An Adaptive Testcase Recommendation System to Engage Students in Learning: A Practice Study in Fundamental Programming Courses

Tien Vu-Van[1], Huy Tran[2], Thanh-Van Le*[3], Hoang-Anh Pham[4], Nguyen Huynh-Tuong[5]
Ho Chi Minh City University of Technology (HCMUT), 268 Ly Thuong Kiet, District 10, Ho Chi Minh City, Vietnam[1,2,3,4]
Vietnam National University Ho Chi Minh City (VNU-HCM), Linh Trung Ward, Ho Chi Minh City, Vietnam[1,2,3,4]
Industrial University of Ho Chi Minh City (IUH), 12 Nguyen Van Bao, Go Vap District, Ho Chi Minh City, Vietnam[5]
*Corresponding Author

***Abstract*—This paper proposes a testcase recommendation system (TRS) to assist beginner-level learners in introductory programming courses with completing assignments on a learning management system (LMS). These learners often struggle to generate complex testcases and handle numerous code errors, leading to disengaging their attention from the study. The proposed TRS addresses this problem by applying the recommendation system using singular value decomposition (SVD) and the zone of proximal development (ZPD) to provide a small and appropriate set of testcases based on the learner's ability. We implement this TRS to the university-level Fundamental Programming courses for evaluation. The data analysis has demonstrated that TRS significantly increases student interactions with the system.**

***Keywords*—Testcases recommendation system (TRS); learning management system (LMS); zone of proximal development (ZPD); singular value decomposition (SVD)**

I. Introduction

A learning management system (LMS) is an educational tool, either in the form of an application or a website, that facilitates interactive online learning, automates administrative tasks, organizes educational content, and records learners' activities [1], [2]. The author in [3] emphasizes that an LMS should be dynamic; that is, it should be active, flexible, customizable, and adaptable. One method to achieve this is to improve the ability to instruct many learners in a personalized manner. As a result, learners would increase their interaction and satisfaction with the system. LMS can serve as a data collection tool for automatic assessment of learning outcomes [4], analysis of learning style [5], and evaluation of learner satisfaction [6].

Moodle[1] is an open-source LMS with a large number of users, with almost 157,289+ registered sites in 241+ countries [7]. Moodle offers various types of assessment questions for learners, such as multiple choice, short answer, matching, etc. On the other hand, programming questions are essential for learners in programming courses. Programming questions require to source code submitted from learners on a set of inputs. Moodle has now supported programming questions through the plugin CodeRunner. Thanks to CodeRunner, Moodle can deliver programming exercises to learners and automatically grade learners' code.

[1]http://moodle.org

Programming exercises usually include two types: lab and assignment. Lab exercises are small code exercises for learners to practice independently after learning a topic in theory. The assignment is a complex exercise with lengthy descriptions and many requirements. Additionally, it is used to test the synthesis of problems. Through the assignment, learners practice breaking down the problem into smaller parts to solve. At the same time, they learn to use a combination of techniques from different programming topics to solve complex problems. A typical evaluation of programming exercises is to run the code on a set of inputs and check if the output matches the expected output that results from running the lecturer's solution code on the same input set. A set of inputs and the corresponding output is called a testcase. The percentage of correct testcases calculates the score for the programming exercise. The number of testcases of an assignment is usually much more than those of a lab exercise. The reason is that assignments often ask for many problems, so many testcases are needed to check possible cases of these problems. Therefore, grading assignments for learners often takes a lot of time. Assignment time is usually given for a relatively long period of about three to six weeks. In contrast, the lab exercises are often done in one week.

An assignment implemented on Moodle can be evaluated in different ways.

- Lecturers manually grade all submissions by looking through the code the learner submits and marking it based on the lecturer's perspective.
- Lecturers leverage an automatic grading system (AGS) to grade the submitted code. The grading tool automatically runs the learner's code through testcases and then checks the code's output match. This grading method will minimize the lecturer's perspective, making it fairer than manual grading.
- Learners are provided with a place to test their code on a set of sample testcases. The problem when grading with the AGS on the lecturer's local computer is that some learners' code does not run the same results as when running the code on the learner's local computer. The causes may be because learners' code depends on the compiler and operating system. For example, a common mistake is that when declaring an integer variable and not initializing a value, some compilers automatically assign the value 0 to the

variable. However, others will not do initialization, and the variable will have a random value. An independent grading environment helps learners minimize differences between the grading environment and the one they run their code. After the learner submits the assignment, the lecturer often uses another set of testcases for grading.

This paper does not study the first method because it has the teacher's subjective opinion when grading. Meanwhile, two remaining methods still have some drawbacks. Learners need help to think of a set of testcases to assess their code before submission. Coming up with testcases is a must-have skill for programmers. In real projects, programmers are also required to write their testcases to examine their solutions. However, it is difficult for beginner learners to think of testcases. Learners would not improve their programming skills when they couldn't think of testcases.

