УДК 004.42(075.8)
DOI: 10.25559/SITITO.019.202303.670-675

Original article

One More Step to Automate a Software Development

Yu. A. Semenov
National Research center "Kurchatov Institute for Theoretical and Experimental physics", Moscow, Russian Federation
Address: 1 Akademicianika Kurchatova Sq., Moscow 123182, Russian Federation
semenov@itep.ru

Abstract
It is proposed to extend the concept of OOP to the practically unlimited sphere. The object may be any entity: channel, file, network, electronic device, etc. For every object should be set some list of possible operations and parameters. That was appeared to be convenient for control code developments. This concept was used to simplify software creation to control and monitor the state of big systems with many components of different nature. The thematic libraries of programs will improve the system efficiency considerably and lessen software error number. We may expect that in the future, routines will be developed by specialists who are not programmers at all.

Keywords: AI, AI power, software development automation, task ranking, software errors

Conflict of interests: the author declares no conflict of interests.

For citation: Semenov Yu.A. One More Step to Automate a Software Development. *Modern Information Technologies and IT-Education.* 2023;19(3):670-675. https://doi.org/10.25559/SITITO.019.202303.670-675

Еще один шаг к автоматизации разработки программного обеспечения

Ю. А. Семенов
ФГБУ «Национальный исследовательский центр "Курчатовский институт"», г. Москва, Российская Федерация
Адрес: 123182, Российская Федерация, г. Москва, пл. Академика Курчатова, д. 1
semenov@itep.ru

Аннотация
Предлагается распространить понятие объектно-ориентированного программирования на практически неограниченную сферу. Объектом может быть любой объект: канал, файл, сеть, электронное устройство и т.д. Для каждого объекта должен быть задан некоторый список возможных операций и параметров. Это оказалось удобным для разработки управляющего кода. Эта концепция была использована для упрощения создания программного обеспечения для управления и мониторинга состояния больших систем со множеством компонентов различной природы. Тематические библиотеки программ значительно повысят эффективность работы системы и уменьшат количество программных ошибок. Можно ожидать, что в будущем подпрограммы будут разрабатываться специалистами, которые вообще не являются программистами.

Ключевые слова: искусственный интеллект, мощность искусственного интеллекта, автоматизация разработки программного обеспечения, ранжирование задач, ошибки программного обеспечения

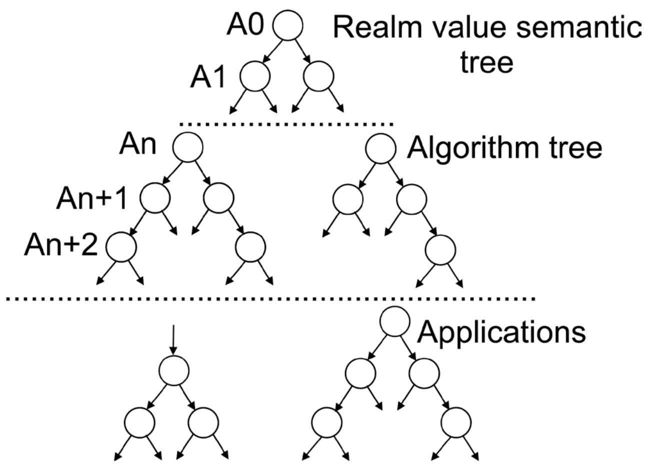
Конфликт интересов: автор заявляет об отсутствии конфликта интересов.

Для цитирования: Семенов Ю. А. Еще один шаг к автоматизации разработки программного обеспечения // Современные информационные технологии и ИТ-образование. 2023. Т. 19, № 3. С. 670-675. https://doi.org/10.25559/SITITO.019.202303.670-675

Introduction

Up to now there are some thousands programming languages from them only about 250 are in active use. The number of areas investigation and technologies is counted by millions. The value of system variable realm fixes this area. The realm value determines a semantic tree, which is a part of the problem tree (see fig. 1). At the output nodes of the tree start new branches of semantic trees of algorithms. “Lower” nodes of algorithm tree determine an algorithmic language of the routine.

