Adaptive assessment and content recommendation in online programming courses: On the use of Elo-rating

BOBAN VESIN, University of South-Eastern Norway and Norwegian University of Science and Technology
KATERINA MANGAROSKA, University of South-Eastern Norway and Norwegian University of Science and Technology
KAMIL AKHUSEYINOGLU, University of Pittsburgh
MICHAIL GIANNAKOS, Norwegian University of Science and Technology

Online learning systems should support students preparedness for professional practice, by equipping them with the necessary skills while keeping them engaged and active. In that regard, the development of online learning systems that support students' development and engagement with programming is a challenging process. Early career computer science professionals are required not only to understand and master numerous programming concepts, but to efficiently learn how to apply them in different contexts. A prerequisite for an effective and engaging learning process is the existence of adaptive and flexible learning environments that are beneficial for both, students and teachers. Students can benefit from personalized content adapted to their individual goals, knowledge, and needs; while teachers can be relieved from the pressure to uniformly and promptly evaluate hundreds of student assignments. This study proposes and puts into practice a method for evaluating learning content difficulty and students' knowledge proficiency utilizing a modified Elo-rating method. The proposed method effectively pairs learning content difficulty with students' proficiency, and creates personalized recommendations based on the generated ratings. The method was implemented in a programming tutoring system and tested with interactive learning content for object oriented-programming. By collecting quantitative and qualitative data from students who used the system for one semester, the findings reveal that the proposed method can generate recommendations that are relevant to students and has the potential to assist teachers in grading students by providing a more holistic understanding of their progress over time.

CCS Concepts: • **Social and professional topics** → *Student assessment*; • **Applied computing** → *E-learning*.

Additional Key Words and Phrases: e-learning, personalisation, ranking students, programming, intelligent tutoring systems

1 INTRODUCTION

Many students taking computer science (CS) programs find programming very challenging, with a high percentage either dropping out or performing poorly [87]. One reason for that is often the way programming is taught. Given the fact that learning programming requires a certain degree of procedural knowledge competence, learning programming needs practice and experience [11, 34, 84]. Unfortunately, teaching and learning content in programming education is often heavy on declarative knowledge, focusing on the features and particularities of

Authors' addresses: Boban Vesin, boban.vesin@usn.no, University of South-Eastern Norway, Raveien 215, Borre, Vestfold, Norway and Norwegian University of Science and Technology, HÃÿgskoleringen 1, Trondheim, Norway; Katerina Mangaroska, katerina.mangaroska@usn.no, University of South-Eastern Norway, Raveien 215, Borre, Vestfold, Norway and Norwegian University of Science and Technology, HÃÿgskoleringen 1, Trondheim, Norway; Kamil Akhuseyinoglu, kaa108@pitt.edu, University of Pittsburgh, 4200 Fifth Ave, Pittsburgh, PA, USA; Michail Giannakos, michaelg@ntnu.no, Norwegian University of Science and Technology, HÃÿgskoleringen 1, Trondheim, Norway.

1946-6226/2022/2-ART $15.00
https://doi.org/10.1145/3511886

various programming languages [49], especially in cases when a new programming language is taught to students with previous programming knowledge [85]. In this regard, novel teaching methods and proper utilization of online environments and resources have the potential to support students' engagement with programming [48]. During the last few years the increased availability of online learning resources has changed not only the way students learn, but also how teachers organize and design learning activities [36]. It is theorized that adaptive learning systems have the capacity to enhance the learning process by offering content and resources that match students' skills and needs [12, 71]. Thus, the demand for online learning resources widely accessible to students with varying skills and backgrounds, has increased the need to accommodate students' individual differences and learning goals [83]. As a result, adaptive learning systems could tailor learning and assessment considering students' individuality, instead of providing one-size-fits-all materials.

In CS and software engineering education, programming is a central part of different study programs, which are not necessarily connected with software development [56]. The increased number of students and the popularity of programming courses have raised multiple challenges and opportunities in offering assessment and feedback on programming assignments [61]. For example, introductory programming classes often have more than 500 students, which makes it difficult for instructors to provide timely, thorough, and uniform assessment [82]. Moreover, instructors are required to check and correct numerous programming assignments that might have similar mistakes; however, due to the volume of programming assignments that need to be graded and the number of instructors involved in these courses, sometimes the same mistakes might be graded differently [88]. Hence, in order to provide efficient and consistent assessment for every student, we argue that automated assessment methods can be adapted and scaled to assist instructors in delivering timely, accurate, and uniform assessment [43, 81]

Despite the great potential of adaptive assessment in programming education, its application does not come without challenges [1]. In programming, every solution can be evaluated considering different aspects (e.g., efficiency of the solution, correct logic but syntactical mistakes, etc.) and, therefore, imposes additional complexity during the assessment process. Moreover, standardized assessment methods fail to measure meaningful forms of human competence, because teachers and learning systems rarely accommodate for learners' diversity [1, 18]. Hence, since standardized Computer-Based Assessment (CBA) fails to accommodate the individuality of students, Adaptive Computer-Based Assessment (ACBA) emerged as a solution for accurate and reliable estimation of proficiency, progress, and reduction of testing time [54]. Consequently, our study attempts to explore how to develop an efficient and scalable method for adaptive and automated assessment of programming assignments utilizing unit testing and a ranking algorithm. To do so, we propose a novel implementation that leverages the Elo-rating algorithm [77], that is originally developed for measuring player strength in chess tournaments [2]. Elo-rating has also found application in the context of educational research for measuring learners' ability and tasks' difficulty [72, 77].

As a first step, we incorporated an adaptive assessment and recommendation module into ProTuS, a system that has been successfully used for learning programming fundamentals [64]. The modified Elo-rating method that we implemented estimates the difficulty of the learning content and students' knowledge in ProTuS. Based on the generated ratings, the system recommends the learning content based on current knowledge and skills demonstrated by the students. In sum, the aim of the study was to explore how accurate and precise the Elo-rating algorithm is for recommending learning content (e.g., coding exercises) and thus, how efficient and scalable the proposed adaptive assessment module is.

Hence, our study addresses the following research questions:

RQ1: *How effective is the Elo-rating algorithm for adaptive assessment in programming tasks?*
RQ2: *How does the proposed method (i.e., Elo-rating algorithm and unit testing) perform in a real-time adaptive assessment process in programming tasks?*

RQ3: *How accurate are the recommendations of learning content based on the accumulated rankings?*
RQ4: *How students' experienced the proposed method and the recognized benefits for programming education?*

The following sections provide a brief overview of the available techniques for automated assessment in e-learning and an overview of recommender systems in education. The proposed method for automatic assessment in educational environments will be presented in Section 3, in which subsection 3.3 explains the recommendation process based on the students' estimated skills and task difficulty implemented in ProTuS. Section 4 presents the study, while the results of the experiments performed are presented in Section 5. The discussion and conclusions will be presented in the final section.

2 RELATED WORK

Adaptive learning systems address several interrelated issues with respect to adaptability, such as learner modeling, automatic assessment of students' knowledge, and determining levels of educational content complexity [3]. Past research shows that adaptability in online learning systems is mainly explored through the development of intelligent tutoring systems [91] or computerized adaptive testing [25]. Studies in the context of programming education indicate the challenges and opportunities of adapting testing to support CS courses [1, 18]. For the purpose of this study, we are looking into state-of-the-art adaptive assessment techniques and methods, as well as the current recommender systems used in online education, with a particular emphasis on studies conducted in programming education [1, 37].

2.1 Adaptive assessment in programming education

Assessment in technology-rich environments has attracted the interest of many educational technology researchers; however, it remains an intriguing and open research issue [63]. On the one hand, it seems that there is an increasing attentiveness to performance-based, formative assessment and learning systems that can effectively support it. On the other hand, many researchers raise concerns about conceptualization and the quality of assessment in these systems [28], as well as the challenges connected with programming education [1, 51].

The development and use of adaptive learning systems attempt to overcome the issue of heavy reliance on summative assessment and the "one-size-fits-all" approach [39], by addressing students' individual needs and capacities through distinct levels of knowledge proficiency, personal learning goals, and different intrinsic motivations that shape learning [4]. Adaptive learning systems provide students with individualized learning content and assignments, while accounting for their current knowledge and skills [77].

Currently, there are many semi-automatic and fully automatic code writing assessment systems, focused either on evaluation of students or on providing an adaptive feedback [17, 24]. The automatic assessment of programming assignments has been studied over the years with various tools, and presented as case studies [6, 15, 19, 57, 67, 68]. Many of the proposed solutions are based on code analysis and differential semantic analysis techniques. However, the accuracy of the generated grades has rarely been evaluated [57]. This raises questions about the quality of assessment these systems display, as well as the way in which assessment is conceptualized and implemented in those systems [28].

2.2 Adaptive assessment methods and their application in programming education

To achieve adaptability, learning systems need to estimate students' level of competence and the educational content difficulty levels [77]. Past research investigated various methods that estimate the difficulty of the learning content [33, 76, 92, 96]. Some of those methods include: proportion correct [29], learner's feedback [40, 69], expert rating [20], and the Elo-rating algorithm [77, 90, 96]. However, within the programming domain, adaptive assessment methods are mostly related to item response theory (IRT) [38, 92], extensions of the Elo-rating algorithm [2, 77, 79, 90, 96], the TrueSkill algorithm [55], or Bayesian networks [14, 44, 70].

One of the most popular methods to obtain skills' estimates is to use models from IRT. Basic IRT models have been built on the assumption of a constant skill [75]. Another possibility is using specialised IRT models, such as Bayesian knowledge tracing (BKT) [21] or performance factor analysis [73]. Despite their frequent use in the field of intelligent tutoring systems [26], these models are complex to use due to calibration on large samples. Moreover, frequent updates are computationally difficult to deal with when using IRT methods, since new skill estimates have to be calculated for each student after a single response for a single task [75].

A promising alternative to the IRT models is the Elo-rating algorithm [35]. This was originally developed for rating chess players, but lately, it has been used in education to overcome some of the gaps imposed by IRT models [77]. For example, the Elo-rating algorithm relies entirely on quantitative evaluation of student's performance and it can model skills that change over time. Next, when a new item is added in the system, there is no need to assign the correct initial item difficulty, as the system will learn and assign difficulty levels for educational content, based on the correctness of the students' answers [75]. Lastly, the Elo-rating algorithm can provide an accurate estimation of the tasks' difficulty with a small sample size, something that is not possible with IRT models [77].

A systematic overview of different variants of the Elo-rating algorithm and their application in education was presented in the work of [76]. The author compared the Elo-rating algorithm to alternative methods and illustrated its application in education. In this study the author demonstrated that the Elo-rating algorithm is computationally "cheap" and very simple to implement, and it is suitable for adaptive practices in educational settings. It is also confirmed that the Elo-rating algorithm is sufficient for guiding adaptive behavior, but it needs at least 100 students to get good estimates of item difficulty. In addition, the authors in [96] also confirmed that the Elo-rating algorithm can provide reliable estimates with a sample size of 200-250 students. Finally, there were attempts to combine IRT with Elo-rating to create an alternative model for adaptive item sequencing [2].

Several theoretical properties of the Elo-rating algorithm, such as stationarity of the mean, variance, and distribution, have been evaluated in [10]. The authors argued that the Elo-rating algorithm as an educational measurement provides correct parameter estimates. In addition, it offers many advantages mainly concerned with its simplicity, flexibility, cheapness and ease of implementation, storing a single parameter for each student and each item, as well as comparable performance to more complex models [76].

