Semi-Automatic Assessment of Modeling Exercises using Supervised Machine Learning

Stephan Krusche
Technical University of Munich
krusche@in.tum.de

Abstract

Motivation: *Modeling is an essential skill in software engineering. With rising numbers of students, introductory courses with hundreds of students are becoming standard. Grading all students' exercise solutions and providing individual feedback is time-consuming.* ***Objectives:*** *This paper describes a semi-automatic assessment approach based on supervised machine learning. It aims to increase the fairness and efficiency of grading and improve the provided feedback quality.*

Method: *While manually assessing the first submitted models, the system learns which elements are correct or wrong and which feedback is appropriate. The system identifies similar model elements in subsequent assessments and suggests how to assess them based on scores and feedback of previous assessments. While reviewing new submissions, reviewers apply the suggestions or adjust them and manually assess the remaining model elements.*

Results: *We empirically evaluated this approach in three modeling exercises in a large software engineering course, each with more than 800 participants, and compared the results with three manually assessed exercises. A quantitative analysis reveals an automatic feedback rate between 65 % and 80 %. Between 4.6 % and 9.6 % of the suggestions had to be manually adjusted.* ***Discussion:*** *Qualitative feedback indicates that semi-automatic assessment reduces the effort and improves consistency. Few participants noted that the proposed feedback sometimes does not fit the context of the submission and that the selection of feedback should be further improved.*

1. Introduction

Software engineering deals with complexity and change. It requires the practical application of knowledge [1, 2]. Modeling a system in UML is an essential practical skill. It allows abstracting the system and leaving out intricate details. It facilitates communication between software engineers on a higher abstraction. Students typically become familiar with modeling in large undergraduate courses in university.

While examples and exercises play a central role in the early phases of cognitive skill acquisition [3], it is time-consuming for instructors to create good modeling exercises. However, carefully developed examples increase the learning outcome [4, 5]. In addition, providing individual feedback is essential in learning in a class environment [6] to improve students' skills. When following an interactive teaching style, including in-class exercises [7, 8, 9], the students submit several models over a semester and further increase the number of needed assessments.

However, with the rising number of students in introductory courses, it is nearly impossible for an instructor to manually grade all students' solutions to modeling exercises and provide individual feedback. For example, at the Technical University of Munich, up to 2,200 students from three different study programs regularly participate in the undergraduate software engineering course. One possible solution is the recruitment of tutors[1] who could take over the assessments. However, it is not easy to find enough skilled tutors who can provide good feedback for students. In addition, the instructor would need to hire, coordinate and introduce the tutors to each exercise and the possible sample solutions, which would require much discussion about the correct and wrong aspects of the exercise.

Even when the university could employ tutors, feedback from multiple persons is not always fair. The experience between tutors differs and is typically lower than the experience of an instructor, who has taught the course for many years and can be considered as a modeling expert. Inexperience can result in grading conflicts depending on how strict each tutor assesses a solution. Grading criteria help to even out the

[1] Tutors are experienced students who have passed the course in the previous year and receive money for helping in the teaching process.

URI: https://hdl.handle.net/10125/79439
978-0-9981331-5-7

inexperience, but tutors can still choose to disregard them. There is typically a sample solution for modeling exercises, so that students can compare their own solution with the sample solution. However, this might not specifically help an individual student to improve her modeling knowledge. The sample solution only describes one possible correct solution for a modeling exercise. To determine whether a solution is similar to the sample solution, students need to have existing modeling skills.

Modeling is a creative task: multiple solutions are possible, and it is not easy to assess immediately whether a solution is correct or not [10, 11]. It is not feasible to define all possible correct solutions. This paper proposes semi-automatic assessment using a customized supervised machine learning approach. While manually assessing the first submitted models, the system learns about the correct and incorrect model elements and can propose suggestions for similar model elements in subsequent assessments. When reviewing new submissions, the reviewers can accept the automatic suggestion or adjust it based on the context of the model and assess all remaining model elements manually.

The paper is structured as follows. Section 2 explains more details about teaching modeling and the challenges concerning creativity and sample solutions. Section 3 discusses related work in the context of assessing submissions to modeling exercises. In Section 4, we present the method for the semi-automatic assessment of modeling exercises using supervised machine learning. Section 5 shows an empirical evaluation of a large undergraduate software engineering course. Section 6 concludes the paper

2. Background

A UML model "captures the important aspects of the thing being modeled from a certain point of view" [12]. It is an abstraction and generalization and omits certain details. This makes models useful for software engineering because both handle the complexity by focusing on relevant details and leaving out the rest. It consists of different diagram types, e.g. UML class diagrams to describe the structure of the system.

Teaching UML modeling to students is a difficult task. While the structural aspects of a model follow certain rules that can be learned by heart, it is challenging for students to map a realistic problem statement to a concrete model. Including all semantics in a correct way is challenging and sometimes not even possible due to abstraction and simplification. Students have to decide on certain trade-offs when modeling a problem which includes higher-level skills.

