Gamification of conceptual modeling education with UML class diagrams: an experimental analysis

Software and Systems Modeling (2026) 25:239–270
https://doi.org/10.1007/s10270-025-01282-5

REGULAR PAPER

Gamification of conceptual modeling education with UML class diagrams: an experimental analysis

Giacomo Garaccione[1] · Riccardo Coppola[1] · Luca Ardito[1] · Marco Torchiano[1]

Received: 4 July 2024 / Revised: 3 January 2025 / Accepted: 6 March 2025 / Published online: 7 April 2025

Abstract
UML has become, throughout the years, the most popular modeling language for the conceptual design of software. However, UML diagrams are frequently flawed with semantic and syntactical errors. One of the main root causes for such issues can be traced back to software modeling education in software engineering curricula, which is typically given less attention than core development activities. The objective of this manuscript is to describe the application of gamification (i.e., the use of game-related mechanics in non-gameful contexts) to increase the motivation and engagement of Master's students in learning the core concepts of UML modeling. Our tool prototype includes typical gamification mechanics such as avatars, achievements, scoring mechanisms, and leaderboards and incorporates a system for automatic validation of the correctness of the student's solution. We empirically evaluated the benefits achieved through the tool by performing a controlled experiment with 280 Master's students. We found that the use of gamification significantly increased the student commitment to perform exercises, the completeness of the exercises, and the semantic quality of the produced diagrams. Through standard usability questionnaires, we also gathered positive responses and attitudes toward the usage of the tool.

Keywords Gamification · UML modeling · Teaching · Information systems

1 Introduction

Class diagrams are one of the instruments provided by the UML family of visual modeling languages, typically employed by software engineers, analysts, and developers during activities pertaining to requirements engineering and Object-Oriented code development. The diagrams are unanimously considered a very useful instrument to provide a shared understanding of the information and data manipulated by the software, and to create a structure of the software that can aid its implementation with object-oriented programming languages, clearly defining all the required classes and the relationships among them.

There are, however, diffuse issues with the understanding of class diagram that lead to an under-utilization by developers and requirements engineering. Among the most crucial, literature has highlighted the perceived complexity of the diagrams [1], pressing time constraints in the software development process [2], a preference for more informal instruments for documentation [3], and finally a lack of training and awareness [2]. The latter point suggests the need for more thorough training in class diagram modeling and an encouragement for junior developers to engage in this activity.

Even if class diagram modeling is typically taught as a part of software engineering programs in Computer Science curricula, the topic is often seen as secondary with respect to software design and development. The pedagogy of conceptual modeling education has been extensively studied in related literature, highlighting a diffuse tendency among students to repeat some specific categories of errors, and the necessity of a significant amount of practice to acquire sufficient proficiency [4]. Several studies in the literature

Communicated by Timothy Lethbridge.

✉ Giacomo Garaccione
giacomo.garaccione@polito.it

Riccardo Coppola
riccardo.coppola@polito.it

Luca Ardito
luca.ardito@polito.it

Marco Torchiano
marco.torchiano@polito.it

[1] Department of Control and Computer Engineering, Politecnico di Torino, Corso Duca degli Abruzzi, 24, 10129 Torino, Italy

have analyzed methodologies and techniques to enhance the motivation and engagement of students. Tools to automatically generate exercises and variations have been proposed; however, the generation of new exercises—albeit indeed beneficial for learning—might not be helpful if the activities to perform are perceived as monotonous and cumbersome. To intrinsically motivate students in performed activities, Gamification (i.e., the application of game design elements in non-game contexts [5]) has affirmed and has been deeply studied in the recent literature as a means capable of having positive impacts on students' learning experience and perception of learning activities [6].

The objective of our study is thus to incorporate gamification mechanics to the teaching of class diagrams, with the objective of fostering motivation and enhancing engagement in software modeling activities. Related software engineering literature has highlighted that gamification mechanics may significantly improve the performance in software development-related activities and the learning outcomes of computer engineering courses [7, 8].

In this paper, we report the outcomes of the application of a gamified learning tool in an Information Systems Master's course, within the Management Engineering program at Politecnico di Torino. The tool, UMLegend, incorporates state-of-the-art gamification mechanics that have been extensively validated by literature reviews focused on gamification in software engineering [9, 10] and includes an evaluation engine to automatically provide quality feedback to the students and ease the instructor's workload. We evaluate the advantages of gamification through an empirical experiment, examining its effects on student productivity and the accuracy of the generated diagrams. We also assess—through questionnaires—the usability of the system and the student's perception of the used mechanics. The present paper is the continuation of our previous work where we conceptualized the tool [11].

The remainder of the manuscript is structured as follows: Section 2 provides background information about gamification in modeling languages education, and ways of evaluating UML class diagrams; Section 3 provides a description of the tool and the gamification mechanics it includes; Section 4 describes the design of the conducted experiments; Section 5 reports the results of the experiment, that are then discussed in Section 6; finally, Section 7 provides a conclusion and future research directions in the topic of Gamified software modeling.

2 Background

In this section, we report background information about gamification and its application to educational activities in software engineering. We also discuss possible means of automatically generating and evaluating UML class diagrams available in the literature.

2.1 Gamification

Gamification is a field that has witnessed a growing popularity in recent years, not limited to the field of computer and software engineering. One of the most common definitions of gamification has been provided by Deterding et al., as "the use of game design elements in non-game contexts". The objective of gamification is the use of mechanics that are typical of game design, to foster specific behaviors and emotions in people who are performing tasks that are either work-related or perceived as non-gameful experiences. Gamification therefore seeks to apply the features that engage people in playing games, to increase the motivation to perform activities that are otherwise perceived as uninteresting or tiresome.

Many frameworks have been described to categorize the different mechanics that contribute to building gamified environments. One of the most popular frameworks is the Octalysis framework by Yu-Kai Chou [12]. The model identifies eight principal dimensions for gamified mechanics (namely, the *core drives*) and discriminates between positive motivators (i.e., prizes for performing correct actions or obtaining correct outcomes) and negative motivators (i.e., penalties in the opposite situations). The model allows developers to characterize and graphically represent the presence and relative importance of the core drives in a specific gamified tool or activity.

The eight core drives are defined as follows: i) epic meaning (feeling that one's activities build to a greater goal); ii) accomplishment (being motivated by having goals to reach and progressively discovering new elements); iii) empowerment (having the freedom to express oneself and act as one prefers, receiving feedback after actions); iv) ownership (possessing and managing goods and items); v) social influence (interacting with peers by cooperating and/or competing with them); vi) scarcity (dealing with rare and desirable elements that require higher effort and commitment); vii) unpredictability (encountering different experiences over time instead of monotonous ones); viii) loss and avoidance (being careful with one's actions due to negative consequences and errors).

A second notorious gamification framework is the MDE framework proposed by Robson et al. [13]: this framework distinguishes three main components in the form of Mechanics, Dynamics, and Emotions.

Game mechanics are the building blocks that compose the game experience, are defined by a game designer, and are fixed for each player for a specific experience (e.g., two different players will interact with the same mechanic). Points, leaderboards, badges, and rewards can be considered

mechanics. Game dynamics are the desired player behaviors that are stimulated during play, they depend on how a player reacts to the game mechanics and are thus variable; emotions are the mental states perceived by a player throughout the game experience, and are essentially the combination of how the mechanics are followed and which dynamics have been generated.