A straightforward method to help beginner learners is to provide all the testcases that learners do incorrectly. However, when there are many errors in testcases, beginner learners must choose which testcase to correct first. Therefore, we propose a recommendation-based method that suggests a subset of a few incorrect testcases that have difficulty levels suitable for the current performance of learners. The main contributions of this paper can be summarized as follows:

- Propose a testcase recommendation system (TRS) that engages the learners in doing assignments by suggesting a small set of testcases adaptive to learners' performance.
- Implement the proposed TRS to fundamental programming courses at our university to investigate the effectiveness in terms of students' ability to learn in a personalized manner and the enhancement of learner interaction.

The rest of this paper is organized as follows. Section II summarizes related works to clarify the scope of our study. Then, Section III describes our proposed method. The implementation and evaluation are presented in Section IV. Finally, Section V provides concluding remarks and future works.

II. Related Work

A recommendation system (RS) will help suggest items to users when there are too many items to select. Recommendation systems (RSs) have become popular and are used extensively in e-commerce and other digital companies [8]. Some well-known examples include Netflix's movie RS [9], Amazon's product RS [10], Google's personalized news, Google's advertisement search, and YouTube for videos [11]. RSs are mainly used for two main tasks: predicting how many ratings a user would give for an item (so-called prediction generation) and recommending a set of items to a user (so-called recommendation generation). RSs collect information on users' past behavior on a set of items and use them for the recommendation. Basically, RSs approaches can be classified into three types: Content-based filtering, Collaborative filtering, and Hybrid one.

- **Content-based filtering (CBF)** provides recommendations based on features of users and items, which are usually created according to users' consuming items. The recommended items are those whose characteristics are similar to the consuming items of the target user.
- **Collaborative filtering (CF)** works on the fact that users with similar behavior will have similar tastes or similar buying habits [8]. CF is divided into memory and model-based approach, based on how the data of the rating matrix are processed [12][13]. The memory-based approach in recommendation systems utilizes similarity measures between users or items to identify their relevant neighbors [8]. These neighbors are then used for recommending or predicting items or users. This approach is easy to implement and interpret the results. However, it requires the entire task rating matrix, making it less suitable for high-dimensional and sparse data. Meanwhile, the model-based approach learns and fits a parameterized model to the user-item rating matrix. This model is then used for providing recommendation tasks. Matrix factorization (MF) is a technique in the model-based approach that gained popularity, especially after the Netflix Prize Contest [13]. MF models are known for their relatively high accuracy, scalability, and dimensionality reduction properties [8].
- **Hybrid approach** combines CBF and CF to provide high predictive accuracy than both [13].

The model-based approach learns the model's parameters with the user-item matrix and uses it to make suggestions. User-item matrix contains user ratings for the items. This rating is similar to the scores obtained by learners for testcases. Therefore, we utilize this method to develop a testcase recommendation system (TRS).

In the domain of testcase recommendation system, previous studies have specifically explored its application in the context of software testing. The authors in [14], [15] examine the test scripts used by automation team and recommends testcases based on source code structural similarity for developing newer testcases. The author in [16] builds a recommender system to find an optimal group of tests to be executed with a code change. The authors in [17] implement an item-based collaborative filtering recommender system that develops a test case prioritizing technique using user interaction data and application modification history information. These studies primarily focus on recommending testcases that are similar to the ones already available for software testing. In contrast, our study concentrates on suggesting testcases that align with the learner's performance. These testcases are utilized by the learner for programming practice.

In this study, we adopt the recommendation algorithm to the learners' score data due to data availability. However, the score has not shown preference as the RS model requires preference as a user's item rating. For example, if a learner gets a high score on a test, they may feel bored because the test may be too easy, leading to disengagement in continuous studying. Therefore, we improve the recommendation algorithm on

the learner's score data by consulting the zone of proximal development (ZPD) theory.

According to the ZPD, if the learning materials are too easy or difficult, the learner will become bored or frustrated. The optimal level of instructional material should be within the "zone" that falls between the learner's upper and lower limits of ability [18]. ZPD is applied in various educational contexts, such as adaptive quiz question recommendations [19] and navigating optimized learning paths [20]. It reduces cognitive load and improves learning outcomes without affecting the learning experience [21]. ZPD is also utilized in specific fields like clinical education [22], participatory scenario planning [23], and divergent thinking [24]. Previous studies [25], [26] have aimed to provide a clearer definition of the ZPD compared to Vygotsky's initial conceptualization [27]. The SZPD criterion is proposed, where $H^* - H > DH$ represents the confused zone, and $H^* - H < -DH$ represents the bored zone [25]. Here, H and DH respectively refer to the goal number of hints and the allowable variation in H to determine the situation within the ZPD.

In summary, we combine recommendation algorithm and ZPD theory to make recommendations on learners' score data.

III. The Proposed TRS

Currently, our approach involves providing a platform to enhance the assignment implementation process. This process includes publishing the assignment specifications, opening a forum for discussion, setting the deadline, providing a designated place for testing with sample testcases (which also serves as the submission place for students' work), and finally opening the TRS.