Bloom's taxonomy classifies learning goals [13]. Modeling requires understanding and analyzing a problem, evaluating different design solutions, and combining them to create new knowledge. Designing a system and modeling it in UML allow for multiple ways how to solve the problem. The freedom in creating UML models has impact on the assessment of modeling exercises. Multiple solutions can be correct and can show important aspects of a problem. There is a risk in teaching that instructors focus on one particular sample solution and that students misunderstand that only this solution is correct and all other solutions are wrong.

However, the nature of modeling is abstraction by hiding complex details. It is difficult to decide which complex details should be hidden and which aspects are necessary to understand the core idea of the problem [14]. Even experienced modelers will come up with different solutions which is one advantage of modeling because it stimulates discussion among the system and facilitates communication. But this also poses a challenge in teaching, especially if instructors do not have enough time to teach students about these aspects of modeling, e.g. in undergraduate courses.

One sample solution cannot cover all aspects of a problem and can constraint the creativity of students. While it allows to compare, it is difficult to decide if a solution is still correct if it differs from the sample solution. Showing a sample solution poses the risk that students learn one out of many representations of a system by heart instead of using their creativity to come up with their own representation. Students should learn how to address a problem with the help of modeling by applying the skill on their own. Instructors then provide guidance with an individual assessment for each student.

Such an assessment should "provide learners an opportunity to demonstrate achievement of outcomes and to demonstrate progress toward the learning outcomes" [15]. It identifies gaps between the student's understanding and the actual knowledge [16]. Individual feedback includes information why certain aspects are incorrect and facilitates learning. Students learn about their mistakes, improve their solution and identify strengths and weaknesses. When instructors assess UML models, they typically scan the submitted model for known (groups of) model elements and evaluate if they are correct in their structure, semantics and visual aspects [17]. In large courses, they face similar models frequently which leads to recurring manual tasks.

3. Related Work

Soler et al. created a learning platform [18] and support exercises for database design [19].

Instructors have to specify the attribute names, which should be used in the solution, and define multiple sample solutions in a prescribed format to assess submissions automatically. Solutions describe classes and associations. The specified attributes from the exercise description are assigned to the classes in the solutions. The system determines the most similar sample solution by matching the attributes.

By comparing the most similar sample solution with the student submission, the system returns errors such as incorrect numbers of classes or associations or multiplicity errors. In contrast to the approach presented in this paper, students do not have to come up with attributes themselves, which is an important skill in modeling. Instructors have to define multiple sample solutions before the assessment starts, which is time-consuming and it is not guaranteed that all possible solutions are covered. The system is limited to UML class diagrams and does not learn which elements are correct or wrong on the fly.

Hasker developed a similar approach [20], but does not require exercise descriptions to specify attribute names. Matching between sample and student solutions is done by name matching, e.g., class names are matched as a whole and for attributes, methods and associations, the system uses substring matching.

Hoggarth and Lockyer [21] shifted the problem of finding similarities between a sample solution and a student solution from the system to the student. An instructor provides one sample solution. When students submit their solution, they are presented a dialog box. It breaks down both solutions in parts and lists them. Student then have to match their model elements to those of the sample solution. The system provides the student a list of feedback describing the differences between the solutions, which acts as a guidance. This approach only allows one sample solution, which might not be similar at all to the student solution. It also assumes that students are familiar with modeling and can match their solution to the sample solution.

Ali et al. [22] designed a system where the student submission is compared to a so-called learner's sample solution. The correctness is checked in three steps: class structure analysis, verification process and language checking. The steps are executed sequentially, only if the previous step did not result in errors. The first step checks the general structure of the UML model, e.g., number of classes and attributes. The second step verifies whether the associations are correct. The last step validates the naming of the classes, attributes and methods. This approach has the disadvantage that the sample solution does not cover all possible solutions. Students have the freedom to choose attribute names themselves, so it is likely that the student solution cannot be compared to the sample solution.

Striewe and Goedicke propose a more flexible solution to assess UML models [23]. They take advantage of UML diagrams being similar to graphs. Instructors can define rules to describe a desired or undesired element. With a graph query engine, the system can verify the rules. Apart from exercise specific rules, they implemented general rules, e.g. checking missing names, roles, cardinalities or directions of associations. Although their solution is more flexible than the other ones, the instructor still has to define the rules manually and it is not guaranteed that all possible rules are covered and that all possible student solutions can be assessed with these rules.

4. Method

Automating recurring tasks in the assessment can reduce the correction effort. At the same time, it can preserve that multiple submissions are considered as correct and would not limit the modeling creativity of students. To support this idea, a modeling assessment framework needs to learn from manual model grading and apply the learned knowledge for the correction of other model submissions enabling a semi-automatic assessment approach as shown in Figure 1.

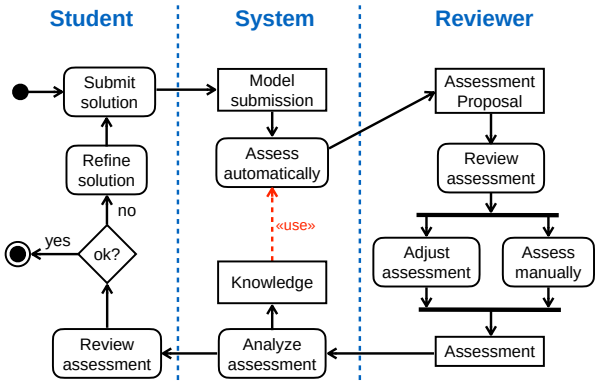

Figure 1. Semi-automatic assessment workflow based on supervised learning.

