applied sciences

MDPI

Article

Automated Assessment and Microlearning Units as Predictors of At-Risk Students and Students' Outcomes in the Introductory Programming Courses

Jan Skalka * and Martin Drlik

Department of Informatics, Faculty of Natural Sciences, Constantine the Philosopher University in Nitra, 94974 Nitra, Slovakia; mdrlik@ukf.sk

* Correspondence: jskalka@ukf.sk

Received: 31 May 2020; Accepted: 25 June 2020; Published: 30 June 2020

Abstract: The number of students who decided to study information technology related study programs is continually increasing. Introductory programming courses represent the most crucial milestone in information technology education and often reflect students' ability to think abstractly and systematically, solve problems, and design their solutions. Even though many students who attend universities have already completed some introductory courses of programming, there is still a large group of students with limited programming skills. This drawback often increases during the first term, and it is often the main reason why students leave study too early. There is a myriad of technologies and tools which can be involved in the programming course to increase students' chances of mastering programming. The introductory programming courses used in this study has been gradually extended over the four academic years with the automated source code assessment of students' programming assignments followed by the implementation of a set of suitably designed microlearning units. The final four datasets were analysed to confirm the suitability of automated assessment and microlearning units as predictors of at-risk students and students' outcomes in the introductory programming courses. The research results proved the significant contribution of automated code assessment in students' learning outcomes in the elementary topics of learning programming. Simultaneously, it proved a moderate to strong dependence between the students' activity and achievement in the activities and final students' outcomes.

Keywords: introductory programming courses; dropout prediction; automated assessment; source code evaluation; microlearning

1. Introduction

The current demand for experts in information technology (IT) as well as the prognosis of the future development on the labour market cause not only the constant growth of computer science education popularity, but also a continual demand for improving the IT skills of the large group of graduates who enter the labour market every year.

The number of students who decided to study IT-related study programs is continually increasing. Higher educational institutions which train the future IT professionals in different study programs react to this situation differently. Many universities admit as many students to study IT-oriented study programs as their capacities allow, besides considering their current ranking and the position in the country or worldwide. Consequently, they often expect that this number of students will naturally decrease during the first months of the term. Even though this process can be considered natural, it opens the discussion, how to teach this large group of newcomers with a different introductory level of IT skills effectively and how to set up the safety net for the at-risk students with higher predisposition to leave a study too early.

Introductory programming courses represent the most crucial milestone in IT education and often reflect students' ability to think abstractly and systematically, solve problems, and design their solutions. Therefore, the level of knowledge of the developers and similar experts (IT specialists, data science specialists, specialists of Internet of Things (IoT) area, etc.) can be considered a key benefit for the emerging labour market.

The required skills of novice programmers can be described as a set of skills learned simultaneously, e.g., semantics, the syntax of languages, problem-solving, computational thinking etc. [1,2]. Knowledge of one or more programming languages, or algorithmic thinking in general, are considered one of the critical IT skills. Even though many students, who come to the universities, have already attended some introductory courses of programming, there is still a large group of students with limited programming skills. This drawback often increases during the first term and is often the main reason why students leave study too early.

There are many approaches, which can be used for identification of the IT students with a higher predisposition to dropout [3]. Even though weak programming skills are not the only reason for drop out of studies, they often represent the most important one. Therefore, it is natural to assume that the detailed analysis of the students' behaviour in introductory programming courses can lead to some relevant indicators, which can estimate the dropout rate as soon as possible, identify at-risk students, and simultaneously help teachers to find a suitable form of intervention.

There is increasing interest in gathering and analysing this data to learn about how students behave. An understanding of student behaviour has a value to educators in predicting success in examinations and other assessments, identifying student difficulties and interventions designed to mitigate them, designing tools that respond to specific behaviours in a way that is helpful to the students [4].

There is a myriad of technologies and tools which can be successfully involved in this process. They allow a more straightforward application of modern educational approaches also to this area of education. As a result, they can eliminate many problems of teaching and learning introductory programming identified in a previous decade by the implementation of new functions directly into integrated development environment (IDE) or learning environments. As an example, the syntax and basic semantic elements of programming languages are tracked continually and verified during the source code writing. Therefore, the students can use code completion, hints showing parameters, as well as a short explanation and focus directly on developing their programming thinking [4].

Many universities implemented these technologies and tools, e.g., different massive open online courses (MOOCs) or learning management systems (LMSs), to support different learning forms. These systems serve as the repositories of the curated educational content at least. Moreover, in case of more advanced IT courses allow submitting the programming assignments and their automated evaluation, writing programming code directly in the embedded editor with syntax highlighting, code peer-reviewing, advanced testing, etc. The integrated functions of LMS and their extensions usually support not only progress monitoring but also many activities, quizzes and content sources integration. Students' activities and achievements are monitored, analysed with appropriate statistical methods, and suitably visualised on personalised dashboards (Figure 1).

It can be assumed that the activities, which require active student's involvement in learning, can bring more relevant results in comparison with the observation of the passive student's presence in the course.

Current technological advancements that allow automated evaluation of the source code written by the student and provide immediate feedback, integrated with other approaches utilising e-learning and microlearning advantages were selected for these study, while they have a potential to engage the students to be more active during their study in introductory programming courses.

Figure 1. An example of the visualisation of the progress monitoring in one of the LMS Moodle courses used in the research. It is possible to identify users with low activity in course.

Therefore, the introductory programming course used in this study has been extended with the automated evaluation of the programming assignments followed by the implementation of a set of suitably designed microlearning units with the following aims:

- To analyse the relationship between the input student's characteristics and his/her final achievements.
- To evaluate the overall contribution of automated evaluation of the programming assignments closely interrelated to the set of suitably designed microlearning units to the students' final grades.
- To research the role of these two elements as possible predictors of identification of at-risk students who fail the introductory programming course.

While the automated programming code evaluation and microlearning units were implemented subsequently during several years, it was possible to collect four different datasets of students' achievements and their activities, which characterise the students' behaviour in the introductory programming course and can be compared with each other.

The structure of the article is as follows. The related work section summarises the current state of the introductory programming education research and emphasises that the understanding the learning process, identifying of the factors, which influence the students' outcomes and the ability to predict the at-risk students are still actual. The third section describes the background of the research, defines the research questions and used methods. The next 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

Becker et al. [5] analysed the evolution of the introductory programming education research over fifty years thoroughly. They confirmed that the design and structure of the course, as well as automated assessment and student's retention and predicting their success based on course enhancement using new tools and technologies still belong to the main categories of the research with increasing interest.

This statement also confirms a comprehensive report [4], which analyses recent trends across the breadth of introductory programming over the last 15 years. While other authors estimated that the

dropout rate among students is about 30–40% [6,7], this study indicates that dropout rates among computing students are not alarmingly high. It has been suggested that the difficulties faced by novices may be a consequence of unrealistic expectations rather than intrinsic complexity of the course irrespective of used educational approaches, tools and technologies.

Many studies have been conducted that tried to identify factors related to success in the learning of programming. Among the identified student characteristics that may contribute to student success in introductory programming courses are prior programming experience, gender, secondary school performance and dislike of programming, intrinsic motivation and comfort level, high school mathematics background, attribution to luck for success/failure, formal training in programming, and perceived understanding of the material [6,8].

Hellas et al. [9] present a systematic literature review of work in the area of predicting student performance. Their analysis shows an increasing amount of research in this area, as well as an increasing variety of techniques used. They did not analyse the relationship between an automated assessment as well as microlearning and predicting students' performance in their review explicitly.

Another systematic review of predicting students' learning outcomes is provided by [10]. The authors identified 14 prediction targets. The student retention/dropout was the second most frequent. Among the feature types used, 151 unique feature types were identified. Student record and performance in the current course, as well as activity and course features and learning behaviour, belong to the most frequent.

Tabanao et al. [6] tried to quantify indicators of novice programmer progress in the task of writing Java programs through the analysis of online protocols of errors. They evaluated the use of these indicators for identifying at-risk students. Even though the derived models could not accurately predict the at-risk students, the authors stated, that described approach can identify the students, who need intervention.

The question of why students drop a computer science course was broadly examined by Kinnunen [7]. The results indicate that several reasons affect students' decision to quit a course. They analysed social, cultural, demographic factors and their impact on students' dropout rate using the interview. The most frequent reasons were the lack of time to make exercises and motivation, firmly joined to the required level of knowledge, which often led to frustration. However, these reasons were also affected by factors such as the perceived difficulty of the course, general difficulties with time managing and planning studies, or the decision to prefer other courses. This study shows that the complexity and extensive variety of factors are involved in students' decision to drop the course. This indicates that simple actions to improve teaching or organisation on a course to reduce the dropout rate may be ineffective. Efficient intervention to the problem requires a combination of many different actions that take into consideration the versatile nature of reasons involved in drop out. They again did not research the impact of educational approach or technology on dropout rate or students' success.

Measuring prior programming knowledge and its impact on the outcomes is the main aim of the paper written by Duran et al. [11]. They proposed and evaluated a novel self-evaluation instrument for measuring prior programming knowledge in introductory programming course to bridge the gap between ad-hoc surveys and standardised tests.