The proposed approach strengthens the internal environment to suggest testcases tailored to each learner's performance. Fig. 1 outlines our assignment implementation process that begins with learners completing a set of sample testcases that are publicly available. The completion rate is determined by the instructor (e.g., 80%). Once learners surpass the completion rate, they are prompted to request more complicated testcases from the recommendation system. Learners practice and debug their code using the provided testcases. Once all testcases have been solved correctly within a limited time, learners can request a new set of testcases. However, there is a limitation on the number of requests per day. This measure reduces the system load and is a basis for applying the ZPD theory.

When a learner submits their code, the system evaluates the correctness of each testcase's result. There is a similarity between the TRS and a typical recommendation system, where learners are considered users, testcases are items, and learner scores correspond to the ratings that users provide for items. Thus, we apply the singular value decomposition (SVD) technique from the typical recommendation system to TRS. Furthermore, the matrix representing user scores for each testcase is what we call ***learner-testcase matrix***, similar to the user-item matrix in a typical recommendation system.

However, in TRS, a higher score from a learner does not necessarily indicate a higher preference for that testcase. A high score could result from an easy testcase, making the learner bored. One approach is to directly ask the learner about their preference for the suggested testcase using explicit profiling. However, this method may annoy and distract learners from the primary assignment goal. In contrast, the implicit profiling method captures user interactions within the system, improving system effectiveness and avoiding the drawbacks of explicit profiling. Combining interactive data from the system with the SZPD allows for suggesting testcases to suit learners' abilities.

Our proposed approach, which utilizes singular value decomposition (SVD) technique and zone of proximal development (ZPD) theory named **SVD-ZPD**, consists of four main steps as follows:

1) **Fitting SVD to predict learner scores for testcases.** The SVD technique is applied to the learner-testcase matrix to predict the scores for any learner. When a learner submits their code, incorrect testcases typically receive a score of 0. With SVD, incorrect testcases will have a non-zero score, indicating the extent of the error. By sorting the incorrect testcases based on these scores, we can identify the testcases that the learner is more likely to answer incorrectly.
2) **Determining the learner's current performance.** We determine the learner's performance state within the ZPD based on the number of times the learner requests a new set of testcases from the TRS. Let R be the goal number of new testcase requests and DR be the allowed variation in R to consider the learner within the ZPD. Let R^* be the actual number of code submissions by the learner on the previous day. We also introduce two constraints: (a) a maximum of requests per day, and (b) the learner must correctly complete all testcases from the previous request to be eligible for a new set of testcases. So, $R^* - R$ represents the learner's current performance. In this scenario, we have:
    - If $R^* - R < -DR$, the learner has not answered many testcases correctly, indicating that the current testcases may be too difficult, and the learner is in the confused zone.
    - If $R^* - R > DR$, the learner quickly answers the testcases and continuously requests new ones, indicating that the current testcases may be too easy and the learner is in the bored zone.
    - In other cases, the learner is in the ZPD zone.
3) **Determining the appropriate difficulty level based on the learner's current performance.** The previous step provides a general guideline for adjusting the difficulty level: decrease the difficulty level if the learner is in the confused zone and increase it if the learner is in the bored zone. This step defines more detailed rules for increasing and decreasing the difficulty level. While there can be multiple approaches, in this initial study, we propose the following simple rules (but we don't limit the approach to these rules):
    - Difficulty level includes three levels: easy, medium, and hard. It is initially set to easy.
    - If the learner is in the bored zone, increase the difficulty level by one adjacent level. If

