilimthana P.M., Sridhara S. High-coverage hint generation for massive courses: Do automated hints help CS1 students? In: *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (*ITiCSE'17*). 2017. P. 182–187. https://doi.org/10.1145/3059009.3059058

25. Hutton G., Meijer E. *Monadic Parser Combinators*. Technical Report NOTTCS-TR-96-4. Department of Computer Science, University of Nottingham. 1996. 38 p. Available from URL: https://www.cs.nott.ac.uk/~pszgmh/monparsing.pdf
26. Phothilimthana P.M., Sridhara S. High-coverage hint generation for massive courses: Do automated hints help CS1 students? In: *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (*ITiCSE'17*). 2017. P. 182–187. https://doi.org/10.1145/3059009.3059058

About the authors

Elena G. Andrianova, Cand. Sci. (Eng.), Associated Professor, Head of the Department of Corporate Information Systems, Institute of Information Technologies, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: andrianova@mirea.ru. Scopus Author ID 57200555430, ResearcherID T-7908-2018, SPIN-code RSCI 9858-3229, http://orcid.org/0000-0001-6418-6797

Liliya A. Demidova, Dr. Sci. (Eng.), Professor, Professor of the Department of Corporate Information Systems, Institute of Information Technologies, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: demidova@mirea.ru. Scopus Author ID 56406258800, ResearcherID R-6077-2016, SPIN-code RSCI 9447-3568, http://orcid.org/0000-0003-4516-3746

Petr N. Sovetov, Cand. Sci. (Eng.), Associated Professor, Department of Corporate Information Systems, Institute of Information Technologies, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: sovetov@mirea.ru. Scopus Author ID 57221375427, SPIN-code RSCI 9999-1460, http://orcid.org/0000-0002-1039-2429

Об авторах

Андрианова Елена Гельевна, к.т.н., доцент, заведующий кафедрой корпоративных информационных систем Института информационных технологий ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: andrianova@mirea.ru. Scopus Author ID 57200555430, ResearcherID T-7908-2018, SPIN-код РИНЦ 9858-3229, http://orcid.org/0000-0001-6418-6797

Демидова Лилия Анатольевна, д.т.н., профессор, профессор кафедры корпоративных информационных систем Института информационных технологий ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: demidova@mirea.ru. Scopus Author ID 56406258800, ResearcherID R-6077-2016, SPIN-код РИНЦ 9447-3568, http://orcid.org/0000-0003-4516-3746

Советов Петр Николаевич, к.т.н., доцент кафедры корпоративных информационных систем Института информационных технологий, ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: sovetov@mirea.ru. Scopus Author ID 57221375427, SPIN-код РИНЦ 9999-1460, http://orcid.org/0000-0002-1039-2429

Translated by V. Glyanchenko
Edited for English language and spelling by Thomas Beavitt