Several groups of researchers developed automated methods for identifying at-risk students early in the semester. Many of these techniques are based on automatically monitoring student activity as the students develop code or advanced data mining and predictive modelling techniques [12]. The research in this area indicates that early success in an introductory programming course is vital for a student's ultimate success. Many researchers have shown that students having difficulty can be identified early in the semester.

An automated assessment (AA), which is in the middle of the interest of this study, represents a tool that allows checking source code automatically and brings a new perspective on learning. The automated assessment of programming assignments has been practised since programming has

been taught, especially in the introductory programming courses. It has different features, which are automatically assessed by various assessment tools and systems [13].

Considering the survey about the AA for the novice programmers in the introductory courses [14], the following base conditions should be fulfilled for successful implementation of AA:

- a clear and unambiguous specification should define the requirement for program work and results, with simultaneous checking whether the programs meet the specification or not;
- sufficient testing of the assignment should be ensured, as each bug disturbs and demotivates beginners;
- environmental support for the discovery of syntactic and semantic errors should exist,
- a suitable mechanism for dealing with the programs with syntactic and semantic errors should be created;
- immediate and corrective feedback for the submitted assignments; feedback helps to build students' knowledge, and habits should be available.

According to Skalka et al. [15] and Rodriguez-del-Pino et al. [16], AA is beneficial for the following reasons:

- the student gains immediate feedback whether the program is correct, and students can use their own pace,
- the teacher gains extra time, instead of time wasted by checking the assignment and identifying and re-explaining repeated errors in past,
- it is possible to teach large groups of students without increasing the demands on teachers, which apply especially in the case of MOOC courses,
- the learning process is more efficient and, due to the errors tracking, speed and quality of the solutions, the individual parts of the process can be fragmented, quantified and described (problematic topics, problematic examples, number of necessary attempts, etc.).

Any of the above-mentioned studies did not provide proof that AA has a significant impact on the students' achievements or significantly reduce a student dropout rate. A large number of studies measure the effectiveness of assessment tools in helping students to write the program and perform the solution [17]. Although AA has been extensively used in programming lessons for several years, research focused on their contribution to programming learning is still limited.

Keuning [18] performed a systematic literature review to find out what kind of feedback is provided, which techniques are used to generate the feedback, how adaptable the feedback is, and how these tools are evaluated. They found that feedback, including AA, mostly focuses on identifying mistakes and less on fixing problems.

According to Akahane et al. [19], the practical results of AA implementation in MOOCs are not trivial. The educational goal can only be achieved through the availability of many high-quality assignments, coupled with coaching in the form of guaranteed rapid and accurate feedback. Applying automated assessment itself is not a solution.

Figueiredo [20] stated that there had been an intense research activity in studying the difficulties in teaching and learning introductory programming. Achieving success or failure early in the introductory programming course is directly related to student continuity in the course of computer science. They proposed a system that allows to suggest exercises and to evaluate the results automatically. They did not examine the proposed approach as a suitable predictor of student's dropout or achievement.

Two groups with and without using AA were compared in the research of Pieterse [21]. The scores of the experimental group showed that students who worked with AA gained a more solid grasp of concepts related to debugging, deployment, and versioning. However, the difference between the means of groups was not statistically significant.

The research conducted by Rubio-Sánchez [22] was focused on improving the average final exam scores using an automated grading (particularly AA). It concluded with the result that the automation

of key processes such as program grading could save a significant amount of scarce resources in introductory courses without a negative impact on academic performance. The author presented significantly higher exam scores for students who used automated grading.

The next research [23] used the Mooshak system, which did not produce clear results concerning whether it helps to reduce the dropout rate or improve the students' achievements. Students use it for testing their programming code and consider that AA deployment is a good idea.

The microlearning is defined as an action-oriented approach offering bite-sized learning that gets learners to learn, act, and practice [24]. Microlearning refers to a didactic concept and approach that uses digital media to deliver small, coherent and self-contained content for short learning activities [25]. Microlearning has become popular, mainly due to the expansion of mobile devices and brings new possibilities for tracking user activities. The development level of front-end web frameworks has made microlearning no longer limited to mobile devices. Web browsers offer the same comfort and quality not only for content displaying and interacting but also for tracking user activity [24].

The content is organised in short lessons (1–5 min) and consists of two types of activities: provision of information (explanation of one idea, concept, or procedure) and verification of understanding (a simple question usually connected to information presented in the previous step). The idea of microlearning is based on every-where and every-time learning and support of learning in small and concluded parts. The goal is to engage students in deeper learning and encouraging them to think about and work with the content of subject/course in their everyday lives. The learning scenario preparation is the most crucial step in building microlearning content because didactic shortcomings in providing content often mean student failure. Course developers should bear in mind that the student receives the content in a dense form, often without linking to the previous lesson, and the time delay between the lessons can be several days [26].

Predicting learning success based on the usage of learning materials similar to microlearning units is described by Leppänen et al. [27]. The authors explore students' usage of online learning materials as a predictor of academic success. They focused on time spent with each paragraph of the online learning material. The authors found that such time can be considered a moderate predictor of student success even when corrected for student time-on-task and that the information can be used to identify at-risk students. In other words, course material usage can be used to predict academic success.

Although the impact of microlearning on students outcomes has been extensively studied in last decade, i.e., its positive or negative impact on learning programming, consideration of the student dropout rate is quite rare and therefore will be analysed later herein [4,5].

The logs of the students' activity in AA and microlearning units are stored similarly as other resources and activities in the e-learning course created in LMS Moodle. The logs will be used later for a possible explanation of the presented study findings considering the fact, that they can uncover the reason, why AA or microlearning have or have not the expected effect. Different resources, including LMS Moodle, have also been analysed in [3] and the academic performance of students in a blended learning form was also analysed. The authors identified a set of variables with well-known predictive value in dropout courses in general.

Many researchers focused their research on the early prediction of the final grade and identification of the threats of dropout by application of different log mining techniques on introductory programming courses [28–30]. These studies analyse all activities of the stakeholders involved in the e-learning course, research their behaviour in different types of course activities, as well as compare different periods of the term. Although these approaches provide some promising results, they usually catch the real students' activity in the course and their interaction with the content and course activities only partially. Moreover, they often aggregate the students' data with different grades to larger groups and compare the behaviour of these groups.

Ihantola et al. [31] provide an overview of the body of knowledge regarding the use of educational data mining and learning analytics focused on the teaching and learning of programming. They defined five challenges, which provide a framework for systematic research in effective using learning analytics

and educational data mining in teaching programming. They identified a learning environment category, which was related to tools and automated testing, grading and feedback, but they did not focus on microlearning or other for educational content.

Sisovic et al. [32] applied educational data mining and learning analytics techniques in order to find out what impacts success in programming. The research was conducted on the dataset compounded of extracted logs from the LMS Moodle and data related to prior knowledge and students' preparation for the study. Classification methods were used to detect connections between prior knowledge and Moodle course activity in relation to final grades. They compared the impact of prior knowledge with the effect of Moodle activity on passing or failing the introductory course of programming. The study showed that some activities and the preparatory seminar contributes to the course success and decreasing of students' dropout rate.

The problem with inactive and low-performance students in e-learning courses was thoroughly analysed in the research conducted by Hussain et al. [33]. They analysed logs from the LMS Moodle and compared several machine learning techniques to predict the low-performance students and identify active and inactive students. The results can have a significant contribution to the LMS Moodle to enable the instructor to focus on groups of the low-performance and inactive students and then give them appropriate feedback. As a result, they assume decreasing student failure and dropout rate in programming courses.

3. Materials and Methods

The authors continually investigated various methods, aspects and approaches to teaching introductory programming courses [24,34]. Over the four years, the authors of the Java introductory programming course introduced new elements into the study every year and inspected their impact on student learning outcomes.

Gradual application of new educational objects into programming courses is implemented to eliminate the reduction of admission requirements to cover the needs of the labour market. The main reason for research is the rising dropout of students in this course and the deterioration of the average results of the subject (Table 1).

Table 1. Structure of final grades and dropout.

Group	Students	A	B	C	D	E	Fx	Dropped Out (%)
2016	51	8	6	4	10	8	15	29.4%
2017	70	12	7	18	7	10	16	22.9%
2018	82	8	5	15	12	8	34	41.5%
2019	102	8	3	18	13	10	50	49.0%

3.1. The Educational Content

The most effective approach to getting the ability on how to solve algorithms and programming problems is training. Ability to acquire these skills depends significantly on the curriculum, and the structure of the programming course, student's willingness and motivation to write many programs. The essential requirement for the long-term quality assurance is the maintenance and continual improvement of the course using instructional design approach, which reflects most of the stakeholders' expectation, divides course content according to the course objectives, types of available resources and course assessments [35]. The suggested approach to reach the best effort is the following:

- develop innovative methodologies in the course;
- make lectures mandatory for the students;
- provide for immediate feedback as soon as possible.

The structure of the introductory programming course is based on the combination of introduction to procedural programming, object-oriented programming, and the graphical user interface (GUI) development.

The procedural programming part lasts five weeks and consists of:

- introduction to Java; variables; input and output;
- conditions; loops; primitive variables;
- string type; nested cycles; syntax and semantic;
- errors and exceptions; indexes and arrays; random numbers; switch structure;
- 2D arrays; streams and files.

