

TECHNOLOGY FOR DEVELOPING PROGRAMMING-RELATED PROFESSIONAL COMPETENCIES OF FUTURE COMPUTER SCIENCE TEACHERS

Rakhmatov Mirzo Mukimovich

Independent Researcher at Bukhara State Pedagogical Institute

Abstract. This article examines the theoretical and methodological foundations of developing programming-related professional competencies among future computer science teachers. It discusses the effective integration of modern programming languages, digital educational resources, interactive teaching methods, problem-based learning, and project-based activities into professional training.

Keywords: future computer science teachers, professional competence, programming competence, programming languages, teaching technology, algorithmic thinking, problem-based learning, project-based learning, digital educational resources, interactive methods, professional training.

Introduction. The rapid development of digital technologies has significantly transformed contemporary education and increased the demand for professionally competent computer science teachers. Today, a computer science teacher is expected not only to possess theoretical knowledge of information technologies but also to demonstrate strong programming skills, algorithmic thinking, digital literacy, problem-solving abilities, and the capacity to organize technology-enhanced learning. Therefore, the development of programming-related professional competencies among future computer science teachers has become one of the priority areas of higher pedagogical education. Programming is an essential component of computer science education because it enables students to understand how digital systems operate, create software products, model various processes, and solve practical problems through computational methods. Learning programming languages also develops logical reasoning, analytical thinking, creativity, accuracy, persistence, and the ability to divide complex problems into manageable stages. For future computer science teachers, these abilities are especially important because they must be able to explain programming concepts clearly, demonstrate algorithms, identify students' difficulties, evaluate program code, and select appropriate instructional methods.

Traditional approaches based mainly on lectures, theoretical explanations, and the reproduction of ready-made code do not always provide sufficient opportunities for developing professional competencies. Students may learn the syntax of a programming language without being able to independently design algorithms, analyse errors, create educational software, or teach programming effectively. Consequently, programming instruction in teacher education should be organised through a

competency-based, practice-oriented, interactive, and professionally directed approach. Programming-related professional competence is an integrated personal and professional quality that includes knowledge of programming concepts, practical coding skills, methodological readiness, independent problem-solving abilities, and responsible use of digital technologies. It requires future teachers to combine subject knowledge with pedagogical, methodological, communicative, and reflective skills. A professionally competent computer science teacher should be capable of selecting a suitable programming language, designing differentiated tasks, managing individual and collaborative activities, using digital learning platforms, and objectively assessing students' achievements.

The programming-related professional competence of a future computer science teacher has a complex structure. It cannot be limited to the ability to write syntactically correct program code. It includes several interconnected components: motivational, cognitive, practical, methodological, communicative, and reflective-evaluative. The motivational component reflects students' interest in programming, their willingness to learn new technologies, and their understanding of the professional importance of programming. A positive attitude towards the subject encourages future teachers to work independently, overcome difficulties, experiment with different solutions, and continuously improve their skills. Motivation can be strengthened through meaningful tasks, successful practical experiences, professional examples, competitions, and projects connected with real educational needs.

The cognitive component includes knowledge of algorithms, data structures, programming paradigms, software development principles, and the syntax and semantics of programming languages. Future teachers should understand variables, data types, operators, conditions, loops, functions, arrays, files, classes, objects, recursion, and exception handling. They should also know how programs are designed, tested, documented, and improved. The practical component involves applying theoretical knowledge to develop working programs. It includes algorithm design, code writing, debugging, testing, optimisation, and software documentation. Students should be able to use programming environments, version-control tools, online coding platforms, and automated testing systems. Practical competence is demonstrated when a student independently solves a problem and explains the logic of the proposed solution.

The methodological component concerns the ability to teach programming effectively. Future teachers must know how to present abstract concepts using understandable examples, visual models, flowcharts, pseudocode, and step-by-step demonstrations. They should be able to formulate learning objectives, prepare lesson plans, select

teaching methods, design tasks of varying complexity, and assess learners' programming achievements. The communicative component includes the ability to explain algorithms and program code clearly, cooperate with others, present project results, and provide constructive feedback. Programming is often perceived as an individual activity; however, professional software development and modern education require effective teamwork and communication. Pair programming, collaborative projects, peer assessment, and group discussions are therefore important elements of teacher training.

The primary objective of the educational technology is to ensure the gradual development of programming knowledge, practical coding abilities, pedagogical skills, and professional independence. Its implementation should be guided by the principles of competency orientation, continuity, professional relevance, activity-based learning, differentiation, integration, collaboration, and reflection. The competency-based principle focuses on students' ability to apply knowledge in practical and pedagogical situations. The principle of continuity requires programming skills to be developed systematically, beginning with basic algorithms and gradually progressing towards complex applications and educational projects. Professional relevance means that programming assignments should reflect the future activities of computer science teachers.

The activity-based principle involves learning through practical action. Students acquire programming competencies more effectively when they design algorithms, write code, test solutions, and correct errors independently. The differentiation principle requires tasks to be adapted to students' abilities and prior experience. Basic tasks can consolidate essential concepts, whereas advanced assignments may involve complex algorithms, software architecture, or interdisciplinary projects.

