

# A Recommendation Module based on Reinforcement Learning to an Intelligent Tutoring System for Software Maintenance

Rodrigo Elias Francisco<sup>1,2</sup> <sup>a</sup> and Flávio de Oliveira Silva<sup>1</sup> <sup>b</sup>

<sup>1</sup>Faculty of Computer, Federal University of Uberlândia (UFU),

Av. João Naves de Ávila, 2121, Block 1A, Room 1A243 - Campus Santa Mônica, Uberlândia, MG, Brazil

<sup>2</sup>Federal Institute Goiano (IF Goiano) - Campus Morrinhos, Rodovia BR153, KM633 Zona Rural, Morrinhos, GO, Brazil

**Keywords:** Intelligent Tutoring System, Software Maintenance, Reinforcement Learning, Q-Learning.

**Abstract:** The demand for qualified professionals to work with Software Maintenance (SM) brings challenges to computer education. These challenges are related to SM's inherent complexity and the teacher's significant work in providing adequate support in practical SM activities. In this context, Artificial Intelligence (AI) based techniques, such as recommendations, can play a central role in developing Intelligent Tutoring Systems (ITS) to focus the teaching-learning process. The literature points out a lack of ITS to SM and that most of them do not use AI-based techniques to recommend content to the students. In this work, we present an Expert Knowledge Module (EKM) for an ITS specially designed for SM. To model the EKM content, we did a deep analysis of the ACM curricula regarding SM topics and the syllabus related to SM from all Brazilian public universities. The content recommendation engine uses the *Q-Learning* algorithm, a well-known Reinforcement Learning (RL) AI-based technique. Using simulation-based experiments, we could verify the efficiency of the *Q-Learning*-based recommendation mechanism to propose contents using the ITS's EKM properly. This work highlights how AI-based techniques can enhance and improve SM's teaching-learning process using ITS and advance this research area.

## 1 INTRODUCTION

Software maintenance (SM) is responsible for about 60% of all costs (Russell and Vinsel, 2017) during the software life-cycle (Russell and Vinsel, 2017) due to high consumption of time and effort (Fernández-Sáez et al., 2018). Learning SM involves handling complex tasks related to the SM process, understanding and modifying software artifacts. Therefore, computer education needs to help students become professionals capable of working with SM (Heckman et al., 2018).

This educational problem brings motivations to design educational tools based on AI for the SM teaching-learning process, such as Intelligent Tutoring Systems (ITS). ITS is a software system to enhance, adapt, and automate (Alkhatlan and Kalita, 2018) the teaching-learning process. Although there is ITS's with different architectures, the ITS generally works with representations of the three types of knowledge: the content, the student, and teaching

strategies.

The literature about ITS for SM presents some gaps. Few works on tutor system for SM and fewer less on ITS to SM. Most of the work in the literature does not detail content recommendations. None of the ones present some recommendation mechanism that uses AI-based techniques.

In this work we present the design of the *Expert Knowledge Module (EKM)* module of a ITS for SM and a content recommendation mechanism based on the *Q-Learning* algorithm, a well-known Reinforcement Learning (RL) AI-based technique used in several areas (Shawky and Badawi, 2018). The main contributions of this work are:

- An ITS with an EKM can handle different types of content related to SM;
- The EKM modeling resulted from a comprehensive analysis of the computer science curriculum proposed by scientific societies, namely the Brazilian Computer Society (SBC) (Zorzo et al., 2017), and the Association for Computing Machinery (ACM) (ACM Computing Curricula Task

<sup>a</sup>  <https://orcid.org/0000-0003-2866-3431>

<sup>b</sup>  <https://orcid.org/0000-0001-7051-7396>

Force, 2013), and the syllabus related to SM used by all Brazilian public universities.

- A content recommendation mechanism that uses the *Q-Learning* algorithm what brings AI to enhance the results of the ITS, and that showed its efficiency in different usage scenarios.

We organized work as follows: section 2 describes the related literature and presents some gaps in this area. She section 3 describes the design of the EMK module and the content recommendation strategy focusing on the SM domain using the *Q-Learning* algorithm. Section 4 describes the simulation process and discusses its results. Finally, Section 5 presents some concluding remarks and future work.