According to the MDE framework, a game designer should choose and integrate the best mechanics that, depending on the game context, can stimulate game dynamics that can trigger the target emotions.

GAME is another framework described by Marczewski [14] and is based on two phases. The first step is planning and designing, with a focus on gathering key information about the target users through surveys, while the second step consists of designing user activities and outcomes. The author identifies four key constructs to target to increase motivation: relatedness, autonomy, mastery, and purpose.

Another relevant framework is the MDA framework (mechanics, dynamics, and aesthetics) [15]: the three constructs that make up the framework represent the components of the game, in terms of data and algorithms, the way mechanics behave depending on player input, and the target emotional response the game is expected to elicit in the player, respectively. The process of defining a gamified experience according to the MDA framework should focus mainly on the goals that the aesthetics aim to reach and the emotions they should arouse, and only after that should come the definition of the dynamics that support these goals, and the mechanics that implement them.

Lastly, the 6D framework described by Werbach and Hunter [16] is another notorious framework worth mentioning. Its name comes from the six steps that compose it: i) define business goals (listing all objectives, identifying key ones and their motivation); ii) delineate target behaviors (identify the behaviors users of the tool will have and metrics and win states for each); iii) describe the players (identifying the users' skill levels and personality types); iv) devise activity cycles (defining feedback loops depending on user actions and progressive challenges); v) do not forget the fun (making sure that the gamified system is actually enjoyable); vi) deploy the appropriate tools (iterative development to constantly improve the gamified tool).

All the frameworks we described place key importance on the design phase and on how the mechanics need to reflect specific goals; among all of them, the Octalysis framework is the one we deem to be the most balanced, due to its focus on including negative aspects which are not explicitly considered by the others. The development of the UMLegend tool has thus followed the rules identified by such a framework and aimed at producing a balanced experience, offering both positive and negative motivators.

2.2 Gamification for software engineering education

While many corporate applications of gamification mechanics have been described in the literature, a natural application of gamification is in education and training. Regarding software engineering education, gamification has been a natural choice to increase motivation in activities typically seen as collateral to actual software development, e.g., software testing, code quality evaluation, software design, and requirements engineering.

Alhammad et al. provided a systematic mapping of the applications of gamification in software engineering education [17]. They underlined that gamification is applied principally in lab sessions, project works, and lectures of traditional SE courses. The primary studies found for the mapping covered different phases of software engineering, with a focus on software construction, software process improvement, and software maintenance. Software design appeared as less fashionable than other software engineering activities for the application of gamification mechanics.

Gasca-Hurtado et al. [18] describe the pedagogic instrument design (PID), a methodology for creating gamified teaching environments for software engineering by identifying the necessary teaching strategies, techniques, and materials. The PID method is defined as a sequence of five steps that are intended to produce pedagogic artifacts and are as follows: i) preparation (the definition of the expected learning goals and competencies earned); ii) design of the gamified instrument (rules to achieve the goal, necessary game materials, roles for involved players and expected responsibilities, step-by-step description of how the game should be played); iii) pilotage (a pilot test run of the game to identify issues); iv) scheduling (adjusting the gamified tool depending on the results of the previous step); v) assessment (evaluating the results obtained with the tool).

Marín [6] describes, through examples of different gamified tools, a set of key lessons derived from applying such techniques to software engineering education. These lessons include, for example, the benefits brought by collaboration on the learning process, the fact that using concrete elements leads to higher motivation, the need to provide theory support with lectures before using gamification to teach complex topics, and the importance of elements such as narration and story (for enjoyment), in-game rewards (for motivation), and leaderboards (for self-esteem).

2.2.1 Gamification for UML class diagrams

In more recent years, several studies have been published discussing the benefits of the application of gamification in modeling courses. To assess the current state of the literature and identify the gap that we aim to address with the

UMLegend tool, we performed a mapping review aimed at identifying the existing tools and serious games (games whose intended purpose is not enjoyment but learning [19]) applied to class diagram education.

To build the pool of sources for the mapping, we have performed a quasi-systematic literature research on the Google Scholar library, by using *gamif* class diagram** as the search string. As exclusion criteria, we have excluded all the papers that were not explicitly tackling class diagrams, that were not applying actual gamification aspects or were not related to serious games, papers in languages other than English, and grey literature papers (e.g., theses, non-peer reviewed papers or preprints).

We have enhanced the set of collected papers with convenience sampling, e.g., related manuscripts that we already analyzed or that were present as references in multiple papers mentioned in the previous sections. We stopped the research by adopting an effort-bounded approach, by stopping our search when a full page of results on the Google Scholar engine contained 10 papers that had all to be excluded.

The search for the mapping was performed in November 2024: a total of 80 papers emerged from the initial search; of these, 17 papers were found to be discussing the implementation of a gamified tool or a serious game for teaching UML; we then excluded one paper, which we found to be a preprint and another that was an older work by us authors, where we described the UMLegend tool in its basic form. The remaining 15 papers constituted the base pool of sources, and we integrated this set with 7 additional sources, which did not come out during the search but were identified separately as relevant for the purpose of our mapping analysis. The full list of papers, including the excluded papers and the result of the mapping, is available online.[1]

We frame the mapping analysis following the guidelines defined by Garousi et al. [20] for conducting literature reviews: the criteria for exclusion were defined and stated systematically, the goal of our analysis is to identify gamified tools and serious games in the literature for UML class diagram education, the stopping criteria we decided to adopt was an effort-bounded one.

Our mapping intends to provide an overview of the existing tools, the game mechanics they employ, and the adherence to Octalysis, our reference framework for the development of gamified tools, in the form of the core drives implemented in the tool.

Jurgelaitis et al. analyzed the application of gamification in teaching UML by using a level-based course structure, involving points, achievements and badges, and leaderboards [7]. A trading system is also present, where students can spend the virtual currency they obtain by completing exercises in exchange for example/tutorial diagrams.

Junior and Farias have presented ModelGame, a quality model to support software modeling learning in a gamified way. The model serves as a reference framework so that instructors can obtain a parameterized way to evaluate UML models created by learners [21], presents missions and scenarios to overcome, and uses progress, points, and feedback to motivate students.

Bucchiarone et al. presented PapyGame, a gamified modeling environment for Papyrus, and conducted several user evaluations that highlighted positive results [22, 23]. The game implements modeling assignments paired with the Hangman game and offers sequential tasks. Successfully completing tasks yields points and allows progression in the game, while failing displays a feedback screen with highlighted errors. Both works share a consensus about the positive impacts of gamification on productivity and learning outcomes for software modeling teaching.

Livovský and Porubän [24] discuss the implementation of a 2-dimensional adventure game where players navigate levels, solve challenges, and face enemies to learn object-oriented topics. An Object Access Tool ties the game and its elements to a UML class diagram, with two possible ways of interactions: players can create new classes in the diagram to generate objects in the game, and can also interact with game objects to see the corresponding class and execute methods.

Classutopia [25, 26] is a role-playing game where students act as a futuristic robot battling against an evil wizard. Students can customize the robot's statistics by creating class diagrams that describe the robot's features and unlock new abilities. Players challenge the villain by fixing defective class diagrams, with correct fixes and errors damaging the enemy and player, respectively, and color-coded feedback being shown directly on the diagrams.

Denden et al. [27] describe an online course on Object Oriented Design Methodology using Moodle. Experience points are obtained by completing learning activities, bonus skill points are awarded for finishing optional teacher assignments, completing a quiz on a specific course topic is rewarded with a badge, and students are ranked in a leaderboard according to their experience points.