The proposed methods are mostly used to track and recommend exercises, where the outcome has a binary value (solved, not solved). However, in the case of programming assignments, the evaluation results cannot be simplified to a binary output, because every solution can be evaluated from different aspects (e.g., how efficient is the solution, number of attempts to solve the problem, time spent to solve the problem, etc.). Consequently, our study attempts to modify the existing Elo-rating algorithm and include additional parameters in the calculations that are important in the context of programming education. In particular, we consider additional factors, such as number of attempts and used time, rather than the solved or not solved status in an attempt to provide adaptiveness and recommendations. The aim of the new algorithm is to provide a simple and efficient method for automatic assessment of programming assignments. The calculated estimates are later used to recommend coding exercises to students that accurately match their knowledge proficiency. The applied algorithm and details about the recommendation process are presented in Section 3.

2.3 Recommender systems in education

Recommender Systems (RS) have been used in e-learning as solutions that consider students' preferences and proficiency, to adapt the learning resources in a way that complement students' goals and needs [7, 31, 83]. RS can support students to achieve a specific learning goal, provide annotation in context, or suggest a sequence of learning resources or activities [65]. Many researchers have examined different recommender techniques and their applicability to personal learning environments and have presented potential strategies for generating

learning content recommendations [58, 65]. Some of the techniques used include content-based filtering [74], user-based collaborative filtering [42], item-based collaborative filtering, stereotypes or demographics-based collaborative filtering, case-based reasoning, and attribute-based techniques [30].

Usage of contextual information in learning systems aims to improve the learning process [27]. Contextual information combined with RS in the e-learning domain results in more accurate recommendations provided to students [93]. In addition to contextual information, different types of feedback from users can also be used to enhance the quality of the recommendations [99]. This includes explicit and implicit feedback, where explicit feedback is a type of feedback that a user gives it intentionally (such as ratings) and implicit feedback is a type of feedback that is collected without users noticing it (such as the amount of time they spent on an item, click history, etc.). This feedback from users can be incorporated in either building separate user profiles or inferring the general preferences of the majority [30, 93].

In sum, the anticipated benefits of RS within programming education are numerous, and thus, it is important to investigate their benefits and undertake further research on how they tackle important challenges in the field, such as pairing students with applicable learning content or providing adequate feedback [8, 51]. Other, additional benefits of RS include enhanced reflection and improvements in the teaching process [16, 31], increase in learning efficiency [98], and automatic assessment [89, 95]. Another important benefit that RS introduce to students is exploratory learning [50], a process that encourages self-initiated, goal-oriented, and self-regulated learning activities. This is an important skill that CS students need to develop, as autonomous learning is critical for their future careers and professional development [66].

3 THE PROPOSED ADAPTIVE ASSESSMENT METHOD

The adaptive learning and assessment method, presented in this paper consists of three main stages (see Figure 1) executed as repetitive cycles:

(1) *Learning sessions.* At this stage students can access and interact with the provided learning resources. They can read about the basic programming concepts presented using text, video, and interactive examples. Students can also engage in solving interactive challenges and coding exercises presented in the system as learning activities that are aligned with the learning content. Since the system is adaptive, the students have the opportunity to follow the generated recommendations, or they can simply select learning activities that they find it useful, challenging, or interesting.
(2) *Adaptive assessment.* Before the assessment takes place, every student and all coding exercises are assigned with initial rankings (student's *knowledge level* and the exercise's *difficulty level*, respectively), represented using the same scale. As the students progress through the learning activities, the system updates their ranking based on their individual performance. This way the assessment is adaptive and continuous because the system evaluates every submitted activity (for example, an attempt to solve a coding exercise) and recalculates students ratings based on their quantitative performance. At the same time, the *difficulty level* of the learning content is also re-calculated after every attempt a student makes. In order to speed up the item difficulty calibration process, all coding exercises were initially assigned with an estimated starting score by the teacher. The details of the implemented algorithm are presented in the section 3.1.
(3) *Recommendations of further learning activities.* The system takes into account the students' knowledge ratings and the difficulty level of the content to further recommend coding exercises that match students' current proficiency. The student has an opportunity to choose whether she/he will follow the recommendation or not. Details of the implemented recommendation process are presented in section 3.3.

Unlike many IRT models, where students' rating is updated after every exercise based solely on the last action of the user, the method presented in our study includes constant recalculation of student proficiency and content difficulty levels after every submission, considering all previous submissions recalculated. In other words, when a

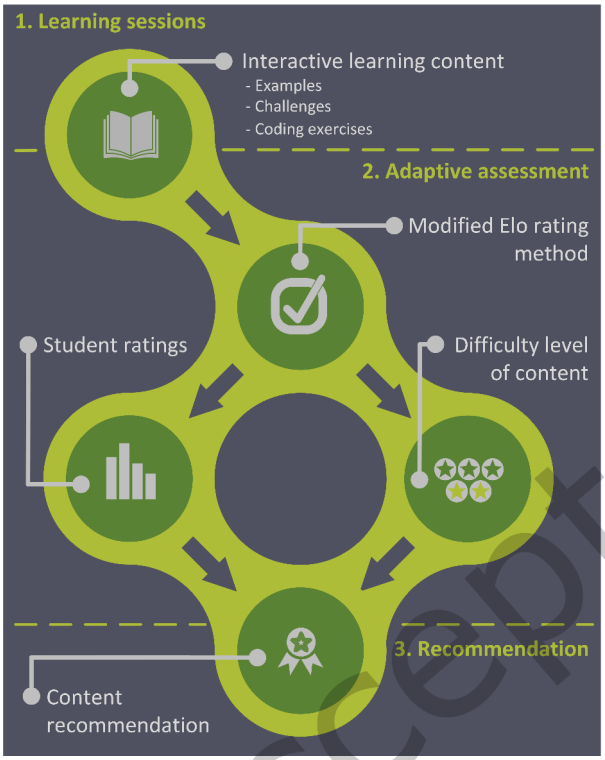

Fig. 1. Learning process in the system

student submits an exercise, the system will rate the student considering the current level of the exercise difficulty calculated not just for that particular exercise, but also recalculating it for all of the previous submitted exercises with their most up-to date difficulty levels. This way, the system allows for more precise evaluation of students' proficiency based on the most recent re-calibration of exercises difficulty level.

Furthermore, the presented method is able to take into account several parameters (e.g., the number of attempts, the time used for solving a task, etc.). Thus, the most important innovation of the presented Elo-rating method is the usage of multi-level estimates in open-ended programming tasks, without being limited to multiple-choice questions or binary-evaluated tasks (correct-incorrect).

3.1 The implemented algorithm

Considering that the Elo-rating method was originally developed for rating chess players, its adaptation to the field of education assumes that the student is the player and the content item (e.g., a coding exercise) is the opponent. Furthermore, the Elo-ratings of the student and the content are represented by a number, which increases or decreases depending on the students' actions with the learning resources. The difference in the ratings between a student and an item serves as a predictor of the outcome of the action. For example, the greater the difference is, the *expectancy level* that the student will solve the exercise is higher.

When students sign in to use the system, they all start with the same initial ranking (i.e., 1300), and the basic logic of the formula considers the changes in the students' performance, thereby students gain points if they perform above their *expectancy level*, and lose points if they perform below their expectancy level [10]. For

example, if a high-rated student successfully solves a less difficult exercise, then a few rating points will be added to their current ranking. However, if a lower-rated student solves an exercise that is above their current rank (i.e., a difficult exercise), the student will receive more ranking points. Consequently, a student who has an underestimated initial rating should in the long run perform better than what the rating system predicts, and thus, gain rating points until the ranking reflects their true knowledge. At the same time, the difficulty level of the learning content is recalculated and adjusted based on the knowledge level of students who managed or failed to solve the exercises. These two strategies allow the rating system to act as a self-correcting system.

The presented algorithm takes into account:

- the number of successful/failed attempts;
- the percentage/rate of successful unit tests;
- the time needed to solve an exercise.

The proposed method estimates the probability that a student i is able to solve the coding exercise j based on its current rank and the difficulty of the coding exercise j. The ratings of the students and the difficulty of the coding exercises are updated as follows:

$$R_i = R_{i-1} + K \cdot (W - W_e) \tag{1}$$

$$D_j = D_{j-1} + K \cdot (W_e - W) \tag{2}$$

Where:

- R_i represents the new rating of a student after the event (i.e., solving the exercise).
- R_{i-1} represents the pre-event rating of a student.
- D_j represents the new level of difficulty of a coding exercise j after the event.
- D_{j-1} represents the pre-event difficulty of the coding exercises.
- K represents the rating point value, i.e., a constant, specifying the sensitivity of the estimate where $K \in \{30, 20, 15, 10\}$. The K-factor determines the âĂIJsensitivityâĂİ of how wins and losses impact the rating. This value is gradually decreased as the student solves more exercises.
- W represents the result from solving the recommended exercise (i.e., the success rate), calculated by the following formula:

$$W = \left(\frac{(A_s + A_o)T_c}{2A_oT_p}\right)(a_id_i - a_it_i) \tag{3}$$

  - A_s represents the number of successful attempts in solving a problem.
  - A_o represents the number of overall attempts to solve a problem.
  - T_p represents the number of performed unit tests.
  - T_c represents the number of correct unit tests.
  - a_i represents a discrimination parameter. This parameter is used to adjust the time parameter discrimination, ensuring balanced influence of outlined time values.
  - d_i represents the time limit.
  - t_i represents the time a student needs to solve a problem i.
- Finally, W_e represents the probability that the student will solve the exercise:

$$W_e = \frac{1}{1 + 10^{\left(\frac{R_{i-1} - D_{j-1}}{400}\right)}} \tag{4}$$

This method, implemented in the programming tutoring system, differs from other implementations of the same algorithm. The difference is in formula (3) which calculates the success rate, W. The proposed method calculates this value based on the ratio between the successful attempts, the overall attempts, the number of unit

tests performed, and the number of correct unit tests. We argue that exercises which were solved correctly at the first attempt can be labeled as "easier" for that particular student, compared to exercises that required multiple solving attempts from the same student. We also argue that exercises which require a student to spend more time (as registered in the log) reaching a solution can be labeled as "difficult" for that particular student. However, these two properties cannot be considered as the sole indicators to distinguish the difficulty level of coding exercises because students are allowed to repeat the assignments until they finally succeed. Hence, formula (3) adds a response time (i.e., time to solve an exercise) in the calculation of the success rate. The authors argue the importance of how quickly a student can understand the problem and break it down into parts. Programming speed also shows the studentâĂŹs ability to think at a high level and that he/she is not overwhelmed with the complexity of the problem. The system ensures that the student is actively engaged with the chosen content by defining an idle timeout value which starts when the student begins solving a coding exercise. The current session is terminated, and the student is logged out if no activity is detected for a specific length of time (i.e., 180 seconds). When the session is terminated this way, the timeout value is subtracted from the overall session duration. Although this method for handling the idle time might not be ideal, it does result in a more accurate estimation of the session time.

Therefore, the answer receives a score based on the correctness of the solution and the response time for that solution t_i [52, 76]. With this rule, the stakes are higher when a student provides a quick response. The faster response can result in a more significant increase of the student's rating in the case of correct submission and a more significant fall in the same rating if the student submits a wrong answer. If the response time is longer, the score converges toward zero (i.e., for both correct and incorrect answers). However, the influence of the time factor should be balanced (with the discrimination factor) to prevent extremely low values, which would significantly decrease the final changes in rankings.