Speaking of artificial intelligence, we usually mean neural networks. Neural networks after learning are adopted to recognize some patterns. The patterns may be graphics, multimedia or set of parameters, characterizing system state. But pattern recognition can be realized and by means of statistical analyses[1] [1-4]. For example, at the remote user recognition by his style of work on a keyboard.



F i g. 1. The semantic tree for network security

Source: Hereinafter in this article all figures were drawn up by the author.

The realrn value determines an algorithm bank and primitive library. For different realm values the same names of programs or primitives may have different work algorithms. Description of the problem may contain names of software modules from the bank, primitive or macro names, and fragments of the Perl-routine, written by user and included into description. Macros are corresponded to the output language, in our case —Perl. As problem description is usually shorter than algorithm description, this method can lessen an error number.

Any node is characterized with level index n and the number of node in the row of semantic tree — j. The nearest nodes have numbers n-1, n+1, and also j-1, j+1. The maximum number of node-neighbours at this formalism is equal 8. On practice some nodes do not contain programs or even problem description. If from the node go many verges, the number of neighbours may exceed 8.

There is also other method to detect nearest neighbours. For example, selection of nodes, situated not more than two verges (nodes) away.

We start from the node corresponding to the new problem. From any node can come any number of verges. A particular semantic tree can correspond to problem or solution description or program which realize an algorithm.

Any node of the graph is characterized with a set of parameters, e. g., number of steps from the tree root or with a list of parent nodes. We have to develop a mechanism for quantitative estimation of known algorithms and those which were created in the process of solution searching. Then after investigation of the most similar algorithms, user may attempt to build a new one. As an estimate of nearness (besides a number of nodes from the root) one may use the number of verges between nodes. From the set of neighbour nodes we select those, which correspond to algorithm descriptions. The main problem to be resolved, is an automatic code generation on the base of task description (task description conversion into algorithm description). In many cases such a procedure may be quite complicated.

Let us try to build semantic tree for a network security realm. Not every graph node corresponds to some algorithm or problem description.

Fragments of semantic tree for network security

In brackets to the right are shown digital identifiers of semantic tree nodes.

1. *Insiders — staff members, LAN-computers (L1.1):*
   a. *MAC-flood (L1.1.1)*
   b. *Illegal file copy (e. g., passwd. L 1.1.2))*
   c. *Illegal DB-bank copy by user (client, staff, depositary etc; L1.1.3.)*
d. *Improper system configuration (L1.1.4)*

2. Outsiders — object or subject outside a LAN (L1.2)
   2a. Attacks of user computer (L1.2.1)
      a. *DoS/DDoS (L1.2.1.1)*
      b. *Buffer overflow attack (L1.2.1.2)*
      c. *Man-in-the-middle (L1.2.1.3)*
      d. *Path-traversal (L1.2.1.4)*
      e. Network worms *(L1.2.1.5)*
      f. Viruses *(L1.2.1.6)*
      g. *Pass guessing (L1.2.1.7)*
      h. *SQL-injection (L1.2.1.8)*
      i. *Rootkit (L1.2.1.9)*
      j. *Poisoned cookies (L1.2.1.10)*
      k. *USB attacks (L1.2.1.11)*
      l. *Sender address spoofing (L1.2.1.12)*
      m. *Hacker’s plugins (L1.2.1.13)*
      n. *File name spoofing (L1.2.1.14)*
      o. *Drive by download (L1.2.1.15)*
      p. *Attacks via equipment (processors, controllers, etc.) (L1.2.1.16. e.g. meltdown)*
      q. *Illegal HTTP-methods (L 1.2.1.17)*
      r. *Vulnerable file requests (L1.2.1.18)*
   2b. USER attack (L1.2.2)
      a. *Phishing — skimming — vishing (L1.2.2.1)*
      b. *SPAM (L1.2.2.2)*

[1] Kurzweil R. How to Create a Mind: The Secret of Human Thought Revealed. Penguin Books; 2013. 352 p.; Penrose R., Gardner M. The Emperor’s New Mind: Concerning Computers, Minds and The Laws of Physics. Oxford University Press; 2002. 640 p.