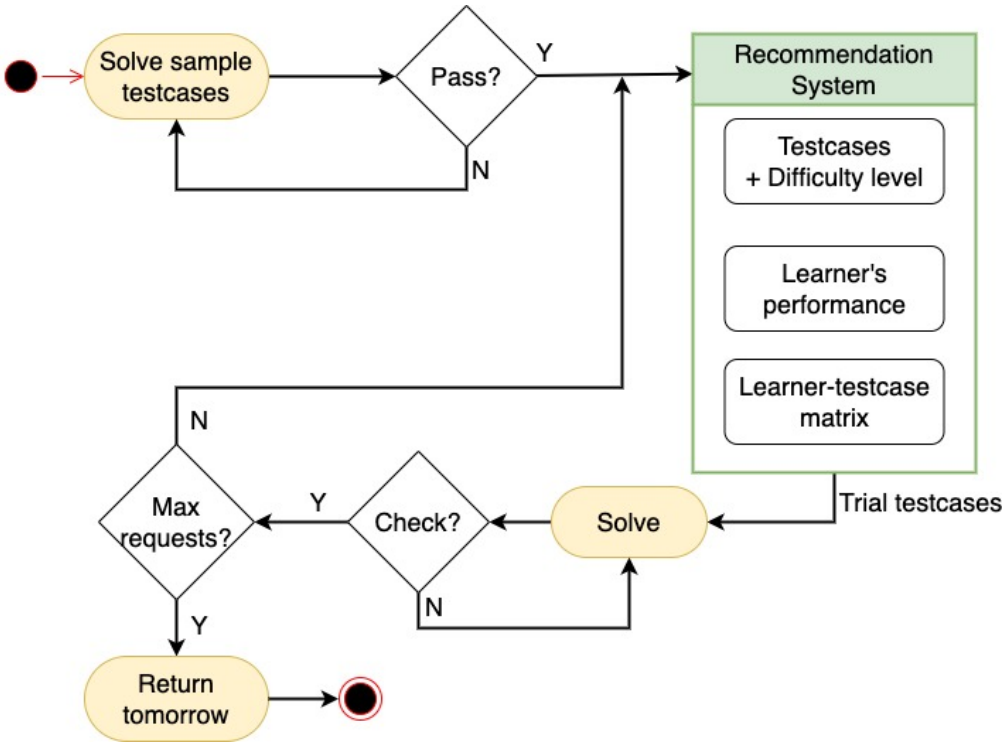


Fig. 1. Illustration of the proposed TRS process.

it is already at the highest level, keep it unchanged.
- If the learner is in the confused zone, decrease the difficulty level by one adjacent level. If it is already at the lowest level, keep it unchanged.
- If the learner is within the ZPD, keep the difficulty level unchanged.

4) **Selecting testcases with the same difficulty level from the previous step in descending order of predicted scores obtained from the first step.**

IV. Evaluation

The proposed TRS is implemented in one assignment of the Fundamental Programming course at our university for the second semester of the 2022–2023 academic year, where Moodle is used as an online learning tool. The effectiveness of TRS is examined by comparing the assignment recorded on Moodle of the three most recent semesters of the same course: the second semester of the 2020-2021 year (SEM-202), the second semester of the 2021-2022 year (SEM-212), and the second semester of the 2022-2023 year (SEM-222).

A. Features of Implementing Assignment

For each semester, we will analyze the information that the system provides to the learners, the main features, and the information that can be obtained for the assignment implementation process.

In SEM-202, learners will receive a set of sample testcases and a place to allow automatic submission. Learners run these sample testcases on their own on the local computer. Learners compare the program's running results with the provided expected results to assess whether the program runs correctly. This set of testcases usually includes only simple testcases and has a small number of testcases. Learners will submit their work on the system before the deadline. Then, the teacher will grade the score on the personal computer and post the score on the system for learners. The drawback of this implementation is that learners can not improve their programming skills because learners have to wait until the deadline expires to receive their marks. While learners are doing the test, they can't see the scores and errors to correct them.

In SEM-212, the system still provides the same materials as SEM-202. Moreover, the system offers an additional place for automatic grading and instant results on sample testcases. This place helps learners receive immediate feedback and ensures the submission runs appropriately in the grading environment. In addition, the interactive information of learners on the system will be more, such as the time of submission, submission, and grading results. The problem with this implementation is that the system only provides one static set of simple testcases. Beginner learners need help to think of complex testcases to improve their programming skills.

In SEM-222, the system still provides the same materials as SEM-212. However, through the proposed TRS, the system offers more testcases suitable for each learner at each time. After correcting the sample testcases, learners can request more testcases from TRS. Then, TRS will provide a small set of testcases that the learner is doing wrong, and these testcases should have a difficulty level matching the learner's current ability. In addition, TRS requires learners to do all testcases correctly to be given a new set of testcases, and they can only

be asked for a maximum of five times a day. The total number of testcases used for suggestion is 124; each request is five testcases.

Table I summarizes the main features of the assignment implementation process over the three semesters, including information for learners, key features, and recorded information. Notably, different from SEM-202 and SEM-212, the last semester is demonstrated by 2 rows, one for sample testcases submission and the other for TRS. As a result, we can see from this table that the system, in addition to TRS, has more key features and can gather more useful information.

B. Comparison

Table II compares the assignment implementation in three semesters on four factors: supportive self-study environment to summarize knowledge, instructional environment, learning and development of learners, and the system's interaction with learners.

We analyze the following three questions regarding the self-study support environment to summarize knowledge. The first question is about the ability to serve a large number of learners (all three semesters have a place to release Assignments to learners, for learners to submit papers, and have an automatic grading method). The second question is how to implement a self-study support environment for many learners (SEM-202 provides a place to submit papers and automatically grade papers after deadlines, SEM-212 delivers a place to submit papers and return results instantly, SEM -222 provides the same environment as SEM-212 and has additional testcase suggestions). The last question is about the ability to evaluate work results for many learners (all three semesters can automatically grade learners' codes).