3.2 The learning content provided in the system

The learning content (i.e., resources) included in the system consists of four types of activities that support individual work. Students can practice and learn programming through the following learning content:

(1) *Explanations (ProTuS).* ProTuS contains reading content (i.e., tutorials) on 15 topics that are aligned with the curriculum presented in the course. However, these 15 topics are not the exhaustive list of topics that are being taught in introductory programming courses [62]. The learning objective behind the reading content was to offer students the opportunity to extend their Object-Oriented Programming (OOP) knowledge on top of their existing knowledge in procedural programming (as students had already undertaken a CS course in Python). Thus, we have developed content for 15 different topics in relation to basic concepts in Python and Java, emphasizing on syntax comparison. An example is provided in Figure 2.
(2) *Examples (MasteryGrids-PCLab).* For each of the proposed topics in the system, there are established examples, called Program Construction EXamples (PCEX) [45]. The idea is to support students in acquiring program construction skills through a new type of *smart and interactive content*. Each PCEX content starts with a worked-out example where it is explained how to write code for a particular problem. The explanations are available for almost all lines in the examples and they focus on why students need to write code in a certain way or why certain programming constructs are used in the code. The explanations for the lines of code are hidden until a student clicks on the line of interest (Figure 3).
(3) *Challenges (MasteryGrids-PCLab).* Following the pedagogical reasoning that examples are more effective when students utilize the knowledge gained immediately to solve a problem similar to the given example, we decided to present a challenge after each example [22]. Therefore, students could try to solve one or more challenges related to the example they had previously viewed in order to consolidate the knowledge gained. Each challenge displays a problem similar to the code that the student had viewed in the example

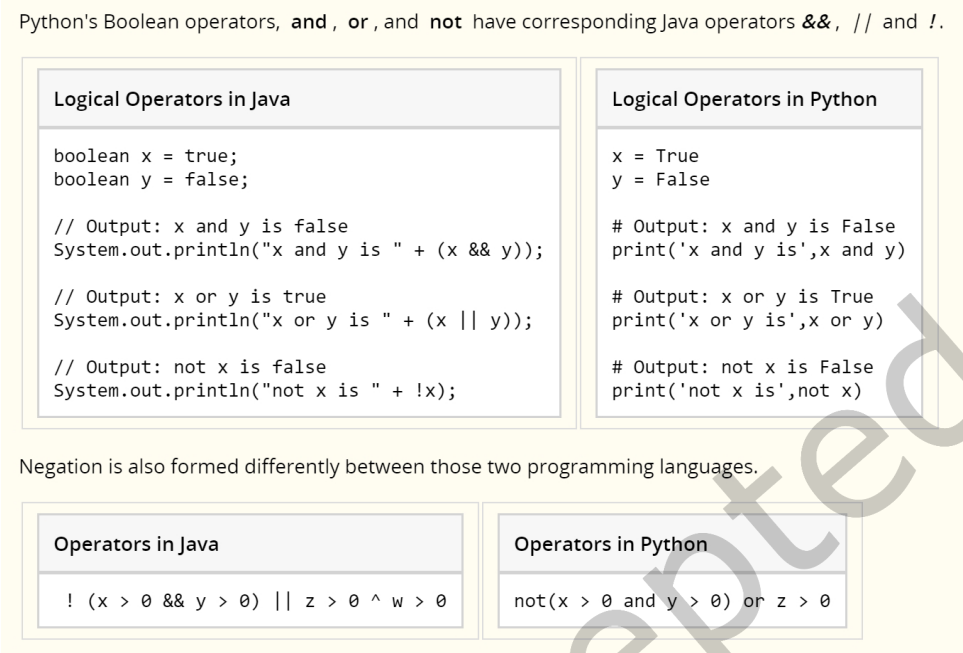

Fig. 2. Syntax comparison in the system

with some blank lines that need to be filled in by dragging and dropping the pieces of code to the blank fields [13]. Immediate feedback and hints are available for each challenge (Figure 4).

(4) *Coding exercises (MasteryGrids-PCRS).* This type of smart content require students to write code or complete a given code to achieve a certain goal. The system that provided this content is known as the Programming Course Resource System (PCRS) [78, 100], whose content server resides at the University of Toronto. Each coding exercise have a problem description and a baseline code. When students submitted their code, the code was tested against a set of unit tests developed for that particular problem and the student received a prompt feedback on whether the written code has passed the tests or not. This automated feedback highlights both syntax and run-time errors - for which specific cases and parameters those errors appeared. The challenges and the coding exercises could be attempted multiple times (Figure 5).

The context of our study was focused primarily on the *coding exercises*, as this content is the only resource to which we have access to make changes in the user modeling technique. This type of interactive content requires students to complete a skeleton code, which after it is submitted, the code is instantly tested against a set of previously developed unit tests. During this process, the system tracks all submissions, ratings from students' feedback, and the time need it for students to solve a particular coding exercise. A faster response time in solving an exercise is not an ideal indicator of student's proficiency; however, we chose this method to calculate a more accurate session time controlling for the idle time.

3.3 The applied recommendation process

The core of the recommendation process is based on the proposed Elo-rating method to estimate students' knowledge and recommend coding exercises that match their current proficiency [63]. In every moment, students receive three to five recommendations of coding exercises that the system has calculated to be the most suitable

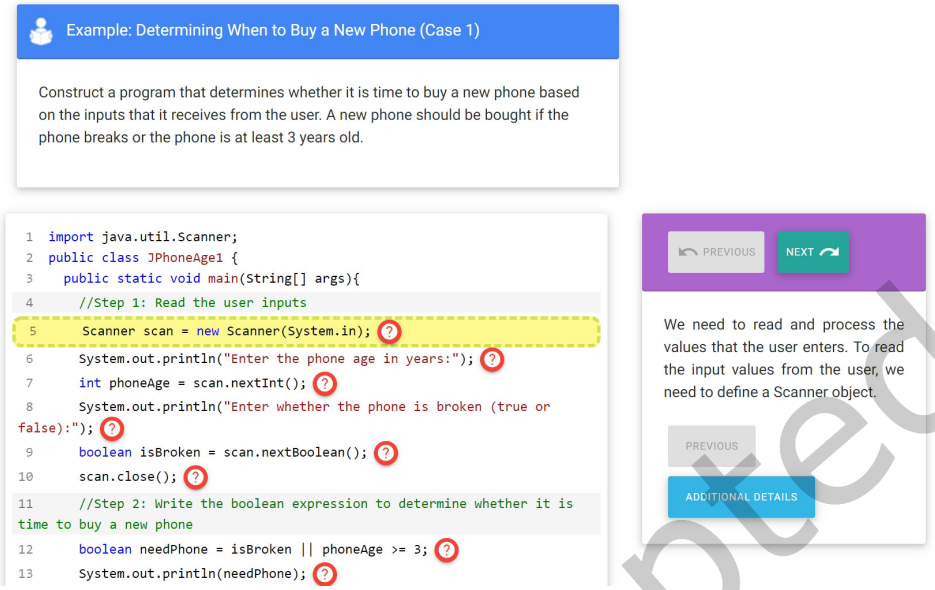

Fig. 3. Display of an Example

for a particular student in relation to that student proficiency in programming (Figure 7). The recommendations run between 3 to 5, depending on the skills a student demonstrates, the difficulty level of the exercise and the process between the two sides once the student engages with the system. Then, it is up to the student to choose between the recommended coding exercises or the other coding exercises that are not currently recommended by the system, but are still displayed in ProTuS (Figure 6). Hence, when a student decides to undertake a more complex coding exercise, that student could gain more ranking points for solving the exercise. However, if a student decides to choose a less complex coding exercise (i.e., potentially less challenging), that student could gain fewer ranking points. Thus, this adaptive strategy further determines the type of coding exercises that could be recommended to students based on their current activities and knowledge proficiency.

4 METHODOLOGY

The main goal of this study was to put into practice the proposed adaptive assessment and content recommendation module in the context of a programming course. In particular, we investigated the efficiency and the performance of the proposed Elo-rating method in an introductory OOP course. We argue that analyzing the method's strengths and weaknesses could provide us with further guidelines on how to improve the ranking technique and the recommendation process.

To tackle the four research questions (RQs) of this paper, we performed the following objectives.

(1) implement an adaptive assessment and content recommendation based on a modified Elo-rating method (as described in section 3);
(2) conduct a semester long study to collect quantitative and qualitative data from students' use and experience; and
(3) perform data analyses to investigate how the proposed method performs in real-time and address the RQs.

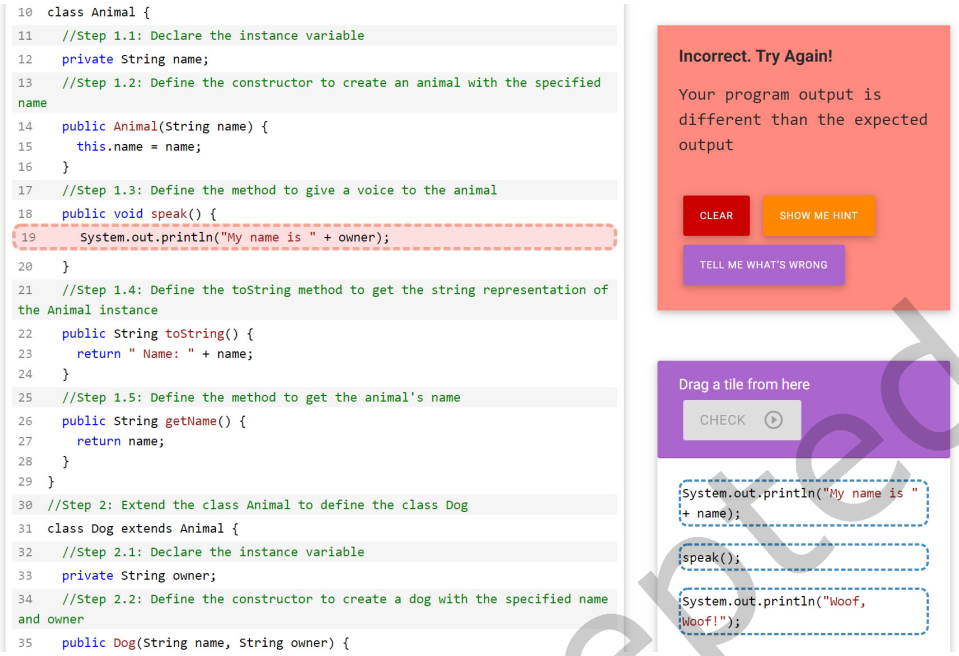

Fig. 4. Display of a Challenge

In this section, we describe the context of the study and the methodological details, such as the measurements used and the performed analyses.

4.1 Context of the study

Before running the study, a prototype module for content recommendation was developed and integrated into ProTuS - programming tutoring system[1], which was hosted at a university's server [94]. The system was available throughout the whole semester to the students who wanted to use a tutoring system with smart content as an additional learning resource to practice their programming skills. Thus, the students were asked to use this module in-the-wild (i.e., whenever they feel the need to use it without being a mandatory part of the course design), and after the end of the semester they were asked to fill out a survey regarding the learning outcomes and the effectiveness of the implemented module.

4.2 Participants

CS students from two public European universities (University of South-Eastern Norway and Norwegian University of Science and Technology), undertaking an introductory course in OOP, were invited to participate in the study and practice their programming skills in Java using ProTuS. The sample consisted of students without previous experience in OOP (considering their study program curriculum). The students had already taken an introductory (procedural) programming course in Python (i.e., CS1) in the previous semester, so we assumed that they had already mastered the fundamentals of procedural programming.

