sustainability

MDPI

Article

Conceptual Framework for Programming Skills Development Based on Microlearning and Automated Source Code Evaluation in Virtual Learning Environment

Jan Skalka [1,*], Martin Drlik [1], Lubomir Benko [1], Jozef Kapusta [1,2], Juan Carlos Rodríguez del Pino [3], Eugenia Smyrnova-Trybulska [4], Anna Stolinska [2], Peter Svec [1] and Pavel Turcinek [5]

1 Department of Informatics, Faculty of Natural Sciences, Constantine the Philosopher University in Nitra, 949 01 Nitra, Slovakia; mdrlik@ukf.sk (M.D.); lbenko@ukf.sk (L.B.); jkapusta@ukf.sk (J.K.); psvec@ukf.sk (P.S.)
2 Institute of Computer Science, Pedagogical University of Krakow, 30-084 Krakow, Poland; anna.stolinska@up.krakow.pl
3 Computing Center of the Department of Informatics and Systems, University of Las Palmas de Gran Canaria, 30, 35001 Las Palmas de Gran Canaria, Spain; jc.rodriguezdelpino@ulpgc.es
4 Institute of Pedagogy, Faculty of Art and Sciences of Education, University of Silesia in Katowice, 40-007 Katowice, Poland; esmyrnova@us.edu.pl
5 Department of Informatics, Faculty of Business and Economics, Mendel University in Brno, 613 00 Brno, Czech Republic; pavel.turcinek@mendelu.cz
* Correspondence: jskalka@ukf.sk

Citation: Skalka, J.; Drlik, M.; Benko, L.; Kapusta, J.; Rodríguez del Pino, J.C.; Smyrnova-Trybulska, E.; Stolinska, A.; Svec, P.; Turcinek, P. Conceptual Framework for Programming Skills Development Based on Microlearning and Automated Source Code Evaluation in Virtual Learning Environment. *Sustainability* **2021**, *13*, 3293. https://doi.org/10.3390/su13063293

Academic Editor: Michail Kalogiannakis

Received: 30 January 2021
Accepted: 5 March 2021
Published: 17 March 2021

Abstract: Understanding how software works and writing a program are currently frequent requirements when hiring employees. The complexity of learning programming often results in educational failures, student frustration and lack of motivation, because different students prefer different learning paths. Although e-learning courses have led to many improvements in the methodology and the supporting technology for more effective programming learning, misunderstanding of programming principles is one of the main reasons for students leaving school early. Universities face a challenging task: how to harmonise students' education, focusing on advanced knowledge in the development of software applications, with students' education in cases where writing code is a new skill. The article proposes a conceptual framework focused on the comprehensive training of future programmers using microlearning and automatic evaluation of source codes to achieve immediate feedback for students. This framework is designed to involve students in the software development of virtual learning environment software that will provide their education, thus ensuring the sustainability of the environment in line with modern development trends. The paper's final part is devoted to verifying the contribution of the presented elements through quantitative research on the introductory parts of the framework. It turned out that although the application of interactive features did not lead to significant measurable progress during the first semester of study, it significantly improved the results of students in subsequent courses focused on advanced programming.

Keywords: conceptual framework; automated assessment; source code automatic evaluation; microlearning; introductory programming courses

1. Introduction

Understanding software and writing a program are currently frequent requirements when hiring employees in many areas of the labour market, not only for information technology (IT) positions. Although education systems at all levels are gradually adapting to this requirement, many skilled pupils abandon their interest in algorithmic thinking and/or application development as teenagers and subsequently choose careers in other fields [1]. In addition to students with excellent IT skills, there are also students who have recently discovered (or rediscovered) IT who decide to study IT. The technological skills of these students are often at a very different level.

As the institutions preparing IT professionals, universities face a challenging task: how to harmonise students' education, which focuses on advanced knowledge with respect to the development of applications, with students' education in cases where writing code is a new skill. An unfortunate consequence of applying this one-size-fits-all approach to education in cases where the students have different levels of skill and knowledge is a natural reduction in student numbers during the first months of study. Many authors have estimated that the student dropout rate is about 30–40% [2–4].

Although the implementation of e-learning courses in LMSs (Learning Management Systems) or MOOCs (Massive Open Online Courses) has led to many improvements in methodology and, especially, in supporting technology for more effective learning of programming, the dropout rate has not decreased significantly [5,6]. However, e-learning remains, in all likelihood, the only practical means of delivering university education, because the established computer science educational concept at universities, based on the combination of lectures and labs, is outdated and inefficient. MOOCs probably remain the best choice for the mass education for students with varying levels of knowledge and skill.

This article aims to present a framework covering the effective education of university students in programming and application development. The framework integrates and interconnects introductory programming courses and software engineering courses. It is based on several years of research results in the preparation of university students through introductory and advanced programming courses.

The idea of the framework is based on involving students in the development of the learning environment in which they are to complete their introductory programming courses. Thus, students go through several levels throughout the educational cycle.

In the first phase, students acquire introductory skills and knowledge in an educational environment with pre-prepared content. This phase is the most important in the education of programmers, because its difficulty is one of the main reasons for discontinuing studies. The scope and methodology must, therefore, be precisely defined and monitored for continuous improvement.

In the second phase, students are involved in creating the educational content of the environment, usually in the form of exercises. Defining tasks and rules for checking deepens students' knowledge and skills, improves their expressive abilities and builds their ability to write automatic tests.

The third phase involves applying the acquired skills and knowledge in order to solve problems and carry out projects. While creating "school" applications is already a daunting task for some students, students with higher levels of skill require real application development. The presented framework is based on the involvement of these students in the software development of the educational environment.

The educational environment must meet strict criteria when implemented, but its lifespan will be short without sustainable development. The reason for this is the constant development of software tools, module updates for security reasons, design changes, user habits, and supported technologies.

Assigning independent developers to the development environment is complicated at the university level. It is much more effective to involve skilled students in the development under the guidance of experienced developers and/or teachers.

Cooperation in the educational system can be beneficial for all participating groups: advanced students will be involved in the development of a real application; younger students will be able to use a growing amount of educational content while having contact with older classmates and being motivated to move into a group of developers; the continuous development of the content and software of the education platform will be ensured.

Learning programming is very stressful due to its complexity, and it is incredibly time-consuming in the beginning. Teachers of programming often encourage students to prepare complete programs in the early stages of the course. This approach is impractical for many students, especially for those who have never written one. Understanding the

problem and transforming it into source code requires many skills and profound theoretical background in several overlapping domains.

Therefore, particular attention is paid to the design of educational objects, activities, and support elements in the first phase of programming learning, which are covered by introductory programming courses. The article presents the proposed framework and verifies the benefits of the concept applied in introductory programming courses.

The research is based on a comparison of educational outcomes measured through the tests and projects evaluation. The goals of the research are divided into two consecutive parts focused on students results.

- The first goal is identifying the immediate impact of interactive activities (microlearning and automated assessment) on the students' results in the introductory programming course. It verifies the contribution of implementing the framework and its interactive elements to students' results in learning the programming language.
- The second goal is identifying the long-term impact of interactive activities on students' results within the course Object Technology. It verifies the contribution of the implementation of the introductory programming courses part of the framework, its interactive elements, forced independence of students and skills acquired in solving automatically assessed tasks to students' results in the application of acquired skills (and knowledge) in an engineering course that places complex demands on the student.

Research questions are designed to identify the benefits of applying the first part of the framework in the short or long term:

RQ1: Does the application of the framework and the interactive elements in it (automatically evaluated programs and micro-study lessons) bring significant improvements for students in the introductory programming courses?

RQ2: Does the application of the framework and interactive elements in introductory programming courses bring significant improvements to students' results in advanced programming courses?

The article has the following structure. The second part summarises the background of the framework proposal and the related research on various aspects of the topics presented in this article. The third part describes the conceptual model resulting from research, the relationships between its components and finally, the research background and used methods are introduced. The fourth section is devoted to the obtained results, which are consequently thoroughly discussed in the next section. The conclusion section summarises the main findings and suggests the direction of future research.

2. Related Work

The complexity of the programming learning process is the main reason for educational failures, student frustration and lack of motivation. The training of programming experts has undergone significant changes in recent decades.

Becker and Quille [7] analysed an evolution in programming education research in the last 50 years. They reported a shift from general and content-oriented approaches to the student-centred approaches closely connected to the e-learning courses available MOOCs and LMSs supplemented by the labs for practising programming languages. Skills assessment has moved from assessing conceptual and cognitive tasks to assessing practical skills (writing, debugging, errors identification) using automated tutoring and assessment systems. At present, the research is focused on the prediction of success and measuring performance and student success.

Luxton-Reily et al. [8] summarised the main teaching techniques used in introductory programming courses as follow:

- Sorting topics—the order of content topics is one of the most common problems in teaching programming. Its application has a significant influence on the understanding of content. The object-oriented and fundamental paradigms are most often used.

- Writing tests—creating automatic tests as soon as possible as part of the programming learning process. This approach focuses on students' understanding of the code by analysing the code by writing tests. The goal is to apply reflective programming instead of trial-and-error programming [9,10].
- Exercises—the main tools for acquiring programming skills based on conventional coding and focused on creative thinking, computational thinking, and sometimes collaborative activities [11].
- Leveraging students' prior knowledge—to use the previous experience of algorithmic-thinking in the teaching process. Many teachers use the construction of algorithms and analogies to connect students' previous understanding of a familiar topic to a new programming content topic [12,13].
- Videos—live coding [14] or content explanation [15] are the main reasons for the recording teaching process. Videos are helpful for study gradual code creation and repeated descriptions of demanding content.

A systematic literature review conducted by Medeiros et al. [16] identified the following main challenges in teaching and learning programming at universities, which solution should be implemented in any contemporary virtual environment or supporting tool for learning programming:

- developing methods and tools for teaching programming definition and evaluation,
- scalability problems—diversity of students in classes,
- keeping students' motivation, engagement and persistence,
- providing immediate feedback and teacher-student communication setting,
- selecting of appropriate programming language,
- curriculum, instructional sequence and students' inadequate mathematical background were inspected only in a few research papers.

Skalka & Drlik [17] stated that building programming skills for university students in the "classic" way is currently obsolete and often encounters a barrier built by modern technologies. Nowadays, students reject passive time-consuming activities and prefer immediate testing, verification, and rapid application of acquired knowledge and skills. The students require the options to learn anytime and anywhere, not only at the university. Simultaneously, they prefer to select the knowledge and skills their usefulness can imagine or employ quickly.

Anindyaputri et al. [18] supposed that adaptive learning systems can improve the learning process. The conclusion of their systematic literature review showed that adaptive learning systems could overcome some problems encountered during learning programming. On the other hand, the complexity of adaptive systems is considered their main disadvantage, limiting their wider use and competitiveness.

Kordaki [19] and Lee et al. [20] mentioned immediate feedback to students as a crucial element, especially for novice programmers, because its absence may result in a further misunderstanding of programming concepts. The feedback is often complicated by several factors naturally contained in university teaching (many students, different starting skills and knowledge levels, different ways of understanding).

Krushke & Seit [21] considered automatically generated feedback implemented in various frameworks and teaching support software systems as a suitable substitute for a human teacher.

2.1. Frameworks and Models

Complex frameworks and models focused on systematic programming learning are rare. Many studies examine methodologies, approaches, failure factors, etc. Many authors bring expanding views to the issue and combine different pedagogical and engineering techniques.

According to a literature review by Luxton-Reily et al. [8], Bloom's taxonomy is widely referenced in the research of introductory programming as a standard for evaluating students' learning. Researchers identified the application of theories and frameworks related to students' progress from exploration through computational thinking, constructivist

learning theories to knowledge acquisition and self-regulated learning theory. Fundamental to the presented framework proposal is the view of cognitive load theory, which supposes that learning gradually degrades because students have to remember more and more items than their working memory capacity can accept.

Fuller et al. in [22] suggested the Matrix Taxonomy, which describes the framework for assessing the learner's computer science and engineering capabilities. The taxonomy is based on the complexity of intrinsic characteristics of computer science and covers the requirements of students' ability to program. The model reflects the fact that understanding the program and independently writing a code are two semi-independent capabilities. Students who acquired the ability to read source code may not necessarily be able to write new programs. Likewise, the ability to write program code does not mean the ability to identify errors, debug programs and correct bugs. Different students use different "learning paths". Some students get the skills to read and debug code first, and other students start with writing code skills instead of the skills to read or debug foreign ideas.