GaMoVR [28–31] is a virtual reality environment where students can solve UML modeling exercises by directly rearranging and connecting classes. A Hangman game is offered as the challenge, with different modes: a textual set of requirements must be modeled, with errors adding pieces to the gallows shown next to the model, a time challenge, and a mode where a diagram with errors is shown and players must shoot at the errors. Completing challenges gives experience points for leveling up, with new levels unlocking new tools and bonuses for the exercises.

Boughzala et al. describe the use of innov8 [32], a serious game where players act as business consultants for a fictional company, collecting information in order to create

[1] https://doi.org/10.6084/m9.figshare.25968082.v2

a target class diagram by interacting with the different game characters. Students create their collective solution, which is compared to a teacher-defined reference, and receive feedback for their resulting diagram.

LearnER [33] is a modeling platform where a textual description is provided and students build a data model by picking elements (e.g., class names, multiplicities) related to the description out of a predefined set. Points can be earned for each exercise, asking for hints reduces the reward, the evaluation produces hints and a progress score, leaderboards for each exercise and the total number of points are also present.

Krugel and Hubwieser [34] present LOOP, an online interactive course on object-oriented programming topics with quizzes, interactive tasks, and programming exercises. A block-based editor can be used to create class diagrams, and direct feedback is offered to help students better understand the theory concepts.

ModelDefenders [35, 36] is an educational tool for teaching model-driven engineering that employs a mutation testing strategy: given a class diagram under test, students are divided into attackers and defenders. Defenders define test cases (sequences of actions for the model and their expected output), and attackers make changes to the model to introduce bugs. Mutants are killed if at least one test case has a different output on the mutated diagram compared to the original model.

Dupuy-Chessa et al. [37] discuss a serious game with a Pictionary-like approach: students are divided into two teams and have to advance on a game board by taking turns in selecting cards. The cards contain specifications for the models that players have to represent, with drawing challenges having limited time and the possibility receiving limited feedback and corrections from the teachers.

PolyGloT-UML [38] is described as a combination of multiple educational tools (Papyrus web editor, 2-dimensional virtual environment) where students can take learning paths and interact with each other in a virtual hub, perform different activities defined by the lecturers (e.g., quizzes, modeling exercises), and complete assignments to earn badges.

Sanjaya and Titan [39] theorize the development of a mobile application for the gamification of SE topics, including class diagrams. Teachers can create interactive quizzes and case studies about system design and requirements, with the possibility to define badges, achievements, and rankings, as well as the conditions to earn them.

We summarize the results of our mapping in Table 1: for each tool, we state whether the tool is either a gamified tool or a serious, game, the game mechanics it offers, and which Core Drives are satisfied.

The results of our mapping show that although there are different solutions in the literature for teaching UML class diagrams with gamification strategies or serious games, none of the existing solutions seem to support the creation of student-defined solutions: the focus is on static exercises with pre-defined answers and modeling options. The UMLegend tool we propose is expected to be the first gamified tool to allow free diagram creation for students, together with automated evaluation in comparison with a teacher-defined solution.

2.3 Evaluation of UML class diagrams

Unified modeling language (UML) class diagrams are fundamental in software engineering and system design, influencing how easily systems can be implemented, maintained, and scaled. Evaluating the quality of these diagrams goes beyond mere syntax checking; it involves determining their effectiveness in representing the system, communicating information, and adhering to design best practices. Several works in related literature have proposed means of evaluating the outcome of class diagrams by comparing them with reference solutions and/or assessing their inherent syntactic and semantic qualities.

In one of the first works in the literature about conceptual modeling evaluation, Lindland et al. introduced a framework for identifying and achieving quality improvement goals in conceptual models [40]. The framework encompasses syntax, semantics, and pragmatics, focusing on language rules, model adherence to domain characteristics, and stakeholder comprehension. While this article provides valuable insights into model quality, its broad focus on conceptual modeling rather than UML-specific criteria makes it less suitable for this study's specific needs.

Anas et al. introduced a semi-automatic approach for comparing student-generated diagrams with teacher-provided ones using graph-matching techniques [41]. The method involves preprocessing diagrams into graphs, matching elements to calculate similarities, and performing formative and summative evaluations. It employs syntactic, structural, and semantic similarity measures to assess lexical similarities, property similarities, and relational similarities among graph elements. This method has been effective in measuring similarities and detecting differences between various UML class diagrams.

Sun et al. presented criteria and guidelines for effective diagram layout to enhance readability [42]. Key criteria include minimizing edge crossings and bends, grouping related elements close together, and avoiding overlapping nodes and edges. The study evaluates how these criteria improve the readability of diagrams created using tools like Borland Together and Rational Rose. The focus here is on the visual layout and its impact on program comprehension, rather than the content or correctness of the diagrams themselves.

Table 1 Summary of existing game-like solutions for UML class diagram education

Tool name	Strategy	Game mechanics	Octalysis core drives
GamoVR [28–31]	Gamification	Levels, points, unlockable elements, feedback	Accomplishment, loss and avoidance
Moodle course (Jurgelaitis et al.) [7]	Gamification	Points, virtual currency, leaderboard, badges	Accomplishment, ownership
ModelDefenders [35, 36]	Gamification	Points, competition, feedback, challenges	Accomplishment, empowerment
LOOP [34]	Serious game	Feedback	Accomplishment
Gamified mobile app [39]	Gamification	Challenges, badges, leaderboards	Accomplishment
LearnER [33]	Serious game	Progress, points, hints, feedback, leaderboards	Accomplishment
ModelGame [21]	Gamification	Points, progress, feedback	Accomplishment
Moodle course (Denden et al.) [27]	Gamification	Points, badges, leaderboard	Accomplishment
innov8 [32]	Serious game	Cooperation	Social influence
Pictionary-like [37]	Serious game	Cooperation, feedback, rewards	Social influence, scarcity, accomplishment
PolyGloT-UML [38]	Gamification	Feedback, challenges, socialization, badges	Accomplishment, ownership, social influence
Classutopia [25, 26]	Serious game	Feedback, challenges, unlockable rewards	Epic meaning, accomplishment, empowerment
2D action-adventure game [24]	Serious game	Levels, feedback	Epic meaning, accomplishment, empowerment, unpredictability
PapyGame [22, 23]	Gamification	Points, feedback, progress, levels	Accomplishment

Bolloju et al. aimed at aiding inexperienced analysts to create high-quality UML artifacts [43]. By analyzing UML class diagrams produced by undergraduate students, the study identifies common errors and suggests training programs to address these issues. The methodology focuses on syntactic, semantic, and pragmatic quality aspects, providing a comprehensive framework for evaluating UML diagrams. This approach ensures that diagrams are not only structurally correct but also relevant to the domain and understandable to all stakeholders. The authors define the following quality aspects for UML class diagrams:

- *Syntactic Quality:* assess if the class diagrams follow the syntactic structure of UML class diagrams. The rules verified for syntactic quality are:
  - Missing cardinality details;
  - Inappropriate naming of classes and associations;
  - Incorrect use of UML symbols.
- *Semantic Quality:* evaluate the accuracy and completeness of the diagrams in representing the intended domain. It is measured in terms of: (i) Validity: all elements and relationships in the diagrams should accurately represent the domain; (ii) Completeness: the diagrams should include all necessary elements and relationships. The rules verified for semantic quality are:
  - Incorrect Cardinality;
  - Aggregation in place of association;
  - Wrong location of attributes or operations;
  - Operations cannot be realized using existing attributes and relationships.
- *Pragmatic Quality:* focus on the understandability of the diagrams from the perspective of stakeholders. The rules verified for pragmatic quality are:
  - Redundant attributes and associations;
  - Specialization with no distinction among subclasses;
  - Inconsistency in styling and conventions.