c. *ARP-spoofing (L1.2.2.3)*
d. *Man-in-the-middle (L1.2.2.4)*
e. *Attacks via protocols TCP, RPC, P2P, etc (L1.2.2.5)*
f. *Back door in media files (L1.2.2.6)*
g. *Cross Site Scripting (L1.2.2.7)*

2c. Attack of LAN (L1.2.3)
a. *DoS/DDoS (L1.2.3.1)*
b. *ARP-spoofing (L1.2.3.2)*
c. *USB attacks (L1.2.3.3)*
d. *DNS forgery (L1.2.3.4)*
e. *Reading out data from computer not connected to network (L1.2.3.5)*
f. *Attacks via proxy (L1.2.3.6)*
g. *Attempt to rise user rights (L1.2.3.7)*
h. *Attacks via insider (L1.2.3.8)*

2d. Attacks of OS (L1.2.4)
a. *APT attacks (L1.2.4.1)*
b. *Zero day attacks (L1.2.4.2)*
c. *Remote File Injection (L1.2.4.3)*
d. *Attack via RPC (L1.2.4.4)*

At my computer — "saturn.itep.ru" have been detected events of the following types: L1.2.1.1 (DOS), L1.2.1.2 (buffer overflow), L1.2.1.4 (*Path-traversal*), L1.2.1.7 (*pass guessing*), L1.2.1.17 (illegal HTTP method), L1.2.1.18 (illegal file), L1.2.2.2 (SPAM).

For todays the most dangerous are attacks L1.2.1.2 (Buffer overflow — attempt to inject code to the operative memory of computer-target).

In some cases, rather difficult to separate DoS-attack versions L1.2.1.1 and L1.2.3.1, and also L1.2.2.1 and L1.2.2.2 (attacks via SPAM, where to be precise we have to analyze a message text).

At the saturn there are routines detecting practically all mentioned types of attacks (excepting cases of OS *L1.2.4*). For some types of attacks we have developed facility to block them.

When we come to the node, corresponding to the problem, which we should describe and solve, one need to realize, is the problem clear understood, and can we describe a solution algorithm. We have to attempt to describe the problem on PDL (Problem Description Language) [5-8].

To simplify a routine building we may use a library of primitives or program modules. A primitive is a piece of soft, carrying out some procedure, e.g., summing an array of numbers.

The history of programming is characterized with an increase of building objects size. With a help of modern neural network we can create a routine, containing several lines of code, but, using primitives, one can get code, solving some application task. It can be a step to a full automatization of software development.

If there is no proper primitive, a program module may be written by the programmer himself.

When we speak of problem description it is related to the algorithm requirement and may contain its fragments. That is why the realm value must be fixed before problem description.

A problem description is usually a client task (sometimes a programmer himself). For an algorithm description is used one of the known algorithmic languages. The number of algorithmic languages shows a plurality of used algorithms. Any translator converts an algorithm description into executable processor code.

To develop a soft for a known algorithm there is no necessity to look for a programmer of high level.

Up to now there is no computers with intellect at the level of university graduate that is why we have to apply the following procedure. At first, we try to find a ready solution in the bank of algorithms. Unfortunately, in our bank there are not so many software modules. If the found module permit modification, it is carried out with the help of input data. Program module in the case must contain code fragments, which activates with modifier from the input parameter list. It is clear that interpretive languages are preferable, as they always provide source code texts[2] [9-12]. At the beginning of 2023 according to the TIOBE-rating the list of 10 the most usable languages looks like: Python (14,16 %), C (11,77 %), C++ (10,36 %), Java (8,35 %), C# (7,65 %), JavaScript, PHP, Visual Basic, SQL, Ассемблер (1,35 %). In the last 10 years several new languages have been appeared, including go[3].

If there is no ready to use solution in the bank, problem description is interpreted as an algorithm written on meta language PDL.

That is why problem description at PDL has no commutative property (permutations of words, lines etc are not allowed).

Primitive and macro names as well as their characteristics have to be known to the user, who writes the routine. That is why PDL is an algorithmic macro language.

The similar model has been used in[4] [13-15].

Let us classify the possible problems according to their complexity.