Malik & Coldwell-Neilson [23] presented a model using ADRI to help novices overcome problems in choosing the right problem-solving strategies. Ali [24] introduced a model based on a modified version of the system development lifecycle (SDLC), where students solve tasks through the planning, coding, and output phases.

Skalka & Drlik [25] presented a conceptual model for programming learning in the mobile platform. The authors highlighted the possibilities of using mobile learning, microlearning and other technologies that support immediate feedback. They emphasised gamification as one of the leading motivational engines.

Alshaye et al. [26] proposed a conceptual framework for learning programming based on problem-based learning. Their proposal guided the online instructors to effectively organise and design teaching tasks and materials to assist students' problem-solving and computer programming skills. However, they did not consider some essential characteristics of the novice programmers, who prefer the possibility of an environment to write their code and receive immediate feedback and different learning paths.

Krpan et al. [27] developed an open-source framework for learning object-programming languages based on project-based learning. This approach does not consider the characteristics mentioned earlier and the fact that project-based learning requires other soft skills, which can distract the students in the initial phases of programming learning.

Khaleel et al. [28] developed a gamification-based learning framework that consists of game elements and programming learning requirements. Rigo & Diehl [29] used a similar approach and concluded that gamification is an excellent motivational resource for learning programming language concepts. These frameworks were narrowly focused on one programming language and did not provide functionality, which is involved in the framework presented in this article.

Labaj et al. [30] introduced an adaptive educational system, ALEF. It provides a comprehensive annotation framework to support interaction and collaboration, motivating students to engage in extensible architecture, and include additional programming languages. This system did not provide an automated assessment of source code.

Ciancarini et al. ([31]) presented a theoretical framework based on a close connection between complex problem solving and computational thinking associated with agile value resulting in "cooperative thinking" defined as a competence encompassed by complex negotiation, continuous learning, group awareness, and group organisation.

Lopez-Fernandez et al. [32] defined a motivational framework composed of instruments, resources, mechanics, and technologies. The results of an empirical study based on its implementation can be summarised as follow:

- Many students perceive motivational aspects at approximately the same level, e.g., the desire to continue their studies, appreciate their abilities, and the hope for the usefulness of their diplomas.
- Students increase their performance when the proposed activities are considered difficult.

- Students are more focused on working with their classmates and positively impacting them, but they do not value the teacher's recognition too much.
- Students are very sensitive to the shortcomings of their environment (physical and virtual resources of the university, opinion of the teacher's abilities, etc.) and value the opportunity to grow in demanding activities. Thanks to these, they feel responsible for the process of their learning and their academic results.

2.2. The Ideas behind the Framework

The implementation of the framework is a conceptual and technological backbone, which is irrelevant without content. The details of the framework have been designed following the latest psychological knowledge and research on user behaviour. Two essential technological concepts provide education in introductory programming courses: automated assessment and microlearning. While automatically evaluated source code (automated assessment) is used in learning programming since the first educational systems, microlearning in its current form is a relatively new educational concept expanding thanks to smartphones. Advanced and software engineering courses are based on the principles of problem and project learning.

2.2.1. Microlearning

Students need to understand the principles, data structures, and commands while they learn fundamental programming languages. They need to try and understand how to create the program using simple commands into more complex structures. Based on the cognitive load theory of Sweller [33], the use of interactive exercises with the repetition of information in appropriate cycles will ensure the mastery of educational content. The suitable approach is to provide information in an interactive form through micro-content and micro-tasks.

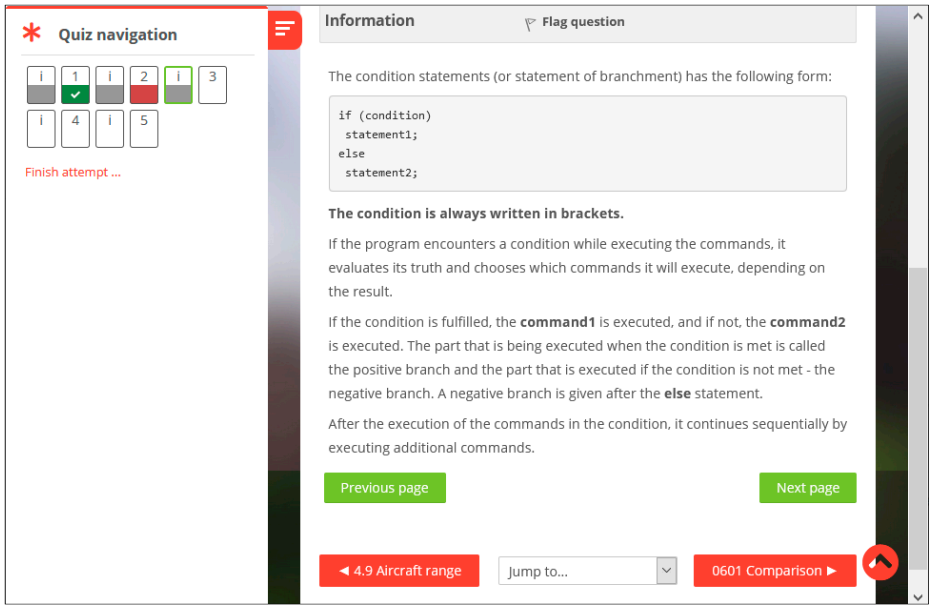
The modern definition of microlearning says that microlearning is an activity-oriented approach that provides learning in small parts, including information and interactive activities to practice [17]. Microlearning offers educational content in short, well-planned units, often through mobile applications that do not require long student attention.

Microlearning is based on the regular rotation of micro-contents and micro-activities. Micro-content is usually presented as short text, sometimes enriched with pictures, tables, diagrams or source codes. Micro-activities require user interactions. These can be in programming learning, e.g., reordering program lines, filling in text or source code, a short answer that represents the result of a program, choosing options or a multi-choice question, and rearranging an expression or commands, etc. Micro-tasks check misunderstandings and consolidate the knowledge gained through micro-content [34].

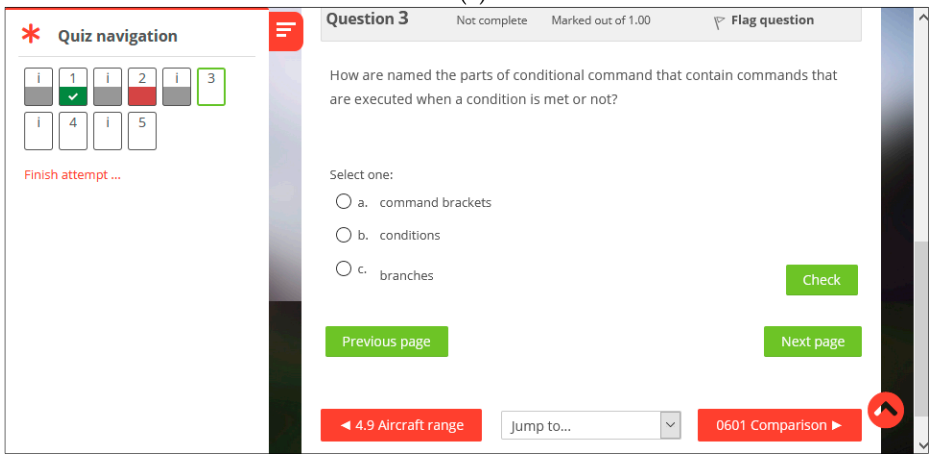
Obviously, due to the range of required skills and the time needed to create programs, microlearning cannot cover the acquisition of all skills necessary to master programming. It is served only as a tool designed to provide introductory information about topic and activities focused on reading, debugging, and source code completion. Created activities require the student to read and understand the written algorithm and prepare him for the final activities, mainly writing the complete program. The exercises requiring whole programs writing are usually placed at the end of chapters—they require the application of information obtained through microlearning.

The typical elements of microlearning lessons (micro-content and micro-question) are presented in Figure 1.

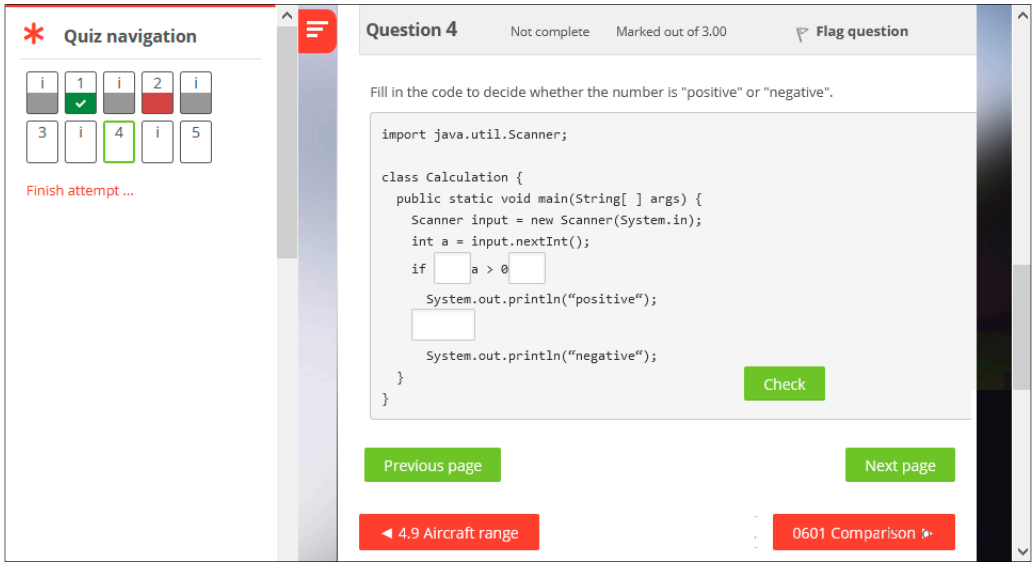
When creating microlearning questions, it is necessary to focus on tasks that verify understanding of the presented content. A typical example is in Figure 1c. However, new terms are often defined in the introductory parts of chapters, which should be memorised to avoid misunderstandings in the following content (Figure 1b). This type of question is needed because some students are able to write a functional program, but they are not able to talk about it and often fail to use terminology correctly.



(**a**)



(**b**)



(**c**)

Figure 1. Example of micro-content (**a**) and micro-tasks incorporated in micro-lessons implementation in the LMS Moodle. Micro-tasks are focuses on memorisation (**b**) or understanding (**c**) of the content.

According to Žuffic & Jurcan [35], the content creators need to define and set only a single teaching goal for one lesson. It is necessary to define a well-thought-out concept of chapters and courses because typical micro-content is focused mainly on important content. Its ideas are based on the elimination of extra content. Considering the situations in which micro-content is used, this requirement is natural and needs to be met.

Although microlearning brings demonstrable results in university education [36–38], its application may not always be beneficial. The typical disadvantage of microlearning is that its application is not suitable for large and complex tasks and is not usually ideal as a primary and only educational strategy. Therefore, it should be used with balance and supplemented with other categories of activities [35,39].

2.2.2. Automated Assessment

The central part of programming teaching is code writing, testing, debugging and optimisation. Assessing the correctness of the code is a challenging task. Automated assessment (AA) has a long history in computer science education. Efforts to automate the evaluation of the correctness of the programs have existed since the mass teaching of programming began (e.g., [40,41]). The demand for immediate feedback has been a part of the programming teaching methodology since didactic research in this area started [42].

Modern principles of automated assessment are based on static or/and dynamic evaluation.

Static evaluation is based on checking the form, structure, content, or source code documentation. This evaluation type is based on the validation of source code without executing the program [43], analysing code and identifying anomalies in textual expression. Static evaluation can be enriched with rules aimed at validating the values of parameters defined in task assignments. Static evaluation is the first option for design-oriented languages (e.g., HTML, CSS) or languages with simple rules (e.g., SQL).

Dynamic evaluation approaches use output results for validation on various levels: the I/O approach and the automated tests approach.

The I/O approach is the simplest approach from the content developer's point of view, with minimal requirements for its capabilities [34]. The authors of the content usually define the input values and expected outputs. Validation is based on a comparison of the values obtained from the students' program with the values defined as expected and correct. The approach offers significant advantages: the definition of test cases is high-speed. Simultaneously, the same test cases can be defined and used for many programming languages. The disadvantages are missing functions to verify the internal structure of the source code (can be solved by extending static evaluation methods) and formatting problems that cause a mismatch between expected and received output. A typical assignment of an automated evaluation task using an I/O approach is shown in Figure 2.