Students submit their models as solution to a specific exercise. In the beginning, there is not enough knowledge for the system to assess automatically, so the assessment proposal will be empty and reviewers needs to assess the complete solution manually. The system analyzes the manual assessment and generates knowledge that can be used to propose automatic assessment for subsequent submissions with similar model elements. Then, reviewers start with a partly assessed submission with automatically proposed

feedback. They can either confirm the proposed assessments in case they are correct, or adjust the assessment in case they are wrong for the current submission, e.g. due to a different context. Following an interactive learning approach [7], students can review the assessment and use it to refine their model and submit another version to the same exercise, if the due date of the exercise has not passed yet.

To enable the system to learn, two core concepts are necessary. First, the system needs to decompose models into smaller elements. Figure 2 shows the important model elements in a UML class diagram: classes, attributes, methods and associations. Other diagrams can be decomposed into different model elements.

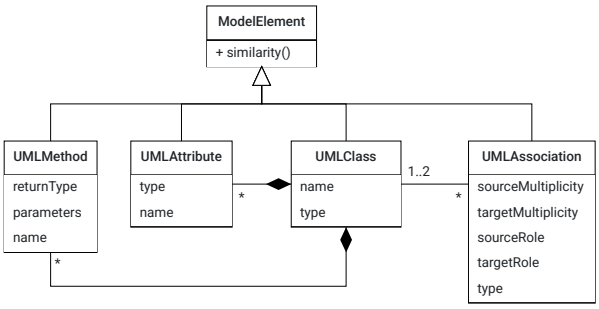

Figure 2. Taxonomy of important model elements

Second, it needs to identify similar model elements in other submissions based on similarity analysis and build model clusters. A cluster includes all similar model elements of all submissions in one exercise. Model elements are considered similar based on their type, name and context. All model elements in one cluster have the same type (e.g. `UMLMethod`), similar properties (e.g. a similar name) and the same context (e.g. a similar parent class). Then, the system can apply the existing assessment of one model element to other elements in the same model cluster. The knowledge in Figure 1 is then the combination of the model clusters (based on similarity analysis) and the manual feedback associated to the model elements in one cluster.

Attributes and methods belong to the same model cluster if they are specified in the same UML class and if they have a similar name. Similarity of names could be measured using a Levenshtein distance metric [24]. The system would assume two names are similar if the normalized Levenshtein distance metric is higher than e.g. 90 %[2]. A more sophisticated similarity metrics using natural language processing, topic modeling and synonym analysis could also be used [25].

[2]The actual computation is more difficult and takes all properties of a model element into account.

UML classes belong to the same cluster if they have a similar name, if they have a similar kind (e.g. abstract, interface) and if they have similar associations to other classes. The last point includes the context of classes to ensure classes are only identified as similar, if they are defined in the same context. Associations belong to the same cluster if they have the same source and target, have the same kind and have similar multiplicities and roles with only small deviations.

When a reviewer assesses one model element in a model cluster, the system learns whether the model element is correct or not and can apply the same score and feedback to all other model elements in the same cluster. By using this approach, each manual assessment creates additional knowledge which can be used for automatic assessment proposals. It makes sure that in the manual reviewing process, multiple solutions can be identified as correct, even if they deviate. And it also focuses on detailed feedback that highlights mistakes on a small scale, i.e. the individual model element. The system can suggest the next manual assessment as the one where the most knowledge is gained by choosing the submission in which the least amount of model elements would include an assessment proposal. Figure 3 shows an example how the system learns.

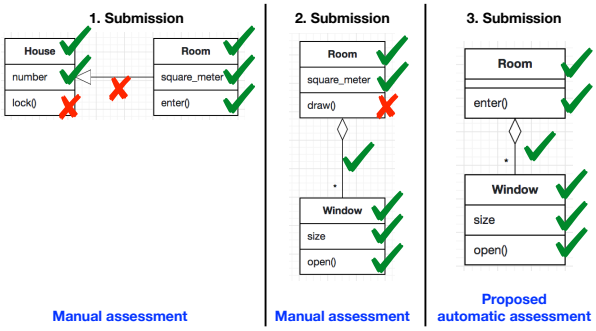

Figure 3. Example how manual assessments lead to knowledge for a proposed automatic assessment

The first two submissions are manually assessed and produce enough knowledge so that the system can propose assessments for all six model elements in the third submission. While it might be tempting to automatically submit the assessment of the third submission, we decided that it is still necessary that a reviewer inspects the model. There are several reasons for this: First, the context and similarity detection are not always correct. Second, missing model elements cannot be identified with this approach. Third, even if the scores (in the example expressed with green ticks and red crosses) are proposed correctly, the textual feedback (not shown in Figure 3) might not fully apply.