In terms of the learner guidance environment, this is an environment that provides feedback to help learners improve their work and programming skills. We analyze the instructional environment according to two questions: Is there an instructional environment for learners? What are the limitations of the effectiveness of the instructional environment? In SEM-202, a forum is provided for learners to ask questions about Assignments. The teacher then answers these questions. The efficacy of a forum depends on the questions posted by learners and the responses provided by instructors and other learners. To examine the effectiveness, appropriate data analysis tools related to the interactive content on the forum are needed. On the other hand, sample testcases are a set of simple testcases for learners to test the code on their own under personal computers. The limitation of sample testcases is that the instructor does not know how learners have utilized them. The completeness level and suitability of the sample testcases for guiding learners cannot be determined. Semester SEM-212 still provides forum and sample testcases. However, sample testcases in SEM-212 are automatically graded and give instant results to learners. The improvement in this method is that the instructor knows whether learners have studied and worked with testcases through their submission attempts. After completing the assignment and grading, the instructor can determine whether the sample testcases are comprehensive enough to guide the learners. Semester SEM-222 not only provides the same environment as SEM-212 but also provides TRS. The challenge is establishing a diverse testcases bank that can be divided into smaller sets to provide appropriate hints based on the learners' abilities at different stages.

Regarding the learning and development of learners, we analyze according to three questions as shown in Table II. Does the system keep track of the learning process: SEM-202 is not recorded because only the last submission is submitted. At the same time, SEM-212 and SEM-222 are recordable at the time of submission and submission code.

We analyze the interaction with learners according to a question about how the system interacts with learners, as shown in the table. The answer consists of two lines describing the information the system receives from the learner and the information the system gives to the learner. The information obtained from the learners was the same over the three semesters. However, the information brought to learners has increased gradually over three semesters. The final semester has the most information for learners. As more information reaches learners, learners can personally practice and improve their programming skills.

C. Statistical Results and Findings

The statistical results after implementing the assignment over three semesters (SEM-202, SEM-212 and SEM-222) are summarized in Table III. The important points of this table should be taken into account as follows.

- This table contains seven information fields including information provided to learners, total number of learners, number of learners who submitted code, number of days having submissions, number of submissions, average number of submissions per learner, and average number of submissions per day.
- The information provided to learners may be sample testcases or information provided from TRS. Note that the information provided by TRS is only available from SEM-222.
- The number of days with submissions in SEM-202 is marked as N/A (not available) since the system does not record individual submission instances. Each learner is only permitted to submit one work for the assignment, so the total number of submissions equals the number of learners. The average number of submissions per day cannot be calculated because the number of days is not recorded.

The following analyses compare SEM-212 with SEM-202 and SEM-222 with SEM-212 regarding sample test cases. Then, the most appropriate context among the three semesters will be identified. Finally, the information from TRS will be analyzed to better understand the system's value.

Considering two semesters, SEM-212 and SEM-202, although the number of learners in SEM-212 is smaller than in SEM-202, the number of submissions is higher. The reason is that SEM-212 can record the information of learners' multiple attempts. In other words, the process implemented in SEM-212 improves the interaction between learners and the system compared to SEM-202. This also serves as an example to

TABLE I. FEATURES OF THE ASSIGNMENT IMPLEMENTATION PROCESS IN THREE SEMESTERS: SEM-202, SEM-212, AND SEM-222

Semester	Provided information for learners	Key features	Recorded Information
SEM-202	Sample testcases	- Automated submission platform for learners. - Learners receive grading results after the deadline.	- Overall grading score.
SEM-212	Sample testcases	- Automated submission platform for learners. - Learners receive grading results after the deadline. - Automated grading and immediate feedback on sample testcases submissions.	- Overall grading score. - Submission history: submission time and grading results for sample testcases.
SEM-222	Sample testcases	- Automated submission platform for learners. - Learners receive grading results after the deadline. - Automated grading and immediate feedback on sample testcases submissions.	- Overall grading score. - Submission history: submission time and grading results for sample testcases.
	Various sets of testcases sent based on individual learners and timing.	- Submissions and requesting testcases for incorrect answers. - Testcase sets tailored to learners' abilities through the TRS system.	- Submission history: submission time and grading results for recommended testcase sets.