Automated tests are a part of the software engineering testing concept. They are widely used and required in the software development process. Using unit testing is one of the necessary skills of modern programmers (JUnit, CUnit, etc.). This approach is currently the most effective way to validate code, which tests the outputs of programs or results and focuses on checking its elementary parts (units) as methods, procedures, algorithms, class states, etc. The ideal goal of unit testing is to verify each part of the written code and allow immediate repetition of testing after modifying any part of the code. The advantage of this approach is greater flexibility, more accurate identification of errors and explanation of mistakes to the user. The disadvantage is the more arduous preparation of the validation itself through writing code.

Writing tests is one of the crucial activities of the presented framework to develop students' programming skills.

The result of automated testing should not only inform about the correctness of the program. Users need to view syntax errors, compiler messages, and test cases with differences between expected and obtained outputs.

Research that represents a measurable improvement in results using AA is not as frequent as research to identify students' views and attitudes towards them. Using a suitable methodology is challenging. The authors mainly focus on perception by students and the simplification of teacher work.

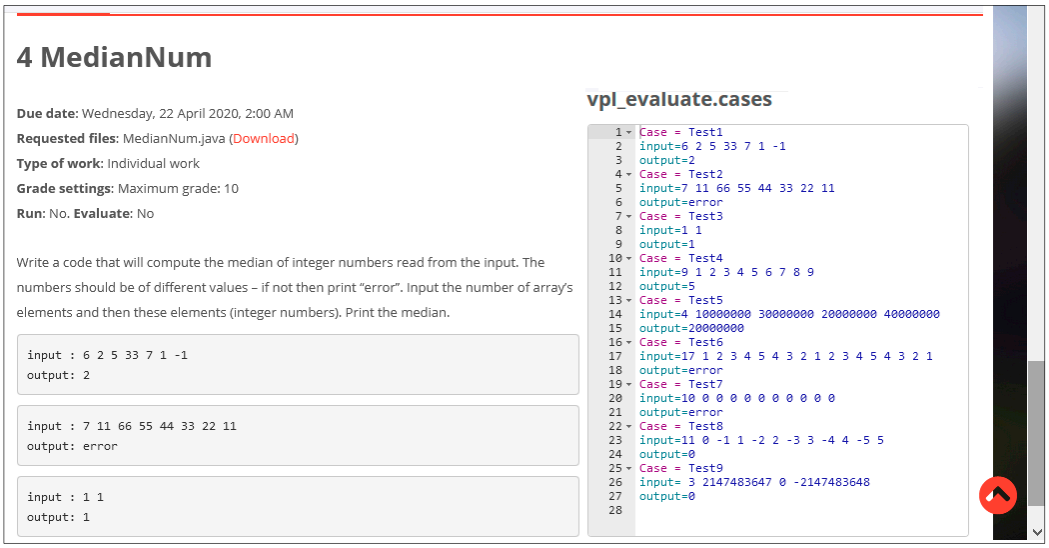


Figure 2. Example of automated evaluation of program code (I/O approach) implemented in the presented framework in the Virtual Programming Lab in LMS Moodle [44]. The task assignment is placed on the left side. Test cases with inputs and expected outputs are placed on the right side.

Alemán [45] compared two groups with and without using AA. The experimental group scores showed that students who worked with AA gained a more reliable experience with debugging, deployment, and versioning. Still, the difference between the means of groups was not statistically significant.

Wilcox [46] used AA as part of automatically evaluated assignments on the final exam. The author presented significantly higher exam scores for students who used automated grading.

Skalka & Drlik [47] concluded that AA does not degrade students' results, but its significant improvement in them was not proved. More detailed research by the same researchers [4] has shown that the use of AA in the introductory lessons of programming courses led to significant student outcomes differences. The initial topics, such as input/output, variables, data types, nested loops, arrays, exceptions, files, etc., cover essential issues. They are often the reason for students' loss of interest in programming because of a misunderstanding of its principles, memorisation problems, or loss of motivation [48,49].

Gaudencio et al. [50] compared the results of evaluating student solutions by teachers and using a computer, concluding that the results are not significantly different. Barra et al. [51] present a case study describing transforming a higher education programming course into an automated student-centred assessment tool due to the COVID-19 pandemic. They inspected the perceptions of students with positive results.

Gordillo [52] stated that automated assessment systems are increasingly used in higher education programming courses since the manual assessment of programming assignments is very time-consuming. He proved, considering his literature review and previous research, that using an automated evaluation tool was beneficial for students due to the increased motivation and quality of their work. At the same time, it helped students expand their practical programming skills.

3. Materials and Methods

The proposal of a conceptual framework covering all phases of programming learning at universities should be connected to appropriate taxonomies. Taxonomies of educational objectives are used worldwide to describe learning outcomes and assessment results, reflecting student learning stages. The effectiveness of taxonomies will be fully reflected in the design of teaching materials and assessments. Structured materials help students advance through taxonomy levels; structured assessments help them gain more in-depth knowledge and master the relationships between the parts of educational material. Many universities use their frameworks to improve learning outcomes and achieve competencies systematically [22,53,54].

The presented framework was designed to cover all parts of Bloom's taxonomy [55] and programming activities in Fuller's matrix [22].

In the presented proposal, the taxonomy is used to cover mastering the programming language and to build comprehensive knowledge and skills to allow the student to create an actual application capable of operating in the current software environment. The use of recent technologies requires acquiring a programming language and the mastery of software development skills (integrated into any modern development environment) and soft skills (communication, problem-solving, working in time, critical thinking). Bloom's taxonomy is applied to all phases of the process of building knowledge and skills, from a programmer starting from zero experience to a professional ready to develop applications in practice.

The movement between the taxonomy levels (Table 1) is ensured by the educational content structure and personalisation of the content selection and its arrangement. Students with better reading skills can begin their studies with tasks focused on reading and supplementing source code, while students who have preferred writing code can solve programming problems first.

Table 1. Revised Bloom's Taxonomy [56].

Knowledge	Cognitive Processes
1. Remember	Recognising, Recalling
2. Understand	Interpreting, Exemplifying, Classifying, Summarising, Inferring, Comparing, Explaining
3. Apply	Executing, Implementing
4. Analyse	Differentiating, Organizing, Attributing
5. Evaluate	Checking, Critiquing
6. Create	Generating, Planning, Producing

Based on [17], the conceptual model should be divided into three levels:

- introductory programming courses dedicated to creating a basis of programming language and mastering basic levels of computational thinking,
- creation of assignments for completed courses as the final phase of the study of introductory programming courses and initial part in engineering courses,
- engineering courses focused on specific technologies (web, server, mobile, database, IoT, etc.) based on programming languages. These courses prepare students to use technologies and solve real problems.

Introductory programming courses should be designed to cover the first four phases of Bloom's taxonomy while developing computational thinking. The mapping to the individual stages of Bloom's taxonomy will be as follows:

- Remember—the student acquires basic knowledge using appropriate elementary pieces of content, combined into logical units in lessons and chapters. The student's understanding of the presented content is verified continuously—usually after the presentation of each information unit, its evaluation in the form of a question follows.

- Understand—the student completes the source code, arranges the program lines, and determines the correct result of an algorithm (or program). They choose the correct form of the proposed algorithm, correct the accuracy of commands, check the right syntax, answer programming theory questions, and write simple programs evaluated by automated methods that provide immediate feedback.
- Apply—the student solves school problems by developing his software programs. The evaluation of source code is automated by comparing the student's outputs with the correct results for a set of input values. If necessary, the student can ask for instructions (help) or directly for the author's solution. He can discuss the ambiguities through a discussion in the implemented social network.
- Analyse—the student solves predefined problems and programs design and data structures for more complex tasks. The procedure and methods of a solution may be unrestricted or limited to time or memory usage. Verification of results is usually performed using automated testing tools.

The finalisation of these activities in these phases will prepare students to increase their knowledge and skills in the last two stages defined in the taxonomy. The gradual transition usually begins as part of the introductory programming courses by final activities and continues as part of the technology courses. Activities that cover this transition aim to create new educational content and new activities that extend the content provided to other students. Simultaneously, they improve the skills level of advanced students.

The coverage of the "evaluation" phase maps the fifth phase of Bloom's taxonomy as follows:

- Evaluate—the student is involved in learning with his/her peers, creates new tasks for the content defined in the introductory courses, and writes automated tests for newly assigned programs. At the same time, he/she participates in discussions, advises beginners and experienced programmers in the community, and evaluates other students' ideas and solutions. The created assignments will be used in the exercises and competitions or tests in courses.

The last phase of Bloom's taxonomy covers the ability to design, create and verify the student's solutions and represents the highest form and application of knowledge:

- Create—the students apply their acquired knowledge and improve their skills in some of the modern development frameworks based on one of the programming languages they receive. The students are involved in developing the learning environment, creates and modifies modules in the courses of their studies and works on real-world problems using the approach and procedures of the business environment. Implementing this phase must include courses focused on technologies (development of mobile applications, frontend and backend frameworks, IoT environments and frameworks, etc.).

Evaluate and Create phases are perceived broadly in this model. The educational part focuses on advanced skills, and knowledge is an integrated part of the educational process. The idea of the framework is based on integrating activities focused on educational content enrichment and system development using the latest technologies. There is no dedicated educational content to cover these parts; this is only a part of the framework—to involve the student in the development process of real systems, which cover students' needs in the first four taxonomy phases. However, it is not enough to master the programming language. One must also know the possibilities of its application, its strengths and weaknesses, and be able to create real deployable applications. Related skills can be acquired by students only via the practical development of real applications. The recommended approach to defining assignments is to use the principles of PBL, cooperative learning and cooperation in teams.

3.1. The Framework Definition

The definition of the framework is designed to prepare students for lifelong learning. The approach used simulates real-life situations on the job: the employees are given a role, a requirement and a knowledge base. They must then develop an application or set of applications to solve the assignment. They can discuss and communicate in teams and with experts during development.

Figure 3 presents the framework from a pedagogical and organisational point of view. The key actors are the students and the educational environment. Students use the learning environment to complete their pathway from the introductory to the final phase of Bloom's taxonomy.

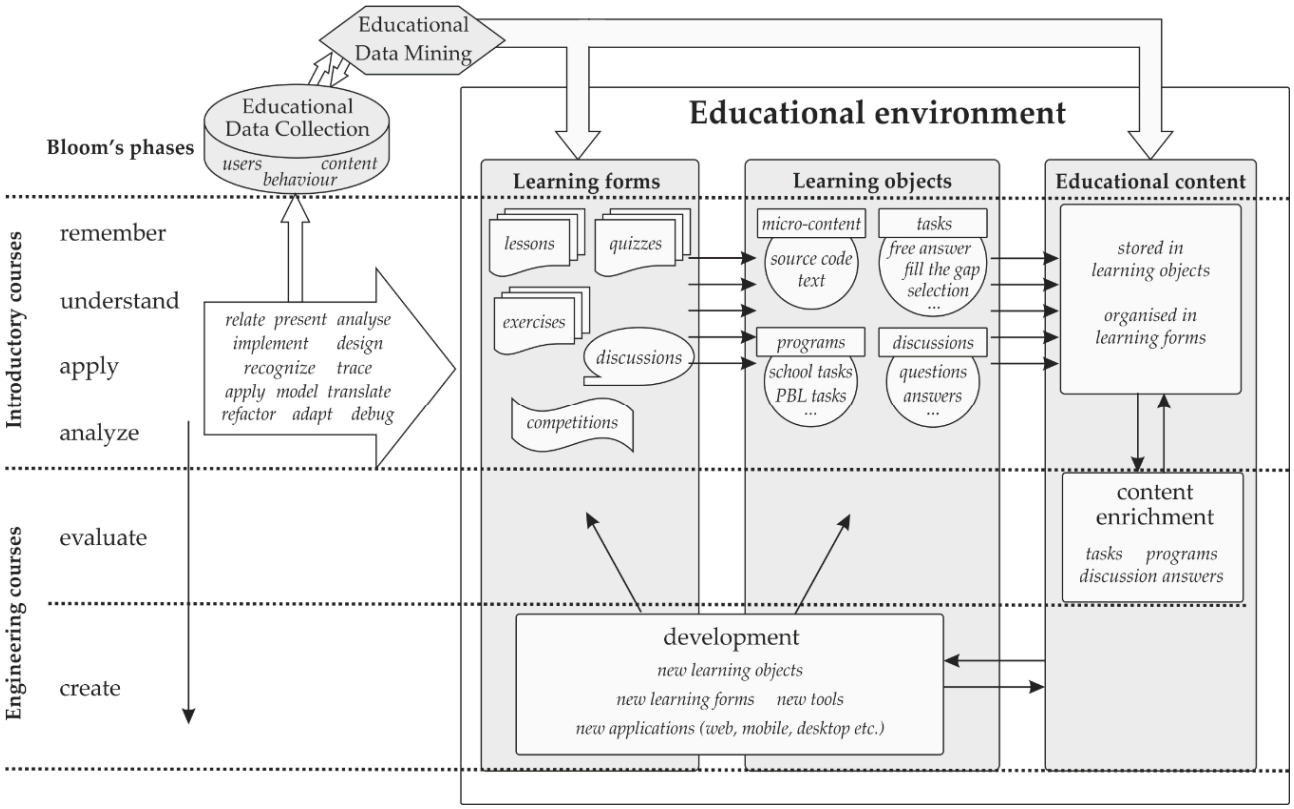


Figure 3. Design of framework from a pedagogical point of view. Students' shift across the stages, according to Bloom's taxonomy from top to bottom.