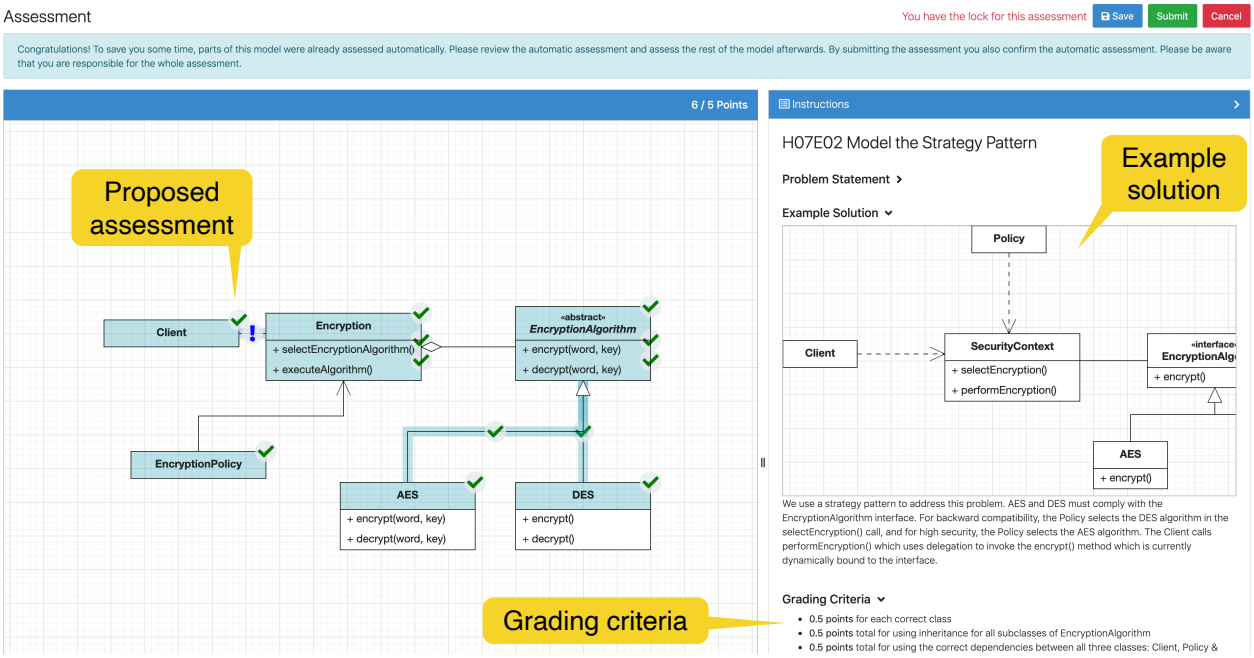

Figure 4. Assessment user interface: the assessment editor on the left includes proposed assessments (highlighted in blue). Instructions on the right include problem statement, example solution and grading criteria.

Figure 4 shows how the assessment user interface. The student submission is on the left. All model elements in blue include automatically proposed feedback. In the shown example most elements are correct (green tick) and only one element has a neutral score (the association between `Client` and `Encryption`) shown with a blue exclamation mark. Some elements are not yet assessed: two associations and the four methods with a white background. The reviewer can see the problem statement on the right side (collapsed in Figure 4). An example solution with an explanation and grading criteria are also shown. A reviewer can simply drag and drop grading criteria to model elements to simplify the review process. Then, a predefined score and feedback is added to the model element which can further be adapted if needed.

5. Evaluation

We implemented the semi-automatic assessment method in Artemis [26], a learning management system that focuses on individual and immediate feedback. Artemis integrates the lightweight online modeling editor Apollon [27] and is open source. We used the method in three exercises in a large undergraduate software engineering course SE1 in the summer 2019 with 1600 participants. SE1 teaches basic software engineering concepts, such as requirements analysis, system design, object design, testing, delivery, UML modeling and design patterns. The only prerequisite is experience with object-oriented programming, e.g. gained in a CS1 course. Exercise participation is voluntary, the final grade is based on an exam.

5.1. Study Design

This section describes the design based on quantitative analysis in six exercises and qualitative analysis in an online survey with 22 tutors.

5.1.1. Exercises In the following six exercises, students had to create a UML class diagram.

`E1` Reverse engineer database tables to a model
`E2` Model build and release management as object model
`E3` Create an analysis object model for Scrum
`E4` Choose a design pattern
`E5` Mapping Java to UML
`E6` Bumpers Analysis Object Model

`E1`, `E2`, and `E3` use semi-automatic assessment as proposed in this paper. `E4`, `E5`, and `E6` are three randomly chosen modeling exercises for a comparison.

`E1` was about reverse engineering database tables into a UML class diagram. Students should leave out unnecessary attributes and methods and group columns logically in an inheritance taxonomy. The exercise

included several hints about the structure of the UML class diagram and was constrained in terms of creativity. While there have been multiple correct solutions, e.g. how to exactly model the inheritance relationship, the model had to match the given database scheme.

In E2, the students had to transform a given informal build and release management workflow into an analysis object model including useful attributes and methods. The resulting analysis object model should include the most important concepts as objects, their properties as attributes and the activities as methods. The given workflow contained several activities such as compiling, testing, building and releasing the source code as well as using the software and providing feedback. It also included typical tools such as version control and continuous integration systems as well as three roles: developer, release manager and user.