TABLE II. COMPARISON OF ASSIGNMENT IMPLEMENTATION IN 3 SEMESTERS

Factor	Related questions	SEM-202	SEM-212	SEM-222
Self-study support environment to summarize knowledge	Is there an environment that supports self-study and automatic grading to serve a large number of learners?	Yes	Yes	Yes
	How to implement a self-study support environment for learners?	The site to submits codes; codes are automatically graded after the deadline	The site to submits codes and returns results instantly on sample testcases; codes are automatically graded after the deadline	The site to submits codes and returns results instantly on sample testcases; the site suggests testcases; codes are automatically graded after the deadline
	Can the learner's work results be assessed?	Yes	Yes	Yes
Instructional environment	Is there an environment that instructs learners?	- Forum - Sample testcases for self-evaluation by learners	- Forum - Sample testcases with instant grading	- Forum - Sample testcases with instant grading - TRS
	What are the limitations of the instructional environment's effectiveness level?	- Forum: relies on learner questions and instructor and peer responses - Sample testcases for self-evaluation by learners: challenge to assess the level of completeness and suitability of these testcases in guiding learners.	- Forum likes on the left - Sample testcases with instant grading: can track learners' engagement with testcases through their submissions; the level of completeness and suitability of the testcases can be assessed on submissions	- Forum and sample testcases with instant grading like on the left - TRS: challenge to establish a diverse testcases bank.
Learners' learning and development	Is it possible to keep track of the learning process?	No (only the last submission is recorded)	Possible (when the learner submits the code)	Possible and enhanced by **TRS**
	Is there a way to keep track of learner development?	No (only the last submission is recorded)	Possible	Possible and enhanced by **TRS**
	Is there a way to support learners' problem-solving skills development?	No	Yes (returns results of grading sample testcases)	Yes (returns results of grading sample testcases; provides testcases hints according to learner's ability)
Interaction with learners	How is the interaction?	- Allow multiple submissions - Grading is done for only the last submission.	- Allow multiple submissions - Grading is done for the final submission; intermediate scores on sample testcases are provided for each submission.	- Allow multiple submissions - Grading is done for the final submission; intermediate scores on sample testcases are provided for each submission; suggested testcases and scores are provided for each submission on the **TRS** system.

support the question in Table II regarding the existence of a method to keep track of the learning process.

The number of learners in SEM-222 (1484) increased by 1.7 times compared to the number of learners in SEM-

TABLE III. STATISTICAL RESULTS AFTER IMPLEMENTING ASSIGNMENT IN 3 SEMESTERS

Semester	Information provided to learners	Total number of learners	Number of learners who submitted code	Number of days having submissions	Number of submissions	Average number of submissions per learner	Average number of submissions per day
SEM-202	Sample testcases	933	852 (91.32%)	N/A	852	1.0	N/A
SEM-212	Sample testcases	864	723 (83.68%)	15	5463	7.56	364.2
SEM-222	Sample testcases	1484	1332 (89.76%)	27	25009	18.78	926.26
	TRS	1484	1082 (72.91%)	14	11427	10.56	816.21

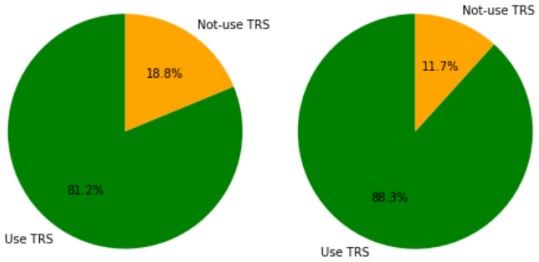
212 (864), and the number of learners participating in code submission increased by 1.8 times. The participation rate in submissions increased (89.76% compared to 83.68%), or in other words, the non-participation rate decreased. Therefore, can it be inferred that the submission system in SEM-222 better supports learners? The number of days with submissions in SEM-222 is nearly double that of SEM-212. So, could the longer submission time in SEM-222 allow learners to have more opportunities to complete their assignments? The number of submissions in SEM-222 (25009) is significantly higher than in SEM-212 (5463). Thus, is this due to the longer allowed submission time or better learner support? The average number of submissions per learner in SEM-222 (18.78) is 2.5 times higher than in SEM-212 (7.56). This indicates that the SEM-222 system supports better interaction compared to SEM-212. Furthermore, if we compare the average number of submissions per day between SEM-222 and SEM-212, the ratio is 2.5 (926.26/364.2), which is higher than the ratio between SEM-222 and SEM-212 (1.7). This suggests that the SEM-222 system provides better support to learners than the SEM-212 system.

The above results indicate that SEM-212 performs better than SEM-202, and SEM-222 performs better than SEM-212. Overall, SEM-222 has the best implementation. The level of system interaction, based on the number of submissions, is 29.4 times higher in SEM-222 (25009/852) compared to the semester with the lowest number of submissions (SEM-202), even though the ratio of the number of learners between these two semesters is only 1.6 (1484/933). Considering only the information regarding sample testcases, SEM-222 displays a significantly higher interaction level than SEM-212 and significantly outperforms SEM-202.

If we consider the additional information from TRS in SEM-222, the number of submissions increases by 45.7% (11427/25009) compared to the number of submissions in the sample testcases. Based on the total number of submissions in SEM-222 (including TRS), the system's interaction level is 42.8 times higher than the semester with the least collected interaction (SEM-202). Although this is the first implementation of TRS, the participation rate is quite impressive at 72.91%. However, it is still lower than the percentage of people who submitted on the sample testcases (89.76%). From this, we can see that students tend to resubmit their assignments on the sample testcases every time they improve their code on TRS. The reason could be that students want to ensure their submitted version is evaluated on the sample testcases. Alternatively, it is possible that students do not trust the consistency between the solutions of the two systems.