[1]ProTuS: https://protus.idi.ntnu.no/

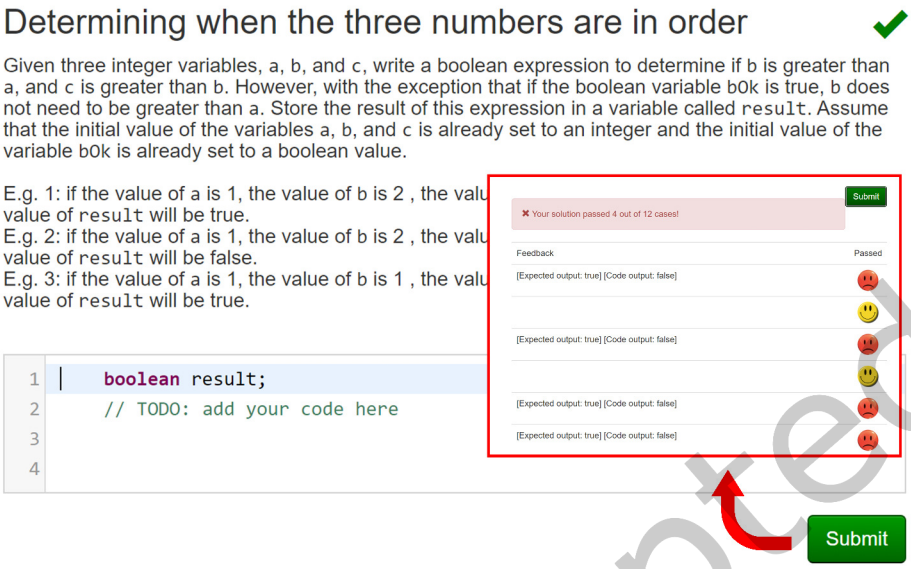

Fig. 5. Display of a coding exercise

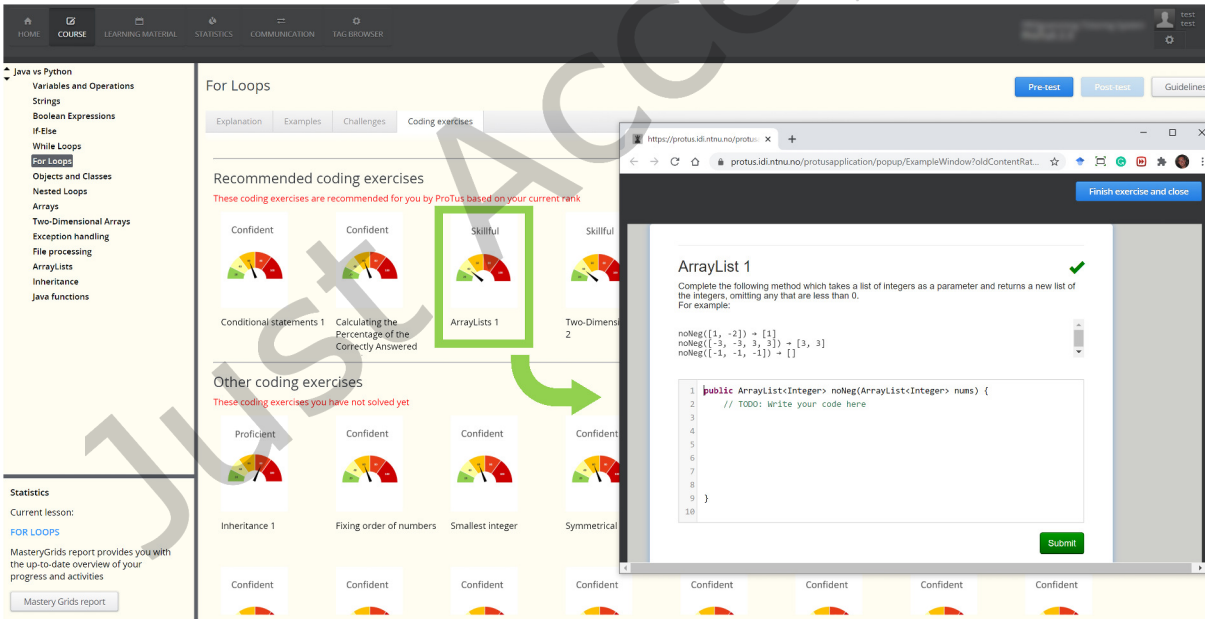

Fig. 6. User interface of the system

Overall, there were 701 students who used the system during the Fall 2019 and Fall 2020 semester. However, one-time visit or very limited interaction with the system do not provide us with meaningful data required

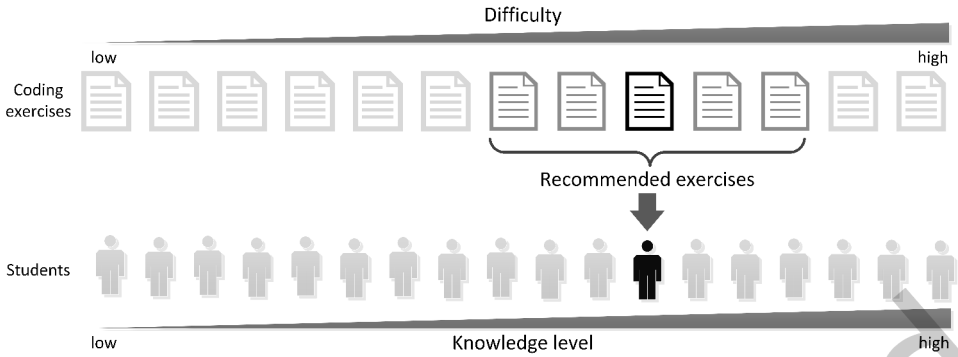

Fig. 7. Recommendation of coding exercises

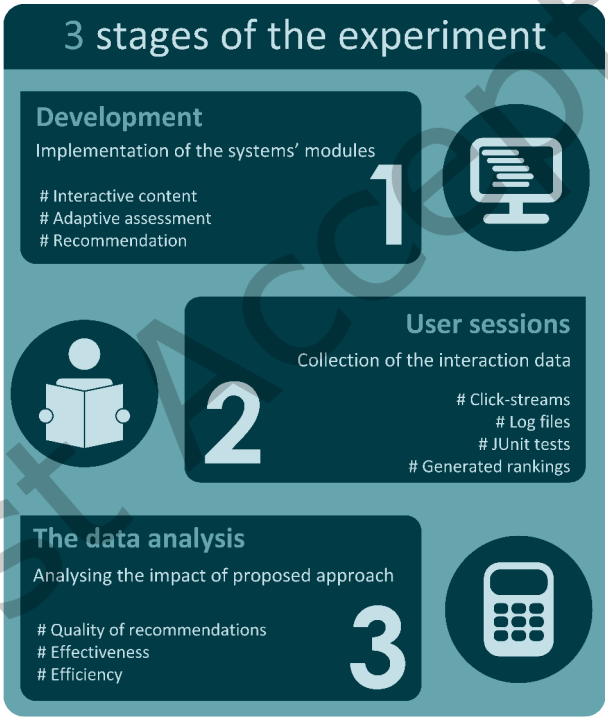

Fig. 8. Stages of the experiment

to investigate how the proposed method performs in real-time. Therefore, the analyzed data sample included only students who have solved more than 10 coding exercises (n=87). The study lasted one semester, with two generations of students (31 students in 2019, and 56 students in 2020).

4.3 Data collection

The following data were collected to evaluate the efficiency and effectiveness of the proposed method (i.e., Elo-rating algorithm combined with unit-tests):

- **Students' interaction data** with ProTuS (e.g., browsing history, session logs, students' clicks on any element of the user interface, duration of actions, students' selection of recommendations, etc.). These records were used to analyze students' activities and their response to the system recommendations.
- **System-generated rankings of students**, which were calculated based on the students' interactions with the learning resources (e.g., number of submission attempts, number of trials/errors with the coding exercises, time to complete the coding exercises, number of submitted solutions, etc.).
- **Survey data** regarding the students' learning outcomes and the effectiveness of the implemented module, collected at the end of the semester (i.e., Fall 2020). The survey consisted of 8 questions, of which 5 were open-ended questions and three were yes-no questions.

4.4 Measurements

To investigate how the proposed method performs in real-time, we introduced metrics to quantify the quality of the recommendations, the effectiveness of the proposed method, and the efficiency of the modified Elo-rating algorithm.

The quality of the recommendations measures the percentage of the generated correct recommendations of coding exercises made by the tutoring system [86]. Thus, to measure the quality of the recommendations (i.e., RQ3), we used the standard precision and recall evaluation metrics [41, 42].

- **Precision** is defined as the percentage of recommended items that truly turn out to be relevant (i.e., consumed by the student).
- **Recall** is defined as the percentage of relevant (i.e., ground-truth positive) items that have been recommended as positive.

These commonly used evaluation metrics are calculated as:

$$Precision = \frac{t_p}{t_p + f_p} \tag{5}$$

$$Recall = \frac{t_p}{t_p + f_n} \tag{6}$$

where:

- t_p (true positive) represents the number of coding exercises recommended and started by the students.
- f_p (false positive) represents the number of coding exercises recommended but not started by the students.
- f_n (false negative) represents the number of coding exercises started by the students but not recommended.

Using precision and recall, we calculated the trade-off between the number of recommended exercises generated by the system and the number of exercises undertaken by the students. In practice, the programming tutoring system creates rankings of the coding exercises based on the implemented Elo-rating algorithm; hence, the top-k coding exercises are always recommended to students. However, the students have the opportunity not to choose the generated system recommendations, and therefore, we decided to look into students' selection of exercises when they choose not to follow the system recommendations. In particular, we were interested to see if the students tend to select coding exercises that are close to their proficiency level or select exercises above/below their current rating.

The Effectiveness of the proposed method is represented through the quality of the system generated grades (i.e., RQ1). To evaluate the quality of the system generated grades, we compared those grades with the actual grades (i.e., based on their final exam) students received at the end of the course. For example, the students

were assigned with the same initial ranking (i.e., 1300) and, after a series of assessments, their ratings roughly fell in the [1100,1500] range (including outliers). These ratings represent a set of **system generated rankings**. Students also received a *course grade* from their teachers because they have attended the university course in OOP from which the students were recruited for the study. Students' interactions with the tutoring system and the outcome of the exercises submitted in that system were not considered in the students final course grades.

4.5 Data analysis

To answer the first RQ (i.e., Elo-rating effectiveness), we compared the grades that the tutoring system generated for each student with the grades that the teacher assigned at the end of the course, by performing a paired sample t-test [47]. The final grades assigned by the teacher were calculated based solely on the final exam. The course grades were scaled to the [1100,1500] interval in order to be comparable to the system generated ratings (the interval has been divided into six equally sized sub-intervals, each representing one grade of the grading scale used at the university where the study took place). Hence, we performed a paired sample t-test [47] that depicts if there is a statistically significant difference between the two means. Although with certain limitations, we argue that this method is suitable due to the thoroughness and acceptability of the final grades assigned from the teacher. Therefore, it has been used to measure the performance of a sample of students in two different activities and analyze their differences. Hence, the null hypothesis states: "There is no significant difference between the grades generated by the system and the final course grades assigned by the instructor".

When it comes to the second RQ (i.e., the efficiency of the proposed method (Elo-rating algorithm and unit testing) in real-time adaptive assessment of programming tasks) we calculated how quickly the system can start giving relevant personalized results. ***The Efficiency*** of the method is evaluated by analyzing the number of calculation steps required (i.e., how many coding exercises a student needs to solve) for the system to generate an accurate student ranking (i.e., RQ2). In other words, it represents the measure of how quickly the system can start giving relevant personalized results. To evaluate the efficiency of the proposed algorithm for an automatic and objective way of generating ratings, we studied the effect of the number of accessed and solved exercises over the performance of the proposed algorithm.