Nikiforova et al. analyzed existing comparison methods and combined metrics such as Shallow Lexical Name Similarity and Attribute Similarity [44]. The method pairs elements from two diagrams based on their semantic meaning, calculates distances between them, and aggregates these distances into a model difference vector. This vector quantifies the differences between diagrams, allowing for a systematic comparison of classes, attributes, methods, and relationships. The approach effectively differentiates diagrams based on both naming conventions and structural differences. The semantical distance defined by the authors is based on four distinct distance vectors, analyzing the differences between the diagram under analysis and a reference (solution) diagram based on four main aspects: classes and interfaces, class attributes, methods, and relations between elements. The higher the resulting distance, the more distant the proposed solution is from the reference solution; a distance equal to 0 indicates a diagram that is equal to the reference solution.

In addition to the aforementioned evaluation rules and criteria, the literature offers several examples of automated evaluators for UML class diagrams: one example is UML-Grade [45], a 6-step approach where teachers define a target UML class diagram for a set of software requirements and the evaluation criteria for the student-produced solutions: an automated comparison between the student and teacher models produces feedback (e.g., scores for each criterion) for both student and teacher.

Modi et al. [46] proposed a tool that implements the UML-Grade approach in the Java language: the tool is able to parse multiple student-generated solutions with a teacher-defined reference and, by comparing the different source XMI files, produce text files with the list of correct elements and missing ones. To the best of our knowledge, the tool has yet to be employed in a study with real student-produced class diagrams.

Another approach was defined by Bian et al. [47], who described an algorithm for evaluating student-made diagrams by identifying matches between their elements and those present in a reference solution. The algorithm uses syntactic, semantic, and structural comparisons to identify matches between the classes and exploits semantic similarity between class names to facilitate the process. The algorithm was implemented in a tool and used to grade a modeling assignment with 20 students: the tool was able to assign grades within less than a 14% difference compared to the lecturer of the course.

Wodel-EDU [48] is an educational tool relying on Wodel [49], a domain-specific language for the creation of mutants (variants of an existing model generated by performing different operations such as creating/editing/deleting an element). By using Wodel, teachers can define the constraints a diagram must have for an exercise, and then use the DSL to automatically perform an evaluation of a student-produced model according to the constraints.

Fauzan et al. [50, 51] proposed an algorithm for computing the similarity between two diagrams by taking a semantic and structural approach: the semantic similarity is computed by identifying pairs of classes that are most similar to one another (e.g., similar enough names, most similar attributes and methods with corresponding types), while the structural similarity is computed by observing the relationships (associations, generalizations, dependencies) between the classes in the two diagrams and creating graphs that represent them, who are then compared to obtain the similarity score.

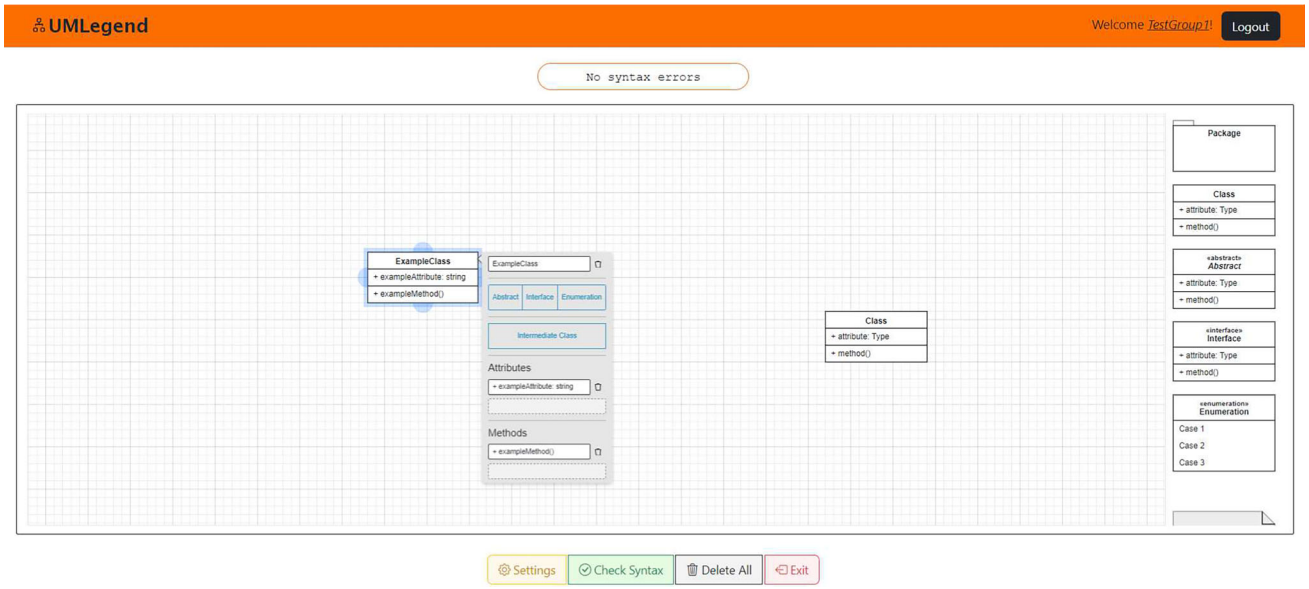


Fig. 1 Base modeler of the UMLegend tool.

2.4 Automated generation of modeling exercises

Related literature has explored the possibility of automatically generating diagram-based exercises for different platforms. A prominent example is the research result by Gomez-Abajo et al. and the aforementioned Wodel-EDU to automatically generate exercises for diagrammatic languages [52]. A strong point of the approach is its generalizability, i.e., the tool is not tailored only for the generation of class diagram exercises but can be used to generate exercises for any task involving diagrams, e.g., electric circuits. To obtain this result, the authors rely on providing a meta-model and its graphical representation to the tool, and also on providing possible modifications, namely *mutants* that can be applied to generate variations of a base exercise. The approach provided by the authors, however, cannot be directly applied to generate alternative exercises when the final objective is drawing a new diagram. It can, in fact, be applied for non-generative exercises like multiple diagram choice, alternative response, and missing word choice. The application of mutations to the solutions of a diagram can be, however, considered in future works for the generation of working alternatives of the same solution, that can be added to the reference solution in order to capture multiple alternative valid solutions for the same exercise.

Foss et al. propose a question generation and evaluation system for UML database design diagrams, called AutoER [53]. The generated exercises are centered on designing database schemas capturing the requirements described in textual problem statements. The tool automatically generates question variants by randomizing parameters in the instructor-created questions, such as the number and names of entities, the textual descriptions of relationships, and the question text order. The tool also randomizes the question generation for the exams, allowing for unique problem sets for each student.

Other approaches have been proposed in the literature for the automated generation of exercises in several different disciplines: examples are the works by Sadigh et al., who analyzed the application of a mutation-based approach to generate exercises for an embedded systems course [54], or by Papasalouros, who used a model-based approach for the automated creation of exercises in Euclidean geometry [55].