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

- introduction into the concept of object-oriented programming; classes, methods and parameters;
- setters; constructors; implicit constructor, overloading etc.;
- static variables, static objects manipulation (e.g., counter);
- the definition of inheritance, relationships between objects and classes, superclass; constructor of superclass;
- polymorphism, overriding, application of inheritance and its principles.

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 subjects covered programming are educated 6 h per week: 2 h lessons, 2 h for the main exercise, and 2 h for a voluntary programming seminar. The formal study can be complete by individual exercises and/or consultations.

The e-learning course in LMS Moodle fully supports the present form of the study. The educational content is primarily composed of interactive activities that require active student participation. The e-learning lessons usually consist of parts presented in Figure 2:

- ppt presentation—presented on lessons;
- video-lectures and video-exercises—recorded on the lessons divided into more topics or chapters;
- AA activities—contains programming assignments and auto evaluation mechanism;
- microlearning activities implemented as quizzes in the course of 2019/2020;
- forums—placed for some summary chapters.

Two practical tests are required to graduate for the successful completion of the course. The difficulty of the tests is determined to consider the knowledge and programming speed of students. The tasks are designed to be demanding of the best students with the aim to differentiate performance in the group of the best ones. Based on many years of experience and experimentation, the limit for passing the test was set at 40% of points.

The final exam consists of a quiz focused on an understanding of written programs and a few essential questions of programming theory. The students passed quiz successfully proceed to the oral examination consists of debate about random topics of programming lessons.

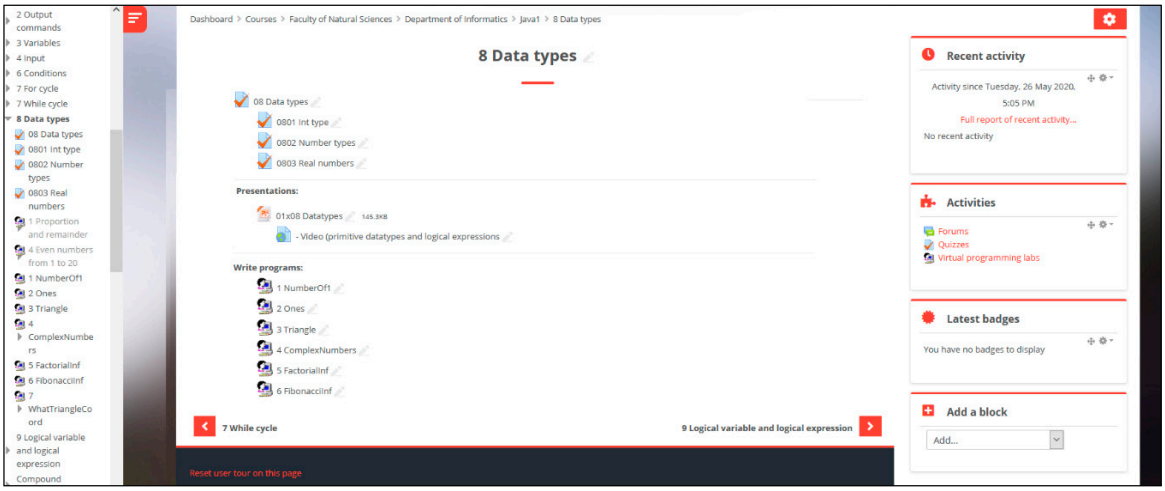


Figure 2. The view of the chapter Data types in the English version of the course of 2019/2020. It consists of quizzes, presentation file, link to the video placed on YouTube and automated source code evaluation activities.

The educational material was changed over four years. Students were provided with the following educational materials in the first year of the experiment (2016/2017):

- presentations of lectures; solved assignments (finished source codes); video recordings of selected programs; video lectures.

Interactive elements were represented by:

- every week's quiz focused on the understanding of topics; voluntary bonus tasks (program development); final tests.

The educational materials were extended in the second year of the experiment (2017/2018) by interactive elements represented by automatic assessment with the ability to solve it anytime, anywhere. The part of structural programming was covered.

The interactive content of the courses was enriched in the third year of the experiment (2018/2019) by automatically evaluated assignments were expanded to all lessons of course (more than 170 exercises), and requirements for successful completion of the semester were expanded by the need to obtain at least 50% of the points in these assignments.

Finally, the interactive content was also enriched in the last year of the experiment (2019/2020) by microlearning activities implemented by quiz object (more than 500 microlessons) and requirements for successful completion of the semester were set to obtain at least 80% of the points in these activities. The structure of a typical quiz is presented in Figure 3.

The gradual building of the structure and content of the courses used in the experiment presents Figure 4.

The department of the authors is one of the departments that actively use LMS tools in the present form of study for many years [36]. The student's activity in the course and the use of LMS activities, in general, are easily traceable. Students used e-learning course as a source of educational resources, activities, exercises, and tests. Using an LMS made it possible to monitor their activity, successes and failures, and overall results of a study.

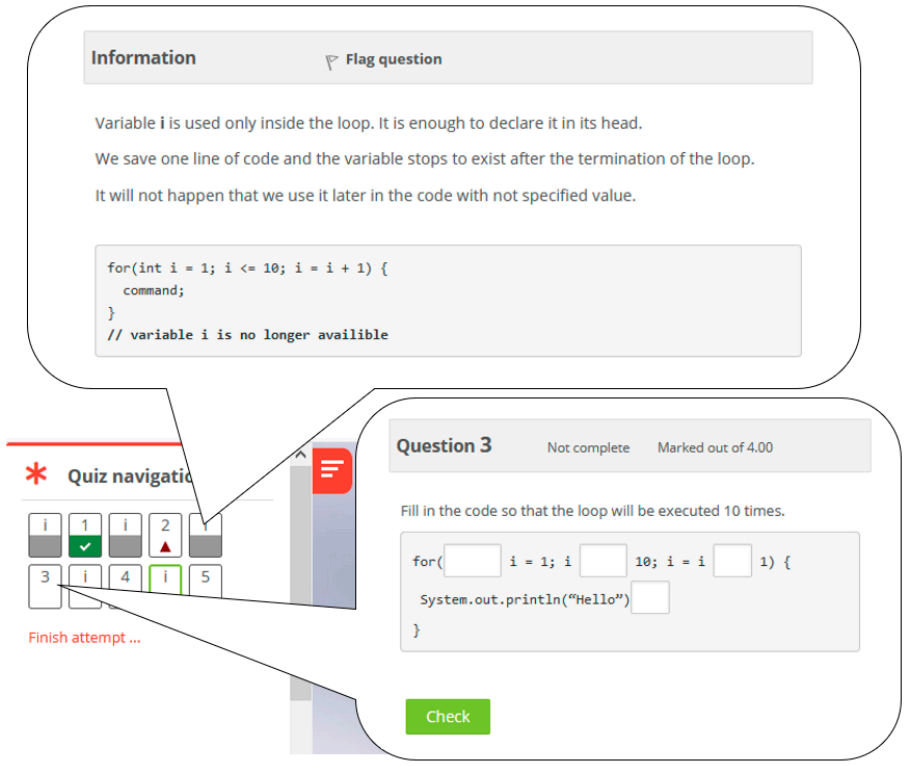


Figure 3. View to the microlearning structure and content represented by the quiz. The suitable rotation of content activities and interactive activities is ensured by the sequence of different types of questions in the quiz.

	2016	2017	2018	2019
sources:	presentations video lectures solved programing assignments	presentations video lectures solved programing assignments	presentations video lectures solved programing assignments	presentations video lectures solved programing assignments **microlessons - content**
activities:	every-week summarisation quiz	every-week summarisation quiz **automated assessment (procedural programming)**	every-week summarisation quiz automated assessment (procedural programming) **automated assessment (class programming)**	**microlessons - quiz** automated assessment (procedural programming) automated assessment (class programming)

Figure 4. The process of content building—the changes between inspected years.

Virtual programming Lab for Moodle [21] (VPL) (Figure 5) was selected as the most appropriate tool for source code evaluation. VPL is the system of the third generation of automated assessment with the support of many common programming languages. The validated code is running in a safe sandbox, and VPL is integrated directly into the Moodle as a plugin. Gaining a complete assessment report and history of validated files (what is fundamental in introductory programming courses) are possible using this tool.