## 2 RELATED WORK

Table 1 details information about the literature related to Tutor Systems for SM found in the literature. First, the table presents if the tutor system is an ITS. When it is an ITS, the table also shows the modules described in work using the same name proposed by the author. The table presents the subject of SM that the research addresses. The table describes if the ITS has a content recommendation mechanism and the corresponding recommendation technique indicating based on AI and highlights if the work describes the tutor module. The subjects related to SM are Software Debugging (SD), Software Refactoring (SR), and Software Understanding (SU).

We found some AI techniques in the literature. Case-based reasoning (CBR) is a strategy that involves the base of a case that is frequently updated and makes it possible to retrieve stored cases to assist in solving a new case. Natural Language Processing (NLP) uses AI algorithms and linguistic knowledge to work with human language in a given language. Q-Learning is a Reinforcement Learning (RL) algorithm that works with the metaphor of an agent inserted in an environment and performs actions to achieve some goal. The reinforcement obtained in the perception of the result makes it possible to create a model at runtime to guide these actions.

A Socratic ITS and its authoring tool (Alshaikh et al., 2021) were proposed for teaching-learning of software understanding. This work applied Natural Language Processing to generate and interpret student dialogues about source code artifacts. They also used dialogues created by experts. The research approached the innermost loop of ITS, where the tutor module performs actions that aim to help the student understand a specific source code artifact through dialogues. The domain module wraps the programs'

source code artifacts and uses the abstract syntax tree. This work did not propose any recommendation strategy.

The work of (Oberhauser, 2017) presents an approach for the recommendation, navigation, and 3D visualization of source code. This work uses a recommendation service based on a theoretical model of program understanding and data related to MethodRank metrics for source code, filtering processing, distance calculations, and points of interest in source code. Their goal was to understand software from an exploratory, analytical, and descriptive cognitive process perspective. This work did not address any AI technique and did not describe any module related to the concept of ITS.

Another work, an ITS for the software debugging domain (Carter and Blank, 2013) that uses the CBR technique. The authors claim that the debugging activity performed by the programmer is inherently case-based. Although this research does not address recommendations, the work implements the following ITS modules: domain, tutor, student, and communication. The domain module comprises syntax, runtime, and logical errors cases.

The Interactive Tutor System *RefacTutor* (Haendler et al., 2019) focus software refactoring. The work strategy uses software understanding as support to approach refactoring.

The analysis of the works presented provides essential information for this research. Few of them focus on Tutor Systems for SM and even less on ITS for SM. The literature found addresses only three SM topics. The only work that addressed content recommendation for SM did not present an AI technique. Furthermore, the papers describe little about how Tutor Systems or ITS for SM modules represent the system domain data.

## 3 DESIGN OF THE ITS FOR SM USING Q-LEARNING

This work is part of a larger research that aims to design, develop and deploy an ITS for SM. This ITS has an architecture which has *Tutor*, *Student*, EKM and *User Interface Modules*. One goal of this ITS is that it can adaptively recommend and offer support to the teaching-learning process of SM. This work focus on the EKM and the content recommendation functionality performed by the Tutor Module.

Our content recommendation mechanism uses RL based on the *Q-Learning* algorithm, which refines its decision model according to the results of previous recommendations. The EKM design started from a

Table 1: Papers on Tutoring Systems for SM.

Reference	ITS (Yes/No/All Modules? Which Modules?)	Subject	Recommend.	Recommend. Technique	AI-Based Technique	Tutor Mod.
(Carter and Blank, 2013)	Yes / All / Modules: domain, tutor, student and commun.	SD	No	No	Case-based reasoning	Yes
(Haendler et al., 2019)	No	SR, SU	No	No	No	No
(Oberhauser, 2017)	Yes / Part. / Module: recommender	SU	Yes	MethodRank	No	No
(Alshaikh et al., 2021)	Yes / Part. / Modules: domain, tutor	SU	No	No	Natural L. Processing	Yes
THIS WORK	Yes / Part. / Modules: exp. knowledge, tutor	SM	Yes	Q-Learning	Q-Learning	Yes

Software Debugging (SD) - Software Maintenance (SM) - Software Refactoring (SR) - Software Understanding (SU)

requirement analysis of the SM educational context, assuming that it would be a component of ITS that could provide an automated recommendation.

### 3.1 Expert Knowledge Module (EKM)

The EKM has an abstraction of the content that supports activities that the ITS provides to students. In this section, we present the rationale used to model the EKM, and we show a categorization of the activities capable of supporting the several topics related to SM.