3 Tool description

We describe in this section the main concepts that compose our tool and the adopted gamified mechanics. The tool has been developed as a React-based web application that implements the mechanics inside a UML modeling canvas.

This modeling canvas, displayed in Fig. 1, lets students easily create diagrams through a drag-and-drop functionality: the available UML constructs on the right can be placed directly on the available space and connected by linking the blue dots on each element.

The modeler also allows students to download a JSON representation of their diagram compatible with the original Apollon editor used as base[2], as well as in a PDF or SVG format.

UMLegend also employs a dedicated admin interface where teachers can define new exercises by specifying

[2] https://apollon.ase.in.tum.de/, last accessed on 04/06/2024

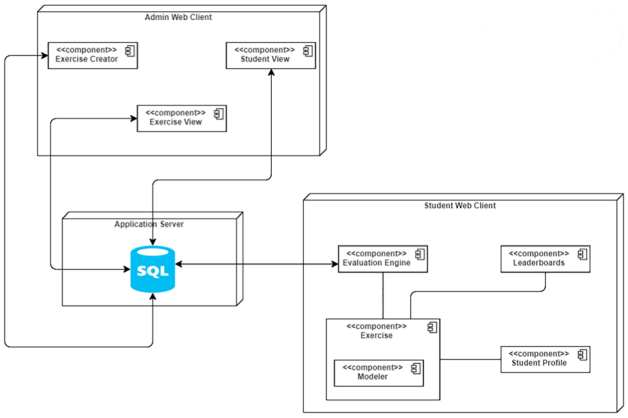


Fig. 2 Component diagram representing the UMLegend tool.

parameters such as textual description, difficulty level, experience reward, and reference solution. The reference solution is determined through a form where teachers can specify all classes, attributes, and associations that identify the expected class diagram representation for an exercise.

The admin dashboard can also be used to view the full list of all students and the saved data for the exercises: the scores obtained by the students, as well as their current diagrams and the results of the evaluations performed on them, can be downloaded to analyze the students' performance.

A visual representation of how the tool is structured can be seen in Fig. 2, the web server is a single entity, while the two clients can be accessed from any device, meaning that students can resume their work thanks to their account storing the entire history of their exercises.

3.1 Gamified mechanics

The gamified mechanics we have implemented in UMLegend can be divided into mechanics that define the general user progression and are intended for long-term usage, and mechanics that affect the context of each specific exercise in response to a student's actions.

3.1.1 Long-term mechanics

The long-term mechanics are those mechanics that build up on the student's progression as they keep solving exercises on the tool. They are meant to give a sense of growth and provide a representation of a student's improvement in modeling; an increasing challenge with higher rewards is used to motivate students.

The first example of a long-term mechanic is in **Levels and Experience Points**: levels represent the natural progression of a student, increasing as they complete exercises.

Experience is earned when a student completes an exercise by reaching a specific completeness threshold inside an exercise, with additional bonuses for good performance (e.g., completing with a completeness score much higher than the threshold, or completing in a few checks).

A second mechanic, intended to offer a way for users to personalize their experience with the tool, comes in the form of **Avatar Customization**: when students open the tool for the first time they have a randomized starting appearance with variable props such as hair, clothing, facial hair, hair color, and accessories.

Additional props and props types can be unlocked after leveling up, providing a reward for completing exercises that goes beyond just leveling up.

The final long-term mechanic is used to implement student competition in the form of **Leaderboards**: students can view their rankings in comparison with their peers and can also show off their customized avatar, displaying the rare props they unlocked by solving exercises.

There are two possible leaderboard types in which users appear: one ranks them by level first and by experience points second, while a different leaderboard type, specific for each exercise, uses the correctness score as a metric.

3.1.2 Exercise-specific mechanics

The exercise-specific mechanics of UMLegend are those mechanics that depend on the diagrams produced by a student when performing an exercise, as they depend on the evaluation of a proposed diagram solution.

A student can launch an evaluation of their diagram at any time through a button and see the changes applied to the tool interface as a consequence of the evaluation.

An impactful mechanic that comes after an evaluation is **Avatar Feedback**: a student's avatar appears on the exercise page and is used to represent the student's reactions to the evaluation. The avatar will change expression the more mistakes are made and the experience reward is reduced, becoming progressively sadder.

This progressive change in avatar expression is used as a negative motivator, to deter students from making careless mistakes and encourage them to be more careful when working. A crying avatar can be seen in Fig. 3(1).

Indicators are a mechanic that can be used for tracking the general status of an exercise: they come in the form of two distinct progress bars, shown in Fig. 3(2).

The first progress bar displays the current completeness percentage, computed after every check as the number of elements in the diagram that match an element in the reference solution over the total elements in the reference solution. This indicator can be used to gauge the distance toward the successful completion of an exercise.

The second bar, instead, shows the experience points that can be obtained once the exercise is completed: the bar starts at the maximum experience reward for the exercise and is

reduced in case there are errors found after a check, with the experience reduction depending on the severity of the mistakes.

As a way to make solving exercises not excessively frustrating, the experience reward has a predefined minimum threshold which is always granted after completion, no matter the number of errors; moreover, it is possible to earn back some of the experience lost when fixing an error found in a previous evaluation.

The results of an evaluation are made available for viewing through the **Error Lists** mechanic: after performing a check, these lists will be filled with detailed explanations of the errors found by the evaluator. Errors are divided into syntax and semantic errors, and pragmatic warnings, following the distinction identified by Bolloju et al. Each error type has a specific color assigned to it, as can be seen in Fig. 3(3), and the error explanation also contains a reference to the corresponding element, in case an error depends on a specific class or attribute.

The color coding of different error types also applies to the **Diagram Feedback** mechanic: each element in a diagram that has an error is colored depending on the error type (orange for syntax errors, red for semantic ones, and blue for pragmatic warnings); the diagram shown in Fig. 3(4) contains an example of this feedback, with a wrong association and two wrong attributes.

This mechanic, combined with the error lists, can be used for identifying with ease which elements in the diagram need to be fixed and the reason behind the errors. In case all errors related to an element are fixed that element is restored to the default coloring.

Lastly, the leaderboard ranking students on the exercise completeness is also accessible directly in the exercise through the button in Fig. 3(5), which opens a menu containing the exercise description and the leaderboard.

Students can check their standing in real-time, with their position being updated directly after every check, allowing for instant gratification in case they manage to get a high position.

3.2 Evaluation engine

UMLegend was conceived as a tool that can be used by students who are performing modeling exercises to get feedback on the correctness of their proposed solution; for this purpose, a built-in evaluation engine that can parse a UML class diagram for errors and violations in comparison with a reference solution is necessary.

The evaluation process is divided into three steps:

1. The diagram is parsed to ensure that there are no violations of the syntax rules defined by the UML class diagram conventions. Missing values (e.g., a class name or an attribute type) are replaced with placeholder values for the second step.
2. The diagram is analyzed again from the semantic point of view, looking for elements that match those in the reference solution.
3. The diagram is inspected again, looking for any situation that is not severe enough to be considered a semantic error but that can also be classified as a pragmatic warning.

3.2.1 Syntax evaluation

The syntax evaluation is performed by analyzing each element of each type (class, attribute, association) and ensuring that all the necessary constraints are respected. An element that does not respect one constraint produces an error of a specific type (depending on the constraint) with which it is associated, producing a list of errors at the end of the parsing process. It is possible for an element to have multiple associated errors, in case it violates more than one constraint.