Educational content is the primary and crucial part of the *Educational environment*. *Educational content* consists of basic units—information, texts, images, and source codes. These elements represent raw but structured content that is displayed to users in learning objects.

Learning objects are represented by different types of questions, microlessons with educational content or program tasks, and are filled by educational content. Each item has its definition, content (question, task, assignment) and mechanisms for its validation and scoring.

Learning objects are arranged in *Learning forms*. The basic structure of the lesson consists of several learning objects. The lessons are organised into chapters that represent the topics of programming courses. Although the content is organised into lessons, the student can choose to read or solve any object in any chapter or lesson. Freedom is a significant feature of the learning environment because each student's learning pathways can be various and different.

While lessons consist of various content and tasks, quizzes contain only assignments of tasks and programs. The exercises offer extended content to provide practice of the skills in the chapter to which they are assigned.

The following important learning form is competition. The competition is a subject consisting of several learning objects described above, assigned to a pair or group of students. Students compete and try to achieve the best time, the best score or the best solution of randomly selected tasks or programs.

The described sections, activities and methodologies are a summary of commonly used approaches. They are integrated into a complex form, which appears within the term *Educational environment*. The educational activities cover the first four parts of Bloom's taxonomy and are included in the introductory programming learning part.

If the framework is to be usable as a long-term and comprehensive tool capable of sustainable development, it is necessary to ensure:

- content validation to identify and correct problem parts in methodologies,
- content enrichment with an emphasis on practice,
- new elements added to keep audience interest,
- modernisation of the form of content provision.

These requirements can be met in line to transfer students to the higher stages of Bloom's taxonomy. The student's connection to the development of the real content building and real software development in engineering courses has at least two contributions: students are involved in real software with immediate feedback from their colleagues, and the software system covering the framework can be modified at any time because its modules, which are developed using current technologies, can be developed and expanded continuously per the content actualisation of the content of engineering courses. These phases are covered as follows.

Content validation for problem identification is covered by the *Educational Data Collection* module, which collects data about every action performed in the learning environment. If a result is part of the user's action (e.g., a score, duration), its value is part of the recorded activity. The Educational Data Mining module evaluates the collected data. If a problem is detected, the correction mechanism is initiated. The results are reviewed by a person (artificial intelligence may be possible in the future), and a solution is proposed (e.g., correction of an incorrect assignment, change of task, addition of new content, etc.). Content validation affects *Educational content* and *Learning forms*.

Content enrichment can be provided by the students. After completing introductory programming courses, the students have comprehensive knowledge of programming language and can solve school and PBL tasks. Moving students to the evaluation phase (of Bloom's taxonomy) means that they can independently assess the correctness of the solutions or propose new tasks. Thanks to their acquired experience, they can identify more demanding parts of courses and come up with tasks that will enable other students to better practice the issue. Simultaneously, students can create automated tests designed to check the correctness of programs submitted by other students. Their role is to communicate with younger colleagues in simple language through discussion and lead them to the right solution for more complex tasks.

Adding new elements is ensured by students' activities defined in engineering courses. The ideal enrichment with new elements consists of proposals of new types of activities that are not implemented in the educational environment yet (e.g., selecting correct program input, pairing input and output values etc.), and creating new tasks based on them. The development of new types of activities is focused on building advanced programming and engineering skills, problem-solving, creativity, responsibility and soft skills in general. The development of new elements is complex; the students have to integrate their product into the existing scheme and co-work with several teams.

It cannot be assumed that every new element will be successfully developed and deployed. Therefore, it is appropriate to consider year-on-year improvement and upgrading of existing activities by new students. The critical requirement of the software update

process is to preserve the educational content and results of users' tasks solved in the past. The development of system modules and related tasks prepares students for challenging tasks implemented in practice and provides them with the significant experience highly valued by all employers.

While the development of new elements requires their implementation into the existing infrastructure, the *modernisation of content provision* pushes students' skills requirements even higher. In addition to typically technological skills, it focuses on developing organisational and managerial skills. The students design and create their own (primary frontend) environment to convey content to users in the form of predefined types of activities.

The authors proposed this concept due to real experience because many new technologies were introduced to the market over several years of application development. Employers, of course, called for the development of student's skills in the latest technologies (e.g., Vue and React frameworks replaced jQuery, Kotlin replaced Java in Android application development, desktop application development moved to a web application development platform, etc.).

Application development and user interface behaviour thus undergo constant changes. The students acquire valuable skills by developing specific cases of applications in the latest development frameworks. The *Educational environment* reaches into modern visual and technological design. Teaching the fundamental programming language does not depend on current technologies, so the actual content remains up-to-date and is offered to more and more students through introductory programming courses.

Activities aimed at developing new elements and new applications connect the needs of practice with the technologies used in professional work. They provide developers with valuable feedback, and often a sense of satisfaction, motivating them to move forward.

The content of the presented framework was primarily designed to cover education in introductory programming courses, but its scope has proved to be much broader. Indeed, the development of elementary skills and the ability to solve problems cannot be separated from the engineering technologies in which these skills are applied in practice. The framework, therefore, covers educational content focused on programming languages, software technologies and software engineering.

Because the framework definition is not adapted to specific programming languages and technologies, the proposed model is universal and provides space for building universal or narrowly specialised implementations.

3.2. Additional Features of Framework

The idea of community learning was created as a part of a constructivist vision [57]. In recent years, there has been a growing interest in exploiting the social nature of learning in the virtual world through all content distribution channels (video, audio, text). Community learning is based on the psychological and social characteristics of students —users want to discuss, are not ashamed to explain or argue, and many of them also want to be seen. Community members act as collaborators, trainers, audiences and knowledge creators. However, their position in the group is, at first sight, equivalent to breaking down the communication and social barriers of less assertive members.

The functions that can be used to support community education can be divided into the following groups within the presented framework:

- Commenting and discussions are an integral part of all activities: commenting on tasks and solutions, helping students solve problems, explaining problems or solutions, discussing defined or free topics, finding topics, etc.
- Group building in the community promotes gamification and competition between classes, universities, countries or freely created groups.
- User presentation is a standard part of social networks. Although the presentation in this type of application is often distorted, it is possible to identify advanced users or users whose knowledge level is similar to others. Users can open a discussion

with them, challenge them to a duel, or at least follow through with standard social networking principles and functions.

- Content development is dedicated to advanced users and allows them to create new tasks and test questions. The primary motivation of this feature is to use the assignments and program for colleagues or students to understand the content better. The parallel benefit is a potential for the permanent expansion of the system with new topics, types and areas. The created tasks become part of a group of tasks intended for practice, and other users can evaluate their quality, accuracy or meaningfulness. The rating of tasks assignment also provides feedback for authors, providing them with the opportunity for analysis and personal improvement.

Learning in the community brings the opportunity to create richer content and a deeper understanding of the relationships between educational units [58,59].

Gamification is currently one of the main tools used in applications to engage the user in application activities. The authors use gamification elements to maintain user engagement and enhance the user experience in applications designed to sell products or services, build a community or realise educational activities. Gamification with a suitable setting is usually used as a tool to ensure internal motivation. The educational process achieves a higher engagement by the students, which affects the quality of their understanding of the presented information and increases their level of satisfaction. Several gamification frameworks are implemented in various applications. They all contain gamification elements aimed at collecting points, gaining skills and competing [60,61].

- Levelling and scoring are essential components of systems using gamification. User behaviour in the system is monitored and evaluated, and desirable behaviour and results are rewarded based on defined rules. Some systems positively assess the regularity of use and reward users for daily use of the application. Depending on the earned points, users progress to higher levels, unlock new features, or gain additional benefits. Many systems also use a currency related to the other rules and can be used to purchase aid or parts of the solution.
- Badge collection is an exciting addition to education systems. Badges are obtained upon successful completion of selected tasks and goals that go beyond the main activities. Badges reflect the skills, knowledge, preferences, interests, or behaviour of users.
- Bonus tasks or contests are an essential part of systems that try to keep users active even after mastering the formal content of lessons. They create a space for comparing skills with other users in various disciplines, e.g., correct answers in a set of several questions, faster programs, faster program writing, shorter source code, etc. Competitions can be aimed at couples or a wider audience.

The advantage of a dedicated learning environment is freedom in the implementation of tracking and evaluating functions [62]. Many user activities can help tune the system: e.g., user behaviour in the system, time spent solving individual tasks or types of tasks, preferences while working in the system, and tracking successes and failures in specific kinds of tasks. The obtained information can modify the system or its content further or further research in the learning process.

User data collection is an essential prerequisite for validating the content and learning pathways as well as predicting user behaviour and the risks of a student's failure.

3.3. Moodle Implementation

The implementation of essential parts of the framework was gradually developed during 2017–2019 in the LMS Moodle environment. The first four parts of the framework were covered by an introductory programming course, which dealt with the Java programming language and object-oriented programming basics.

The experience gained in courses Java I. and Java II. was further developed by the course Object Technology realised in the 3rd semester of study. This course was one of the courses dedicated to the final phase of the framework. Teaching in this course was

realised without the development of modules of the educational system—students applied the acquired knowledge and skills "only" in solving practical school tasks.

The following research, which verifies the importance and effectiveness of implementing the framework, aims to identify the significance of the impact of technologies and methodologies used in the introductory programming course and their effects on students' skills and knowledge in solving complex problems in the Object Technologies course.

Over the three years, the Java introductory programming course authors introduced new types of elements into the study every year and inspected their impact on student learning outcomes.

- In 2016, teaching was carried out using methods that did not use automated assessment and microlearning concepts. The students' results achieved this year will be considered the control group results.
- In 2017, the first AAs were added to a programming course and enabled students to solve expanding (voluntary) tasks in Java fundamentals (first half of the course).
- In 2018, AAs became mandatory, and their number increased to more than 150 and covered all the Java course content.

The implementation of the framework ended in 2019. The introductory programming course was modified to the new form with content structured into micro-lessons and micro-tasks. The gradual building of the course according to individual content elements is shown in Table 2.

Table 2. The process of content building—the changes between years 2016–2019

Knowledge	New Content Elements	New Activity Elements
2016	presentations video lectures solved programming assignment	every-week summarisation quiz
2017		automated assessment (procedural programming)
2018		automated assessment (class programming)
2019	micro-lesson content	micro-lesson quizzes

3.4. The Educational Content

The introductory programming course structure is based on the combination of introduction to procedural programming, object-oriented programming, and graphical user interface (GUI) development. The content is summarised in [63] and is divided into the following chapters:

The procedural programming part lasts five weeks and consists of:

- The Java language, Output commands, Variables, Loading the values
- Conditions, Loops, Numeric data types, Other data types, String
- Nested loops and effectivity, Multiple conditionals, Exceptions
- Arrays, Array processing, Random numbers, 2D arrays
- Files

The second part is focused on the objects and class type. It lasts four weeks and consists of:

- Introduction to object-oriented programming, Methods, Encapsulation
- Constructors, Class examples
- Static variables, Class examples II.
- Inheritance
- Polymorphism

The last part of the course is focused on GUI proposal and applications with a simple GUI. This part lasts two weeks and consists of:

- basic components (button, text fields, check and radio buttons);
- components with data models (lists, tables).

The courses are taught 6 h per week: 2 h lessons, 2 h for the mandatory, and 2 h for a voluntary programming seminar. Intensive home preparation is a matter of course—students have to solve 15 programs during the week on average.