To model the EKM, we carried out a survey of SM topics in the curriculum of all Brazilian Federal Universities that have courses in the computing area, such as Computer Science and Software Engineering, to contribute to the creation of the EKM. We chose all Brazilian Federal Universities because they have geographic coverage throughout Brazil, which allowed for contemplating the diversity of SM curricula. In addition, to gather the requirements of the EMK, we added to this survey the Reference Curricula proposed by the Brazilian Computer Society (SBC) (Zorzo et al., 2017), and Association for Computing Machinery (ACM) (ACM Computing Curricula Task Force, 2013) concerning SM.

The survey of topics covered in SM was necessary to support the pedagogical and technical decisions around the construction of an ITS.

The pedagogical assumptions consider the possibility of teaching SM so that learning begins with practical experiences, which gradually contributes to increasing the conceptual knowledge.

The technical assumptions considered an EKM for the SM domain that can be reused, expanded, and aligned with the architectural design of an ITS. The EKM should contribute to the operationalization of the SM content by using AI based algorithms. The

content modeling should define a semantic that eases in creating and modifying activities in ITS. Finally, we consider in the design decisions that the content managed by the EKM and recommended by the *Tutor Module* should facilitate the process of evaluating AI algorithms in instantiating them in an ITS suited to SM.

We organize the topics raised into four categories: A - *Software Understanding*, B - *SM Practice*, C - *SM Testing*, and D - *Concepts Understanding*. This categorization allowed the planning of the types of activities addressed by ITS, as shown in Table 2. We used Bloom's Revised Taxonomy (Amer, 2006) to estimate the dimensions of the cognitive process necessary by students in each type of activity.

Next, we will present each of these categories and the topics covered in the SM curricula:

- Category A - *Software Understanding* includes the following topics: understanding software artifacts, reuse, legacy systems, and concerns and concern location. This category reflects the need for the SM student to build mental models about software representation to make decisions of change in SM.
- Category B - *SM Practice* has the following topics: types of software maintenance, refactoring, covers SM practice, reverse engineering, reengineering, and debugging. This category contributes to the proposal of practical SM activities involving source code for specific systems that may include these topics.
- Category C - *SM Testing* encompasses only testing in the context of SM
- Category D - *Concepts Understanding* was divided into two subcategories: D1 - *SM Process*, D2 - *Software Maintainability*.
  - D1 - *SM Process* includes the following topics:

Table 2: Activities categorized with Bloom's Taxonomy.

Activity Type	Rememb.	Underst.	Apply	Analyze	Evaluate	Create	Nº Dimens.
A1 - Understand the result of execution (source code)	x	x					2
A2 - Order the program (parson puzzles)	x	x	x	x			4
A3_1 - Reply to abstract (source code)	x	x	x	x			4
A3_2 - Reply to abstract (UML models)	x	x	x	x			4
B1 - Defect Fix type SM	x	x	x	x	x	x	6
B2 - SM of the Environ. Adaptation type	x	x	x	x	x	x	6
B3 - SM of Add Funct. type	x	x	x	x	x	x	6
B4 - Refactoring	x	x	x	x			4
C1 - Tests	x	x	x	x	x		5
D1 - SM Process	x	x					2
D2 - Maintainability of Software	x	x					2

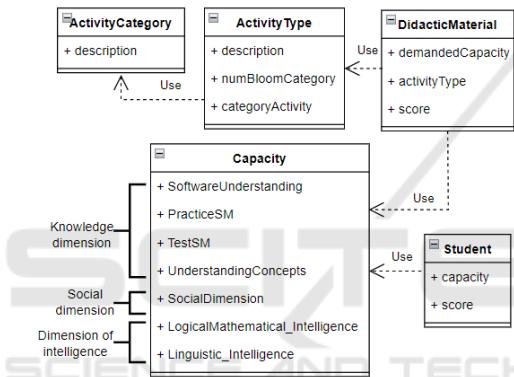


Figure 1: Expert Knowledge Module Partial Class Diagram.

process, migration, the relationship between SM types, configuration and change management, effort estimation, software repository mining, evolution laws, SM generical, prediction Analysis, Impact Analysis, Software Evolution, and Integration and Continuous Delivery.