To answer the third RQ and investigate the accuracy of provided content recommendations we used the standard precision and recall evaluation metrics. Details about the measurments are provided in the section 4.4.

When it comes to the fourth RQ (i.e., studentsâĂŹ experience with the proposed method) we conducted a thematic analysis of the qualitative data collected from students at the end of the study (Fall 2020 semester). In particular, we applied an elicitation technique and conducted a thematic analysis on the survey data following Braun and Clarke [9] six-step framework. Unlike many qualitative methodologies, this analysis is not tied to a particular theoretical perspective [59]. Because the nature of the study was exploratory, so was the analysis; thus, we used an inductive approach and looked for themes that emerged from the text [9]. The survey data was divided between two researchers who coded the data individually. Later, the differences were settled, and we end the process achieving a high inter-rater reliability (CohenâĂŹs k = 0.78). The analysis led to four themes and eight unique codes *(i.e., practice options, learner characteristics, assisted guidance, ITS benefits, personal choices, individual needs, useful content, and current drawbacks).* Most of the coded nodes are associated with one theme, but some are associated with more than one theme.

5 RESULTS

The students who used the system and were included in our study (a total of 701 students registered into the system), engaged in 4528 user-sessions with 6920 visited content (students' visits to examples, challenges, or coding exercises). A typical student spent circa 49 minutes within the system. A total of 1016 coding exercises have been tried to solve. The detailed overview of the collected data is presented in Table 1.

Table 1. Users' session details

Users' session details	
Overall number of systems' users	701
Number of solved coding exercises	1016
Number of active users (solved more than 10 exercises)	87
Overall number of user sessions	4528
Average overall session time of a student	48m54s
Number of visited content	6920
The number of students that responded to the survey	34

Table 2. Results of recommendation in ProTuS

Educational content	**Precision**	**Recall**	**Recommendation technique**
Coding exercises	0.65	0.62	Elo-rating
Examples and challenges	0.64	0.63	Adaptive sequencing recommendations

Quality of the recommendations

Precision and recall have been used to evaluate the quality of the content recommendations (Table 2). The results showed that the probability of the implemented Elo-based algorithm recommending relevant coding exercises was 0.62 (recall), and the probability that all of the recommended coding exercises were relevant was 0.65 (precision).

To further analyze the behavior of the students on an individual level, *Precision* and *Recall* were calculated for each coding exercise (Figure 9) and student individually (Figure 10). As one can notice, the figures contain fewer dots than the reported number of students and visited content, but this is due to the overlaps at some points in the graph.

Analyzing the content in Figure 9, one can notice that majority of content lies above the 0.5 point mark of precision, meaning that the algorithm returned substantially more relevant results than irrelevant ones. Looking at the individual choices of students (Figure 10), one can observe that the students could be roughly divided into two groups. The first group includes students who almost blindly followed the recommendations (recall is 1 or in close proximity), although they were not instructed or asked to do so, but freely choose to solve the exercises that match their proficiency level. The other group of students almost completely ignored the recommendations and preferred to select coding exercises based on a topic or concepts they were struggling with during learning.

For the purpose of our study and as already mentioned in the data analysis section, we decided to look into the students' selection of coding exercises when they were not following the recommendations. Thus, we analyzed the average difference between the current rank of every student and the difficulty of the chosen exercise. The results are presented in Figure 11 and they describe how values for a single student are spread across the entire range (the outliers have been removed).

It can be observed that 50% of the choices that students made fall into a small range of [-44,83], meaning that overall, the students selected exercises similar to their own ratings. On average, they tended to select the exercises slightly above their rank, which is indicated by the median of 8, showing the ambition to achieve better results. However, their choices are spread along a wide range, so a significant number of students experimented with exercises of high and low difficulty.

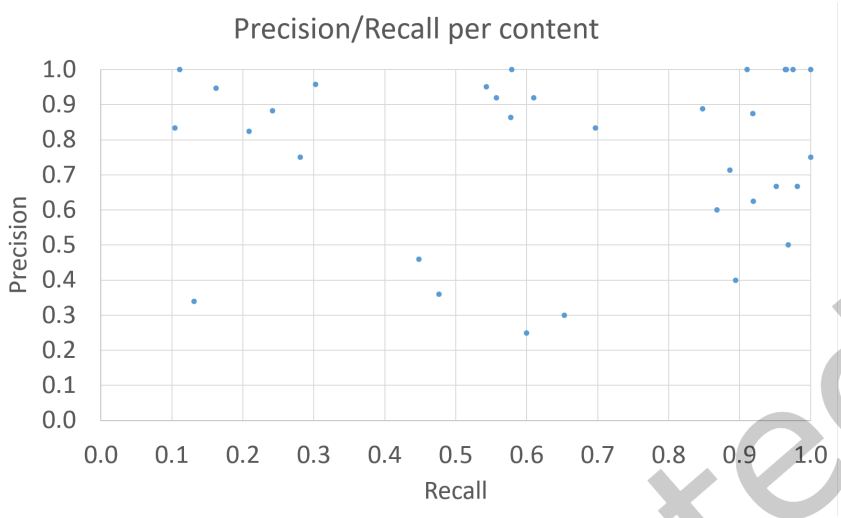

Fig. 9. Precision/recall plot of the performed recommendations - per content

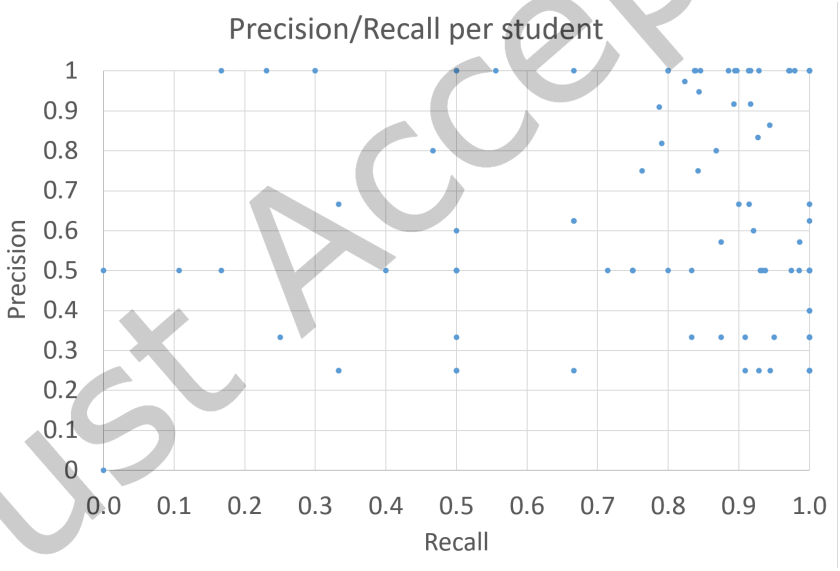

Fig. 10. Precision/recall plot of the performed recommendations - per student

Effectiveness

A paired t-test was conducted in order to determine whether there is a significant difference between the grades generated by the system (i.e., system rankings) and the grades that are given by the instructor. Table 3 displays the descriptive statistics for these two sets of grades. We can see from the two means that the system generated grades have a similar mean (mean = 1410.51) to the course grades (mean = 1408). Moreover, the standard deviation

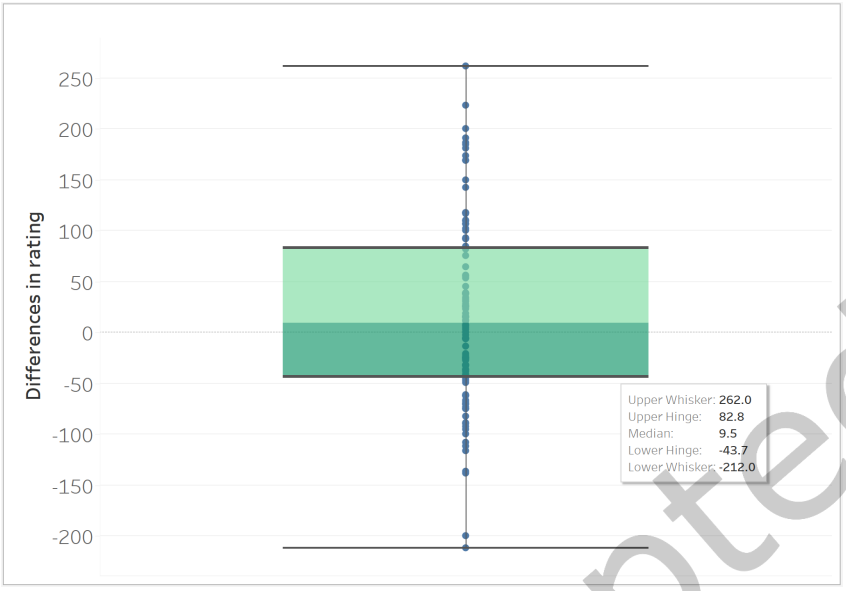

Fig. 11. Differences in the difficulty level of exercises and the current

Table 3. Descriptive statistics of two sets of grades

	Mean	N	Standard deviation	Standard error
Systems' generated grades	1410.51	109	113	11
Course grades	1408	109	184.26	17.65

Table 4. Dependent t-test results

	Mean	r	95% confidence interval		t	df	p-value
			Lower	Upper			
Pair: 2 sets of grades (df$ELOrating and df$Course.points)	2.1955	0.011	-37.58	41.97	0.1094	108	0.9131

shows that the grades given by the instructor have a higher deviation compared to the system grades, which are very close to the mean.

The results of the performed paired t-test are presented in Table 4. One can notice that there is not a significant difference between the grades generated by the system and the course grades assigned by the teacher, ($t(108) = 2.1955$, $p = 0.91$, $r = 0.01$), meaning that with 95% confidence we can keep the null hypothesis.

Efficiency

In order to test the efficiency of the proposed method, we investigated how many steps are needed for the proposed algorithm to generate an accurate student rating. In practice, this means that we investigated how many coding exercises a student needs to solve (or try to solve) in order to be ranked accurately. To increase the accuracy of rank estimation, the analysis focused only on students who solved at least 10 coding exercises.

Figure 12 shows how the ratings of the most active students changed over time. The chart displays only students who completed at least 10 exercises. The x-axis represents the number of coding exercises students solved, while the y-axis presents their ranking in a particular moment. Thus, it can be observed that the biggest changes in rating occurred after the first 6 to 7 solved exercises. After the seventh step, ratings only slightly converged toward their final ratings. Based on that fact, it can be concluded that the requirement for the successful rating of students with the proposed method requires no more than 7 steps; the optimum can be achieved in a maximum of 10 iterations. If we compare our findings with findings reported in the past, we can say that the efficiency of the proposed method does not lag behind other more popular methods used in digital assessment [2, 77, 97]. However, additional evaluations are needed to compare the efficiency of the proposed method with other popular methods that are currently in use.