To examine the detailed impact of TRS on the sample testcases, the group of learners who submitted assignments in SEM-222 needs to be divided into two smaller groups: the group that used the TRS system (Use TRS) and the group that did not use the TRS system (Not-use TRS). Fig. 2(a) shows the percentage distribution of learners between the Use TRS and Not-use TRS groups. And Fig. 2(b) illustrates the percentage distribution of submissions between the Use TRS group (22082) and the Not-use TRS group (2927) on the system of sample testcases. Let's consider the index that measures the level of interaction through submissions on the system referred to as the average interaction index. It is calculated by dividing the number of submissions by (the number of people who submitted multiplied by the number of days of submission).

- The average interaction index of the group of learners on the sample testcases in SEM-212: 5463 / (723 * 15) = 0.5.
- The average interaction index of the group of learners on the sample testcases in SEM-222: 25009 / (1332 * 27) = 0.70.
- The average interaction index of the group of learners on TRS in SEM-222: 11427 / (1082 * 14) = 0.75.
- The average interaction index of the Use TRS group on the sample testcases in SEM-222: 22082 / (1082 * 27) = 0.76.
- The average interaction index of the Not-use TRS group on the sample testcases in SEM-222: 2927 / (250 * 27) = 0.43.



a. Number of learners b. Number of submissions

Fig. 2. Comparison between Use and Not-use TRS in SEM-222.

The average interaction score on the TRS system (0.75) is higher than the average interaction score on the sample test-

cases system during the same semester (0.70). This indicates that the TRS group has a higher level of interaction with the sample testcases in SEM-222.

Subsequently, let's examine the level of interaction between the Use TRS group and the Not-use TRS group on the sample testcases in SEM-222. The data collected in Table III shows that the number of students using TRS is 1082, and the number of students not using TRS is (1332 - 1082) = 250. The ratio of the number of people between the two groups is 1082 / (1332 - 1082) = 1082 / 250 = 4.3, while the submission ratio on the sample testcases between these two groups is 22082 / 2927 = 7.5 (1.7 times higher than the ratio of students in the two groups). This indicates that the Use TRS group has nearly twice the interaction with the sample testcases compared to the Not-use TRS group. We can also calculate similar results by determining the ratio between the average number of submissions of the Use TRS group (22082 / 1082) and the Not-Use TRS group (2927 / 250).

The average interaction score of the student group on the sample testcases in SEM-212 (0.5) is lower and approximately equal to the average interaction score of the Not-use TRS group on the sample testcases in SEM-222 (0.43) - meaning 0.5/0.43 = 1.16 times higher. This indicates that the Not-use TRS group in SEM-222 has a slightly lower interaction level than the student group in the semester without TRS support (SEM-212). This is a less active group in the exercise process and does not actively utilize the support from the teaching environment.

The average interaction score of the Use TRS group on the sample testcases in SEM-222 (0.76) is significantly higher (about 1.5 times) than the average interaction score in SEM-212 (0.5). This indicates that the Use TRS group in SEM-222 interacts more actively than the other groups (Not-use TRS in SEM-222 and the student group in SEM-212). In other words, this is the contribution of the TRS system.

V. Conclusion

This paper proposed a testcase recommendation system (TRS) for assisting learners in completing assignments in introductory programming courses. TRS provides a small set of testcases adaptive to the learner's current level of proficiency. Using learners' performance data, we propose a new testcase recommendation process based on the SVD model and the ZPD theory.

TRS was implemented and deployed in the university-level fundamental programming course in the second semester of the 2022-2023 year. Then, we investigated TRS's effectiveness by conducting a comparison with two previous semesters (SEM-202 and SEM-212) without using TRS. The statistical results have shown that SEM-222 had the highest level of interaction with learners among the three semesters (2.5 times higher than SEM-202). Additionally, the proposed TRS received acceptance and significant interaction from learners during its initial semester.

Our future work is to examine the effectiveness of TRS for learners in greater depth and make more comparisons with other testcase recommendation methods. Moreover, we aim to identify strategies for gathering information on learners' satisfaction, particularly in a new environment that supports additional testcase suggestions for assignments. We will also explore the integration of automated techniques that generate testcases using a formal descriptive language. Furthermore, our investigation will involve developing a testcase bank to provide diverse and differentiated recommendations. Lastly, the solution will be packaged as a module for seamless integration into various learning management systems (LMS), specifically focusing on Moodle LMS.

Acknowledgment

This research is funded by Vietnam National University Ho Chi Minh City (VNU-HCM) under grant number DS2022-20-07. The authors also acknowledge Ho Chi Minh City University of Technology (HCMUT), VNU-HCM for supporting this study.

References

[1] D. Turnbull, R. Chugh, and J. Luck, "Learning Management Systems, An Overview," *Encyclopedia of Education and Information Technologies*, pp. 1052–1058, 2020.

[2] ——, "An Overview of the Common Elements of Learning Management System Policies in Higher Education Institutions," *TechTrends*, pp. 855–867, 2022.