Class 1 corresponds to written and debugged routines, where may be included real time modifications. A user can change input parameters and modifiers.

Modifiers help to update the routine, but these modifications must be programmed in advance.

Class 2 suppose, that routine may be modified by user with a help of introduction of primitives or any other plug-in.

Class 3 corresponds to a case, when an algorithm must be determined by user (chooses one from the proposed list). The program may be written by himself, using existing libraries. The solution search may be done with looking through known algorithms or by program modification of found solutions. To make programming more effective and flexible the library must be rather big. At first, one need to determine realm, that permit to select the primitive library and algorithm bank. To automate the process, we may use standard macro-fragments in Perl (<perl>perl-code</perl>) - these are not primitives, as they doesn't solve any problems, but simplify the software development. These macros may have names and form libraries. Macros are used to solve problems of classes 2 and 3.

Class 4 is a case when problem is described, but solution algorithm is unknown. The modern AI-systems can't solve such problems.

[2] Semenov Yu.A. *Dinamika ispol'zovaniya raznyh yazykov programmirovaniya za poslednee desyatiletie* [Ratings of programming languages]. In: Semenov Yu.A. *Protokoly Internet dlja jelektronnoj torgovli* [Internet Protocols for Electronic Commerce]. Moscow: Gorjachaja linija – Telekom, 2003. Available at: http://book.itep.ru/10/languges.htm (accessed 19.06.2023). (In Russ.); TIOBE Index [Electronic resource] // TIOBE Software BV, 2023. Available at: https://www.tiobe.com/tiobe-index/ (accessed 19.06.2023).

[3] The Go Programming Language [Electronic resource] // Google, 2023. Available at: https://go.dev/ (accessed 19.06.2023).

[4] Wolfram: Computation Meets Knowledge [Electronic resource] // Wolfram Research, 2023. Available at: https://www.wolfram.com (accessed 19.06.2023).

Even the man, who have found a solution, often has no idea of the way he has come to this result. There are also problems, which have no solution at all.

The complexity of class 4 tasks may vary in a very wide range. Naturally, a man, who can solve such a problem, must have rather high level of education and skills. But these knowledge and skills are necessary but not sufficient to reach a success. One of the examples of class 4 task is a determination of software error number in a code.

If problem may be divided and converted into several subproblem, each of which may be solved modifying one of the known algorithms, then the problem we may classify as class 3. At the problem division, each of parts will be considered of the same class, or lower. At the big number of subdivision, the complexity may be concluded in these divisions interaction. The part interaction problem may be even more hard than of any of the parts.

There is a possibility to create routines for automatic code generation for classes 2-3. For solving tasks of class 4 we need artificial intelligence higher than any postgraduate has. Existing neural solutions will not fit. See AI-considerations in[5] [16, 17].

To solve the problem, we need to select rules for description of objects: graphical, multimedia (MPEG-2-7), hardware, network, software, textual etc. If in OOP languages objects are software modules, here object may be also channel, file, multimedia entity, data base, a set of status parameters, piece of hardware equipment, text fragment and so on. For every object we have to fix a set of operations, which can be used and form program operators, realizing the function. These operators are being described and included in PDL. It will simplify programming and represents a step to a higher level language. The programming automatization is one of the most actual directions of developments [18-22].

In frame of this technology we may build e.g. a distributed backup system, in this case copies of all elements of virtual memory are created. This is a VLAN with build in backup option. The file names should have a prefix, characterizing the file origin. It may be a last four digits of IP-address. This is a form of internal cloud and need some security measures [23-25].

These technics may be useful to secure systems containing IoT-elements. It will provide a possibility to detect intrusion attempts and restore soft of IoT after attack or software deterioration.

In object description are set upper and lower limits for input parameters (maximum request length, temperature, load value, number of acceptable pass guessing attempts, administrator e-mail, legal IP, DB-pass.

The list of possible objects

a. Data base and tables
b. Files access_log and error_log (journal files, for example, secure, IDS or PDS)
c. Arrays of numbers or symbol strings
d. Flows of messages, bytes or packets
e. Measurement results in form of lists and files
f. Text files
g. Graphical objects (pictures, multimedia)
h. Queues
i. Operating and virtual memory
j. Hardware (processor, switch, router, firewall, ...)
k. Network packet
l. Object interaction protocol
m. Personal data including genome, photo and so on.