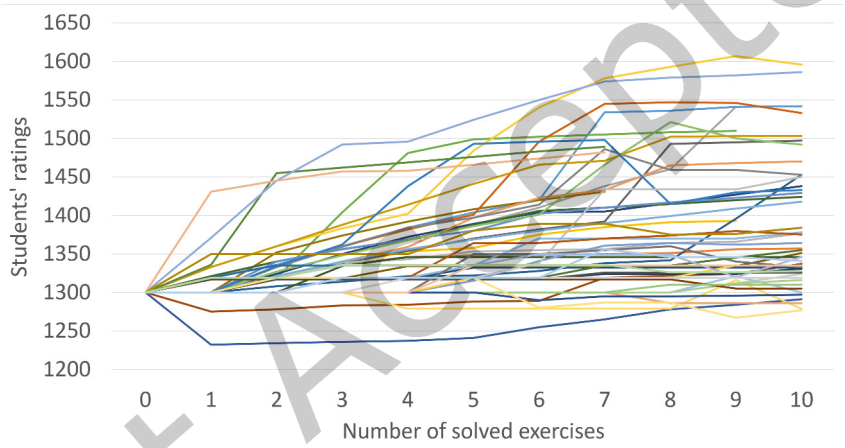

Fig. 12. Trends of students' ratings

Themes from the thematic analysis

Theme 1: Supporting students' empowerment, confidence and problem solving skills. When it comes to learning with ITS, the majority of the students agreed that their voluntary exposure to the system throughout the semester supported the learning outcomes of the OOP course (82.4% of 34 participants). The system offered relevant content that allowed students to engage with "learning by doing" practices, with most the students identifying this as the factor which led to actual learning, - *"I learned more by actually coding and troubleshooting errors rather than watching a two hour lecture"*. Moreover, the proposed activity empowered students to achieve actual learning outcomes by engaging to "trial and error" and "search for new knowledge" practices, especially for the novices, who had to search online for particular concepts, terminology, or to understand the way tasks are composed and worded; - *"I had to do research to be able to solve some of the tasks, which in turn helped me to improve my knowledge"*. Several students also said: - *"[the system] helped me [to learn] by making me search for new knowledge"* and - *"it [the tasks] made me think and research before answering [submitting the code]"*.

In general, almost all students found the usage of the system helpful toward improving their proficiency and some of their personal beliefs/feelings toward programming. Some students expressed that - *"The immediate feedback from the system when it tells you if your code works or not, gives you confidence to try different approaches"*, - *"I got better at debugging and understanding logic how to approach different problems"*, as well as - *"It helped me to become more confident to write the correct syntax"*. At the end, the main benefits from learning with ITS that the students listed in the survey included: - *"It's [offers] short tasks with not too much [acceptable] complexity"*; - *"I learn the things I actually need for the exam"*; - *"I can complete the tasks while being able to verify [immediately] the result"*; and - *"the explanations [the automatic feedback I get] after I submit the code"*.

Theme 2: Complementary nature of adaptive assessment in programming education and the importance of a tailored "assessment trajectory". This theme elicited from the participants views on the use of the proposed adaptive assessment method as an additional content source in programming education. Students said that there is a good alignment between the course outline and the adaptive assessment method of the system, which is the reason they positively viewed and used the system as an additional resource to practice Java and improve their skills. Another reason that adds toward their optimistic behavior for usage of adaptive assessment in programming education was the possibility to choose content that one finds it interesting, relevant, or need it at a particular moment during the learning process, with integrated automatic feedback (e.g., syntax errors, run-time errors for specific cases and values). One student pointed out - *"[this system] is not liner. It gives me opportunity to jump to a topic I feel I need, where I can find various levels of difficulty for the same concept I study"*.

Approximately half of the students expressed a positive attitude regarding the "assessment trajectory" they received from the system itself; - *"I particularly enjoyed the guidance from easy to headache tasks"*. However, although some students found the system helpful to - *"gradually tackle more difficult [programming] problems"*, majority of the students (i.e., 64.7% of 34 participants) did not follow the path lined by the system for them. Many students expressed their concerns regarding the recommendation process; in fact, 64.7% stated that the system recommended tasks that did not meet their needs; - *"I followed my own selection [of tasks] because the recommendations were not aligned to my needs"*, - *"I did not find the recommendations useful or easy to tackle"*; and - *"I followed my own path, because I know what I need"*. In addition, several students said that they used both ways, the generated recommendations by the system and their own selection, and they used their own selection when they could not understand or solve the recommended tasks.

Theme 3: The promise of the confluence of adaptive assessment with other advanced capabilities (e.g., gamification, adaptive navigation) to support programming education. Students communicated several ideas about what they expect from adaptive assessment that can assist them to improve the way they learn. First, they pointed out to the need for more gamification elements in the system; - *"There is [already] a certain gamification which makes learning fun in [Anonymous]. I would be even more motivated to use the system if directed more heavily towards gamification"*. Second, students reported some issues with the user interface. They said: - *"The cold start in [Anonymous] is not easy"*; - *"Although it is frustrating [to use the system] at times, it actually helped me learn. I just expect more intuitive interface"*; and - *"I often felt confused on where I was on the website, and both the recommendations and the navigation felt like more of an obstacle rather than being helpful. When I found the Master grids overview I completely stopped using all other methods of navigation. The [MasteryGrids] panel gives a very intuitive way of seeing what I have and I have not done"*.

Theme 4: Offering "on the spot" support and feedback in programming education. Last, students positively expressed their attitude toward the immediate feedback (e.g., syntax errors, run-time errors for specific cases and values) they received after submitting the written code; - *"It helped me explore various approaches I had in my mind and see how my code worked [based on each of those approaches]"*, which is something they expect to be even more adapted in future; - *"Well, I found that it was quite difficult to see what aspects of the code that*

was wrong when I got errors. Like when i got 14/15 correct on one task, and it gave no message of what the last task was, only a Sad emoji". In future versions of the system, it will be beneficial to improve the recommendation process, i.e., align the generated recommended tasks with the students' needs which currently the students find it challenging; - *"I expect the recommended path to be the best way to gain skills efficiently"*.

6 DISCUSSION AND CONCLUSION

Assessment is a key component of education as it helps students to reflect on their progress, be aware of the knowledge gaps they have, and act as guidance where to focus to further enhance the development of their skills [32, 37, 63]. Recent studies in programming education indicate that automatic assessment has the capacity to successfully prepare (even sometimes better compared to demonstrators and lecturers) students for situations where they have to write the code by themselves; thus, motivating the development of self-sufficiency [60]. Moreover, automatic digital assessment has the capacity to motivate and encourage students' skills development as it provides early and continuous feedback per activity (i.e., after every submitted exercise in a system) [37], as well as to increase programming practice and decrease fraudulent behavior [5].

The ultimate goal is not only to automate the assessment of programming assignments, but to utilize computerized adaptive practice in online or blended programming courses. Therefore, the aim of the study was to explore how accurate and precise the Elo-rating algorithm is for recommending learning content (e.g., coding exercises) and, thus, how efficient and scalable the proposed adaptive assessment module is. As a first step toward automated assessment in introductory OOP programming courses, we explored how to develop a simple and efficient method for adaptive and automatic assessment of programming assignments, leveraging smart and open learning content [12], utilizing unit tests, and a state-of-the-art ranking algorithm (i.e., Elo-rating) [77]. With considerable caution and the respective contextual limitations, we want to report positive findings for the feasibility and potential of using this modified Elo-rating method in recommending coding exercises.

Our findings show that the implemented modified Elo-rating algorithm returns substantially more relevant recommendations than irrelevant ones (see Figure 4). Moreover, the Elo-rating method finds relevant coding exercises for a particular student 65% of the time, and it is 62% precise in recommending the proportion that is actually relevant for that particular student. The numbers might not appear very high, but our results are comparable with results coming from the adaptive sequencing recommendation method [46] used for recommending examples and challenges.

In addition, we observed two core groups of students, one who constantly followed and selected the recommended exercises without being instructed or asked to do so, and another who were creating their own learning paths. We argue that the group who selected the recommended coding exercises found this recommended learning path comfortable enough to follow it throughout the learning experience, confirming that the reported values are adequate as a proof of concept. On the other hand, the students from the second group who were creating their own learning trajectory and did not select any of the recommended content resulted in a good matching of the selected coding exercises with their competence level. In particular, the results presented in Figure 11 show that those students were quite good at selecting coding exercises that were aligned with their rankings. They avoided selecting exercises that were far beyond their knowledge competence levels, or were very easy for them to solve. Therefore, the results indicate that some students are aware of their own expertise and possible knowledge gaps, leading them to be confident in creating and choosing learning trajectories that keep them engaged and inflow (i.e., "completely involved in something to the point of forgetting time, fatigue, and everything else but the activity itself") [23, p.15], assisting them in progressing toward their own learning goals. For those two core groups to be supported, future research in programming education can develop dedicated methods and system functionalities that scaffold those two strategies and support students to select/receive better recommendations.

Regarding the method's effectiveness, we compared the ratings that the system generated with the grades that the teacher assigned to the students at the end of the course. The dependent t-test showed that there is no significant difference between the ratings generated from the system and students' final grades. The discovered and reported differences have their own limitations (e.g., the way those rankings were made, the time period over which those rankings were influenced, the competences they are based on), for which we would welcome further investigations in the proposed method, but also which inform our contemporary teaching and assessment approaches (e.g., adequacy of exams and exercises to evaluate programming knowledge).

We also calculated the method's efficiency by considering the number of steps required to generate an accurate student rating. The number of steps was represented through the number of correctly solved and submitted exercises, which in our case is seven steps. However, we have not compared the efficiency of our method to another methods that are often used in adaptive RS, which is planned for the next study. We also want to implement a method based on IRT and see if that method might give better results than the proposed modified Elo-rating method.

Finally, we have analyzed the students' quantitative insights on the use of the proposed adaptive assessment method. The students indicated overall positive views and particularly highlighted four areas that the proposed method was beneficial for them (as they came out from the thematic analysis of their responses). First, it supported their confidence and problem-solving skills, and empowered them to sustain their efforts to solve the problem and even seek external knowledge. They also said that the use of the system helped them to achieve actual learning outcomes by engaging to "trial and error" and "search for new knowledge" practices. Second, the adaptive assessment served as a complementary resource for them and they found the tailored "assessment trajectory" helpful. Third, the students recognized the importance of coupling adaptive assessment with gamification capabilities. They indicated the importance of intensifying the use of advanced capabilities (e.g., further gamification, navigation), indicating that this is likely to lead to increased motivation and engagement with the content. The fourth benefit indicated from the students has to do with the system's ability to provide "on the spot" feedback and amplify their mental abilities and processes that are very useful in programming education.

6.1 Limitations and future work

The findings of this paper support our initial proposition that adaptive assessment and content recommendation have inherent benefits for learning programming, which would enable more capacity to support introductory courses in CS education. Although our study presents some positive findings, it also indicates some shortcomings and limitations that are worth mentioning. First, learning efficiency and effectiveness, and recommendation accuracy may not be related. This is one of the evaluation challenges in RS that is still open. However, we have not measured learning gain in this study, which is an interesting assumption that we want to explore in future.

Second, the participants of our study, although they represent an appropriate sample for our study (e.g., CS students with basic procedural programming knowledge who are learning OOP), were of a limited number, and the potential bias of the selection method (only the ones who engaged the most) might have produced slightly different results than the needs of the typical CS student. Nevertheless, the results provide a valid proof of concept and are reported with considerable caution, aiming to provide a springboard in CS education research that will allow us to proceed to further studies.

Third, recommendation accuracy is not sufficient to infer satisfaction [80]. So even if the recommendations generated by the recommender system are accurate, students may not be satisfied with the recommended learning paths, even though we observed one group of students who had been following all of the recommendations throughout the semester. Users' satisfaction includes many psychological aspects and further studies with students

are needed to empower productive conversations and embrace a human-centered perspective toward improving the design of adaptive assessment "at-scale" [53].

The last limitation we would like to report is connected with the evaluation of the effectiveness of the proposed adaptive assessment. We compared the effectiveness with students' grades; however, students' final grades might be heavily affected by the content taught and focused on the teacher and the content provided in the learning system. The learning system contains 15 topics that are aligned with the curriculum taught at the university, but those 15 topics are not an exhaustive list of the topics the instructors teach in the OOP courses. Therefore, it is likely that the teacher's focus influenced the content used for the assessment of the course, and is slightly different from the one the learning system uses.