E3 built on the previously taught knowledge about Scrum [28]. Student had to create an analysis object model with the most important core concepts of Scrum and their relationships. The idea was to model the Scrum artifacts (e.g. backlog item) and meetings (e.g. sprint planning meeting) as objects and add proper attributes and methods (e.g. estimate() on the object backlog item). The exercises E2 and E3 included more freedom than E1, because the exercise was not to create a complete and fully consistent model of the given theories, but to express the most important concepts in an understandable way.

E4 asks the students to choose an appropriate software design pattern [29] based on a problem statement and to model the concrete situation using this design pattern. While the exercise provides some kind of creativity, the problem statement narrows down the choice of patterns and includes several key words that student should use in the resulting UML class diagram as class name, attributes or methods.

In E5, the students had to reverse engineer Java code into a UML diagram. Correct solutions to the exercise only allowed a limited number of choices, e.g. whether to use aggregation, composition or unidirectional associations, or the explicit use of default values (role names, multiplicities, etc.). Therefore, the creative nature of the exercise was limited.

Student had to extend an existing analysis object model in E6 based on new requirements in the context of the game Bumpers. In Bumpers, one player steers a car and crashes it in a specified way into other cars that are randomly controlled by the computer. The students experienced two sprints as in-class exercises and had to work on the third sprint as homework including the implementation of new simple requirements and the modeling of the extension to the UML class diagram of the previous sprint. The concrete realization included a couple of creative aspects, e.g. in the choice of the concrete mode of crash detection. E6 focused on the the modeling part of the homework exercise.

5.1.2. Quantitative Analysis We evaluate the resulting data of all six exercises in a quantitative analysis and **compare** the data of exercises with semi-automatic assessment (E1, E2 and E3) with the data of exercises with manual assessment (E4, E5, E6). We obtained the data by executing SQL statements against the database of Artemis. The data consists of real student submissions and assessments by tutors. The analysis focuses on the assessment type of the individual model elements (automatic, adjusted or manual). By analyzing the amount of automatically proposed feedback for model elements compared to the total number of feedback items for model elements, we gain information about the percentage of automatically assessed model elements (*automatic assessment rate*).

In order to compare the evaluation results of different exercises with each other, a way to classify modeling exercises in terms of *variability* is necessary. The variability provides information about the number of possible solutions to an exercise and we expect that it is closely coupled to the automatic assessment rate of the system. For example, an exercise with only one possible solution creates less model clusters which increases the potential of automatic assessments compared to an exercise with multiple different possible solutions. One way of analyzing the variability is the comparison of the problem statements of the exercises.

However, this does not constitute a formal method, as it does not yield an unambiguous, comparable benchmark and it is subject to the experience of the person who evaluates the problem statement. Therefore, we introduce a variability index that specifies a distinct value for each modeling exercise. It takes the average amount of model elements per submission and the number of model clusters into account:

- $a = avg.\ elements\ per\ submission$
- $p = \#model\ clusters\ with\ positive\ feedback$
- $c = \#all\ model\ clusters$
- $e = \#assessed\ elements$

We define two variability indices v_1 and v_2:

$$v_1 = \frac{p}{a} \tag{1}$$

$$v_2 = \frac{c}{a} \tag{2}$$

Definition (1) takes all model clusters into account that have a positive score. It represents the number of different model elements of all submissions that the reviewers considered as correct. The average number of model elements per submission is an indication about the size of the solutions. Both numbers in relation provide the variability of the corresponding exercise, i.e. it is an indicator of the amount of possible correct solutions. The higher the index, the greater is the variability of the exercise.

If the number of correct model elements is much higher than the average number of model elements per submission, there must be multiple different possible solutions. If both numbers are almost identical, the elements in the correct solutions do not vary much which indicates that there is only one correct solution to the exercise. Equation (2) is similar, but considers all model clusters instead. In contrast to equation (1), it provides information about the variety of all student submissions, not only the correct ones. For a better comparison between exercises, the variability values should be between 0 % and 100 %. Therefore, we adjust the equations in the following way:

$$v_p = \frac{p - a}{e - a} \tag{3}$$

$$v_c = \frac{c - a}{e - a} \tag{4}$$

v_p describes the variability in terms of good solutions, while v_c describes it in terms of all solutions. If the number of model clusters equals the average number of elements per submission, the submissions do not vary much. In this case, v_p and v_c converge to 0. If the number of model clusters equals the number of assessed elements, every model cluster contains only one element and there are no similar model elements among the submissions. Then, the normalized equations converge to 1 and express that variability is high.

5.1.3. Qualitative Analysis To evaluate the quality of the proposed system from a users perspective, we conduct a qualitative analysis in the form of a user survey. The survey targets the 45 tutors that used the new approach for the assessment of the exercises in the SE1 course. It consists of different questions that focus on the quality of the automatic assessments and the impact on the manual assessment process.

Since the tutors have used the pure manual and the new semi-automatic assessment approach in SE1 in the summer term 2019, the survey is suitable for a direct comparison of both approaches. The online survey asked the tutors what they like and what they do not like about the approach using free text answers. We invited all 45 tutors of SE1 to participate in the survey. 22 of the tutors (response rate 48.9 %) completed the survey.