The proposed technology can be implemented through four interrelated stages: diagnostic-motivational, theoretical-orientational, practical-creative, and reflective-evaluative. At the diagnostic-motivational stage, students' initial knowledge, practical abilities, interests, and learning needs are identified. Diagnostic tests, questionnaires, interviews, and short programming tasks can be used for this purpose. The results help the teacher select appropriate content and determine task difficulty. At this stage, students should also become familiar with the professional importance of programming and the expected learning outcomes.

The theoretical-orientational stage develops students' fundamental understanding of algorithms and programming concepts. Explanations should be supported by practical examples, visualisations, flowcharts, pseudocode, and code demonstrations. Instead of memorising individual commands, students should understand the relationship

between a problem, its mathematical or logical model, the algorithm, and its software implementation. At the practical-creative stage, students solve programming problems, complete laboratory work, participate in collaborative activities, and develop individual or group projects. Tasks should gradually increase in complexity. Initially, students may work with simple calculations, conditional statements, and loops. Later, they can create modular programs, work with data structures, process files, apply object-oriented programming, and develop educational applications.

The selection of programming languages should correspond to educational objectives, students' level of preparation, and future professional needs. Languages with clear syntax and extensive educational resources are especially suitable for introducing programming. Python, for example, allows students to focus on algorithmic logic without being distracted by excessively complicated syntactic structures. It can be used for teaching basic programming, data processing, visualisation, automation, and introductory artificial intelligence concepts. Web programming technologies can help future teachers create interactive educational resources. By studying HTML, CSS, and JavaScript, students can design websites, online exercises, quizzes, and digital learning materials. Object-oriented languages can be introduced at more advanced stages to develop an understanding of software structure, modularity, inheritance, encapsulation, and reusable code.

Integrated development environments, online compilers, interactive coding platforms, learning management systems, and automatic code-assessment systems can improve the efficiency of instruction. These tools provide immediate feedback, simplify code testing, help teachers monitor progress, and support independent learning. However, digital tools should be selected according to pedagogical value rather than technological novelty. Artificial intelligence tools can support code explanation, error detection, task generation, and personalised feedback. Nevertheless, students should not use such tools merely to obtain ready-made solutions. They must critically evaluate generated code, check its correctness, explain its structure, and modify it independently. Ethical use, academic integrity, data security, and authorship should therefore be addressed as part of professional training. Programming tasks should be organised according to increasing levels of complexity. Reproductive tasks help students understand syntax and reproduce basic algorithms. Productive tasks require them to adapt known methods to new conditions. Creative tasks involve independently designing algorithms and developing original software products. Professionally oriented assignments should combine programming and teaching methodology. For example, students may be asked to:

- create a program demonstrating the operation of sorting algorithms;

- develop an electronic quiz for a school informatics lesson;
- prepare visual material explaining loops or conditional statements;
- identify and correct typical errors in learners' programs;
- design differentiated programming tasks for different ability levels;
- create an educational game or interactive simulator;
- develop assessment criteria for a practical programming assignment;
- conduct and analyse a model lesson on programming.

Such tasks allow students to act simultaneously as programmers, instructional designers, and future teachers. They also help bridge the gap between university study and professional practice. Three levels of competence development can be distinguished. At the basic level, students understand fundamental concepts and complete standard tasks with assistance. At the intermediate level, they independently solve typical problems, identify errors, and apply appropriate programming methods. At the advanced level, they design original solutions, develop complete educational projects, justify technological choices, and effectively teach programming concepts to others. A comprehensive assessment system may combine tests, laboratory assignments, project work, oral explanations, code reviews, portfolios, self-assessment, and peer assessment. An electronic portfolio is particularly valuable because it demonstrates students' development over time and may include program codes, lesson plans, educational resources, reflective reports, project presentations, and assessment materials. The systematic implementation of the proposed technology is expected to strengthen future computer science teachers' programming knowledge, practical skills, methodological readiness, and professional independence. Students learn to design algorithms, develop and test software, explain programming concepts clearly, organise collaborative learning, and use digital tools responsibly. As a result, future teachers become capable of conducting practice-oriented programming lessons, adapting instruction to students' individual needs, creating digital educational resources, and evaluating learning outcomes objectively. The technology also promotes creativity, critical thinking, teamwork, self-development, and readiness to learn emerging technologies throughout professional life.

References

1. Mamarajabov M.E., Toshtemirov D.E., Yuldashev O'.A. Informatika o'qitish metodikasi: darslik. — Toshkent: Bookmany Print, 2023.
2. Rustamov H.Sh. Matematika va informatika o'qitish metodikasi: darslik. — Buxoro, 2022. — 257 b.
3. Zaripov N.N. Zamonaviy dasturlash tillari: darslik. — Buxoro, 2024. — 146 b.

4. Grover S., Pea R. Computational thinking in K–12: A review of the state of the field // Educational Researcher. — 2013. — Vol. 42, No. 1. — P. 38–43.