Algorithm 1 contains the algorithm of the syntax evaluation process, while the full list of syntax rules defining all the possible violations elements can have is presented in Appendix A.1.

```
Input  : Diagram elements
Output: List of syntax errors
Define an empty array of syntax errors synErrs;
if there are no classes in the diagram then
    Add an error to synErrs for the empty diagram;
    return synErrs;
else
    Iterate through each type of element in the diagram;
    foreach class in the diagram do
        Add an error to synErrs for each syntax rule not
        respected by the class;
    end
    foreach attribute in the diagram do
        Add an error to synErrs for each syntax rule not
        respected by the attribute;
    end
    foreach association in the diagram do
        Add an error to synErrs for each syntax rule not
        respected by the association;
    end
    foreach intermediate class in the diagram do
        Add an error to synErrs for each syntax rule not
        respected by the intermediate class;
    end
    return synErrs;
end
```

Algorithm 1: Syntax Evaluation Algorithm

In case elements are missing relevant values such as names or association multiplicities, placeholder values are inserted

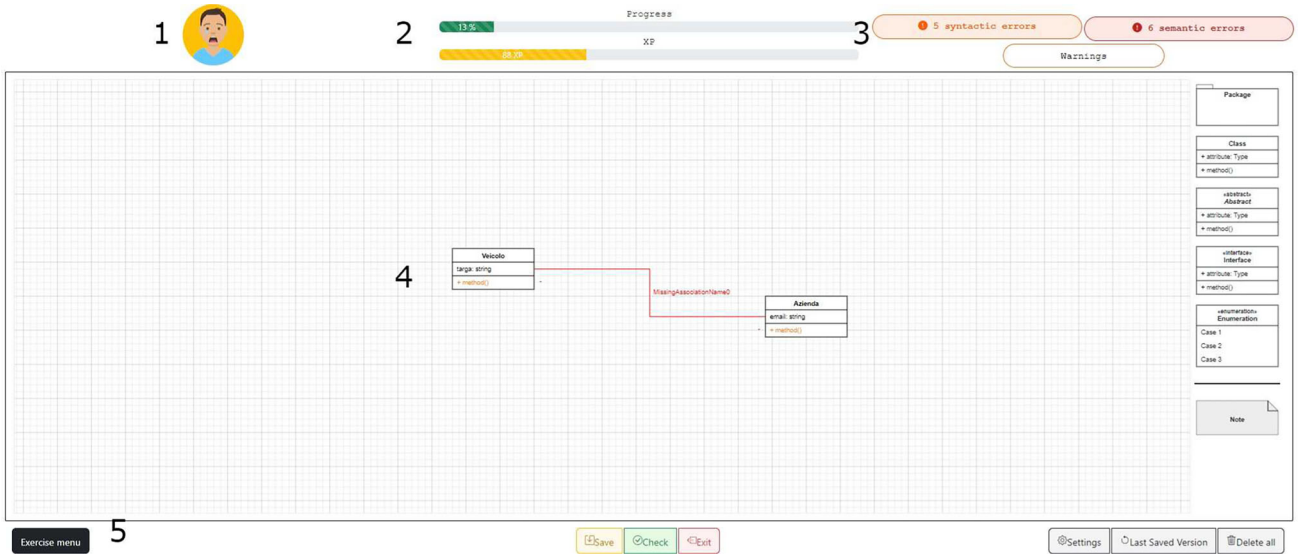


Fig. 3 Exercise page of the tool showing: 1) Student Avatar, 2) Indicators, 3) Error Lists, 4) Diagram Feedback, 5) Exercise Menu.

before performing the semantic evaluation; the elements that have placeholder values are not considered during the semantic evaluation, meaning that they do not cause an error for not matching an expected element.

3.2.2 Semantic evaluation

The semantic evaluation has a much higher complexity compared to the syntax process, mainly because semantic violations are exercise-specific and require the definition of a solution structure for each exercise.

This structure defines the elements that are expected to have an existing match in a student's diagram and expects the following:

- A list of class objects. Each class object must have an identifying name, a list of synonyms for its name, a weight that indicates whether the class is mandatory or optional, a list of names that cannot be used for its attributes, and a list of attribute objects.
- Each attribute object must have an identifying name, a list of synonyms for the name, a list of allowed types, and a value that specifies whether the attribute can also be modeled as a separate class.
- A list of association objects. Each association must have a name, a list of synonyms for the name, the names of the source and destination classes, and a list of allowed multiplicities for both the source and the destination class.
- A list of intermediate class objects. Each intermediate class object must have the name of the class that acts as intermediate, and the names of the two classes it connects.
- A list of generalization objects, with each object specifying the parent class and the list of classes that act as children.
- A list of names that cannot be used to name classes (e.g., concepts that should not be included such as an application or the device on which the software is deployed).
- A list of names that cannot be used to name associations (e.g., infinite verbs).
- A list of class pairs that cannot be connected through an association (e.g., concepts that should not interact).

The semantic evaluation identifies matching diagram elements that match the expected ones by means of string comparison with the Levenshtein distance: when looking for a solution class, for example, the name of every class is compared with the reference class name using the distance to identify the most similar one.

The distance calculation for all types of names also takes synonyms into account and is normalized to account for the different lengths of the two compared strings; an element is considered a match if its name has a distance value to the expected name (or one of its synonyms) equal or lower than 25%.

The semantic evaluation algorithm, shown in Algorithm 2, follows a similar logic to the syntax evaluation process, in that it also operates iteratively through both the diagram elements and the reference solution elements.

3.2.3 Pragmatic evaluation

The pragmatic evaluation is performed at the end of the semantic one, following the logic defined in Algorithm 3.

```
Input  : Diagram elements
Output: List of semantic errors
Define an empty array of semantic errors, semErrs;
foreach class in the reference solution do
    if there is no matching class in the diagram and the class
    weight is STRONG then
        Add an error to semErrs;
    end
end
foreach class in the diagram that matches a reference class do
    foreach attribute in the corresponding reference class do
        if the attribute is not present in the diagram class then
            Add an error to semErrs;
        end
        if the attribute is present but its type is not among the
        allowed ones then
            Add an error to semErrs;
        end
        if there is a class matching it but the attribute cannot be
        modeled as a class then
            Add an error to semErrs;
        end
    end
    foreach attribute in the diagram class do
        if the attribute name is not allowed then
            Add an error to semErrs;
        end
    end
end
foreach intermediate class in the reference solution do
    if either of the two classes it connects do not have a matching
    diagram class then
        Add an error to semErrs;
    end
    if there is no matching intermediate class in the diagram then
        Add an error to semErrs;
    end
end
foreach reference association do
    if both source and destination classes have matching
    diagram classes then
        if there is no association between the source and
        destination diagram classes then
            Add an error to semErrs;
        end
        if at least one cardinality value is not among the allowed
        values then
            Add an error to semErrs;
        end
    end
end
foreach diagram association do
    if there is no matching solution association then
        if the association name is not allowed then
            Add an error to semErrs;
        end
        if the association connects two classes that cannot have
        an association then
            Add an error to semErrs;
        end
    end
end
return semErrs;
```

Algorithm 2: Semantic Evaluation

This evaluation is less severe, as it is used to identify small inconsistencies in a diagram rather than serious errors.