6.2 Implications

In sum, the major implication for practice from our findings is that the modified Elo-rating method could effectively pair educational content complexity with students' proficiency, leading to recommending relevant coding exercises. This might create a more engaging learning setting for students, as they would not feel uncomfortably challenged or bored by the complexity of the exercises compared to their current knowledge. Moreover, the idea behind our method is not to replace the instructors in grading students, but to assist them in assessing programming assignments from large courses throughout the semester (e.g., to adapt and scale the assessment process), so that the assigned grades at the end of the course are uniform representation of the knowledge students' have gained, and more accurate considering their effort throughout the semester, rather than the exam at the end. Also, as reported in our findings, there are students who select and follow recommendations as their learning path, which may lead to sustaining higher levels of engagement and performance, the two critical components for successful learning. Finally, as programming is a subject that students mainly learn in digital learning environments, a prerequisite for an effective and engaging learning process is the existence of adaptive and flexible learning settings that can personalize and adapt learning content to individual learners.

REFERENCES

[1] Pedro Henriques Abreu, Daniel Castro Silva, and Anabela Gomes. 2018. Multiple-Choice Questions in Programming Courses: Can We Use Them and Are Students Motivated by Them? *ACM Transactions on Computing Education (TOCE)* 19, 1 (2018), 1–16.

[2] Margit Antal. 2013. On the use of elo rating for adaptive assessment. *Studia Universitatis Babes-Bolyai, Informatica* 58, 1 (2013), 29–41.

[3] Lora Aroyo and Darina Dicheva. 2004. The new challenges for e-learning: The educational semantic web. *Journal of Educational Technology & Society* 7, 4 (2004), 59–69.

[4] Jannicke M Baalsrud-Hauge, Ioana A Stanescu, Sylvester Arnab, Pablo Moreno Ger, Theodore Lim, Angel Serrano-Laguna, Petros Lameras, Maurice Hendrix, Kristian Kiili, Manuel Ninaus, et al. 2015. Learning through analytics architecture to scaffold learning experience through technology-based methods. *International Journal of Serious Games* 2, 1 (2015), 29–44.

[5] João Paulo Barros, Luís Estevens, Rui Dias, Rui Pais, and Elisabete Soeiro. 2003. Using lab exams to ensure programming practice in an introductory programming course. *ACM SIGCSE Bulletin* 35, 3 (2003), 16–20.

[6] Luciana Benotti, Mara Cecilia Martnez, and Fernando Schapachnik. 2017. A tool for introducing computer science with automatic formative assessment. *IEEE Transactions on Learning Technologies* 11, 2 (2017), 179–192.

[7] Outmane Bourkoukou, Essaid El Bachari, and Mohamed El Adnani. 2017. A recommender model in e-learning environment. *Arabian Journal for Science and Engineering* 42, 2 (2017), 607–617.

[8] Grant Braught, John MacCormick, and Tim Wahls. 2010. The benefits of pairing by ability. In *Proceedings of the 41st ACM technical symposium on Computer science education*. 249–253.

[9] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.

[10] Matthieu JS Brinkhuis and G Maris. 2009. Dynamic parameter estimation in student monitoring systems. *Measurement and Research Department Reports (Rep. No. 2009-1). Arnhem: Cito* (2009).

[11] Erveson Bruno, Bruno Alexandre, Rafael Ferreira Mello, Taciana Pontual Falcão, Boban Vesin, and Dragan Gašević. 2021. Applications of learning analytics in high schools: a Systematic Literature review. *Frontiers in Artificial Intelligence* (2021), 132.

[12] Peter Brusilovsky, Stephen Edwards, Amruth Kumar, Lauri Malmi, Luciana Benotti, Duane Buck, Petri Ihantola, Rikki Prince, Teemu Sirkiä, Sergey Sosnovsky, et al. 2014. Increasing adoption of smart learning content for computer science education. In *Proceedings of*

the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference. 31–57.
[13] Peter Brusilovsky, Sibel Somyürek, Julio Guerra, Roya Hosseini, Vladimir Zadorozhny, and Paula J Durlach. 2016. Open social student modeling for personalized learning. *IEEE Transactions on Emerging Topics in Computing* 4, 3 (2016), 450–461.
[14] Cory J Butz, Shan Hua, and R Brien Maguire. 2006. A web-based bayesian intelligent tutoring system for computer programming. *Web Intelligence and Agent Systems: An International Journal* 4, 1 (2006), 77–97.
[15] Dimitra I Chatzopoulou and Anastasios A Economides. 2010. Adaptive assessment of student's knowledge in programming courses. *Journal of Computer Assisted Learning* 26, 4 (2010), 258–269.
[16] Wei Chen, Zhendong Niu, Xiangyu Zhao, and Yi Li. 2014. A hybrid recommendation algorithm adapted in e-learning environments. *World Wide Web* 17, 2 (2014), 271–284.
[17] M Choy, S Lam, CK Poon, FL Wang, YT Yu, and L Yuen. 2007. Towards blended learning of computer programming supported by an automated system. *Blended Learning* 9 (2007).
[18] Sanja Maravić Čisar, Petar Čisar, and Robert Pinter. 2016. Evaluation of knowledge in Object Oriented Programming course with computer adaptive tests. *Computers & education* 92 (2016), 142–160.
[19] Benjamin Clegg, Siobhán North, Phil McMinn, and Gordon Fraser. 2019. Simulating student mistakes to evaluate the fairness of automated grading. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training*. IEEE Press, 121–125.
[20] Ricardo Conejo, Beatriz Barros, and Manuel F Bertoa. 2018. Automated assessment of complex programming tasks using SIETTE. *IEEE Transactions on Learning Technologies* (2018).
[21] Albert T Corbett and John R Anderson. 1994. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction* 4, 4 (1994), 253–278.
[22] Kent J Crippen and Boyd L Earl. 2007. The impact of web-based worked examples and self-explanation on performance, problem solving, and self-efficacy. *Computers & Education* 49, 3 (2007), 809–821.
[23] Mihaly Csikszentmihalyi, Sami Abuhamdeh, and Jeanne Nakamura. 2014. Flow. In *Flow and the foundations of positive psychology*. Springer, 227–238.
[24] Ole Halvor Dahl and Olav Fykse. 2018. *Combining Elo Rating and Collaborative Filtering to improve Learner Ability Estimation in an e-learning Context*. Master's thesis. NTNU.
[25] Rafael Jaime De Ayala. 2013. *The theory and practice of item response theory*. Guilford Publications.
[26] Benjamin Deonovic, Michael Yudelson, Maria Bolsinova, Meirav Attali, and Gunter Maris. 2018. Learning meets assessment. *Behaviormetrika* 45, 2 (2018), 457–474.
[27] Michael Derntl and Karin Anna Hummel. 2005. Modeling context-aware e-learning scenarios. In *Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops. Third IEEE International Conference on*. IEEE, 337–342.
[28] K DiCerbo, VJ Shute, and Yoon Jeon Kim. 2017. The Future of assessment in technology rich environments: Psychometric considerations. *Learning, design, and technology: An international compendium of theory, research, practice, and policy* (2017), 1–21.
[29] Kenneth A Dodge, Roberta R Murphy, and Kathy Buchsbaum. 1984. The assessment of intention-cue detection skills in children: Implications for developmental psychopathology. *Child development* (1984), 163–173.
[30] Hendrik Drachsler, Hans GK Hummel, and Rob Koper. 2008. Personal recommender systems for learners in lifelong learning networks: the requirements, techniques and model. *International Journal of Learning Technology* 3, 4 (2008), 404–423.
[31] Pragya Dwivedi and Kamal K Bharadwaj. 2015. e-Learning recommender system for a group of learners based on the unified learner profile approach. *Expert Systems* 32, 2 (2015), 264–276.
[32] Stephen H Edwards and Manuel A Perez-Quinones. 2008. Web-CAT: automatically grading programming assignments. In *Proceedings of the 13th annual conference on Innovation and technology in computer science education*. 328–328.
[33] Tomáš Effenberger, Jaroslav Čechák, and Radek Pelánek. 2019. Measuring Difficulty of Introductory Programming Tasks. In *Proceedings of the Sixth (2019) ACM Conference on Learning@ Scale*. ACM, 28.
[34] Margaret Ellis, Clifford A Shaffer, and Stephen H Edwards. 2019. Approaches for coordinating eTextbooks, online programming practice, automated grading, and more into one course. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 126–132.
[35] Arpad E Elo. 1978. *The rating of chessplayers, past and present*. Arco Pub.
[36] Javier Escobar-Avila, Deborah Venuti, Massimiliano Di Penta, and Sonia Haiduc. 2019. A survey on online learning preferences for computer science and programming. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training*. IEEE Press, 170–181.
[37] Nickolas Falkner, Rebecca Vivian, David Piper, and Katrina Falkner. 2014. Increasing the effectiveness of automated assessment by increasing marking granularity and feedback units. In *Proceedings of the 45th ACM technical symposium on Computer science education*. 9–14.
[38] Michal Forišek. 2009. Using item response theory to rate (not only) programmers. *Olympiads in Informatics* 3 (2009), 3–16.