- *D2 - Software Maintainability* has the following topics: maintenance problems, maintenance cost, maintainability and metrics for software maintainability, and (re)documentation.

As a result of the assumptions, design decisions, the survey of SM topics, and its categorization, presented in Table 2, we modeled the EKM. Figure 1 presents a partial class diagram of the EKM classes.

Activity types are represented by the class *ActivityType* and are categorized by *ActivityCategory*. *DidacticMaterial*, which represents the activities, is related to a given *ActivityType* and has the fields *score* and *demandedCapacity* to represent the score and the capacity that the student needs to be able to get it right

the question in the simulation.

*Capacity* has three dimensions: knowledge dimension, social dimension, and intelligence dimension. This work emphasizes only the dimensions of knowledge, subdivided into software understanding, SM practice, tests, and concept understanding. *Student* represents the student, who has a grade ("score") and capacity.

### 3.2 Content Recommendation

An ITS needs a model capable of allowing its *Tutor Module* to make appropriate pedagogical decisions. These decisions include recommending content considering the student's learning status concerning the SM content at a given time. We choose to use the *Q-Learning* algorithm, which builds and improves the model at run-time and does not depend on prior knowledge of experts to build the model.

The *Q-Learning* (Shawky and Badawi, 2018) algorithm, used in this proposal to recommend contents, uses an RL algorithm that models the best actions for certain states and stores this model in the Q matrix. The equation, presented in 1 represents the way the algorithm updates the matrix. At certain moments random choices occur in a controlled manner.

$$Q(s, a) = Q(s, a) + \alpha[r + \gamma * (\max Q(s', a')) - Q(s, a)] \quad (1)$$

The update of the Q matrix occurs whenever the algorithm performs an action recommendation and does its execution.  $Q(s, a)$  represents the benefit of doing action  $a$  in state  $s$ .  $\alpha$  and  $\gamma$  are the learning rate and discount factor parameters. The  $\max Q(s', a')$  function returns the *score* of the best possible action ( $a'$ ) in the new state ( $s'$ ).

To use *Q-Learning* it is necessary to model the states and actions of the environment in question and to calibrate their parameters. In a simplified way, this algorithm represents an agent that performs actions in the background and receives reinforcement, which can be positive or negative, on these actions. Upon receiving the reinforcement, the algorithm updates its model with the memory of the result of the executed action. In our *Q-Learning* configuration, the recommended actions represent a Didactic Material (DM), and the states represent the *scores* of the students.

## 4 SIMULATION AND DISCUSSION

Using simulation, we evaluated the EKM and the content recommendation mechanism based on the *Q-Learning* algorithm. In this work, the evaluation focuses only on the *knowledge dimension* considering the student *Capacity* presented in Figure 1. This section details the simulation process and presents a discussion about the results.

### 4.1 Simulation Process

Figure 2 details the simulation process and presents the sequence of activities between the following actors: the teacher, the ITS, and the student coordinated by the simulator. Initially, the teacher selects a set of students and a set of DM's and configures the learning session with the appropriate *Q-Learning* parameters. The simulator will traverse the student set. During this process, the simulator first verifies if there is a DM available. If yes, the ITS uses the *Q-Learning* algorithm and the student's current profile to *recommend a DM*. The student *solve the DM*. In the simulation process, the simulator infers success or failure for the DM. When no more DM is available for the student, the simulator displays the results.

The simulator infers success or failure by verifying if the student's capacity (attribute *Student.capacity*) is sufficient to solve the DM considering its *DM.demandedCapacity*. If the student has enough capacity, the simulator updates the current student *capacity*, its *score* and the list of solved DM's.

Using the simulator, we create a set of students in the ITS using a default constructor. The initial parameter for the simulator is the size of the set. Initially all the students capacity attributes (*softwareUnderstanding*, *practiceSM*, *testSM* and *understandingConcepts*) are set to zero. The student *score* of student represents the number of activities solved correctly and initially is also zero.

The simulator creates a set of DM to mimic the DM selection by the teacher. The simulator receives the size of the DM set and creates activities of every type (*ActivityType*) to the given size. The simulator calculates the required capacity for each activity. The attribute *categoryActivity* defines the activity category and indicates knowledge dimension attributes associated with the attribute *demandedCapacity*. The Algorithm 1 describes the calculation of the demanded knowledge capacity and the *score* for a given DM.