These inconsistencies usually consist of minor and easily fixable errors such as the diagram missing a class that is not considered mandatory, having a class that could also be represented as an attribute of another class, or two reference classes that should be connected being both present but without an association.

3.2.4 Limitations

The current implementation of the evaluation engine presents two limitations: the first, and most relevant, is the fact that it supports only one solution structure, with its set of classes, attributes, and associations; an alternative representation of the same problem with different classes would be considered wrong by an automated evaluation but be judged correct by a human evaluator able to understand the similarities.

The second limitation comes from needing to know and define all possible synonyms and alternative ways to express the same concept for all components of a diagram: not considering a synonym for a class name, for example, may lead to multiple students using the synonym and erroneously receiving negative feedback.

3.3 Solution creation

As mentioned at the beginning of this Section, the UMLegend tool allows teachers to create reference solutions for the exercises they supply to the students. This procedure is aided by a dedicated form on the teacher web page where it is possible to define the information for the elements expected to be in the students' diagrams such as classes (name, synonyms, relevance), attributes (name, allowed type(s), synonyms, class), associations (name, connected classes, multiplicities), forbidden associations, inheritance relationships, and forbidden names for attributes and classes.

The procedure for designing a solution is also supported by an editor, identical to the one used for solving the exercises, where teachers can draw the original solution and define the basic concepts, which can then be further refined by using the form. Figure 4 shows how the modeling canvas appears when creating the reference solution for a new exercise and the form for editing the solution created in that way, displaying the classes present in the diagram and their synonyms added by using the form.

The decision to include a modeling canvas for defining a reference solution in addition to a textual form was made to facilitate the process of defining a new exercise, which can be a long and time-consuming process. We expect the process of defining a solution with UMLegend to be done according to the following steps:

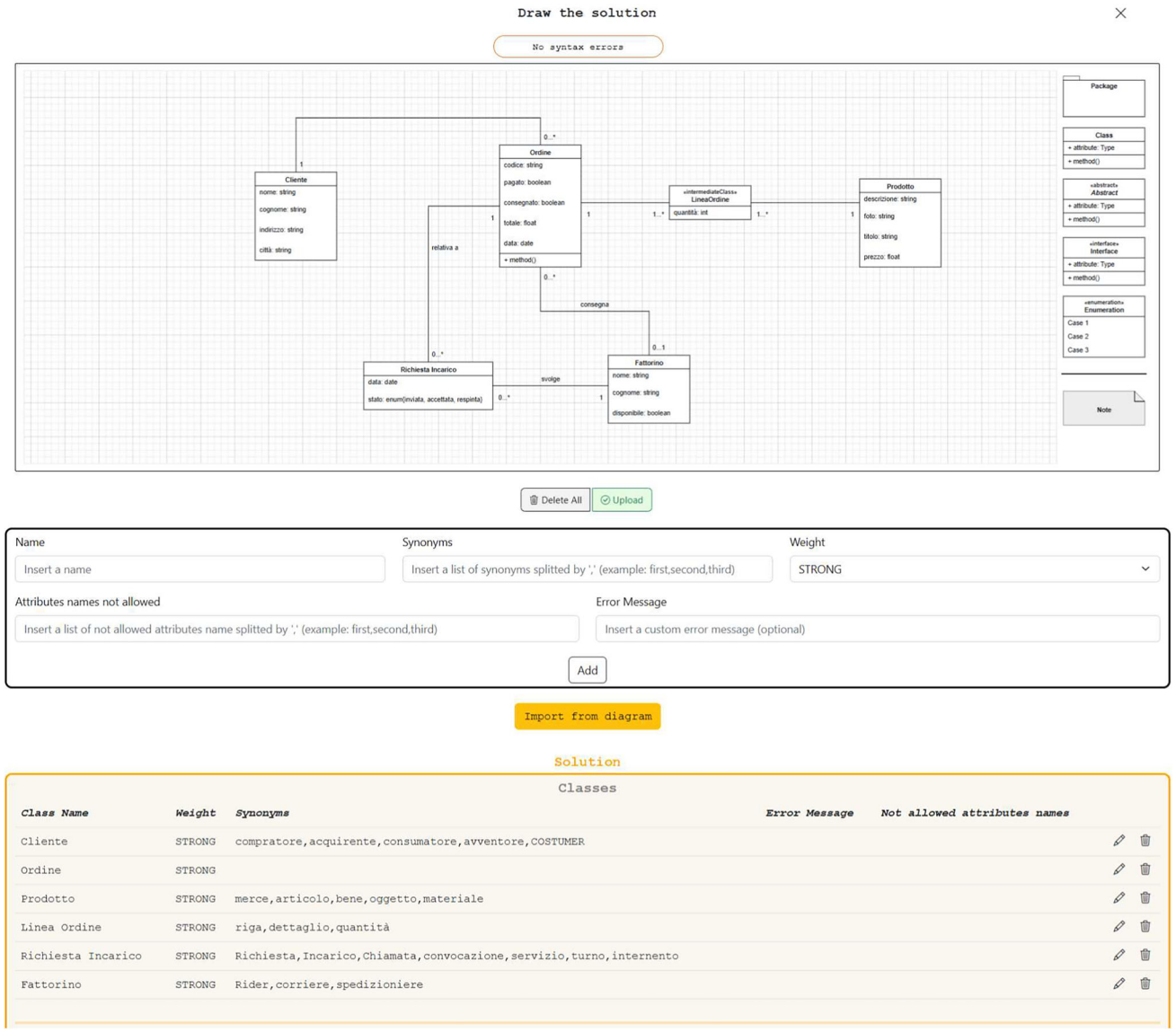


Fig. 4 Modeling canvas for the creation of a reference solution and form for editing the elements of a created solution.

1. Identify a real-world application that can serve as the base for the exercise. Write a natural language description of the application, the actors involved, the main concepts that compose the application, and their characteristics.
2. All professors involved in the definition of the exercise, who are supposed to be knowledgeable in UML modeling, draft their base solution according to the natural language description.
3. The solutions are assessed and compared by all professors together, who converge on a final solution to be used as the reference. Synonyms for the different elements (classes, attributes, associations) are defined in this phase.
4. One professor uses the form and modeling canvas to save the reference solution on the tool, including all synonyms.

The time required for completing this procedure depends on the expected difficulty of the exercise: we assume that a fairly skilled professor should take around 1 hour to create the reference solution of an average-level exercise, and 2-3 hours for harder exercises. The time needed to converge on the final solution, instead, depends on the number of professors involved and on the variability between their solutions: if the proposed solutions are highly different from one another it may take more time to reach a consensus; we assume a time window of around 2 hours to define a complete solution in case of multiple professors. The final step, entering the solution in the tool, is something that can be done in at most 30 minutes for highly complex exercises with several elements in the reference solution.

```
Input  : Diagram elements
Output: List of pragmatic warnings
Define an empty array of pragmatic warnings, pragWarns;
foreach class in the reference solution do
    if there is no matching class in the diagram and the class
    weight is WEAK then
        Add a warning to pragWarns;
    end
end
foreach class in the diagram that matches a reference class do
    foreach attribute in the corresponding reference class do
        if there is a class matching it and the attribute can also be
        modeled as a class then
            Add a warning to pragWarns;
        end
    end
end
foreach intermediate class in the reference solution do
    if a diagram class matches the intermediate class that is
    connected to the expected source and destination classes is
    not defined as an intermediate class then
        Add a warning to pragWarns;
    end
end
foreach diagram class that does not match a solution class do
    if the class name is close enough to the name of a missing
    reference class then
        Add a warning to pragWarns
    end
end
foreach diagram class that is considered unnecessary do
    Add a warning to pragWarns
end
foreach reference association do
    if source and destination reference classes both have
    matching diagram classes then
        if there is no association between the two diagram classes
        and one of the two reference classes has a WEAK weight
        then
            Add a warning to pragWarns
        end
    end
end
foreach diagram association that does not match a reference
association do
    if the association is unnecessary then
        Add a warning to pragWarns
    end
end
foreach reference generalization object do
    foreach diagram class that matches a reference child class do
        if the diagram class does not have an inheritance
        association with a diagram class that matches the
        reference parent class then
            Add a warning to pragWarns
        end
    end
end
return pragWarns
```

Algorithm 3: Pragmatic Evaluation

In terms of skill needed by a professor to be able to define a reference solution, we assume that the basic skill level required is that of novice software engineering lecturers, as we believe being able to teach about conceptual modeling to be the only necessary prerequisite to create a solution.