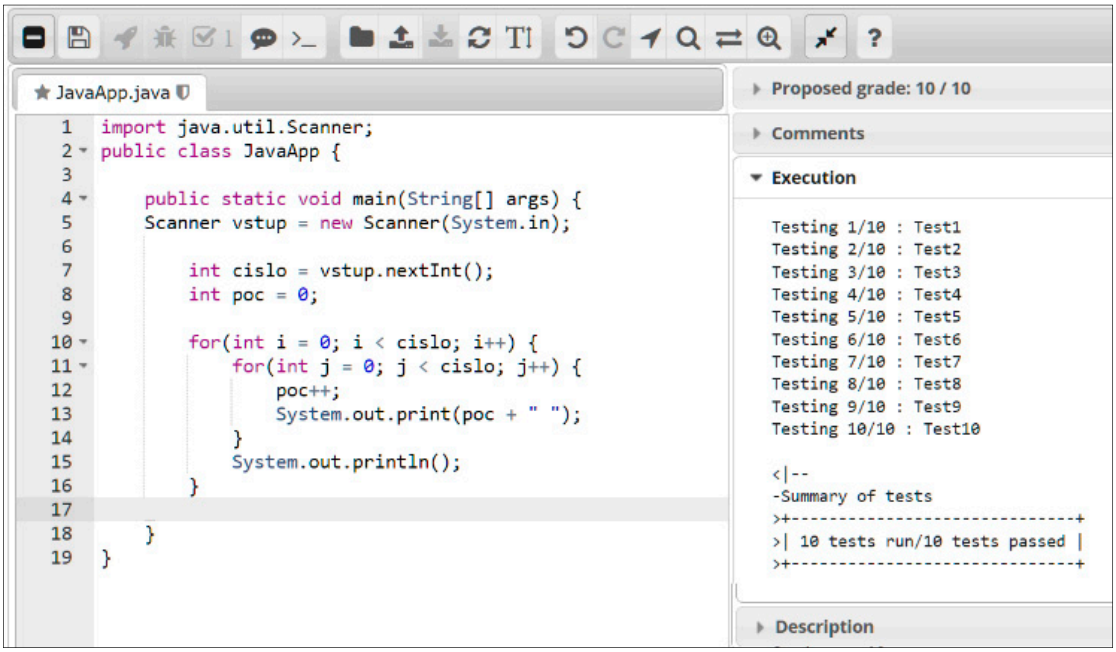


Figure 5. Virtual programming lab for Moodle [21].

3.2. The Goals

The goals of the research are divided into three consecutive parts focused on the application of the interactive activities of students.

- The identification and estimation of the impact of selected interactive activities in the LMS environment on the final test results. The significance of the automatic evaluation of programs and using microlearning activities are separately investigated.
- The relationship between the achieved results expressed by the obtained points from the tests and the interim results of the writing programs and solving microlearning activities will be identified.
- The possibility of identifying problem students since their activities within interactive activities, or the possibility of estimating their marks on the basis of their results in these activities will be presented.

3.3. Research Questions

In accordance with changes in content structure, the research questions are based on a comparison of educational outcomes measured through the tests. The content inspection will be divided into two parts: introductory to programming (structural part) and classes (object-oriented part). The hypotheses inspect approaches to minimise dropout.

Hypothesis 1 (H1). *The use of LMS objects for the automatic evaluation of source codes improves students' results in the area of introduction to programming.*

A comparison of 2016 and 2017 groups will be used to verify this hypothesis. In 2017, the LMS object "virtual programming lab" was used for the first time and its scope covered parts of the Java language aimed at explaining the basic control and data structures. This hypothesis should verify the importance of using automatic programs evaluation for students beginning programming.

Hypothesis 2 (H2). *The use of LMS objects for the automatic evaluation of source codes improves students' results in object-oriented programming.*

A comparison of groups 2017 and 2018 will be used to verify this hypothesis. In 2018 has been programming course extended by parts covered object-oriented chapters. The number of LMS virtual programming labs increased to 170, which represents a burden on students with 15–20 assignments per week. The hypothesis should verify the contribution of VPL to teaching advanced programming topics.

Following the division of the course into two parts, other hypotheses will be examined to verify the effectiveness of microlearning.

Hypothesis 3 (H3). *The use of microlearning principles improves students' results in the area of introduction to programming.*

Hypothesis 4 (H4). *The use of microlearning principles improves students' results in object-oriented programming.*

In order to find elements that allow predicting the risk of unsuccessful completion of the course, the existence of dependence between the success of students in individual activities and the overall success of the course will be identified. The following dependencies will be inspected:

- the relationship between the results of entrance exams (points of secondary level of study) and the result of the course,
- the relationship between course activity and course achievement, and
- the relationship between the results in the VPL and microlearning activities and the result in the course.

3.4. Characteristics of the Respondents

Comprehensive research has been carried out on a sample of 51–102 students per year. The structure of the student's groups attending the introductory programming courses is rather diverse because of different skills reached in secondary schools, various experience with programming, and different level of computational thinking. It is very complicated to define homogenous groups with optimal size and start teaching on the different entrance level. The students with excellent programming skills as well as students who have never made any program have the same starting position.

The students who participated in the experiment were divided into four groups. Every group consists of computer science first-year students at the age of 18–24. Only students who have studied the subject for the first time are in the groups. Students who repeat the study of programming were excluded from the research.

The entrance results of student groups were obtained from university applications and reflected the grades acquired in high-school study and secondary school competitions. These results are rated on a scale 0–40. The statistical characteristics of the entrance results are presented in Figure 6. A year-on-year decline in students' initial scoring can be seen at first glance. This situation may be partly due to demography and partly to an increase in the number of first-year students.

3.5. Measurement Tools

The main tools used for research were:

- the list of admitted students with the number of points awarded in the admission process used primarily in pretest;
- the test results of students in two tests realised after introductory course part and object-oriented course part used for evaluation of used objects and method;
- LMS Moodle log used for the assessment of the activities of the students and dependency identification; the parts of the log were processed to use for relevant calculation.

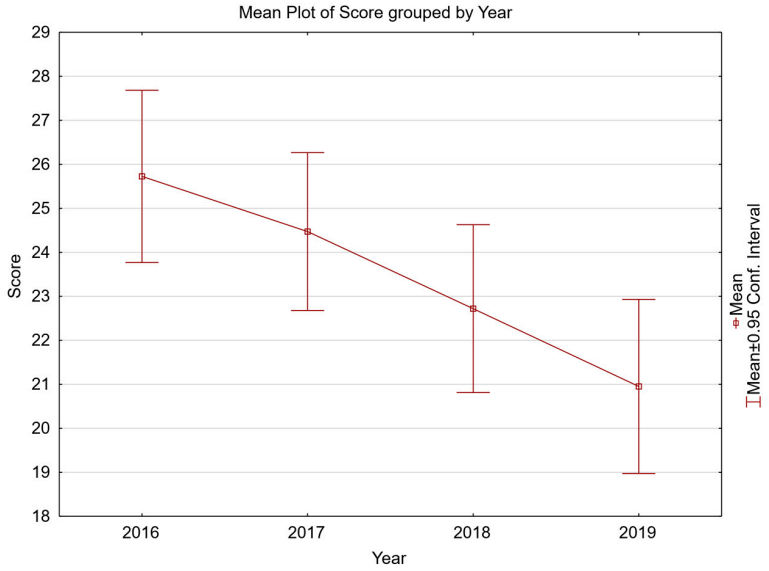


Figure 6. Graphical visualisation of statistical characteristics of groups (points awarded in the admission procedure).

4. Results

The differences between the results of students' groups for four years were inspected in the following research. The groups learnt via different educational sources, which are described in part as the educational content.

4.1. Pretest

The first step before the research is proof that student's groups were well-modelled by a normal distribution according to the level of their performance at the high school. These values are reflected by the score obtained in the admission procedure. Simultaneously, these values do not depend on the previous students' experience in programming and thus do not distort the prerequisites for mastering the course. The range of awarded points was between 0 and 40.

The Kolmogorov–Smirnov test was used to prove that the distributions of groups match the characteristics of a normal distribution. All the essential characteristics are listed in Table 2.

Table 2. 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
2017	70	24.47	24	7.52	0.10	0.45
2018	82	22.72	23	8.68	0.09	0.55
2019	102	20.95	21.5	10.07	0.08	0.50

The data in every group of students do not differ significantly from a normal distribution. The *p*-values prove normal distributions at the 5% significance level. All groups were well-modelled by a normal distribution. Characteristics of the groups were visualised in Figure 6.

Levene's test to assess the variance of two groups was used in the next step. The null hypothesis that two normal populations have the same variance was used to verify the equality of variance for all the pairs of groups. The requirement of homogeneity is met when the result is not significant. The results are presented in Table 3.

The hypothesis about the equality of variance at the 5% significance level was rejected in pairs 2016–2019 and 2017–2019. The equality of variance in other cases is not significantly different. Therefore, it is possible to accept the assumption that the results of the group can be compared.

Table 3. The proof of the equality of variance.

	Valid N1	Valid N2	Std.Dev.1	Std.Dev.2	F-Statistic	*p*-Value	
2016–2017	51	70	6.96	7.52	0.78	0.997	
2016–2018	51	82	6.96	8.68	0.82	0.084	
2016–2019	51	102	6.96	10.07	0.87	**0.001**	significantly different
2017–2018	70	82	7.52	8.68	0.87	0.078	
2017–2019	70	102	7.52	10.07	0.90	**0.001**	significantly different
2018–2019	82	102	8.68	10.07	0.92	0.081	

Normal distribution and homogeneity of variances give assumptions for the analysis of variance (ANOVA test) used for investigating whether the population means of the groups are equal., it is necessary to compare the means of the following pairs to validate hypotheses H1–H4:

- 2016–2017 (H1)—to research the benefit of VPL activities in introductory programming parts,
- 2017–2018 (H2)—to evaluate the benefit of VPL activities in object–oriented programming parts,
- 2018–2019 (H3, H4) – to inspect the benefit of microlearning activities in both (introductory and object-oriented programming) parts.

The descriptive statistic and results are presented in Table 4.

Table 4. The proof of the equality of mean.

Groups	Valid N1	Valid N2	F-Statistic	*p*-Value	Result
2016–2017	51	70	0.87	0.35	not significantly different
2017–2018	70	82	1.74	0.19	not significantly different
2018–2019	82	102	1.58	0.21	not significantly different

The *p*-value corresponding to the F-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 all monitored pairs at the level of significance of 5%.