Practical skills and the ability to apply the acquired knowledge are verified by two comprehensive tests focused on practical skills at the middle and end of the semester. The complexity of the assignments did not change between the years during the performed experiment. The tasks are designed so that only the best students can complete them in a limited time. The aim is to realistically compare and differentiate students' abilities on both sides of the Gaussian curve. Based on many years of experience and experiments, the limit for successfully completing the test was set at 40% of points.

The final exam consists of a quiz aimed at understanding the finished programs and several basic questions of programming theory. The final exam results are not part of the research because they do not bring new and relevant information—in programming, the emphasis is on applying the acquired knowledge and skills in solving practical tasks.

The structure of the course Object Technology is focused on JavaEE technologies. This course is optional, but most students choose it to prepare for project solutions in practice. It is aimed at the practical development of applications, which corresponds to its structure: one hour of lectures, two hours of seminars and two hours for homework activities.

The content consists of the following topics:

- JavaEE—Java Enterprise Edition
- JDBC—Java DataBase Connectivity
- Servlets—essential elements of server applications
- Sessions and user identification
- Servlet application development—actual application with thread synchronisation and session guarding
- JPA, ORM—Java Persistent API, Object-Relation Mapping

Completing the course requires the students to finish 3–5 projects with increasing difficulty depending on the individual topics. Students solve projects individually to master each of the technologies at a sufficient level. A typical example is the e-shop development with standard shopping processes and order management.

Students evaluate projects through a blind peer review and by teachers. Students are evaluated by points obtained for solving projects. The relevant, measurable indicator is the percentage of students' success based on the number of points obtained that they could get within the course.

3.5. Definition of Hypotheses

According to defined research questions, the research compares educational outcomes measured through the tests and projects evaluation. The following hypotheses are set to identify the significance of the contribution of new elements and methodologies to the evaluated student outcomes.

Hypothesis 1 (H1). *The application of the framework and its new interactive objects improve students' results in the introductory programming course.*

Students' outputs in 2016 and 2019 will be compared. While any elements used in the presented framework were not implemented in 2016, interactive activities (automated assessment and microlearning) were used in 2019. Simultaneously, new methodologies for their inclusion in teaching were implemented.

Hypothesis 2 (H2). *The application of the framework and its new interactive objects improve students' results in the advanced programming course.*

A comparison of student group results started in 2016 and 2019 will be used to verify this hypothesis. Because the Object Technology course is implemented in the 3rd semester, student results from 2017 and 2020 will be compared. The hypothesis should verify the contribution of automated assessment and microlearning to learning advanced programming topics.

3.6. Characteristics of the Respondents

Comprehensive research has been carried out on a sample of 51–102 students per year. Changes in society in recent years have affected the number of students admitted to universities. The number of first-year students in the research workplace has doubled in four years period, which significantly impacts students' quality.

Every group consisted of computer science first-year students at the age of 18–24. Only students who have studied the subject for the first time were included in the groups. Students who repeated the study of programming were excluded from the research.

The entrance results of students were obtained from university applications and reflected the grades acquired in high-school study and secondary school competitions. These results were rated on a scale of 0–40. Only students whose number of points obtained in entrance examinations based on high school results have reached the limit of at least 10 points were included in the research. In 2016, this limit represented the lower limit for the admission of a student to university studies. This requirement reduced the number of students involved in research after implementing the framework. Therefore, only 87 students met this requirement in 2019.

Nevertheless, the groups of students who take the introductory programming courses were quite diverse every year. It was caused by the various skills acquired in secondary schools, different programming experiences, and various computational thinking levels. The statistical characteristics of students' entrance results are presented in Figure 4.

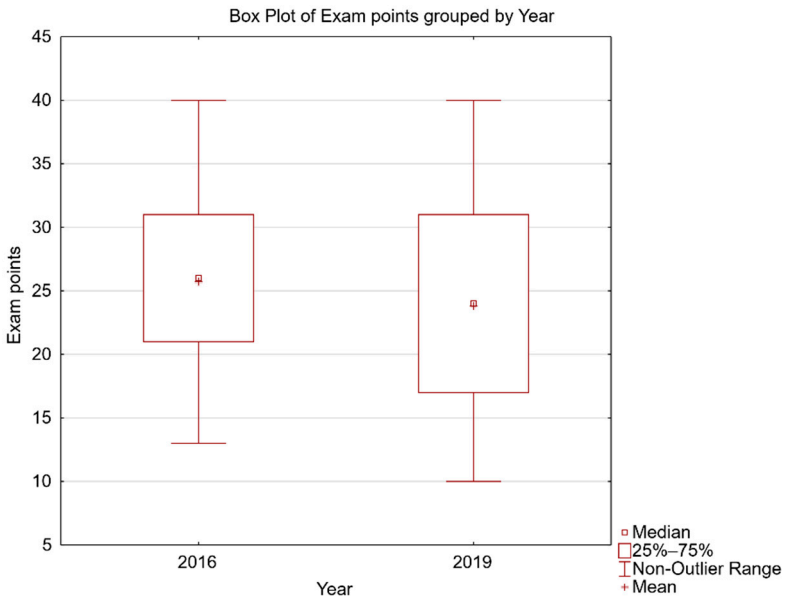


Figure 4. Graphical visualisation of the statistical characteristics of groups inspected in an introductory programming course (points awarded in the admission procedure).

When the second hypothesis was verified, only students who completed the introductory course of programming in one year and the course of Object Technology in the

following year were included in the inspected sample. The groups reduced to 37 students in 2017 and 36 students in 2020.

The characteristics of students based on the results of the entrance exam are shown in Figure 5.

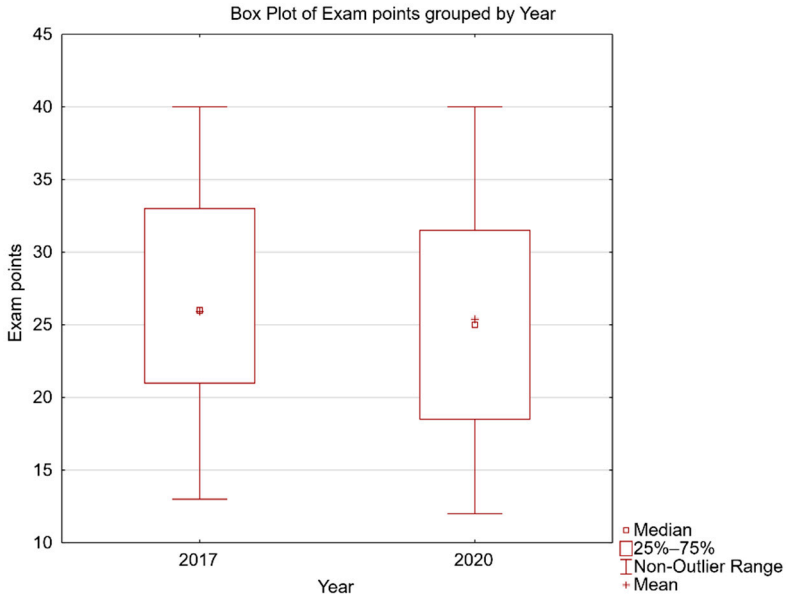


Figure 5. Graphical visualisation of statistical characteristics of groups inspected in advanced programming course (points awarded in the admission procedure).

The primary sources used for research were:

- The list of admitted students with the number of points awarded in the admission process used in the pre-tests.
- Students' test results in two tests realised in the introductory programming course to evaluate Hypothesis 1 (H1).
- The results of the evaluation of student projects in the advanced course of programming (Object Technology) expressed as a percentage representing the number of points obtained against the number of points achievable. The results of second-year students in 2017 and 2020 are considered to evaluate Hypothesis 2 (H2).

4. Results

The differences between the results of students' groups were inspected in the following research. The first part compares students' results based on H1—Improvement in the introductory programming course based on the implementation of the framework. The second part checks the benefits of the framework application in the advanced programming course (H2).

4.1. H1—Improvement of Students' Results in the Introductory Programming Course

4.1.1. Pre-Test

The first step is proof that student's groups were well-modelled by a normal distribution. These values reflected the score obtained in the admission procedure. These values were independent of the previous students' experience in programming and thus did not distort the prerequisites for mastering the course. The range of awarded points was between 10 and 40.

The results of the Kolmogorov–Smirnov test are listed in Table 3. The data in both groups of students do not differ significantly from a normal distribution at the 5% significance level—the groups are well-modelled by a normal distribution.

Table 3. Proof of normal distribution.

Group	Count	Mean	Median	Std. dev.	D (val. of K-S Test)	*p*-Value
2016	51	25.73	26	6.96	0.10	0.59
2019	87	23.81	24	7.84	0.10	0.37

Levene's test compared the variances of groups with the null hypothesis that both populations (with normal distribution) have the same variances. The requirement of homogeneity is met when the result is not significant. The results are presented in Table 4.

Table 4. The proof of the equality of variance.

Valid N1	Valid N2	Std. dev. 1	Std. dev. 2	F-Statistic	*p*-Value
51	87	4.03	3.86	1.84	0.18

The equality of variance at the 5% significance level is not significantly different. Therefore, it is possible to accept the assumption that the results of the group can be compared.

The last part of the pre-test is the comparison of means in groups. Because the equality of variance is not significantly different, the Tukey test can investigate whether the population means of groups are equal. The results are presented in Table 5.

Table 5. The proof of the equality of means.

Valid N1	Valid N2	Q-Statistic	*p*-Value
51	87	2.03	0.15

The *p*-value corresponding to the Tukey HSD Q statistic is higher than 0.05, which means that the results are not significantly different at the 5% significance level. The result identifies comparable characteristics in inspected groups.

As a result, the research intention can be realised because of the differences between the educational outcomes in students' groups.

4.1.2. Post-Test

The H1 hypothesis requires a comparison of students' results measured by two tests. The tests were focused on the practical experience of students and consisted of several (3–5) assignments that covered the entire content of the first or second half of the course content.

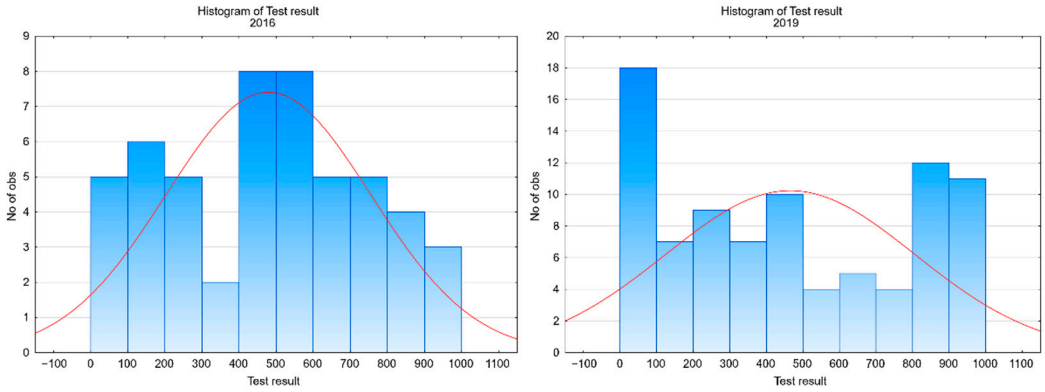
The results were collected from the LMS Moodle environment, and they did not contain the results of students who failed.

The assumptions of normal distribution and homogeneity of variances are met. The results of the Shapiro-Wilk test are presented in Table 6.

Table 6. Assumptions of normal distribution and homogeneity of variances.

Shapiro-Wilk Test of Normality						
Group	**Count**	**Mean**	**Median**	**Std. dev.**	**W-Stat**	***p*-Value**
2016	51	475.96	480.60	274.63	0.97	0.19
2019	87	462.43	427.22	339.07	0.91	0.00002

The data in group 2016 do not differ significantly from a normal distribution. On the other hand, data in group 2019 are significantly different (p-value < 0.05). The histograms of tests results are presented in Figure 6. A visual representation confirms this fact. The means of the groups are close, but medians and standard deviations are visibly different.



(**a**) (**b**)

Figure 6. Histogram of test score distribution in monitored groups: (**a**) Test score in the group trained without advanced elements in 2016; (**b**) Test score in the group of students used all advanced elements in 2019.

Therefore, a non-parametric Mann–Whitney U-test should be used to investigate whether the results are significantly different. The null hypothesis assumes that the means are not significantly different. The distribution of values is approximately normal. Therefore, the z-score should be used. The values of the test are presented in Table 7.

Table 7. Mann-Whitney U-test results.