[39] Dragan Gašević, Shane Dawson, Tim Rogers, and Danijela Gasevic. 2016. Learning analytics should not promote one size fits all: The effects of instructional conditions in predicting academic success. *The Internet and Higher Education* 28 (2016), 68–84.
[40] Aldo Gordillo. 2019. Effect of an Instructor-Centered Tool for Automatic Assessment of Programming Assignments on Students' Perceptions and Performance. *Sustainability* 11, 20 (2019), 5568.
[41] Asela Gunawardana and Guy Shani. 2009. A survey of accuracy evaluation metrics of recommendation tasks. *Journal of Machine Learning Research* 10, Dec (2009), 2935–2962.
[42] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. 2004. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)* 22, 1 (2004), 5–53.
[43] Colin A Higgins, Geoffrey Gray, Pavlos Symeonidis, and Athanasios Tsintsifas. 2005. Automated assessment and experiences of teaching programming. *Journal on Educational Resources in Computing (JERIC)* 5, 3 (2005), 5–es.
[44] Danial Hooshyar, Rodina Binti Ahmad, Moslem Yousefi, Moein Fathi, Shi-Jinn Horng, and Heuiseok Lim. 2016. Applying an online game-based formative assessment in a flowchart-based intelligent tutoring system for improving problem-solving skills. *Computers & Education* 94 (2016), 18–36.
[45] Roya Hosseini, Kamil Akhuseyinoglu, Andrew Petersen, Christian D Schunn, and Peter Brusilovsky. 2018. PCEX: Interactive Program Construction Examples for Learning Programming. In *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*. ACM, 5.
[46] Roya Hosseini, I-Han Hsiao, Julio Guerra, and Peter Brusilovsky. 2015. What should i do next? adaptive sequencing in the context of open social student modeling. In *Design for teaching and learning in a networked world*. Springer, 155–168.
[47] Henry Hsu and Peter A Lachenbruch. 2007. Paired t test. *Wiley encyclopedia of clinical trials* (2007), 1–3.
[48] Christopher David Hundhausen, Daniel M Olivares, and Adam S Carter. 2017. IDE-based learning analytics for computing education: a process model, critical review, and research agenda. *ACM Transactions on Computing Education (TOCE)* 17, 3 (2017), 1–26.
[49] Gwo-Haur Hwang, Beyin Chen, Ru-Shan Chen, Ting-Ting Wu, and Yu-Ling Lai. 2019. Differences between studentsâĂŹ learning behaviors and performances of adopting a competitive game-based item bank practice approach for learning procedural and declarative knowledge. *Interactive Learning Environments* 27, 5-6 (2019), 740–753.
[50] David H Jonassen. 1991. Hypertext as instructional design. *Educational technology research and development* 39, 1 (1991), 83–92.
[51] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education (TOCE)* 19, 1 (2018), 1–43.
[52] Sharon Klinkenberg, Marthe Straatemeier, and Han LJ van der Maas. 2011. Computer adaptive practice of maths ability using a new item response model for on the fly ability and difficulty estimation. *Computers & Education* 57, 2 (2011), 1813–1824.
[53] Chinmay Kulkarni. 2019. Design Perspectives of Learning at Scale: Scaling Efficiency and Empowerment. In *Proceedings of the Sixth (2019) ACM Conference on Learning@ Scale*. 1–11.
[54] Fotis Lazarinis, Steve Green, and Elaine Pearson. 2010. Creating personalized assessments based on learner knowledge and objectives in a hypermedia Web testing application. *Computers & Education* 55, 4 (2010), 1732–1743.
[55] Youngjin Lee. 2019. Estimating student ability and problem difficulty using item response theory (IRT) and TrueSkill. *Information Discovery and Delivery* 47, 2 (2019), 67–75.
[56] Michael Leisner and Philipp Brune. 2019. Good-bye localhost: a cloud-based web IDE for teaching Java EE web development to non-computer science majors. In *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings*. IEEE Press, 268–269.
[57] Xiao Liu, Shuai Wang, Pei Wang, and Dinghao Wu. 2019. Automatic grading of programming assignments: an approach based on formal semantics. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training*. IEEE Press, 126–137.
[58] Jie Lu, Dianshuang Wu, Mingsong Mao, Wei Wang, and Guangquan Zhang. 2015. Recommender system application developments: a survey. *Decision Support Systems* 74 (2015), 12–32.
[59] Moira Maguire and Brid Delahunt. 2017. Doing a thematic analysis: A practical, step-by-step guide for learning and teaching scholars. *All Ireland Journal of Higher Education* 9, 3 (2017).
[60] Phil Maguire, Rebecca Maguire, and Robert Kelly. 2017. Using automatic machine assessment to teach computer programming. *Computer Science Education* 27, 3-4 (2017), 197–214.
[61] Katerina Mangaroska, Roberto Martinez-Maldonado, Boban Vesin, and Dragan Gašević. 2021. Challenges and opportunities of multimodal data in human learning: The computer science students' perspective. *Journal of Computer Assisted Learning* (2021).
[62] Katerina Mangaroska, Boban Vesin, and Michail Giannakos. 2019. Cross-platform analytics: A step towards personalization and adaptation in education. In *Proceedings of the 9th International Conference on Learning Analytics & Knowledge*. ACM, 71–75.
[63] Katerina Mangaroska, Boban Vesin, and Michail Giannakos. 2019. Elo-rating method: Towards adaptive assessment in e-learning. In *2019 IEEE 19th International Conference on Advanced Learning Technologies (ICALT)*, Vol. 2161. IEEE, 380–382.
[64] Katerina Mangaroska, Boban Vesin, Vassilis Kostakos, Peter Brusilovsky, and Michail N Giannakos. 2021. Architecting Analytics Across Multiple E-Learning Systems to Enhance Learning Design. *IEEE Transactions on Learning Technologies* 14, 2 (2021), 173–188.

[65] Nikos Manouselis, Hendrik Drachsler, Riina Vuorikari, Hans Hummel, and Rob Koper. 2011. Recommender systems in technology enhanced learning. In *Recommender systems handbook*. Springer, 387–415.
[66] Robert McCartney, Jonas Boustedt, Anna Eckerdal, Kate Sanders, Lynda Thomas, and Carol Zander. 2016. Why computing students learn on their own: Motivation for self-directed learning of computing. *ACM Transactions on Computing Education (TOCE)* 16, 1 (2016), 1–18.
[67] Pablo Molins-Ruano, Carlos González-Sacristán, F Díez, P Rodriguez, and GM Sacha. 2014. Adaptive Model for Computer-Assisted Assessment in Programming Skills. *arXiv preprint arXiv:1403.1465* (2014).
[68] Eerik Muuli, Kaspar Papli, Eno Tõnisson, Marina Lepp, Tauno Palts, Reelika Suviste, Merilin Säde, and Piret Luik. 2017. Automatic assessment of programming assignments using image recognition. In *European Conference on Technology Enhanced Learning*. Springer, 153–163.
[69] Pavol Navrat and Jozef Tvarozek. 2014. Online programming exercises for summative assessment in university courses. In *Proceedings of the 15th International Conference on Computer Systems and Technologies*. ACM, 341–348.
[70] Mahfudzah Othman, Siti Hana Quzaima Alias, and Nur Fajrina Mohd Rashidi. 2016. Enhanced Collaborative E-learning for Programming Using Open Learner Model. *Computing Research & Innovation (CRINN), Vol. 1, November 2016* (2016), 64.
[71] Steven Oxman, William Wong, and D Innovations. 2014. White paper: Adaptive learning systems. *Integrated Education Solutions* (2014), 6–7.
[72] Maciej Pankiewicz. 2020. Measuring task difficulty for online learning environments where multiple attempts are allowed-the Elo rating algorithm approach.. In *EDM*.
[73] Philip I Pavlik Jr, Hao Cen, and Kenneth R Koedinger. 2009. Performance Factors Analysis–A New Alternative to Knowledge Tracing. *Online Submission* (2009).
[74] Michael J Pazzani and Daniel Billsus. 2007. Content-based recommendation systems. In *The adaptive web*. Springer, 325–341.
[75] Radek Pelánek. 2014. Application of time decay functions and the elo system in student modeling. In *Educational Data Mining 2014*. Citeseer.
[76] Radek Pelánek. 2016. Applications of the Elo rating system in adaptive educational systems. *Computers & Education* 98 (2016), 169–179.
[77] Radek Pelánek, Jan Papoušek, Jiří Řihák, Vít Stanislav, and Juraj Nižnan. 2017. Elo-based learner modeling for the adaptive practice of facts. *User Modeling and User-Adapted Interaction* 27, 1 (2017), 89–118.
[78] Andrew Petersen. 2018. Programming Course Resource System (PCRS). https://mcs.utm.utoronto.ca/~pcrs/pcrs/
[79] Wolter Pieters, Sanne HG van der Ven, and Christian W Probst. 2012. A move in the security measurement stalemate: Elo-style ratings to quantify vulnerability. In *Proceedings of the 2012 New Security Paradigms Workshop*. ACM, 1–14.
[80] Pearl Pu, Li Chen, and Rong Hu. 2012. Evaluating recommender systems from the user's perspective: survey of the state of the art. *User Modeling and User-Adapted Interaction* 22, 4-5 (2012), 317–355.
[81] Yigal Rosen, Ilia Rushkin, Andrew Ang, Colin Federicks, Dustin Tingley, and Mary Jean Blink. 2017. Designing adaptive assessments in MOOCs. In *Proceedings of the Fourth (2017) ACM Conference on Learning@ Scale*. 233–236.
[82] Riku Saikkonen, Lauri Malmi, and Ari Korhonen. 2001. Fully automatic assessment of programming exercises. In *ACM Sigcse Bulletin*, Vol. 33. ACM, 133–136.
[83] Avi Segal, Kobi Gal, Guy Shani, and Bracha Shapira. 2019. A difficulty ranking approach to personalization in E-learning. *International Journal of Human-Computer Studies* 130 (2019), 261–272.
[84] Ruiqi Shen, Donghee Yvette Wohn, and Michael J Lee. 2019. Comparison of Learning Programming Between Interactive Computer Tutors and Human Teachers. In *Proceedings of the ACM Conference on Global Computing Education*. 2–8.
[85] Nischal Shrestha. 2018. Towards Supporting Knowledge Transfer of Programming Languages. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 275–276.
[86] Jiangbo Shu, Xiaoxuan Shen, Hai Liu, Baolin Yi, and Zhaoli Zhang. 2018. A content-based recommendation algorithm for learning resources. *Multimedia Systems* 24, 2 (2018), 163–173.
[87] Andrew Simon, Luxton-Reilly, Ibrahim Albluwi, Brett A Becker, Michail Giannakos, Amruth N Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory programming: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*. 55–106.
[88] Thomas Staubitz, Hauke Klement, Jan Renz, Ralf Teusner, and Christoph Meinel. 2015. Towards practical programming exercises and automated assessment in Massive Open Online Courses. In *Teaching, Assessment, and Learning for Engineering (TALE), 2015 IEEE International Conference on*. IEEE, 23–30.
[89] Tiffany Ya Tang and Gordon McCalla. 2005. Smart recommendation for an evolving e-learning system: Architecture and experiment. *International Journal on elearning* 4, 1 (2005), 105.
[90] Robert K Tsutakawa and Jane C Johnson. 1990. The effect of uncertainty of item parameter estimation on ability estimates. *Psychometrika* 55, 2 (1990), 371–390.
[91] Kurt Vanlehn. 2006. The behavior of tutoring systems. *International journal of artificial intelligence in education* 16, 3 (2006), 227–265.

[92] Yehiry Lucelly Pulido Vega, Juan Carlos Guevara Bolaños, Gloria Milena Fernández Nieto, and Silvia Margatira Baldiris. 2012. Application of item response theory (IRT) for the generation of adaptive assessments in an introductory course on object-oriented programming. In *Frontiers in Education Conference (FIE), 2012*. IEEE, 1–4.

[93] Katrien Verbert, Nikos Manouselis, Xavier Ochoa, Martin Wolpers, Hendrik Drachsler, Ivana Bosnic, and Erik Duval. 2012. Context-aware recommender systems for learning: a survey and future challenges. *IEEE Transactions on Learning Technologies* 5, 4 (2012), 318–335.

[94] Boban Vesin, Katerina Mangaroska, and Michail Giannakos. 2018. Learning in smart environments: user-centered design and analytics of an adaptive learning system. *Smart Learning Environments* 5, 1 (2018), 1–21.

[95] Tzu-Hua Wang. 2014. Developing an assessment-centered e-Learning system for improving student learning effectiveness. *Computers & Education* 73 (2014), 189–203.

[96] Kelly Wauters, Piet Desmet, and Wim Van Den Noortgate. 2012. Item difficulty estimation: An auspicious collaboration between data and judgment. *Computers & Education* 58, 4 (2012), 1183–1193.

[97] Kelly Wauters, Piet Desmet, and Wim Van Noortgate. 2010. Monitoring learners' proficiency: weight adaptation in the elo rating system. In *Educational Data Mining 2011*.

[98] Dianshuang Wu, Jie Lu, and Guangquan Zhang. 2015. A fuzzy tree matching-based personalized e-learning recommender system. *IEEE Transactions on Fuzzy Systems* 23, 6 (2015), 2412–2426.

[99] Xiao Yu, Xiang Ren, Yizhou Sun, Bradley Sturt, Urvashi Khandelwal, Quanquan Gu, Brandon Norick, and Jiawei Han. 2013. Recommendation in heterogeneous information networks with implicit user feedback. In *Proceedings of the 7th ACM conference on Recommender systems*. ACM, 347–350.

[100] Daniel Zingaro, Yuliya Cherenkova, Olessia Karpova, and Andrew Petersen. 2013. Facilitating code-writing in PI classes. In *Proceeding of the 44th ACM technical symposium on Computer science education*. 585–590.