[3] S. Yildirim, N. Temur, A. Kocaman, and Y. Goktas, "What makes a good LMS: an analytical approach to assessment of LMSs," in *Information Technology Based Proceedings of the Fifth International Conference on Higher Education and Training*, 2004, pp. 125–130.

[4] J. K. Strakos, M. A. Douglas, B. McCormick, and M. Wright, "A learning management system-based approach to assess learning outcomes in operations management courses," *International Journal of Management Education*, vol. 21, no. 2, p. 100802, 2023.

[5] C. Lwande, L. Muchemi, and R. Oboko, "Identifying learning styles and cognitive traits in a learning management system," *Heliyon*, vol. 7, no. 8, p. e07701, 2021.

[6] N.-T. Nguyen, "A study on satisfaction of users towards learning management system at international university–vietnam national university hcmc," *Asia Pacific Management Review*, vol. 26, no. 4, pp. 186–196, 2021.

[7] J. Cole and H. Foster, *Using Moodle: Teaching with the Popular Open Source Course Management System*. O'Reilly Media, Inc, 2007.

[8] R. Mehta and K. Rana, "A review on matrix factorization techniques in recommender systems," in *2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA)*, 2017, pp. 269–274.

[9] Y. Koren, "Factorization meets the neighborhood: A multifaceted collaborative filtering model," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2008, p. 426–434.

[10] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: item-to-item collaborative filtering," *IEEE Internet Computing*, vol. 7, no. 1, pp. 76–80, 2003.

[11] C. C. Aggarwal, "An Introduction to Recommender Systems," *Recommender Systems: The Textbook*, pp. 1–28, 2016.

[12] J. Lee, M. Sun, and G. Lebanon, "A Comparative Study of Collaborative Filtering Algorithms," *CoRR*, vol. abs/1205.3193, 2012. [Online]. Available: http://arxiv.org/abs/1205.3193

[13] J. Bennett and S. Lanning, "The netflix prize," in *Proceedings of KDD cup and workshop*, vol. 2007. New York, 2007, p. 35.

[14] S. B. John, D. Gaur, and A. Siddiqui, "Test case Recommendation for regression with Named Entity Recognition for test step prediction," in *Proceedings of the 4th Biennial International Conference on Nascent Technologies in Engineering (ICNTE)*, 2021, pp. 1–6.

[15] S. Shimmi and M. Rahimi, "Leveraging Code-Test Co-Evolution Patterns for Automated Test Case Recommendation," in *Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test (AST)*, 2022, p. 65–76.

[16] S. Nandan, "Test case recommendation system," *International Journal of Advance Research, Ideas and Innovations in Technology*, vol. 5(2), 2019.

[17] M. Azizi and H. Do, "A Collaborative Filtering Recommender System for Test Case Prioritization in Web Applications," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. Association for Computing Machinery, 2018, p. 1560–1567.

[18] H. Xie, D. Zou, R. Zhang, M. Wang, and R. Kwan, "Personalized word learning for university students: a profile-based method for e-learning systems," *Journal of Computing in Higher Education*, vol. 31, pp. 273–289, 2019.

[19] K. Mao, Q. Dong, Y. Wang, and D. Honga, "An Exploratory Approach to Intelligent Quiz Question Recommendation," *Procedia Computer Science*, vol. 207, pp. 4065–4074, 2022.

[20] R. Baker, W. Ma, Y. Zhao, S. Wang, and Z. Ma, "The Results of Implementing Zone of Proximal Development on Learning Outcomes," *International Educational Data Mining Society*, 2020.

[21] C. Ferguson, E. L. van den Broek, and H. van Oostendorp, "AI-Induced Guidance: Preserving the Optimal Zone of Proximal Development," *Computers and Education: Artificial Intelligence*, vol. 3, p. 100089, 2022.

[22] L. D. Kantar, S. Ezzeddine, and U. Rizk, "Rethinking clinical instruction through the zone of proximal development," *Nurse Education Today*, vol. 95, p. 104595, 2020.

[23] S. Poskitt, K. A. Waylen, and A. Ainslie, "Applying pedagogical theories to understand learning in participatory scenario planning," *Futures*, vol. 128, p. 102710, 2021.

[24] D. G. Dumas, Y. Dong, and M. Leveling, "The zone of proximal creativity: What dynamic assessment of divergent thinking reveals about students' latent class membership," *Contemporary Educational Psychology*, vol. 67, p. 102013, 2021.

[25] T. Murray and I. Arroyo, "Toward measuring and maintaining the zone of proximal development in adaptive instructional systems," in *Intelligent Tutoring Systems*, S. A. Cerri, G. Gouardères, and F. Paraguaçu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 749–758.

[26] J. V. Wertsch, "The zone of proximal development: Some conceptual issues," *New Directions for Child and Adolescent Development*, vol. 1984, no. 23, pp. 7–18, 1984.

[27] L. Vygotsky *et al.*, *Interaction between learning and development*. Linköpings universitet, 2011.