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

4.2. Post-Tests

The following hypotheses require a comparison of students' results measured by tests. The tests consist of four or five assignments to write programs. The tasks are only practical and measure the student's ability to write and debug the program. The assignments cover the entire content of the relevant part of the course.

The automatic evaluation of source code is used in the first step of evaluation, followed by the evaluation realised by a human evaluator. Source code of every program is checked and also evaluated by the teacher. If the evaluation is realised by more than one teacher, the rule that one task is checked only by one teacher is followed to ensure the same conditions for all students' solutions.

4.2.1. H1—Using VPL Improves Student Performance in Introductory Chapters of Programming

The assumptions of normal distribution and homogeneity of variances are met. The results of the Kolmogorov–Smirnov test and Levene's test are presented in Table 5.

The data in groups do not differ significantly from a normal distribution. The *p*-values prove normal distributions at the 5% significance level and the groups were well-modelled by a normal distribution. The results of the Levene's test is not significant. It means that the variances of groups are not significantly different.

The ANOVA test was used to compare the results of the student tests. The test compares the means of groups at the 5% significance level. The null hypothesis in ANOVA assumes that there is

not a significant difference in means. The alternative hypothesis captures any difference in means. The results are summarised in Table 6.

Table 5. Assumptions of normal distribution and homogeneity of variances.

The Results of the Kolmogorov-Smirnov Test of Normality						
Group	**Count**	**Mean**	**Median**	**Std. Dev.**	**D (Val. of K-S Test)**	***p*-Value**
2016	51	235.97	227.80	153.29	0.10	0.60
2017	70	317.12	366.65	157.82	0.15	0.09
The Results of the Levene's Test of Homogeneity of Variance						
	Valid N1	**Valid N2**	**Std.Dev.1**	**Std.Dev.2**	**F-Statistic**	***p*-Value**
2016–2017	51	70	6.96	7.52	0.02	0.90

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

	DF	**Sum of Square**	**Mean Square**	**F-Statistic**	***p*-Value**
Groups (between groups)	1	194,258.95	194,258.95	7.99	0.0055
Error (within groups)	119	2,893,546.19	24,315.51		
Total	120	3,087,805.14	25,731.71		

Since p-value < α, H_0 is rejected, the difference between the means of groups can be considered large enough to be statistically significant. While p-value equals 0.00552176, [p ($x \leq F$) = 0.994478], the chance of type1 error (rejecting a correct H_0) is small: 0.005522 (0.55%).

The hypothesis H1 was not rejected, and thus the use of automated evaluation of the students' programs in the introductory programming courses brought significant differences in student outcomes.

4.2.2. H2—Using VPL Improves Student Performance in Advanced Chapters of Programming

The same procedure was applied for testing hypothesis H2. The results of the final test realised at the end of the semester are used. The results of the Kolmogorov–Smirnov test and Levene's test are presented in Table 7.

Table 7. Assumptions of normal distribution and homogeneity of variances.

The Results of the Kolmogorov-Smirnov Test of Normality						
Group	**Count**	**Mean**	**Median**	**Std. Dev.**	**D (Val. of K-S Test)**	***p*-Value**
2017	70	228.39	225	143.17	0.083	0.68
2018	82	231.34	250	158.88	0.099	0.37
The Results of the Levene's Test of Homogeneity of Variance						
	Valid N1	**Valid N2**	**Std.Dev.1**	**Std.Dev.2**	**F-Statistic**	***p*-Value**
2016–2017	70	82	79.69	77.30	2.42	0.12

The data in the observed groups, as well as the variances of groups, are not significantly different from a normal distribution.

Again, the ANOVA test at the 5% significance level was used to compare student test results. The null hypothesis assumes that there is not any significant difference in means. The results are summarised in Table 8.

Since p-value > α, H_0 is accepted. The difference between the means of groups is not large enough to be considered statistically significant. The p-value equals 0.905179, [p($x \leq F$) = 0.0948215]. This means that, if H_0 were rejected, the chance of type1 error (rejecting a correct H_0) would be too high: 0.9052 (90.52%).

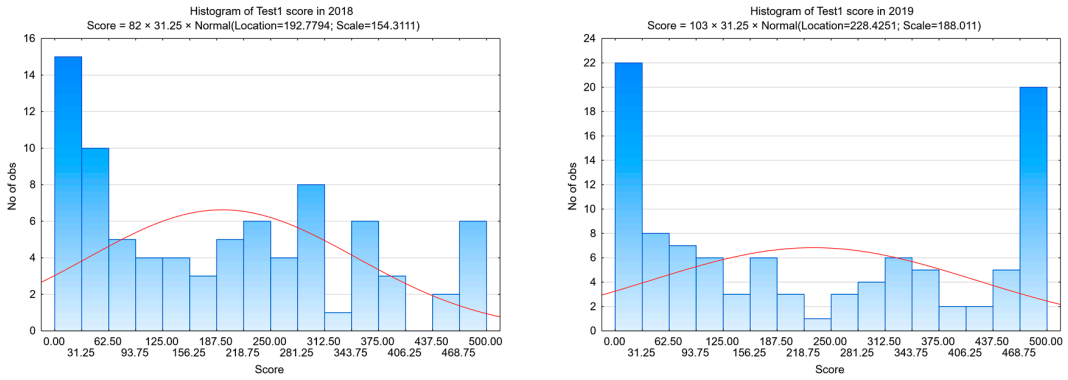
Table 8. Assumptions of normal distribution and homogeneity of variances.

	DF	Sum of Square	Mean Square	F-Statistic	*p*-Value
Groups (between groups)	1	328.32	328.32	0.014	0.91
Error (within groups)	150	3,458,909.89	23,059.40		
Total	151	3,459,238.21	22,908.86		

It follows that the hypothesis H2 cannot be accepted. In other words, the use of automated assessment object in the advanced topics of introductory programming course does not cause significant differences in student's outcomes.

4.2.3. H3—The Use of Microlearning Improves Student Performance in Introductory Chapters of Programming

The compared groups are groups of students from 2018 and 2019 year. The results of the test realised in half of the semester are used. The test score distribution of the test is demonstrated in Figure 7.



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

Figure 7. Histogram of test score distribution in monitored groups: (**a**) Test score in the group trained without microlearning in 2018; (**b**) Test score in the group of students taught with microlearning in 2019.

The results of the Kolmogorov-Smirnov test are presented in Table 9.

Table 9. The results of the Kolmogorov-Smirnov test of normality for test results in group 2018 and group 2019.

Group	Count	Mean	Median	Std. Dev.	D (Val. of K-S Test)	*p*-Value
2018	82	192.78	187.61	154.31	0.13	0.130
2019	103	228.43	177.78	188.01	0.13	0.048

The data in 2018 group does not significantly differ from a normal distribution, but the distribution of the test results in the 2019 group is significantly different from a normal distribution. Therefore, a non-parametric Mann–Whitney U-test should be used to investigate if 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 10.

The U-value is 3820.5, the Z-Score is 1.11105, and the *p*-value is 0.1335. The result is not significant at $p < 0.05$. Since *p*-value $> \alpha$, the hypothesis is accepted, the difference between the averages of all groups is not large enough to be statistically significant.

Table 10. Mann-Whitney U-test results.

	2018	2019	Combined
Sum of ranks	7223.5	9981.5	17,205
Mean of ranks	88.09	96.91	93
Expected sum of ranks	7626	9579	
Expected mean of ranks	93	93	
U-value	4625.5	**3820.5**	
Expected *U*-value	4223	4223	
Standard Deviation			361.82

Therefore, the hypothesis H3 cannot be accepted, and it means the implementation of the microlearning activities in the introductory topics of programming courses does not lead to the significant differences in student outcomes.

4.2.4. H4—The Use of Microlearning Improves Student Performance in Advanced Chapters of Programming

The last hypothesis H4 compared groups of students between 2018 and 2019. The results of the final test written at the end of the semester are used. The examined sample does not include students who failed the study of the course during the semester and did not attend the second test.

The graphical presentation of test score distribution is demonstrated in Figure 8.