The object description format

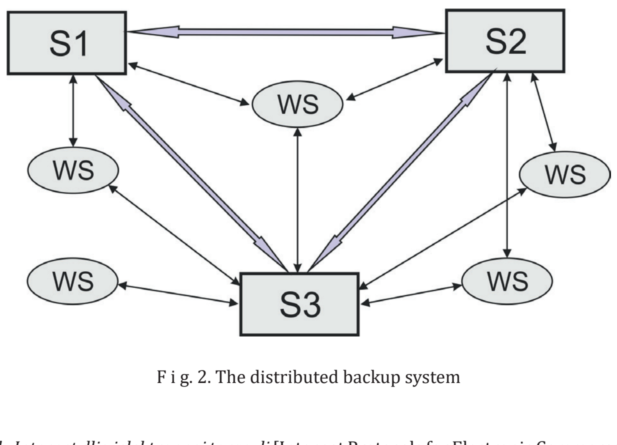
1. The object name
2. The class (digital, textual, program, hardware, network...)
3. Parameters (including position, size, creation time, defence level (cryptodefence, control summing, certificates)
4. Possible operations under object
5. Owner name, address
6. User access rights
7. Range of possible parameter values for the object
8. Possibility to modify or copy object for other users
9. Backup copy frequency
10. Protocols to inform of some events (via e-mail, SMS etc.)

Let us consider an example of problem of class 2-3. It can be a case, when one need to put data into DB table. At first, we have to create a DB itself. Data base and table have to be described as objects. To solve the problem, we need to add some new words in the operator and attribute lists of PDL. In the program we have to anticipate a control of input parameter correctness in subroutine or introduced at dialog.

At decreasing of memory prices there is a possibility to build a distributed backup system, where any file may be stored in several copies. Such a system permits to work more effectively with diskless computers, in particular with IoT-objects, increasing their security.

All data exchanges must be cryptographically defended. An access to computers, where backup files are stored, is secured by the multiparameter authentication.

The total server numbers may vary in a wide range, on fig. 2 there are 5 working stations (WS), that may exchange files with servers and each other. Files can be stored at servers and WS, if its IP is in a permitted object list. Objects of type S and WS should be described in advance and all necessary primitives were included in corresponding library. As PDL interpreter has modular structure, there was no problem to adopt it for this modification.



F i g. 2. The distributed backup system

[5] Semenov Yu.A. *Iskusstvennyj intellekt* [Artificial intelligence]. In: Semenov Yu.A. *Protokoly Internet dlja jelektronnoj torgovli* [Internet Protocols for Electronic Commerce]. Moscow: Gorjachaja linija – Telekom, 2003. Available at: http://book.itep.ru/4/7/ii.htm (accessed 19.06.2023). (In Russ.); Kaku M. The Future of the Mind, The Scientific Quest to Understand, Enhance, and Empower the Mind. New York: Doubleday Publ.; 2014. 400 p.; Kotler S. Tomorrowland. Our Journey from Science Fiction to Science Fact. New Harvest: Uncorrected Proof edition; 2015. 304 p.; Watson R. Digital vs Human: how we'll live, love, and think in the future. Scribe US Publ., 2016, 288 p.

Conclusion

To find a solution for some new task of class 4 algorithmically is quite unrealistic. Simple neural networks also do not suit. One should try a network of neural networks, which have interaction protocols with mechanisms of self organization and optimization. These protocols may be built into new AI-chips[6]. There are no such protocols up to now, and they should be developed in the nearest future.