Algorithm 1: Calculate Demand Capacity and DM Score.

---

```

Let n be the number of DM for a given Activity
Type: DM[i] | i>=1, i<=n;
Let j be the indication of the Activity Category and
Demanded Capacity (knowledge dimension) of
the DM[i];
i=1;
while i<=n do
    DM[i].demandedCapacity[j] = i *
        (numBloomDimensions / n);
    DM[i].score = numBloomDimensions / n;
    i++;
end

```

---

To configure the learning sessions, the teacher describes the average number of times a student tries to solve a DM and the parameters of the *Q-Learning* algorithm: positive reinforcement, negative reinforcement, learning rate, discount factor, and indicator of the percentage of exploitation.

The recommendation of DM's follows the sequence of generated students. Initially, the simulator creates a Q matrix. For each student, while the student has not reached the maximum *score* or the maximum number of attempts, the ITS recommends a DM to the student. The simulator changes the student state and updates the Q matrix.

### 4.2 Results and Discussion

To evaluate the content recommendation module according to the EKM proposed, we did several simulations using different parameters configurations for the *Q-Learning* algorithm. We used ten (10) different configurations with the same simulation scenario.

The simulation scenario consisted of ten (10) students and a set with 110 DM's for each student. We created ten (10) DM's for each of the eleven (11) types of activity presented in Table 2. The goal of each experiment was to use the recommendation system until each student reached the maximum score, in this case, 110 considering one point to each DM's correctly solved.

Table 3 presents the different settings for the *Q-Learning* algorithm in each experiment. The *Max.*

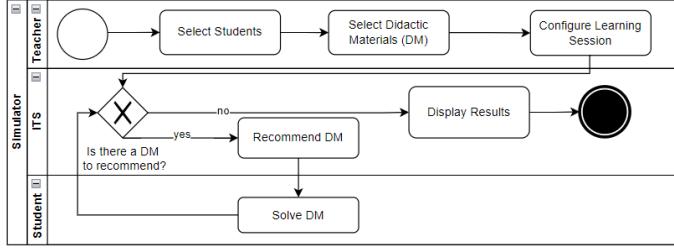


Figure 2: Simulation Process.

Table 3: Experiments and Results of Simulation for the Content Recommendation.

Experiment	1	2	3	4	5	6	7	8	9	10
Max. Number of Attempts	330	330	440	660	770	550	550	550	550	550
Exploration Percentage (%)	50	33,3	25	20	20	16,6	16,6	16,66	12,5	12,5
Positive Reinforcement	4	4	4	10	8	9	15	15	13	13
Negative Reinforcement	-4	-4	-4	-9	-8	-9	-15	-15	-13	-13
Learning Rate ( $\alpha$ )	0,12	0,12	0,12	0,16	0,16	0,4	0,4	0,5	0,5	0,7
Discount Factor ( $\gamma$ )	0,9	0,9	0,9	0,7	0,7	0,8	0,8	0,8	0,8	0,7
Number of Recommendations:	1473	1538	1569	1388	1408	1303	1276	1342	1326	1361

*Number of Attempts* sets the maximum number of attempts a student makes to solve the complete DM set. The Exploration Percentage defines the policy of the Q-Learning algorithm to explore the environment. Positive reinforcement, negative reinforcement, learning rate, and discount factor are parameters used by Q-Learning to update its model upon receiving reinforcement. The *Number of Recommendations* indicates the total recommendations that were necessary for all students to perform all DM's correctly. This number is related to the efficiency of the experiment, and the lower is better.

Experiments number 6 and 7 had the best results, as they needed fewer recommendations to reach the maximum score. These experiments have very similar configurations, and the only difference was the parameters referring to positive and negative reinforcement. There are indications that a relatively low exploitation percentage, as in the case of 16.6%, contributes to the efficiency of the experiment, which is related to the idea of privileging the Q-Learning algorithm model instead to work randomly. Experiments 2 and 3 had the worst results. We can observe that their exploitation percentages are among the highest, which indicates a more random behavior.

The simulation contributed to evaluating the content recommendation functionality of the Tutor Component with the EKM at design time. Furthermore, the experiments showed that the EKM could be properly operationalized by the simulation process and by the content recommendation functionality of the Tutor Component.

To highlight the benefits of the recommendation module based on the Q-Learning algorithm, we will

describe a hypothetical scenario with no recommendation using two different types of students. We modeled these students using average data from undergraduate computer science students (Tomkin et al., 2018).