	2016	**2019**	**Combined**
Sum of ranks	3611	5980	9591
Mean of ranks	70.8	68.74	69.5
Expected sum of ranks	3544.5	6046.5	
Expected mean of ranks	69.5	69.5	
U-value	2152	2285	
Expected U-value	2218.5	2218.5	
Standard Deviation			226.71

The U-value is 2152, the Z-Score is 0.29, and the p-value is 0.77. The result is not significant ($p < 0.05$). Since p-value > α, the hypothesis is accepted, the difference between the averages of groups is not large enough to be statistically significant.

The results show that the hypothesis H1 cannot be accepted. It means that implementing the framework in the introductory programming course does not lead to significant differences in student outcomes.

4.2. H2—Improvement of Students' Results in the Advanced Programming Course

4.2.1. Pre-Test

Students' subsets from the groups examined in the previous hypothesis are selected to verify the hypothesis as follows. These students have completed the optional course in Objective Technology 12 months after starting their university programming studies. The groups are named 2017 and 2020 because they completed the course of advanced programming in these years. However, they are a subset of students examined in the previous hypothesis, who started the study in 2016 and 2019.

Table 8 shows the characteristics of groups and the confirmation of the normal distribution of the score of admission procedure. The range of awarded points was between 10 and 40.

Table 8. Proof of normal distribution (Kolmogorov–Smirnov test).

Group	Count	Mean	Median	Std. dev.	D (val. of K-S Test)	*p*-Value
2017	37	25.92	26	7.52	0.11	0.77
2020	36	25.39	25	7.71	0.12	0.67

The data in both groups of students do not differ significantly from a normal distribution at the 5% significance level—the groups are well-modelled by a normal distribution.

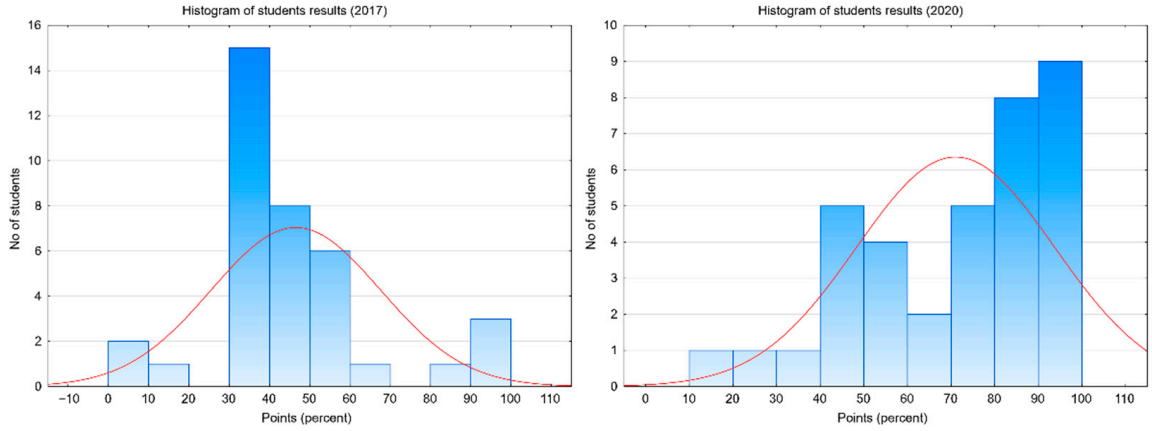
Levene's test was used to assess the variances of the two groups. The difference between the variances of all the groups is not big enough to be statistically significant. The *p*-value is 0.83, [p(x $\leq$ F) = 0.166249], and it is possible to accept the assumption that the group results can be compared.

The Tukey test realised the comparison of means in groups at the 5% significance level: the *p*-value corresponding to the Tukey HSD Q statistic (0.42) is 0.78, which means that the results are not significantly different.

The research intention can be realised because of the differences between the educational outcomes in groups of students can be compared.

4.2.2. Post-Test

The H2 hypothesis requires a comparison of students' results measured by project evaluation. Projects are solved continuously throughout the semester. The final evaluation represents the sum of points obtained and the total number of points that could be obtained in the course in the current semester. The results were collected from the LMS Moodle environment. The results of the compared groups are shown in the histogram in Figure 7.



(**a**) (**b**)

Figure 7. Histogram of results distribution in monitored groups in advanced programming course: (**a**) Course score in the group trained without advanced elements in 2017; (**b**) Course score in the group of students used all advanced elements in 2020.

At first glance, it is clear that the data does not have a normal distribution. Evidence of this assertion is the results of the Shapiro-Wilk test shown in Table 9.

Table 9. Shapiro-Wilk test results for students results in an advanced programming course.

Shapiro-Wilk Test of Normality						
Group	**Count**	**Mean**	**Median**	**Std. dev.**	**W-Stat**	***p*-Value**
2017	37	46.24	42.7	20.96	0.87	0.0004
2020	36	70.66	75.9	22.60	0.91	0.0050

The data in both groups are significantly different from a normal distribution (*p*-value < 0.05); therefore, a non-parametric Mann–Whitney U-test will be used again to investigate if the results are significantly different.

The null hypothesis assumes that the means are not significantly different. The test statistic Z equals −4.204568, which is not in the 95% critical value accepted range: (−1.9600, 1.9600). Value U = 284.50, is not in the 95% accepted range (488.4000, 0.02594). The *p*-value is 0.000026. Since *p*-value < α, H_0 is rejected. The difference between the means of groups can be considered large enough to be statistically significant.

The hypothesis H_1 was not rejected, and thus the application of the framework brought significant differences in student outcomes in advanced programming courses.

4.3. Dependencies Inspection

Dependencies between the measurable indicators of groups were further analysed to obtain a comprehensive view of the investigated characteristics of observed groups.

First, the correlations between the entrance exams results and the introductory and advanced programming course results were examined. Subsequently, the correlations between the results of the initial programming course and the course for advanced programming were investigated. Figure 8 shows graphs for each comparison, and Table 10 shows the Pearson coefficients for all observed pairs.

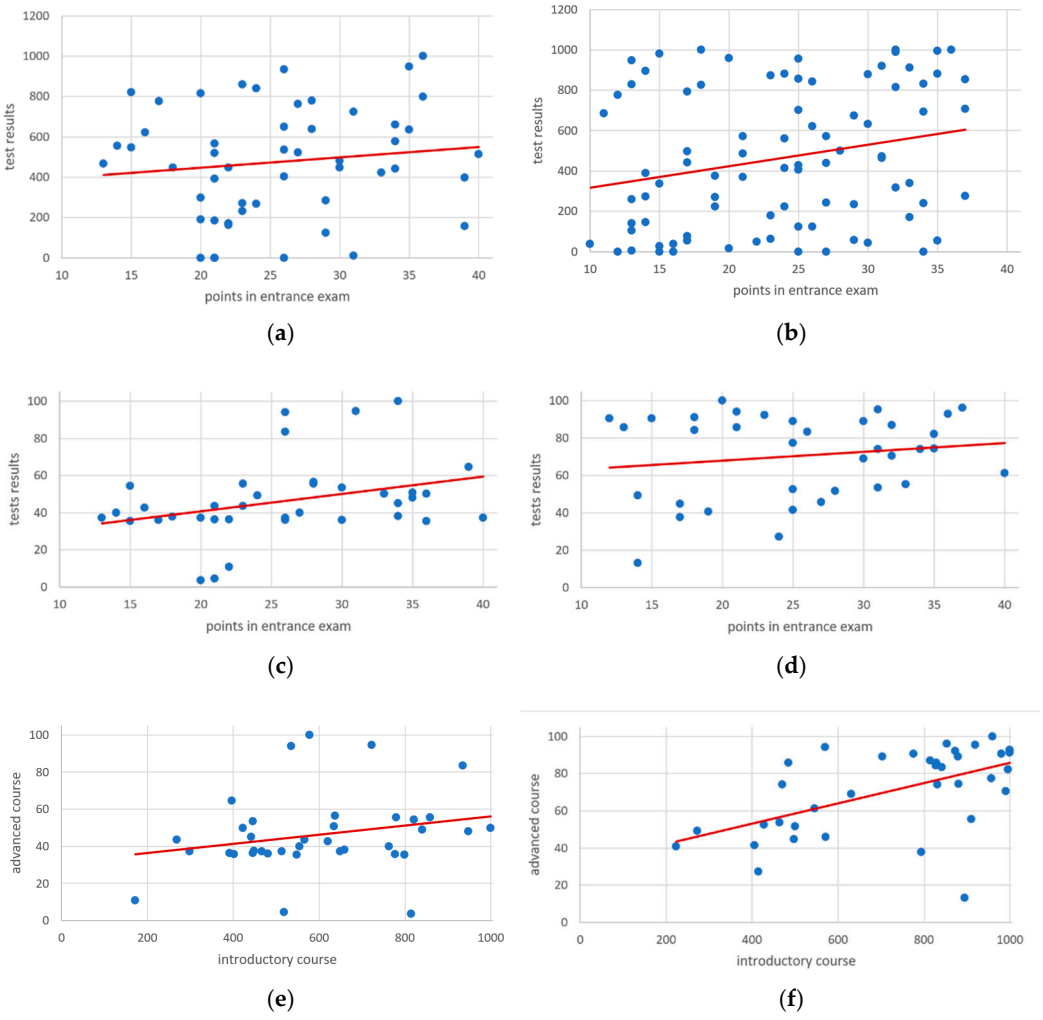


Figure 8. The graphs expressing the correlation between the observed characteristics and their significance: (**a**) relationship between results of entrance exam and an introductory course in group 2016 (2017), (**b**) relationship between results of entrance exam and introductory course in group 2019 (2020), (**c**) relationship between results of entrance exam and advanced course results in group 2016 (2017), (**d**) relationship between results of entrance exam and advanced course results in group 2019 (2020), (**e**) relationship between introductory course results and advanced course results in group 2016 (2017), (**f**) relationship between introductory course results and advanced course results in group 2019 (2020).

Table 10. Pearson correlation coefficient between the characteristics of students results.

Group	2016–2017	2019–2020
entrance exam—introductory course results	0.13	0.24
entrance exam—advanced course results	0.33	0.16
introductory course results—advanced course results	0.24	0.55

Except for the correlation between the introductory course results and the advanced programming course results in the experimental group, the analysis shows very little or no correlation. This result indicates that the student's high school results (summarised in the entrance exam results) do not affect or weakly affect the results achieved in programming courses.

The positive correlation expressed by the Pearson correlation coefficient of 0.55 shows that the results obtained in the introductory programming courses after applying the proposed framework positively influenced the results in the advanced programming course.

5. Discussion

The research was focused on identifying the significance of using the framework in the introductory parts of programming learning. It compares students' results in two phases of study at the end of the study of the introductory programming course and after the completion of the technological course in the third semester.

Hypothesis 1 (H1). *The application of the framework and its new interactive objects improves students' results in the introductory programming course.*

This hypothesis cannot be accepted. Students' results based on a pair of tests covering the course content did not show a significant difference between the group from 2016 studying according to classical blended learning method and the group in 2019, whose study was covered by the proposed framework and its interactive activities. The difference between the results is not significant. Even though the average of the experimental group is slightly lower, no relationship has been identified between the entrance exam results and the introductory course results; it is appropriate to focus on the differences in the behaviour of groups during the semester when identifying the reasons.

Students from group 2016 continuously submitted a small number of tasks that teachers evaluated. Most of these tasks were submitted at once. During their studies in 2019, students were forced to work continuously. The possibility of submitting programs and studying micro-learning content was made available only for a certain period. Students were forced to perform tasks during the semester continuously.

The approach applied in 2019 divided students into two groups during the first half of the introductory programming course. The first test results in Figure 9 show that some students had very poor, and some achieved excellent, results. The number of students with average results was small. Although the results approached a normal distribution by the end of the introductory course, there was a division between students who are satisfied with continuous work and students who take longer to get used to continuous study in the first half of the course. Some students dropped out early.

Although the application of the framework did not bring significant benefits in the introductory programming course, students perceive the use of microlearning and automatically evaluated program codes positively. Based on the already published results of the questionnaire aimed at finding out the student's attitudes [4], it can be stated that:

- 69% of users agreed that AAs help students understand educational content.
- 76% of users agreed that AAs help students to practice educational content.
- 81% of users agreed that microlearning helps students understand educational content.
- 85% of users agreed that microlearning helps students practice educational content.

- 73% of users agreed that microlearning could be used as the primary way of programming learning.