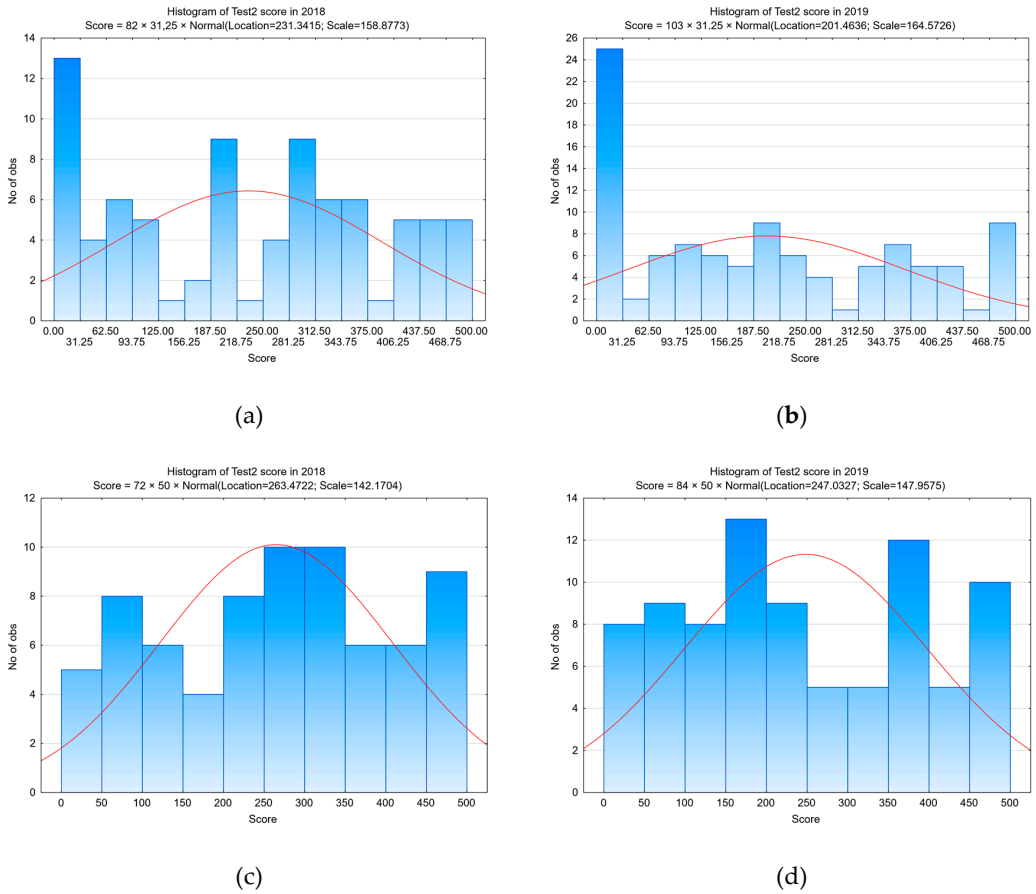


Figure 8. Histogram of test score distribution in monitored groups: (**a**) Test score in the group trained without microlearning in 2018; (**b**) Test score in the group taught with microlearning in 2019; (**c**) Test score in the group trained without microlearning in 2018 without students who failed during the semester; (**d**) Test score in the group trained with microlearning in 2019 without students who failed during the semester. The red curve represents the outline of the normal distribution of values.

The results of the Kolmogorov–Smirnov test in examined samples are presented in Table 11.

Table 11. Assumptions of normal distribution and homogeneity of variances.

The Results of the Kolmogorov-Smirnov Test of Normality for Results						
Group	**Count**	**Mean**	**Median**	**Std. Dev.**	**D (Val. of K-S Test)**	***p*-Value**
2018	72	263.47	292.50	142.17	0.088	0.596
2019	84	247.03	220.50	147.96	0.100	0.345
The Results of the Levene's Test of Homogeneity of Variance						
	Valid N1	**Valid N2**	**Std.Dev.1**	**Std.Dev.2**	**F-Statistic**	***p*-Value**
2018–2019	72	84	74.28	74.43	0.32	0.57

The data in groups is not significantly different from a normal distribution. Their variances do not significantly differ too.

The ANOVA test was used to compare student test results at the 5% significance level. The results are shown in Table 12.

Table 12. Assumptions of normal distribution and homogeneity of variances.

	DF	**Sum of Square**	**Mean Square**	**F-Statistic**	***p*-Value**
Groups (between groups)	1	10,477.64	10,477.64	0.50	0.48
Error (within groups)	154	3,252,069.67	21,117.34		
Total	155	3,262,547.30	21,048.69		

Since p-value > α, H_0 is accepted. Consequently, the difference between the means of the groups is not considered large enough to be statistically significant. p-value equals 0.482255, [p(x ≤ F) = 0.517745]. This means that if H_0 were rejected, the chance of type1 error (rejecting a correct H_0) would be too high: 0.4823 (48.23%).

Therefore, the hypothesis H4 cannot be accepted. Therefore, the use of microlearning in the advanced chapters of programming course has no significant impact on student's outcomes.

4.2.5. Dependencies Inspection

A revealing of the dependence between individual variables is relatively simple in some cases. However, in others, the expected influence of variables is not proven. Based on simple reasoning, it could be assumed that students who have demonstrated a lower initial assessment in the admission process will have more significant problems in their studies.

The inspection of the relation between the results of entrance exams (points of secondary level of study) and the result of the course showed a low correlation. The results are based on Pearson correlation coefficient, which is used to measure the strength of a linear association between two variables. The results are summarised in Table 13.

Table 13. Pearson correlation coefficient between results of entrance admission process and grades of tests in individual experimental groups.

Group	**Structural Programming Test**		**Object-Oriented Programming Test**	
2016	−0.093	weak	0.339	weak
2017	0.074	weak	0.231	weak
2018	0.362	weak	0.379	weak
2019	0.274	weak	0.296	weak

The dependences between inspected variables in groups are weak. However, it is possible to observe that the dependency between the entrance process values and results of the second test is a

little bit stronger. The reason is that some of the students dropped out, while they were students with lower input rating (differences are statistically undetectable).

The LMS allows monitoring of students' activity within the study period by logging the operations performed by the students in the virtual environment (Figure 9).

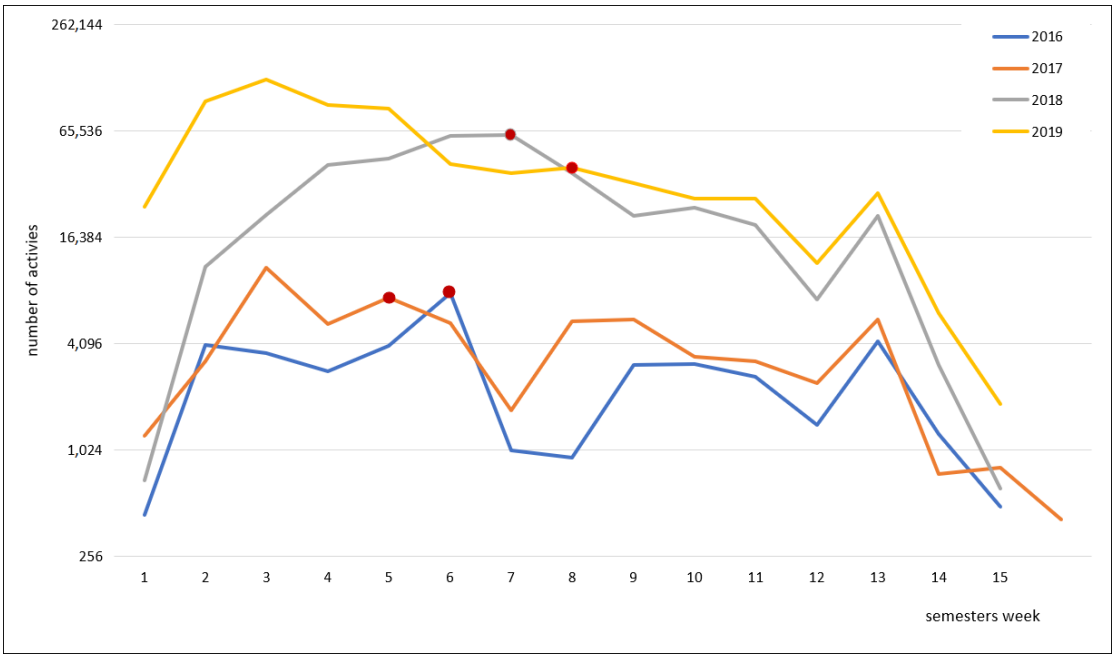


Figure 9. The student's activity expressed by the number of interactions performed in the course in the course during the semester - the x-axis shows the weeks of the semester (from the 1st week to the last (14th or 15th) week of the semester). The y-axis represents number of student's activities on a logarithmic scale. The red dots indicate the date of the first test. The last test was realised in the 13th week.

Considering the log visualisation (Figure 9), the students' activity is very similar over the course of all four years. The most significant increase in activity can be observed before the first test. The activity declines subsequently and never reaches the value it had before the first test. The time before the final test also causes the growth of the students' activities, but these activities are not in that quantity as before the first test.

A subjective explanation of this phenomenon is based on communication and feedback with the students using anonymous questionnaires and interviews. The students said that they were afraid of the unknown before the first test, and they studied as much as possible. Later, before the second test, they were with less stress due to the known requirements for the expected skills and practices. Although the students' activity has increased significantly due to the use of VPL in the third and fourth observed year, the curve copies the previous ones.