The scenario is the same: a set of ten (10) students and a set with 110 DM's, ten (10) different DM's for each of the eleven (11) types of activity presented in Table 2. In each cycle, each type of student tries to solve all the DM's. We assumed one ability to hit a DM for each cycle, and between cycles, we assumed an increment of this ability.

Table 4 presents these values and details the hypothetical scenario considering the types of students. The *Number of Attempts (set size)* is the size of the set of DM's that the student needs to solve in each cycle. *Increment (Inc.)* shows the increase in the correctness capacity in a specific cycle for a given type of student. The *Ability to Hit* shows the total capacity to solve each DM correctly after the increment of the capacity. *Number of Hits (No. Hits)* is the number of DM's that the student solved correctly in the cycle. The table summarizes the overall performance for each type of student, presenting the total *Number of Attempts* for one and ten students. For both student types, the increment of hit ability occurs in a non-linear way. The hypothetical Student Type #1 starts with a slight hit ability of 20% with a gradual increment (0%, 10%, 10%, 20%, 30%). In contrast, the hypothetical student of type #2 starts with a correctness capacity of 50% with different gradual increments (0%, 15%, 15%, 20%) in each cycle.

Figure 3 presents a comparison between the two types of hypothetical students. The vertical line of the

Table 4: Hypothetical scenario of DM resolutions.

Cycle	Student Type #1			Student Type #2				
	No. Attempts (set size)	Inc.	Ability to Hit	No. Hits	No. Attempts (set size)	Inc.	Ability to Hit	No. Hits
1	110	0%	20%	22	110	0%	50%	55
2	88	10%	30%	26,4	55	15%	65%	35,75
3	61,6	10%	40%	24,64	19,25	15%	80%	15,4
4	36,96	10%	50%	18,48	3,85	20%	100%	3,85
5	18,48	20%	70%	12,93	-	-	-	-
6	5,54	30%	100%	5,54	-	-	-	-
<b>No. of Attempts (1 student)</b>	<b>320,584</b>				<b>188,1</b>			

chart displays the cumulative number of DM's hits, the maximum of which is 110. The horizontal line of the chart shows the cycles in which students try to solve the complete set of DM's, where the Student Type #1 uses six cycles, and student type #2 uses four cycles. Student Type #2 has a higher DM's resolution capability than Student Type #1, which allows it to resolve DM's faster.



Figure 3: Comparison between the Students Types.

Student of Type #1, which can be considered a student with lower ability, would need approximately 320 attempts to solve the 110 DM's, and the total for ten students of this type would require around 3206 attempts to solve these 110 DM's. The Student Type #2, who is considered a student with average ability, would need 188,1 attempts to solve these 110 DM's, and ten students of this type would need 1881 attempts to solve these 110 DM's.

Content recommendation and the EKM for ITS for SM presented in this work, according to the simulation's parameters described in Experiment 7, allow ten virtual students to solve the 110 DM's with 1276 recommendations. This information enables us to estimate the improvement that the proposal can bring to the efficiency of the SM teaching-learning process when considering two types of hypothetical students.

The recommendation method and the EKM proposed in this work reduce by approximately 60,2 % of the number of attempts compared to the Type #1 students, so instead of 3206 attempts, only 1276 ones

are necessary to solve the complete DM set by all students. When considering students Type #2, which has a higher ability from the beginning, we can see a reduction of approximately 32,2 %, with 1276 attempts to solve all the DM set compared to 1881 attempts.

Thus, the results of this work overcome the hypothetical scenario described, whose types of students designed meet beginner (Student Type #1) and average (Student Type #2) ability levels.

## 5 CONCLUSION

The literature on Tutor Systems for SM has gaps available for investigation. Furthermore, there are few initiatives to address the concept of ITS for SM. In the literature, we did not find a content recommendation method for SM applied in the context of an ITS, which shows that this research contributes to advancing the state-of-art. Moreover, the only mechanism of recommending SM-related content found in the literature does not use any AI-based technique.

We have realized the importance of providing Computer Science students with learning based on SM's practical and conceptual experience and the difficulty of this for teachers. An ITS can be very promising in solving this problem.