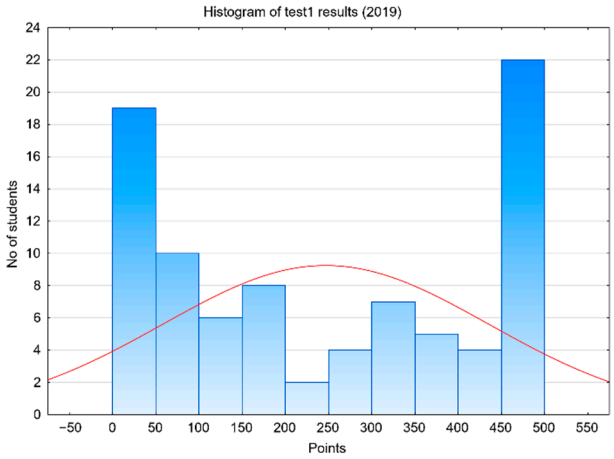


Figure 9. Histogram of first test results distribution in group 2019 in an introductory programming course.

Although the last statement is quite vague, and microlearning certainly fails to cover all the programming instruction needs, the overall satisfaction of students with the framework is high.

Hypothesis 2 (H2). *The application of the framework and its new interactive objects improves students' results in advanced programming courses.*

The hypothesis was not rejected. The use of the first four parts of the framework and application of automatic evaluation of programs and microlearning activities demonstrate significant benefits for students in the long-term perspective.

The Pearson correlation coefficient between the introductory programming course and advanced course results shows the mediate dependence. The introductory course activities and the need for continuous problem solving, testing of created programs, reading test results, finding correct answers in micro-lessons, and other activities impacted building knowledge, skills, and habits supporting successful mastery of the advanced programming course (Object Technology).

This result confirmed the findings of researchers who have studied the use of AA in higher education. They found that the services of AA positively affect building the skills needed in advanced programming and engineering courses.

Alemán presented in [45] the results of a study performed on programming students in the CS2 course. The experimental group score showed that the students who used AA in their learning were more motivated to study and understood the concepts of debugging, deployment, and versioning more thoroughly. This finding supports the idea that automated assessment supports students' skills and ability to use programming tools.

Barra et al. show in [51] that AA was perceived very positively by students. The authors state that automated student-centred assessment systems can help students if they are appropriately integrated into the courses teaching methods. The students said that if the use of AA were optional, they would undoubtedly decide to use it.

According to Pieterse [64], AA has the potential to provide strong support to MOOC participants. However, it states that AA is not in itself a comprehensive or sufficient solution and suggests several principles for the submission and evaluation of AA:

- It is necessary to allow students to resubmit their work.
- It is impossible to expect that the outputs of students' programs will exactly match the specified output. It should be easy to allow variations in output formatting.

- It is necessary to provide qualitative feedback. If a student's program generates an error, the system should give text advice associated with the specific output.
- It is suitable to provide student's statistical information of the increments they have made. Students can be motivated by knowing what their performance is in relation to their peers.

An important element of the improvement was probably the students' activity, which increased many times between 2016 and 2019. A comparison of activity in the LMS Moodle system is shown in Figure 10.

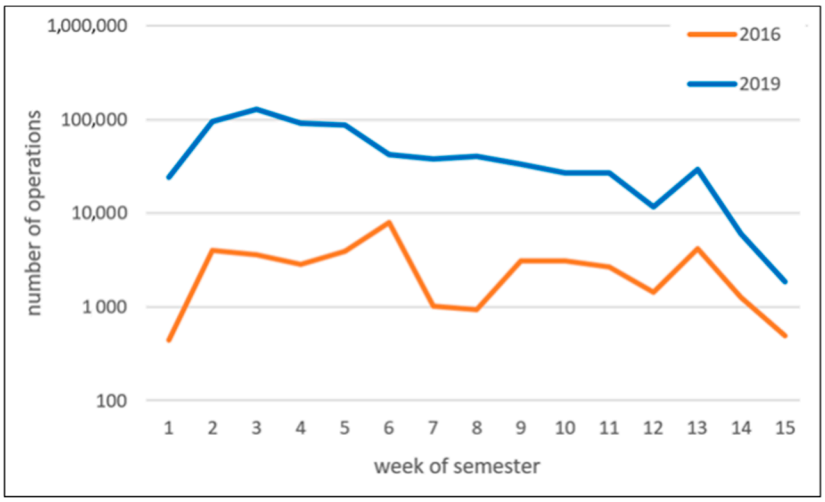


Figure 10. The activity of students expressed by the number of interactions performed in the course during the semester—the x-axis shows the weeks of the semester (from the 1st week to the last (14th or 15th) week of the semester). The y-axis represents the number of students' activities on a logarithmic scale.

An important element of the introductory course, which probably also influenced the students' results in the advanced programming course, was implementing discussions about problems and reports on incorrect or difficult to understand assignments and the subsequent debate. By discussing issues, students improve their communication skills, formulate questions accurately and use the correct terminology. During education, a simple form of gamification was also applied. Students collected points for solved tasks and badges for completing a weekly series of tasks.

The use of LMS Moodle to cover the activities grouped in the first four phases of the presented framework was satisfactory. However, the idea of using LMS Moodle for the following stages of the framework was not applicable. The development of Moodle modules requires a specific approach, the mastering of which by students is hugely time-consuming, and the benefits of mastering it are limited in practice.

For this reason, an independent software solution was chosen for the next application of the framework. The system was developed and deployed in the autumn semester 2020/2021 with the content and scope used in the presented research. In the current semester (spring 2020/2021), the students content creation phase will be started. In the following semester, these students will create system modules. PHP (Laravel) on the back-end and Vue on the frontend were selected as a development platform. This combination is more understandable for students than Moodle scripts and also provides a broader application in practice.

6. Conclusions

In contrast to other existing frameworks, the presented framework is complex—it covers university students' training in using programming languages from the introduction to the development of real applications. Introductory programming courses are understood

as only a part of the educational process focused on technological knowledge and skills development. The educational environment presented in the article as the central part of the conceptual framework is an instrument and a target of the student's training.

The first four phases of the presented framework are elaborated very well in several of the research studies mentioned in this article. The other two phases of the present framework have so far only been implemented in basic terms. Tasks requiring a comprehensive view of the issue from students were defined and implemented. Methods based on problem-based learning have been applied, and in the following semesters, which were not described in the article, activities aimed at the development of team problem-solving take place.

The implementation of the introduction part of the framework was in LMS Moodle. It used quizzes for microlearning activities and Virtual Programming Lab exercises for automatic source code evaluation. Even though the implementation of the initial stages in LMS Moodle can be considered successful, the use of Moodle encountered system limitations. The main restrictions are the static course structure, which did not support the efficient display of many objects (too slow download in the user view) and complicated work with gamification elements. Logging user activity and learning analytics, one of the most important parts for further research, did not provide detailed information about user behaviour. Obtaining detailed information about source code fixes by users was tedious. The ability to customise the user's view of the educational content was low, etc.

Implementation of the engineering parts of the framework requires in-depth knowledge about the LMS system and the use of spaghetti code in PHP, which significantly limited the possibilities of developing students' skills. The students' requirements proved to be too high, as not all students can master the structure and methods used in LMS Moodle in the third semester of study. Also, some parts of the source codes are relatively outdated, and the creation of new modules focused only on PHP is not beneficial for current practical training. Therefore, the use of LMS Moodle as an environment, which students would modify within the courses of advanced programming, was consequently rejected.

The logical step for further development was to create a stand-alone, fully adaptable system in-house that primarily supports the requirements of the framework and is based on recent popular and widely used technologies. The development of a suitable system began in 2019 based on the definition in [34] and is currently being deployed in a pilot phase.

The development of the new system will also consider the results achieved in research carried out in secondary education [65] and in the gamification for this group [66] so that the proposed system can be extended to teaching programming to younger students.

Author Contributions: Conceptualization, J.S., M.D., J.C.R.d.P. and P.S.; Data curation, J.S., M.D., L.B., J.C.R.d.P., E.S.-T., A.S. and P.T.; Formal analysis, J.S., L.B., J.K. and P.S.; Funding acquisition, J.S., M.D., L.B., J.K., J.C.R.d.P., E.S.-T., A.S., P.S. and P.T.; Investigation, J.S., M.D., L.B., J.K., J.C.R.d.P., E.S.-T., A.S., P.S. and P.T.; Methodology, J.S. and M.D.; Project administration, J.S., M.D., J.K., J.C.R.d.P., E.S.-T., A.S., P.S. and P.T.; Resources, J.S., M.D., L.B., J.K., J.C.R.d.P., E.S.-T., A.S., P.S. and P.T.; Software, J.S., L.B., J.C.R.d.P. and P.S.; Supervision, J.S. and M.D.; Validation, J.S., J.K., E.S.-T., A.S. and P.T.; Visualization, J.S. and M.D.; Writing—original draft, J.S. and M.D.; Writing—review & editing, J.S. and M.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by European Commission under the ERASMUS+ Programme 2018, KA2, grant number: 2018-1-SK01-KA203-046382 "Work-Based Learning in Future IT Professionals Education" and the Cultural and educational agency of the Ministry of Education of the Slovak Republic, grant number: KEGA029UKF-4/2018 "Innovative Methods in Programming Education in the University Education of Teachers and IT Professionals".