References

[1] Ahmed Z., Kinjol F.J., Ananya I.J. Comparative Analysis of Six Programming Languages Based on Readability, Writability, and Reliability. In: 2021 24th International Conference on Computer and Information Technology (ICCIT). Dhaka, Bangladesh: IEEE Computer Society; 2021. p. 1-6. https:doi.org/10.1109/ICCIT54785.2021.9689813

[2] Farooq M.Sh, Khan S.A., Ahmad F., Islam S., Abid A. An Evaluation Framework and Comparative Analysis of the Widely Used First Programming Languages. *PLOS ONE.* 2014;9(2):e88941. https://doi.org/10.1371/journal.pone.0088941

[3] Khurana D., Koli A., Khatter K., Singh S. Natural language processing: state of the art, current trends and challenges. *Multimedia Tools and Applications*. 2023;82(3):3713-3744. https://doi.org/10.1007/s11042-022-13428-4

[4] Lloyd S. Ultimate Physical Limits to Computation. *Nature.* 2000;406:1047-1054. https://doi.org/10.1038/35023282

[5] Dorenskaya E.A., Kulikovskaya A.A., Semenov Y.A. *Yazyk opisaniya problemy i issledovanie ego vozmozhnostej* [Exploring Possibilities of Language for Describing the Problem]. *Modern Information Technologies and IT-Education*. 2020;16(3):653-663. (In Russ., abstract in Eng.) https:doi.org/10.25559/SITITO.16.202003.653-663

[6] Casado M., Foster N., Guha A. Abstractions for software-defined networks. *Communications of the ACM.* 2014;57(10):86-95. https://doi.org/10.1145/2661061.2661063

[7] Hu Y., Zou D., Peng J., Wu Y., Shan J., Jin H. TreeCen: Building Tree Graph for Scalable Semantic Code Clone Detection. In: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22). New York, NY, USA: Association for Computing Machinery; 2023. Article number: 109. 12 p. https://doi.org/10.1145/3551349.3556927

[8] Gong Y., Huang W., Wang W., Lei Y. A survey on software defined networking and its applications. *Frontiers of Computer Science*. 2015;9:827-845. https://doi.org/10.1007/s11704-015-3448-z

[9] Kulikovskya A.A., Dorenskaya E.A., Semenov Yu.A. *Kolichestvennye harakteristiki bezopasnosti programm* [Quantitative Security Characteristics of Perl Programs]. *Modern Information Technologies and IT-Education*. 2022;18(4):855-860. (In Russ., abstract in Eng.) https://doi.org/10.25559/SITITO.18.202204.855-860

[10] Dorenskaya E.A., Semenov Yu.A. *Metod opredeleniya kontekstnyh znachenij slov i dokumentov* [The determination method for contextual meanings of words and documents]. *Modern Information Technologies and IT-Education*. 2018;14(4):896-902. (In Russ., abstract in Eng.) https://doi.org/10.25559/SITITO.14.201804.896-902

[11] Kádár I. The optimization of a symbolic execution engine for detecting runtime errors. *Acta Cybernetica.* 2017;23(2):573-597. https://doi.org/10.14232/actacyb.23.2.2017.9

[12] Dorenskaya E.A., Semenov Yu.A. O *tehnologii programmirovaniya, orientirovannoj na minimizaciyu oshibok* [About the programming techniques, oriented to minimize errors]. *Modern Information Technologies and IT-Education*. 2017;13(2):50-56. (In Russ., abstract in Eng.) https://doi.org/10.25559/SITITO.2017.2.226

[13] Nefdt R.M. Scientific modelling in generative grammar and the dynamic turn in syntax. *Linguistics and Philosophy*. 2016;39(5):357-394. https://doi.org/10.1007/s10988-016-9193-4

[14] Barba-Guaman L.R. et al. Using wolfram software to improve reading comprehension in mathematics for software engineering students. In: 2018 13th Iberian Conference on Information Systems and Technologies (CISTI). Caceres, Spain: IEEE Computer Society; 2018. p. 1-4. https://doi.org/10.23919/CISTI.2018.8399388

[15] M egan CrouseNneji G.U., Deng J., Shakher S.S, Monday H.N., Agomuo D., Ukwuoma C.C. A Multimedia Computer Aided Learning Software. In: 2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON). Vancouver, BC, Canada: IEEE Computer Society; 2018. p. 807-813. https://doi.org/10.1109/IEMCON.2018.8614770