5.2. Objectives

We formulate the following hypotheses with respect to the semi-automatic assessment approach that was implemented in the modeling assessment framework Compass in Artemis [26]:

H1 **Reduced assessment effort**: The approach reduces the manual assessment effort.

H2 **Improved feedback quality**: The approach improves the quality of the feedback.

While the proposed system aims to reduce the manual assessment effort, it must also focus on the quality of the resulting feedback at the same time. The quality has a direct impact on the students' learning progress. Therefore, the main goals are to reduce the manual effort, while at the same time improving the quality of the provided feedback in terms of fairness, consistency and understandability. High quality feedback takes the context of the exercise into account and helps students to improve their understanding and prevents misconceptions.

5.3. Results

This section structures the results according to the different evaluation methods. It describes the outcome of the quantitative analysis of the data from the four exercises of the case study. It shows the most meaningful answers of the online survey with the tutors.

5.3.1. Quantitative Analysis The quantitative analysis evaluates data from modeling exercises where the proposed method was used. Table 1 shows the results of the analysis. Each column represents one of the exercises `E1` - `E3`. The table lists the number of assessed submissions and the total number of assessed elements as well as the ratio (i.e. how many model elements have been assessed).

In the three exercises, between 836 and 887 students submitted their model solution. The submissions of exercise `E2` contain the most elements in total and also the most assessed elements. This exercise has the highest average amount of elements and assessed elements per submission. The feedback items represent the assessment of single model elements of the submissions.

Exercise	E1	E2	E3
Assessed submissions	887	877	836
Total elements	17,558	32,954	21,361
Assessed elements (e)	14,035	23,942	12,894
Assessed elements (ratio)	79.9 %	72.7 %	60.4 %
Avg. elem. per submission (a)	19.8	37.6	25.5
Avg. assessed elem. per submission	15.8	27.3	15.4
Automatic feedback	76.1 %	80.3 %	65.0 %
Adjusted feedback	7.0 %	4.6 %	9.6 %
Manual feedback	17.0 %	15.1 %	25.4 %
Model clusters (c)	943	2,641	2,232
Avg. elements per model cluster	14.8	9.1	5.8
Optimal manual correction effort	6.7 %	11.0 %	17.3 %
Max. automatic assessment rate	93.2 %	89.0 %	82.7 %
Variability index v_p	1.4 %	7.1 %	10.0 %
Variability index v_c	6.6 %	10.9 %	17.2 %

Table 1. Results of the quantitative analysis for exercises `E1` - `E3` (assessed semi-automatically)

The next three rows show the respective proportion of the different feedback types and add up to 100 %:

- **Automatic**: The feedback was created by the system automatically and it was not adjusted.
- **Adjusted**: The feedback was created by the system automatically, but the score of feedback text was changed by a reviewer.
- **Manual**: The feedback was manually created by a reviewer as the system could not (yet) review the corresponding model element.

In the three exercises, the automatic assessment rate is between 65.0 % and 80.3 %. This number show how many model elements could be assessed automatically by the system. `E1` has the smallest number of model clusters with 943, `E2` has the highest number `E2` with 2,641. `E1` has the highest amount of elements per model cluster with 14.8. The optimal correction effort is the relation of model clusters to assessed model elements. It describes the maximum amount of model elements that reviewers would have to assess manually in an optimal scenario, i.e. without any duplicated assessment of similar model elements. From `E1` to `E3`, this parameter lies between 6.7 % and 17.3 %.

Table 1 shows the theoretical maximum automatic assessment rate, which is the maximum amount of model elements that can be assessed automatically by the system with the given data. It is the optimal correction effort subtracted from 100 %. The last row in the table displays the variability indices. v_c is 6.6 % for `E1`, 10.9 % for `E2` and 17.2 % for `E3`.

Exercise	E4	E5	E6
Assessed submissions	927	921	1,111
Total elements	25,135	10,020	48,771
Assessed elements (e)	8,494	4,937	6,092
Assessed elements (ratio)	33.8 %	49.3 %	12.5 %
Avg. elem. per submission (a)	27.1	10.9	43.9
Avg. assessed elem. per submission	9.2	5.4	5.5
Model clusters (c)	1,453	251	1,936
Avg. elements per model cluster	5.9	19.7	3.2
Optimal manual correction effort	17.1 %	5.1 %	31.8 %
Max. automatic assessment rate	82.9 %	94.9 %	68.2 %
Variability index v_c	16.8 %	4.9 %	31.3 %

Table 2. Results of the quantitative analysis for exercises `E4` - `E6` (assessed manually)

For a comparison, we calculated the same values with three completely manually assessed modeling exercises `E4` - `E6` in the same SE1 course. Table 2 shows the results. `E4` was a medium exercise with 27.1 model elements per submission and a variability of 16.8 %. `E5` was a rather small exercise with 10.9 model elements per submission and a small variability of 4.9 %. It therefore would have the smallest optimal manual correction effort. `E6` was a relatively large exercise with 43.9 model elements per submission and a greater variability of 31.3 %.