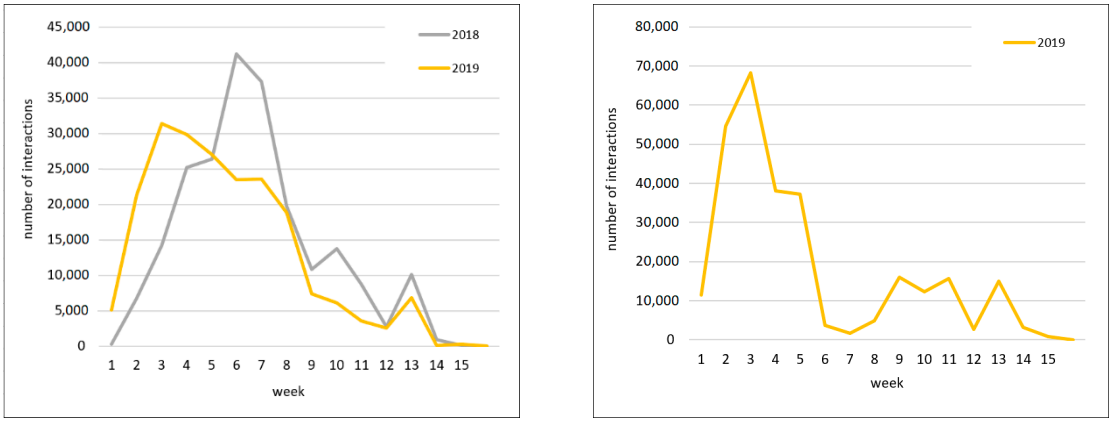
Therefore, it is useful to visualise the scope of VPL and microlearning activities in 2018 and 2019 to obtain a complete picture of the interactive activities (Figure 10).

The graphs in Figures 9 and 10 confirm that interactive activities represent a significant part of users' activity, and thus represent a suitable tool for identifying the study behaviour of the students.

This statement about the significance of these activities in the educational process is also supported by the identification of dependencies between students' test results and VPL and quiz results presented in Table 14. The dependence between activity in the course and final evaluation has not been proven.

The Pearson correlation coefficient expresses moderate to strong dependence between final students' achievements and results achieved solving interactive activities in courses. It can be argued that the overall success of the students depends on the ongoing results in individual interactive

activities, while early warning must be based on the ongoing results in these activities. The influence of the user's activity in the course on his or her success is inconclusive.



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

Figure 10. The interactive activities in courses 2018 and 2019 during the weeks of the semester: (**a**) automated source code evaluation activities; (**b**) microlearning activities.

Table 14. Pearson correlation coefficient between the test results and interactive student activities results.

Group	Test 1—VPL1	Test2—VPL2	Sum of Tests—Microlearning
2018	0.512	0.721	-
2019	0.563	0.683	0.577

5. Discussion and Conclusions

Based on the verification of the established hypotheses on the sample of more than 300 students in total, it can be specified that the use of additional interactive concepts in introductory programming courses brings the following benefits:

Hypothesis 1 (H1). *The use of LMS objects for the automatic evaluation of source codes improves students' results in the area of introduction to programming.*

The hypothesis was not rejected. The use of automatic evaluation of programs in the introductory chapters of programming courses brought significant differences in student outcomes. These introductory chapters cover essential topics as input/output, variables, data types, nested loops, arrays, exceptions, files, etc. and they are often the reason for students' loss of interest in programming because of a misunderstanding of its principles, problems with memorisation, or loss of the motivation [37,38].

Successful completion of this part of the introductory course is usually a prerequisite for mastering the second part of the course of the programming course, which is oriented to object-oriented programming. It is often (but not always) the case that students who fail the introductory part do not continue their studies. These students repeat the course a year or more often.

The finding that the use of VPL is significant brings these students an increased chance of successful completion of the course. The result is in accordance with the results presented in [21,23,39]. At the same time, it should be noted that applying automated assessment is not a solution itself. The quality of the automated assessment is also essential.

Hypothesis 2 (H2). *The use of LMS objects for the automatic evaluation of source codes improves students' results in object-oriented programming.*

The hypothesis cannot be accepted, and the use of automated assessment object in the advanced chapters of introductory programming course does not lead to the significant differences in student's outcomes. The reason for this statement is mainly the fact that this part of the introductory programming course is completed by students who have completed the first part successfully after the most significant dropout occurs. Students who have completed the first part of the course probably have enough motivation to master programming. Their skills have shifted to a matching level to understand the concepts and principles of programming and, in essence, the concept of object-oriented programming. The groups of students in both monitored years have stabilised at approximately the same level of knowledge. As a result, they do not differ significantly between the years.

Hypothesis 3 (H3). *The use of microlearning principles improves students' results in the area of introduction to programming.*

The hypothesis cannot be accepted. The use of microlearning activities in the introductory chapters of programming courses does not imply significant differences in student outcomes. Although microlearning in its modern form represents a new approach in university education and can produce better results than traditional methods in various areas, it did not ensure a significant change in the level of knowledge of the experimental group of students [40,41]. The reasons for this can be as follows:

- microlearning was applied in combination with the automatically evaluated assignments and its addition was minimised or overlapped,
- many of the different sources and the obligation to develop microlearning lessons caused overload for weaker students and caused them to resign.

The results of the questionnaire survey presented below support the first claim. The histogram shown in Figure 7b presents the group division into students with excellent and students with weak (resp. zero) results, while the number of students with average results was minimised. The reasons for and real effects of microlearning on the obtained level of knowledge or the dropout rate of the students should be the subject of further research.

Hypothesis 4 (H4). *The use of microlearning principles improves students' results in object-oriented programming.*

The hypothesis cannot be again accepted. The use of microlearning in the advanced chapters of introductory programming course does not cause significant differences in students' outcomes. The reasons are similar to the reasons mentioned in a discussion about the hypothesis H2.

The dependencies between the achieved VPL score and the final test score, as well as the dependencies between the microlearning activities and the final score, were identified in a section focused on the verification of the dependencies between the success of students in the inspected activities and the results in the course. The dependencies are identified from the moderate to strong level and place these LMS activities into the position of a tool with the assumptions to identify the risk of dropout.

The 2019 group responded to the questionnaire survey conducted after the end of the semester, and the results collected also correspond to the research findings. Thereby, students evaluated the subjective contribution of the monitored activities to their learning.

The main facts are summarised in Table 15. The students expressed their opinions using a seven-point Likert scale questionnaire.

The questionnaire shows that both types of activities are received positively by more than 70% of users. Activities are identified as resources that play a significant role in building skills and knowledge. Activities with automated code evaluation are accepted more positively. The students identified the main benefits as immediate control and feedback on the submitted program, learning from one's own mistakes, practical learning, and possibility to repeat solutions in the case of error.

Table 15. Subjective perception of VPL and microlearning activities by students.

Group	7	6	5	4	3	2	1
VPL assignments helped me understand the curriculum	42.31	11.54	15.38	17.31	1.92	5.77	5.77
VPL assignments helped me practice the curriculum	51.92	11.54	13.46	13.46	1.92	0.00	7.69
microlearning assignments helped me understand the curriculum	34.62	21.15	25.00	11.54	3.85	1.92	1.92
microlearning assignments helped me practice the curriculum	34.62	25.00	25.00	7.69	3.85	1.92	1.92
microlearning was used as a main way of learning	34.62	25.00	13.46	15.38	1.92	7.69	1.92

The benefits of microlearning activities were most often mentioned: a better understanding of the content, preparation for the following lessons, quick acquisition of information, brief content of the lesson, possibility of practice. Moreover, students would expect similar use of both types of objects in other programming courses.

The research result aimed at verifying the effectiveness of interactive activities for successful completion of the course is the confirmation of a significant benefit of the implementation of automatically evaluated programs into the introductory programming courses. At the same time, it was found that this benefit is not significant in advanced programming topics. The advantage of the educational content transformation into the microlearning form did not bring a significant improvement in the students' results despite the positive perception of this activity by the students.

Even though the significant contribution of interactive elements was confirmed only in the case of automatically evaluated programs in the first half of the course, the dependence between the overall results of students and the results in the activities was proven. Further research will focus on the identification of students at risk of a dropout based on their behaviour during the solving of the mentioned activities.

Author Contributions: Conceptualisation, J.S. and M.D.; methodology, J.S. and M.D.; investigation, J.S. and M.D.; resources, J.S. and M.D.; data curation, J.S. and M.D.; writing—original draft preparation, J.S. and M.D.; writing—review and editing, J.S. and M.D.; funding acquisition, 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: KEGA 029UKF-4/2018 "Innovative Methods in Programming Education in the University Education of Teachers and IT Professionals."

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

References

1. Malik, S.I. Improvements in Introductory Programming Course: Action Research Insights and Outcomes. *Syst. Pract. Action Res.* **2018**, *31*, 637–656. [CrossRef]
2. Mason, R.; Cooper, G.; Raadt, M. Trends in introductory programming courses in Australian Universities-Languages, environments and pedagogy. In Proceedings of the Fourteenth Australasian Computing Education Conference (ACE2012), Melbourne, Australia, 31 January–3 February 2012; Volume 123, pp. 33–42.
3. Chango, W.; Cerezo, R.; Romero, C. Predicting academic performance of university students from multi-sources data in blended learning. In Proceedings of the Second International Conference on Data Science, E-Learning and Information Systems, Tromsø, Norway, 2–5 December 2019. [CrossRef]
4. Luxton-Reilly, A.; Albluwi, I.; Becker, B.A.; Giannakos, M.; Kumar, A.; Ott, L.M.; Paterson, J.; Adrir Scott, M.; Sheard, J.; Szabo, C. Introductory programming: A systematic literature review. In Proceedings of the ITiCSE '18 Companion, Larnaca, Cypru, 2–4 July 2018.
5. Becker, B.A.; Quille, K. 50 years of CS1 at SIGCSE: A review of the evolution of introductory programming education research. In Proceedings of the SIGCSE 2019-Proceedings 50th ACM Technical Symposium Computer Science Education, Minneapolis, MN, USA, 27 February–2 March 2019; Volume 1, pp. 338–344. [CrossRef]