[16] Ali S. et al. Explainable Artificial Intelligence (XAI): What we know and what is left to attain Trustworthy Artificial Intelligence. *Information Fusion*. 2023;99:101805. https://doi.org/10.1016/j.inffus.2023.101805

[17] Alkhalifa R., Kochkina E., Zubiaga A. Building for tomorrow: Assessing the temporal persistence of text classifiers. *Information Processing & Management*. 2023;60(2):103200. https://doi.org/10.1016/j.ipm.2022.103200

[18] MacGregor R., Trinder P., Loidl H.-W. Improving GHC Haskell NUMA profiling. In: Proceedings of the 9th ACM SIGPLAN International Workshop on Functional High-Performance and Numerical Computing (FHPNC 2021). New York, NY, USA: Association for Computing Machinery; 2021. p. 1-12. https://doi.org/10.1145/3471873.3472974

[19] Shapiro J. Programming language challenges in systems codes: why systems programmers still use C, and what to do about it. In: Proceedings of the 3rd workshop on Programming languages and operating systems: linguistic support for modern operating systems (PLOS '06). New York, NY, USA: Association for Computing Machinery; 2006. 9 p. https://doi.org/10.1145/1215995.1216004

[6] AI Chips 2023-2033 [Electronic resource] // IDTechEx, 2023. Available at: https://www.idtechex.com/en/research-report/ai-chips-2023-2033/937 (accessed 19.06.2023).

[20] Guo Y., Chen Z., Chen L., Xu W., Li Y., Zhou Y., Xu B. Generating Python Type Annotations from Type Inference: How Far Are We? *ACM Transactions on Software Engineering and Methodology*. 2024;33(5):123. https://doi.org/10.1145/3652153

[21] Ziegler A., Kalliamvakou E., Li X. A., Rice A., Rifkin D. Measuring GitHub Copilot's Impact on Productivity. *Communications of the ACM*. 2024;67(3):54-63. https://doi.org/10.1145/3633453

[22] Silva T.C., Boos C.F., Junkes-Cunha M., Azevedo F.M. Automatization of a protocol for the postural assessment of patients with Chronic Obstructive Pulmonary Disease. In: AFRICON 2015. Addis Ababa, Ethiopia: IEEE Computer Society; 2015. p. 1-5. https://doi.org/10.1109/AFRCON.2015.7331918

[23] Sonnekalb T., Heinze T.S., Mäder P. Deep security analysis of program code. *Empirical Software Engineering*. 2022;27(1):2. https://doi.org/10.1007/s10664-021-10029-x

[24] Villalón-Fonseca R. The nature of security: A conceptual framework for integral-comprehensive modeling of IT security and cybersecurity. *Computers & Security*. 2022;120:102805. https://doi.org/10.1016/j.cose.2022.102805

[25] Bouke M.A. Software Development Security. In: CISSP Exam Certification Companion. *Certification Study Companion Series*. Berkeley, CA: Apress; 2023. p. 645-725. https://doi.org/10.1007/979-8-8688-0057-3_10

Submitted 19.06.2023; approved after reviewing 21.08.2023; accepted for publication 17.09.2023.
Поступила 19.06.2023; одобрена после рецензирования 21.08.2023; принята к публикации 17.09.2023.

About the author:

Yuri A. Semenov, Lead Researcher of the Institute for Theoretical and Experimental Physics named by A. I. Alikhanov of National Research center "Kurchatov Institute for Theoretical and Experimental physics" (1 Akademicianika Kurchatova Sq., Moscow 123182, Russian Federation), Cand. Sci. (Phys.-Math.), **ORCID: https://orcid.org/0000-0002-3855-3650**, semenov@itep.ru

The author has read and approved the final manuscript.

Об авторе:

Семенов Юрий Алексеевич, ведущий научный сотрудник Института теоретической и экспериментальной физики имени А. И. Алиханова, ФГБУ «Национальный исследовательский центр "Курчатовский институт"» (123182, Российская Федерация, г. Москва, пл. Академика Курчатова, д. 1), кандидат физико-математических наук, **ORCID: https://orcid.org/0000-0002-3855-3650**, semenov@itep.ru

Автор прочитал и одобрил окончательный вариант рукописи.