In contrast to the exercise `E1` - `E3` shown in Table 1 the ratio of assessed elements is significantly smaller. The main reason is that feedback was not automatically applied to other submissions.

5.3.2. Qualitative Analysis The survey asked tutors what they like and what they do not like about the approach. They answered in free text. The following list presents a representative selection of the most meaningful answers to the question what they like:

- "Most of the time it got the score correct."
- "Clear visualization of automatically assessed items."
- "I got an insight to the Feedback of the other tutors which led to more consistency in grading."
- "Easy models were corrected very good."
- "The more assessments are done, the less I have to assess manually. I can focus on the key elements."
- "Good overview about what parts of the diagram are correct in the first view, I could see for what other tutors gave points."

The majority of the replies stated that the proposed score was mostly correct. The tutors liked the visualization of the automatic assessments. Another

advantage that several tutors mentioned is that they got an overview of how other tutors graded the elements which increases the consistency of the assessments. The tutors did not like the following aspects:

- "Feedback message are not accurate sometimes."
- "It was faster for me to assess without automatic assessment, my feedback was more complete than the automated one. I was slower because we were told to double check and not trust the automated assessment. Reviewing all feedback and scores was exhausting."
- "Feedback was sometimes inconsistent with scores."

25 % of the tutors mentioned that the proposed feedback was not always matching the context of the elements and had to be adjusted. Two tutors consider the semi-automatic approach to be more time consuming than the purely manual assessment workflow.

5.4. Discussion

The evaluation shows that the semi-automatic assessment approach reduces the manual assessment effort of the tutors, thus supporting **H1**. The survey reveals that not all tutors consider the new approach as helpful. One of the problems is the selection of the proposed feedback as it is too specific to the situation of a submission in most cases and does not fit the context of similar elements in other submissions.

One way of improving the approach would be to increase the influence of the elements' context on the similarity calculations. This could increase the accuracy of the feedback as the model clusters are more context specific. However, it might also result in a lot of model clusters with very few or even only one single element and thus decrease the automatic assessment rate. We should be careful with the influence of the context and also consider other approaches for selecting and assigning automatic feedback instead.

With respect to **H2**, we can observe that the number of assessed elements between purely manually graded exercises (`E4 - E6`) increases when using the semi-automatic approach (`E1 - E3`). In conversations with students, we found that the perceived quality increased in the exercises with semi-automatic assessments. The number of complaints was lower than for manually assessed ones. However, we were not yet able to quantitatively assess the quality and consistency of the contained feedback texts and compare it with manually graded exercises. It is not feasible to analyze the whole data set, but it would be interesting to analyze representative, random samples. Therefore, we have first anecdotal evidence that supports **H2**, but additional empirical evaluations are needed.

5.5. Limitations

This section analyzes the limitations of the evaluation regarding the validity of the results. The first limitation concerns the significance of the online survey. 22 tutors (48.9 %) who used the system completed the survey. This number is too small to obtain reliable results. The validity of this evaluation is limited by the similar reasons regarding wording and understanding.

While the tutors had previous experience in modeling, most of them had only limited modeling skills and it was the first time for them to assess models. A general issue with user questionnaires as the only quality assessment method is the negativity bias. Users are more likely to remember negative than positive experiences and, therefore, provide more negative feedback.

We cannot be sure that all of the tutors actually reviewed every single automatically assessed element. It could be the case that some tutors relied too much on the automatic assessments which would distort the results of the quantitative analysis. To derive reliable findings from evaluation results, the size of the conducted study is one of the decisive aspects. The three exercises on which the proposed system was tested may be too few to make general statements about the effectiveness of the new semi-automatic assessment approach.

6. Conclusion

Modeling is not a deterministic activity. Instead, it includes creativity: there are typically multiple correct solutions to a given problem. In this paper, we presented a semi-automatic approach for assessing solutions to modeling exercises. Based on a customized machine learning approach including a supervised classification and an automatic similarity detection for clustering, we developed a system including three main components:

1. **Apollon:** A modeling editor that allows students to create and submit models.
2. **Artemis**: An assessment editor to manually grade the student submissions and to provide feedback.
3. **Compass:** An automatic assessment component that categorizes similar elements and learns which model elements are correct and wrong by analyzing manual assessments.

Apollon is open source[3] and allows to model freely and without an account[4]. Artemis is open source[5] and is used in many courses in more than ten universities.

[3] `https://github.com/ls1intum/Apollon`
[4] `https://apollon.ase.in.tum.de`
[5] `https://github.com/ls1intum/Artemis`

We applied the approach in three exercises in which more than 800 students participated. In each exercise, this approach was able to propose automatic feedback to more than 65 % of the model elements automatically. A qualitative study with the tutors shows that the approach has several advantages, but some tutors are skeptical, and the feedback text selection needs to be improved. For example, showing how many other tutors have already manually assessed or confirmed the score could improve this issue. In addition, adding predefined options for feedback in the grading criteria could simplify the feedback selection.