6. 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 Seventh International Workshop on Computing Education Research, ICER 2011, Providence, RI, USA, 8–9 August 2011. [CrossRef]
7. Kinnunen, P.; Malmi, L. Why Students Drop out CS1 Course? In Proceedings of the Second International Workshop on Computing Education Research, New York, NY, USA, 10–12 September 2006; pp. 97–108. [CrossRef]
8. Bergin, S.; Reilly, R. Programming: Factors That Influence Success. In Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education, St. Louis, MO, USA, 23–27 February 2005; pp. 411–415. [CrossRef]
9. Hellas, A.; Nam Liao, S.; Ihantola, P.; Petersen, A.; Ajanovski, V.V.; Gutica, M.; Hynninen, T.; Knutas, A.; Leinonen, J.; Messom, C. Predicting academic performance: A systematic literature review. In Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE, Larnaca, Cyprus, 2–4 July 2018; pp. 175–199. [CrossRef]
10. Hu, X.; Cheong, C.W.L.; Ding, W.; Woo, M. A Systematic Review of Studies on Predicting Student Learning Outcomes Using Learning Analytics. In Proceedings of the Seventh International Learning Analytics & Knowledge Conference, New York, NY, USA, 13–17 March 2017; pp. 528–529. [CrossRef]
11. Duran, R.; Rybicki, J.M.; Hellas, A.; Suoranta, S. Towards a common instrument for measuring prior programming knowledge. In Proceedings of the 2019 ACM Conference on International Computing Education Research Conference, ITiCSE, London, UK, 11–15 November 2019; pp. 443–449. [CrossRef]
12. Costa, E.B.; Fonseca, B.; Santana, M.A.; de Araújo, F.F.; Rego, J. Evaluating the effectiveness of educational data mining techniques for early prediction of students' academic failure in introductory programming courses. *Comput. Hum. Behav.* **2017**, *73*, 247–256. [CrossRef]
13. Galan, D.; Heradio, R.; Vargas, H.; Abad, I.; Cerrada, J.A. Automated Assessment of Computer Programming Practices: The 8-Years UNED Experience. *IEEE Access* **2019**, *7*, 130113–130119. [CrossRef]
14. Luchoomun, T.; Chumroo, M.; Ramnarain-Seetohul, V. A Knowledge Based System for Automated Assessment of Short Structured Questions. In Proceedings of the 2019 IEEE Global Engineering Education Conference (EDUCON), Dubai, UAE, 8–11 April 2019; pp. 1349–1352. [CrossRef]
15. Skalka, J.; Drlík, M.; Obonya, J. Automated Assessment in Learning and Teaching Programming Languages using Virtual Learning Environment. In Proceedings of the 2019 IEEE Global Engineering Education Conference (EDUCON), Dubai, UAE, 8–11 April 2019; pp. 689–697. [CrossRef]
16. Rodriguez-del-Pino, J. A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features. In Proceedings of the International Conference on e-Learning, e-Business, Enterprise Information Systems, & e-Government, Las Vegas, NV, USA, 16–19 July 2012.
17. Salleh, S.M.; Shukur, Z.; Judi, H.M. Analysis of Research in Programming Teaching Tools: An Initial Review. *Procedia Soc. Behav. Sci.* **2013**, *103*, 127–135. [CrossRef]
18. Keuning, H.; Jeuring, J.; Heeren, B. A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. *ACM Trans. Comput. Educ.* **2018**, *19*, 1–43. [CrossRef]
19. Akahane, Y.; Kitaya, H.; Inoue, U. Design and evaluation of automated scoring Java programming assignments. In Proceedings of the 2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Takamatsu, Japan, 1–3 June 2015; pp. 1–6. [CrossRef]
20. Figueiredo, J.; García-Peñalvo, F.J. Building skills in introductory programming. *ACM Int. Conf. Proc. Ser.* **2018**, 46–50. [CrossRef]
21. Pieterse, V. Automated Assessment of Programming Assignments. In Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research, Arnhem, The Netherlands, 4–5 April 2013; pp. 45–56.
22. Wilcox, C. The Role of Automation in Undergraduate Computer Science Education. In Proceedings of the 46th ACM Technical Symposium on Computer Science Education, Kansas City, MO, USA, 4–7 March 2015; pp. 90–95. [CrossRef]
23. Rubio-Sánchez, M.; Kinnunen, P.; Pareja-Flores, C.; Velázquez-Iturbide, Á. Student Perception and Usage of an Automated Programming Assessment Tool. *Comput. Hum. Behav.* **2014**, *31*, 453–460. [CrossRef]
24. Skalka, J.; Drlík, M. *Conceptual Framework of Microlearning-Based Training Mobile Application for Improving Programming Skills*; Springer: Cham, Switzerland, 2018.

25. Göschlberger, B.; Bruck, P.A. Gamification in Mobile and Workplace Integrated Microlearning. In Proceedings of the 19th International Conference on Information Integration and Web-Based Applications & Services, Salzburg, Austria, 4–6 December 2017; pp. 545–552. [CrossRef]
26. Skalka, J. Data processing methods in the development of the microlearning-based framework for teaching programming languages. In Proceedings of the 12th International Scientific Conference on Distance Learning in Applied Informatics (DIVAI), Štúrovo, Slovakia, 2–4 May 2018; pp. 503–512.
27. Leppänen, L.; Leinonen, J.; Ihantola, P.; Hellas, A. Predicting academic success based on learning material usage. In Proceedings of the SIGITE 2017-Proceedings 18th Annual Conference Information Technology Education, Rochester, NY, USA, 4–7 October 2017; pp. 13–18. [CrossRef]
28. Figueira, Á. Mining moodle logs for grade prediction: A methodology walk-through. *ACM Int. Conf. Proc. Ser.* **2017**. Part F1322. [CrossRef]
29. Félix, I.M.; Ambrósio, A.P.; Neves, P.S.; Siqueira, J.; Brancher, J.D. Moodle predicta: A data mining tool for student follow up. In Proceedings of the CSEDU 2017-Proceedings 9th International Conference Computer Support Education, Porto, Portugal, 21–23 April 2017; Volume 1, no. Csedu. pp. 339–346. [CrossRef]
30. Kadoić, N.; Oreški, D. Analysis of student behavior and success based on logs in Moodle. In Proceedings of the 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 21–25 May 2018; pp. 654–659. [CrossRef]
31. Ihantola, P.; Vihavainen, A.; Ahadi, A.; Butler, M.; Börstler, J.; Edwards, S.H.; Isohanni, E.; Korhonen, A.; Petersen, A.; Rivers, K.; et al. Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies. In Proceedings of the 20th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE), Vilnius, Lithuania, 4–8 July 2015; Volume 16. [CrossRef]
32. Sisovic, S.; Matetic, M.; Bakaric, M.B. Mining student data to assess the impact of moodle activities and prior knowledge on programming course success. *ACM Int. Conf. Proc. Ser.* **2015**, *1008*, 366–373. [CrossRef]
33. Hussain, M.; Hussain, S.; Zhang, W.; Zhu, W.; Theodorou, P.; Abidi, S.M.R. Mining moodle data to detect the inactive and low-performance students during the moodle course. *ACM Int. Conf. Proc. Ser.* **2018**, 133–140. [CrossRef]
34. Skalka, J.; Drlík, M. Educational Model for Improving Programming Skills Based on Conceptual Microlearning Framework BT-The Challenges of the Digital Transformation in Education. In *The Challenges of the Digital Transformation in Education*; Springer: Berlin, Germany, 2020; pp. 923–934.
35. Huet, I.; Pacheco, O.R.; Tavares, J.; Weir, G. New challenges in teaching introductory programming courses: A case study. In Proceedings of the 34th Annual Frontiers in Education, Savannah, GA, USA, 20–23 October 2004; Volume 1, pp. T2H/5–T2H/9. [CrossRef]
36. Drlik, M.; Skalka, J. Virtual Faculty Development Using Top-down Implementation Strategy and Adapted EES Model. *Procedia Soc. Behav. Sci.* **2011**, *28*, 616–621. [CrossRef]
37. Jenkins, T. On the difficulty of learning to program. In Proceedings of the 3rd Annual Conference LTSN Centre Information Computer Science, Loughborough, UK, 27–29 August 2002; Volume 4, pp. 53–58.
38. Gomes, A.J.; Santos, A.N.; Mendes, A.J. A Study on Students' Behaviours and Attitudes towards Learning to Program. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*; ACM: New York, NY, USA, 2012; pp. 132–137. [CrossRef]
39. Aleman, J.L.F. Automated Assessment in a Programming Tools Course. *IEEE Trans. Educ.* **2011**, *54*, 576–581. [CrossRef]
40. Mohammed, G.S.; Wakil, K.; Nawroly, S.S. The Effectiveness of Microlearning to Improve Students' Learning Ability. *Int. J. Res. Rev.* **2018**, *3*, 32–38. Available online: https://www.ijere.com/article/the-effectiveness-of-microlearning-to-improve-students-learning-ability (accessed on 31 May 2020).
41. Kapp, K.M.; Defelice, R.A. *Microlearning: Short and Sweet*; American Society for Training & Development: Alexandria, VA, USA, 2019.