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Publicly available datasets were analysed in this study. This data can be found here: https://fitped.eu/images/datasets/data-framework.zip (accessed on 10 January 2021).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Henriksen, D.; Mishra, P.; Fisser, P. Infusing creativity and technology in 21st century education: A systemic view for change. *Educ. Technol. Soc.* **2016**, *19*, 27–37.
2. Kinnunen, P.; Malmi, L. Why students drop out CS1 course? In Proceedings of the ICER 2006—Proceedings of the 2nd International Computing Education Research Workshop, Canterbury, UK, 9–10 September 2006; Volume 2006.
3. Tabanao, E.S.; Rodrigo, M.M.T.; Jadud, M.C. Predicting at-risk novice Java programmers through the analysis of online protocols. In Proceedings of the ICER'11—Proceedings of the ACM SIGCSE 2011 International Computing Education Research Workshop, Providence, RI, USA, 8–9 August 2011.
4. Skalka, J.; Drlik, M. Automated assessment and microlearning units as predictors of at-risk students and students' outcomes in the introductory programming courses. *Appl. Sci.* **2020**, *10*, 4566. [CrossRef]
5. Othman, J.; Wahab, N.A. The Uncommon Approaches of Teaching the Programming Courses: The Perspective of Experienced Lecturers. In *Computing Research Innovation (CRINN)*; Lulu: Morrisville, NC, USA, 2016; Volume 1.
6. Chen, Y.; Zhang, M. MOOC student dropout: Pattern and prevention. In Proceedings of the ACM Turing 50th Celebration Conference, Shanghai, China, 12 May 2017. [CrossRef]
7. Becker, B.A.; Quille, K. 50 Years of CS1 at SIGCSE. In Proceedings of the SIGCSE '19: The 50th ACM Technical Symposium on Computer Science Education, Minneapolis, MN, USA, 27 February–2 March 2019.
8. Luxton-Reilly, A.; Becker, B.A.; Ott, L.; Simon; Giannakos, M.; Paterson, J.; Albluwi, I.; Kumar, A.N.; Scott, M.J.; Sheard, J.; et al. A review of introductory programming research 2003–2017. In Proceedings of the Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE, Larnaca, Cyprus, 1–3 July 2018.
9. Briggs, T.; Girard, C.D. Tools and Techniques for Test-Driven Learning in CS1. *J. Comput. Sci. Coll.* **2007**, *22*, 37–43.
10. Edwards, S.H. Using software testing to move students from trial-and-error to reflection-in-action. In Proceedings of the Proceedings of the SIGCSE Technical Symposium on Computer Science Education, Norfolk, VA, USA, 3–7 March 2004.
11. Miller, L.D.; Soh, L.K.; Chiriacescu, V.; Ingraham, E.; Shell, D.F.; Hazley, M.P. Integrating computational and creative thinking to improve learning and performance in CS1. In Proceedings of the SIGCSE 2014—Proceedings of the 45th ACM Technical Symposium on Computer Science Education, Atlanta, GA, USA, 5–8 March 2014.
12. Chen, T.Y.; Lewandowski, G.; McCartney, R.; Sanders, K.; Simon, B. Commonsense computing: Using student sorting abilities to improve instruction. In Proceedings of the SIGCSE 2007: 38th SIGCSE Technical Symposium on Computer Science Education, Covington, KY, USA, 7–11 March 2007.
13. Gonzalez, G. Constructivism in an introduction to programming course. *J. Comput. Sci. Coll.* **2004**, *19*, 299–305.
14. Bennedsen, J.; Caspersen, M.E. Revealing the programming process. *ACM SIGCSE Bull.* **2005**, *37*. [CrossRef]
15. Murphy, L.; Wolff, D. Creating video podcasts for CS1: Lessons learned. *J. Comput. Sci. Coll.* **2009**, *25*, 152–158.
16. Medeiros, R.P.; Ramalho, G.L.; Falcao, T.P. A Systematic Literature Review on Teaching and Learning Introductory Programming in Higher Education. *IEEE Trans. Educ.* **2019**, *62*. [CrossRef]
17. Skalka, J.; Drlík, M. Educational Model for Improving Programming Skills Based on Conceptual Microlearning Framework BT. In *The Challenges of the Digital Transformation in Education*; Auer, M.E., Tsiatsos, T., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 923–934. [CrossRef]
18. Anindyaputri, N.A.; Yuana, R.A.; Hatta, P. Enhancing Students' Ability in Learning Process of Programming Language using Adaptive Learning Systems: A Literature Review. *Open Eng.* **2020**, *10*. [CrossRef]
19. Kordaki, M. A drawing and multi-representational computer environment for beginners' learning of programming using C: Design and pilot formative evaluation. *Comput. Educ.* **2010**, *54*. [CrossRef]
20. Lee, D.M.C.; Rodrigo, M.M.T.; Baker, R.S.J.D.; Sugay, J.O.; Coronel, A. Exploring the relationship between novice programmer confusion and achievement. In *Proceedings of the Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer: Cham, Switzerlad, 2011; Volume 6974 LNCS.
21. Krusche, S.; Seitz, A. ArTEMiS—An automatic assessment management system for interactive learning. In Proceedings of the SIGCSE 2018—Proceedings of the 49th ACM Technical Symposium on Computer Science Education, Baltimore, MD, USA, 21–24 February 2018; pp. 284–289. [CrossRef]
22. Fuller, U.; Johnson, C.G.; Ahoniemi, T.; Cukierman, D.; Hernán-Losada, I.; Jackova, J.; Lahtinen, E.; Lewis, T.L.; Thompson, D.M.; Riedesel, C.; et al. Developing a computer science-specific learning taxonomy. *ACM SIGCSE Bull.* **2007**, *39*. [CrossRef]
23. Malik, S.I.; Coldwell-Neilson, J. A model for teaching an introductory programming course using ADRI. *Educ. Inf. Technol.* **2017**, *22*. [CrossRef]
24. Ali, A. A Conceptual Model for Learning to Program in Introductory Programming Courses. *Issues Inf. Sci. Inf. Technol.* **2009**, *6*. [CrossRef]
25. Skalka, J.; Drlík, M. *Conceptual Framework of Microlearning-Based Training Mobile Application for Improving Programming Skills*; Springer: Cham, Switzerland, 2018; Volume 725, ISBN 9783319751740.
26. Alshaye, I.; Tasir, Z.; Jumaat, N.F. The Conceptual Framework of Online Problem-Based Learning Towards Problem-Solving Ability and Programming Skills. In Proceedings of the 2019 IEEE Conference on e-Learning, e-Management and e-Services, IC3e 2019, Penang, Malaysia, 19–21 November 2019.
27. Krpan, D.; Mladenović, S.; Zaharija, G. The framework for project based learning of object-oriented programming. *Int. J. Eng. Educ.* **2019**, *35*, 1366–1377.

28. Khaleel, F.L.; Ashaari, N.S.; Wook, T.S.M.T.; Ismail, A. Methodology for developing gamification-based learning programming language framework. In Proceedings of the 2017 6th International Conference on Electrical Engineering and Informatics: Sustainable Society Through Digital Innovation, ICEEI 2017, Langkawi, Malaysia, 25–27 November 2017; Volume 2017.
29. Da Silva, S.J.R.; Rigo, S.J.; Diehl, P. Tri-lua: Using gamification as support learning programming language. In Proceedings of the European Conference on Games-based Learning, Paisley, UK, 6–7 October 2016.
30. Labaj, M.; Šimko, M.; Tvarožek, J.; Bieliková, M. Integrated environment for learning programming. In *Proceedings of the Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer: Cham, Switzerland, 2014; Volume 8719 LNCS.
31. Ciancarini, P.; Missiroli, M.; Russo, D. Cooperative Thinking: Analysing a new framework for software engineering education. *J. Syst. Softw.* **2019**, *157*. [CrossRef]
32. López-Fernández, D.; Alarcón, P.P.; Tovar, E. Motivation in engineering education a framework supported by evaluation instruments and enhancement resources. In Proceedings of the IEEE Global Engineering Education Conference, EDUCON, Tallinn, Estonia, 18–20 March 2015; Volume 2015.
33. Sweller, J. Cognitive Load During Problem Solving: Effects on Learning. *Cogn. Sci.* **1988**, *12*. [CrossRef]
34. Skalka, J.; Drlik, M. Priscilla—Proposal of System Architecture for Programming Learning and Teaching Environment. In Proceedings of the 2018 IEEE 12th International Conference on Application of Information and Communication Technologies (AICT), Almaty, Kazakhstan, 17–19 October 2018. [CrossRef]
35. Žufic, J.; Jurcan, B. Micro Learning and EduPsy LMS. In Proceedings of the Central European Conference on Information and Intelligent Systems, Varazdin, Croatia, 23–25 September 2015; Volume 2015, pp. 115–120.
36. Jomah, O.; Masoud, A.K.; Kishore, X.P.; Aurelia, S. Micro Learning: A Modernised Education System. *Brain. Broad Res. Artif. Intell. Neurosci.* **2016**, *7*, 103–110.
37. Grevtseva, Y.; Willems, J.; Adachi, C. Social media as a tool for microlearning in the context of higher education. In Proceedings of the 4th European Conference on Social Media, ECSM 2017, Vilnius, Lithuania, 3–4 July 2017.
38. Polasek, R.; Javorcik, T. Results of pilot study into the application of microlearning in teaching the subject computer architecture and operating system basics. In Proceedings of the 2019 International Symposium on Educational Technology, ISET 2019, Hradec Králové, Czech Republic, 2–4 July 2019.
39. Lim, C.; Ryu, J.; Martindale, T.; Kim, N.; Park, S. Learning, Design, and Technology in South Korea: A Report on the AECT-Korean Society for Educational Technology (KSET) Panel Discussion. *TechTrends* **2019**, *63*. [CrossRef]
40. Jones, N.D.; Gomard, C.K.; Sestoft, P. *Partial Evaluation and Automatic Program Generation*; Prentice-Hall, Inc.: Upper Saddle River, NJ, USA, 1999; ISBN 0130202495.
41. Selby, R.W.; Porter, A.A. Learning from Examples: Generation and Evaluation of Decision Trees for Software Resource Analysis. *IEEE Trans. Softw. Eng.* **1988**, *14*. [CrossRef]
42. Daly, C. Roboprof and an introductory computer programming course. *ACM SIGCSE Bull.* **1999**, *31*, 155–158. [CrossRef]
43. Zheng, J.; Williams, L.; Nagappan, N.; Snipes, W.; Hudepohl, J.P.; Vouk, M.A. On the value of static analysis for fault detection in software. *IEEE Trans. Softw. Eng.* **2006**, *32*. [CrossRef]
44. Rodríguez-del-Pino, J.C.; Rubio-Royo, E.; Hernández-Figueroa, Z. A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features. In Proceedings of the Conference on e-Learning, e-Business, Entreprise Information Systems e-Government, Las Vegas, NV, USA, 16–19 July 2012.
45. Fernández Alemán, J.L. Automated assessment in a programming tools course. *IEEE Trans. Educ.* **2011**, *54*. [CrossRef]
46. Wilcox, C. The role of automation in undergraduate computer science education. In Proceedings of the SIGCSE 2015—Proceedings of the 46th ACM Technical Symposium on Computer Science Education, Kansas City, MO, USA, 4–7 March 2015.
47. Skalka, J.; Drlik, M.; Obonya, J. Automated Assessment in Learning and Teaching Programming Languages using Virtual Learning Environment. In Proceedings of the IEEE Global Engineering Education Conference (EDUCON2019), Dubai, United Arab Emirates, 8–11 April 2019. [CrossRef]
48. Jenkins, T. On the Difficulty of Learning to Program. In Proceedings of the 3rd Annual Conference of the LTSN Centre for Information and Computer Sciences, Loughborough, UK, 23–29 August 2002; Volume 4, No. 2002. pp. 53–58.
49. Gomes, A.J.; Santos, Á.N.; Mendes, A.J. A study on students' behaviours and attitudes towards learning to program. In Proceedings of the Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE, Haifa, Israel, 3–5 July 2012.
50. Gaudencio, M.; Dantas, A.; Guerrero, D.D.S. Can computers compare student code solutions as well as teachers? In Proceedings of the SIGCSE 2014—Proceedings of the 45th ACM Technical Symposium on Computer Science Education, Atlanta, GA, USA, 5–8 March 2014.
51. Barra, E.; López-Pernas, S.; Alonso, A.; Sánchez-Rada, J.F.; Gordillo, A.; Quemada, J. Automated Assessment in Programming Courses: A Case Study during the COVID-19 Era. *Sustainability* **2020**, *12*, 7451. [CrossRef]
52. Gordillo, A. Effect of an instructor-centered tool for automatic assessment of programming assignments on students' perceptions and performance. *Sustainability* **2019**, *11*, 5568. [CrossRef]
53. Tovar, E.; Soto, Ó. Are new coming computer engineering students well prepared to begin future studies programs based on competences in the european higher education area? In Proceedings of the Frontiers in Education Conference, FIE, San Antonio, TX, USA, 18–21 October 2009.

54. Bekki, J.M.; Dalrymple, O.; Butler, C.S. A mastery-based learning approach for undergraduate engineering programs. In Proceedings of the Frontiers in Education Conference, FIE, Seattle, WA, USA, 3–6 October 2012.
55. Bloom, B.S. *Taxonomy of Educational Objectives Book 1: Cognitive Domain*; Addison Wesley Publishing Company: Boston, MA, USA, 1984; ISBN 0582280109.
56. Anderson, L.W.; Krathwohl, D.R.; Bloom, B.S. *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*; Longman: Harlow, UK, 2001.
57. Kafai, Y.B. *Constructionism in Practice: Designing, Thinking, and Learning in A Digital World*; Routledge: Abingdon-on-Thames, UK, 2012.
58. Palloff, R.M.; Pratt, K. *Collaborating Online: Learning Together in Community*; John Wiley & Sons: Hoboken, NJ, USA, 2010; Volume 32.
59. Wenger, E. Communities of practice and social learning systems: The career of a concept. In *Social Learning Systems and Communities of Practice*; Springer: London, UK, 2010.
60. Hunicke, R.; Leblanc, M.; Zubek, R. MDA: A formal approach to game design and game research. In Proceedings of the AAAI Workshop—Technical Report, San Jose, CA, USA, 25–29 July 2004; Volume WS-04-04.
61. Chou, Y.-K. *Actionable Gamification: Beyond Points, Badges, and Leaderboards*; Octalysis Media: Fremont, CA, USA, 2016.
62. Wise, A.F.; Vytasek, J.; Hausknecht, S.; Zhao, Y. Developing learning analytics design knowledge in the "middle space": The student tuning model and align design framework for learning analytics use. *Online Learn. J.* **2016**, *20*. [CrossRef]
63. Skalka, J.; Benko, L.; Boryczka, M.; Landa, J.; Rodríguez-del-Pino, J.C. *Java Fundamental*; FITPED: Nitra, Slovakia, 2020. [CrossRef]
64. Pieterse, V. Automated Assessment of Programming Assignments. In Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research, Heerlen, The Netherlands, April 2013; pp. 45–56.
65. Papadakis, S.; Kalogiannakis, M.; Orfanakis, V.; Zaranis, N. The appropriateness of scratch and app inventor as educational environments for teaching introductory programming in primary and secondary education. *Int. J. Web-Based Learn. Teach. Technol.* **2017**, *12*, 58–77. [CrossRef]
66. Papadakis, S.; Kalogiannakis, M. Using gamification for supporting an introductory programming course. The case of classcraft in a secondary education classroom. In *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*; Springer: Cham, Switerland, 2018; Volume 229.