The Artemis version used for the evaluation did not yet support multiple submissions. Tutors should be able to start assessing solutions while students are still working on the exercise. Students who submit early and receive an assessment before the due date have the chance to include the feedback to improve their model. This improvement will allow students to show that they have improved their modeling skills and receive another assessment.

References

[1] T. Connolly, M. Stansfield, and T. Hainey, "An application of games-based learning within software engineering," *British Journal of Educational Technology*, vol. 38, no. 3, pp. 416–428, 2007.

[2] D. Shaffer, "Pedagogical praxis: The professions as models for postindustrial education," *Teachers College Record*, vol. 106, no. 7, pp. 1401–1421, 2004.

[3] K. VanLehn, "Cognitive skill acquisition," *Annual Review of Psychology*, vol. 47, pp. 513–539, 1996.

[4] J. Sweller and G. A. Cooper, "The use of worked examples as a substitute for problem solving in learning algebra," *Cognition and Instruction*, vol. 2, no. 1, pp. 59–89, 1985.

[5] J. G. Trafton and B. J. Reiser, "Studying examples and solving problems: Contributions to skill acquisition," tech. rep., Naval HCI Research Lab, Washington, DC, USA, 1993.

[6] R. Higgins, P. Hartley, and A. Skelton, "The conscientious consumer: Reconsidering the role of assessment feedback in student learning," *Studies in higher education*, vol. 27, no. 1, pp. 53–64, 2002.

[7] S. Krusche, A. Seitz, J. Börstler, and B. Bruegge, "Interactive learning: Increasing student participation through shorter exercise cycles," in *Proceedings of the 19th Australasian Computing Education Conference*, pp. 17–26, ACM, 2017.

[8] S. Krusche, N. von Frankenberg, and S. Afifi, "Experiences of a software engineering course based on interactive learning," in *Proceedings of the 19th SEUH*, pp. 32–40, 2017.

[9] S. Krusche and A. Seitz, "Increasing the Interactivity in Software Engineering MOOCs - A Case Study," in *Proceedings of the 52nd HICSS*, pp. 1–10, 2019.

[10] C. Lange, B. DuBois, M. Chaudron, and S. Demeyer, "An experimental investigation of uml modeling conventions," in *International Conference on Model Driven Engineering Languages and Systems*, pp. 27–41, Springer, 2006.

[11] C. Lange and M. Chaudron, "An empirical assessment of completeness in uml designs," in *Proceedings of the 8th International Conference on Empirical Assessment in Software Engineering*, pp. 111–121, 2004.

[12] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language. Reference Manual*. Addison Wesley, 1999.

[13] B. Bloom, "Taxonomy of educational objectives: The classification of educational goals," 1956.

[14] S. Krusche, D. Dzvonyar, H. Xu, and B. Bruegge, "Software theater—teaching demo-oriented prototyping," *Transaction on Computing Education*, vol. 18, July 2018.

[15] T. Stavredes and T. Herder, *A guide to online course design: Strategies for student success*. John Wiley & Sons, 2014.

[16] A. Ramaprasad, "On the definition of feedback," *Behavioral Science*, vol. 28, no. 1, pp. 4–13, 1983.

[17] J. MacCreery and B. Tenbergen, "On the syntactic, semantic, and pragmatic quality of students' conceptual models," in *Proceedings of the 52nd HICSS*, 2019.

[18] J. Soler, J. Poch, E. Barrabés, D. Juher, and J. Ripoll, "A tool for the continuous assessment and improvement of the student's skills in a mathematics course," in *Technologies de l'Information et de la Communication*, pp. 105–110, 2002.

[19] J. Soler, I. Boada, F. Prados, J. Poch, and R. Fabregat, "A web-based e-learning tool for uml class diagrams," in *Education Engineering*, pp. 973–979, IEEE, 2010.

[20] R. Hasker, "Umlgrader: An automated class diagram grader," *Journal of Computing Sciences in Colleges*, vol. 27, no. 1, pp. 47–54, 2011.

[21] G. Hoggarth and M. Lockyer, "An automated student diagram assessment system," in *SIGCSE Bulletin*, vol. 30, pp. 122–124, ACM, 1998.

[22] N. Ali, Z. Shukur, and S. Idris, "A design of an assessment system for uml class diagram," in *International Conference on Computational Science and its Applications*, pp. 539–546, IEEE, 2007.

[23] M. Striewe and M. Goedicke, "Automated checks on uml diagrams," in *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, pp. 38–42, ACM, 2011.

[24] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *Transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1091–1095, 2007.

[25] N. Indurkhya and F. Damerau, *Handbook of natural language processing*, vol. 2. CRC, 2010.

[26] S. Krusche and A. Seitz, "ArTEMiS: An Automatic Assessment Management System for Interactive Learning," in *Proceedings of the 49th SIGCSE*, pp. 284–289, ACM, 2018.

[27] S. Krusche, N. von Frankenberg, L. M. Reimer, and B. Bruegge, "An Interactive Learning Method to Engage Students in Modeling," in *Proceedings of the 42nd ICSE*, pp. 12–22, ACM, 2020.

[28] K. Schwaber, "Scrum development process," in *Proceedings of the OOPSLA Workshop on Business Object Design and Information*, 1995.

[29] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson, 1994.