This work contributes to this area and presents an ITS with an EKM capable of handling different types of content related to SM. The EKM modeling resulted from a comprehensive analysis of the computer science curriculum based on scientific societies, namely ACM and SBC, and the syllabus related to SM from all Brazilian public universities. The EKM brings the capacity representation, which includes the SM knowledge dimension and the social and intellectual dimensions; the student representation, which consists of the ability and the score; and the necessary models for the SM content, which are activity category, activity type, and the DM.

Other researchers can explore the EKM modeling

presented to reuse and refine it to build an ITS suitable to SM that can help the SM teaching-learning process.

Our contribution goes further in this area, and we introduce a content recommendation method based on the Q-Learning algorithm what brings AI to enhance the results of the ITS.

The experiments showed that the content recommendation, through the Q-Learning algorithm, manages to make recommendations that improve over time. The experiment that reached the maximum score performed 1276 DM's recommendations, and this indicates the efficiency of Q-Learning algorithm. As there are ten students and 110 DM's, an ideal situation where students do not wrong DM's is composed of 1100 recommendations, which is very close to the experiment's value.

Using two types of hypothetical students, one with a low learning ability, called *Student of Type #1*, and another one with an average learning ability, called *Student of Type #2*, we made a comparison with our recommendation method. The results showed that our recommendation method improved approximately 60.2 % compared to students with lower ability and 32.2 % to students with average ability.

We intend to investigate data clustering algorithms to extend the ITS student's module for SM in future work. We will evaluate its influence on the content recommendation. Moreover, we will deploy and assess our ITS for SM in the educational context and corporate SM training using a real-world experimental scenario.

## ACKNOWLEDGEMENTS

This work is inside UFU-CAPES.Print Program. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. This research also received the support from PROPP/UFU.

## REFERENCES

- ACM Computing Curricula Task Force, editor (2013). *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, Inc.
- Alkhatlan, A. and Kalita, J. (2018). Intelligent tutoring systems: A comprehensive historical survey with recent developments. *arXiv preprint arXiv:1812.09628*.
- Alshaikh, Z., Tamang, L., and Rus, V. (2021). Experiments with Auto-generated Socratic Dialogue for Source Code Understanding:. In *Proceedings of the 13th International Conference on Computer Supported Education*, pages 35–44, Online Streaming, — Select a Country —. SCITEPRESS - Science and Technology Publications.
- Amer, A. (2006). Reflections on bloom's revised taxonomy. *Electronic Journal of Research in Educational Psychology*, 4(1):213–230.
- Carter, E. and Blank, G. D. (2013). An intelligent tutoring system to teach debugging. In *International Conference on Artificial Intelligence in Education*, pages 872–875. Springer.
- Fernández-Sáez, A. M., Chaudron, M. R., and Genero, M. (2018). An industrial case study on the use of uml in software maintenance and its perceived benefits and hurdles. *Empirical Software Engineering*, 23(6):3281–3345.
- Haendler, T., Neumann, G., and Smirnov, F. (2019). Refactutor: an interactive tutoring system for software refactoring. In *International Conference on Computer Supported Education*, pages 236–261. Springer.
- Heckman, S., Stolee, K., and Parnin, C. (2018). 10+ years of teaching software engineering with itrust: the good, the bad, and the ugly. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 1–4. IEEE.
- Oberhauser, R. (2017). Visitr: 3d visualization for code visitation trail recommendations. *International Journal on Advances in Software Volume 10, Number 1 & 2, 2017*.
- Russell, A. and Vinsel, L. (2017). Andrew Russell and Lee Vinsel. Let's Get Excited About Maintenance! Available in <https://www.nytimes.com/2017/07/22/opinion/sunday/lets-get-excited-about-maintenance.html>. Accessed in November 13, 2020.
- Shawky, D. and Badawi, A. (2018). A reinforcement learning-based adaptive learning system. In *International Conference on Advanced Machine Learning Technologies and Applications*, pages 221–231. Springer.
- Tomkin, J. H., West, M., and Herman, G. L. (2018). An Improved Grade Point Average, With Applications to CS Undergraduate Education Analytics. *ACM Transactions on Computing Education*, 18(4):17:1–17:16.
- Zorzo, A. F., Nunes, D., Steinmacher, I., Leite, J. C., Araujo, R., Correia, R. C. M., and Martins, S. (2017). Referenciais de Formação para os Cursos de Graduação em Computação 2017.