Currently, there is no way for teachers to compare their solutions directly in the UMLegend tool, it is only possible to produce them through the generic modeler, export them, and compare them; in the future, we envision the possibility for teachers to add their proposed solutions directly to an exercise and the ability to add annotations and suggestions to the diagrams made by others.

4 Experiment design

We conducted an experiment to investigate the impact of gamification on student productivity and correctness when performing UML modeling exercises.

The experiment also aimed at gathering the students' opinions on the gamified tool's usability and appreciation, as well as feedback for improvement.

Table 2 reports our research objectives, which we have defined following the Goal-Question-Metric template [56].

4.1 Research questions

We define the following research questions to frame the experiment design:

- **RQ1:** Does gamification improve the students' *Productivity* in modeling UML diagrams compared to non-gamified UML modeling? *We expect gamification to provide a more engaging environment which leads students to make more changes in their diagrams and have them evaluated more frequently.* We refine the question further into:
  - **RQ1.1:** Does the application of gamification impact the *Size* of the diagrams made by the students?
  - **RQ1.2:** Does the application of gamification impact the *Attempts* made by students?
- **RQ2:** Does gamification improve the *Correctness* of the diagrams modeled by the students compared to non-gamified UML modeling? *We expect gamification to improve student performance by increasing diagram completeness and reducing the errors made by the students.* We refine the question further into:
  - **RQ2.1:** Does the application of gamification impact the *Completeness* of the diagrams produced by the students?

Table 2 GQM template for the experiment

Object of study	Learning of UML modeling
Purpose	Compare the traditional and gamified approach
Focus	Productivity, correctness, student reception
Context	Modeling tools
Perspective	Teachers

  - **RQ2.2:** Does the application of gamification impact the *Errors* found in the diagrams produced by the students?

- **RQ3:** What is the students' *Perception* of gamified UML modeling? *We assume gamification is positively perceived by the students with more involvement and willingness to adopt it.* We refine the question further into:
  - **RQ3.1:** What is the perceived usability of the gamified UML modeling tool?
  - **RQ3.2:** What is the appreciation of the gamified characteristics and mechanics in the gamified UML modeling tool?
  - **RQ3.3:** What are the issues and improvement areas for the gamified UML modeling tool?

4.2 Design

We organized the study as a within-subjects 2x2 full factorial (crossover) experiment during which we administered the treatment (application of gamification) with two distinct versions: a *Vanilla* version, with no gamification features, and a *Gamified* version with the complete implementation.

We implemented the Vanilla version as a modification of the Gamified platform without the detailed feedback and the impact of the evaluation on the various mechanics (e.g., experience points, completeness bar, avatar reactions). The leaderboards also took into account only the scores obtained when using the Gamified version.

The Vanilla version included the underlying evaluation engine as well, but its feedback was less precise (e.g., it would state that at least one element was missing without explaining which one).

Feedback is considered a game mechanic in the different frameworks we have considered, particularly according to the Octalysis classification, which is what we mostly followed in the design of the game mechanics. However, we deemed it necessary to include at least a minimum amount of feedback even in the vanilla version of the tool, since our previous expertise within a similar study led to very low scores for the students if feedback was not provided at all [57]. We therefore consider basic feedback a necessary aspect of a tool, even if not prominently developed with gamification purposes, and we compare it against more detailed feedback in the fully Gamified solution, to evaluate how such details might be guiding students toward a more correct solution.

We could have decided to implement the full feedback for both versions of the tool, but that would have meant not including the feedback as a game mechanic, and the mechanic has been found to be effective in the literature.

Table 3 Experiment design

	Task 1 Exercise	Tool	Task 2 Exercise	Tool
Group 1	Exercise A	Vanilla	Exercise B	Gamified
Group 2	Exercise A	Gamified	Exercise B	Vanilla
Group 3	Exercise B	Vanilla	Exercise A	Gamified
Group 4	Exercise B	Gamified	Exercise A	Vanilla

All the participants received both treatments and performed the two exercises that made up the experiment in two consecutive tasks, with a task being a combination of exercise and treatment.

Exercise and treatment were both offered in two possible orders, leading to a total of four possible combinations, which we summarize in Table 3.

Each combination of treatment and exercise identified a group, and we randomly assigned the participants to these four groups, ensuring that all groups had the same number of participants.

The two exercises selected for the experiment consisted of two descriptions of real-life situations that were based on the use of an Information System and required students to model a UML class diagram that could adequately represent the necessary software concepts.

We selected the two exercises among the ones offered in past exams of the course: the two exercises were judged to have an average difficulty level and solvable by students at the end of the course in an adequate way.

4.3 Operationalization of variables

We present in this section the variables we have considered in our experiment to answer our research questions.

For both RQ1 and RQ2, we considered the *Treatment* as the independent variable of interest: we compared the results obtained by students when using the Vanilla version with those obtained with the Gamified version.

Two additional co-factors we have considered as possible confounding factors for our analysis are the *Exercise* and the *Group* to which we have randomly assigned the students: we have performed our statistical analysis by comparing the various metrics according to the possible values of the two factors.

We have considered two dependent variables to answer RQ1: the *Size* of the diagram, computed as the sum of classes, attributes, and associations found in a diagram, and the number of *Attempts* made by a student when performing the exercise: each separate evaluation, triggered by a student using the *Check* button of the exercise modeler, is counted as an attempt. The number of attempts is counted separately for both the Vanilla and Gamified versions, starting from 0 when a student first accesses an exercise and increasing for each check, with no limit to the number of attempts that can be made. At any attempt if a previously spotted error is corrected, the experience points are returned to the player. Otherwise, if the error is still present in the current attempt and has not been fixed, the penalty in experience points is doubled. This behavior is applied in the Vanilla version as well: the changes in experience points are not shown to the students but are still tracked internally by the tool, to provide information about a student's performance in both versions.

We expect both size and number of attempts to increase with the use of gamification. We imagine in fact that the presence of several gamified elements can be a powerful motivator for users to increase the number of modeled entities and to check the correctness of their solutions with higher frequency.

To answer RQ2.1, we have considered the *Completeness* of an exercise as the dependent variable, while we considered three separate metrics for answering RQ2.2: these three metrics relate to the Syntactic and Semantic Errors, and the Pragmatic Warnings found in the students' diagrams.

We expect an increase in completeness in the Gamified version against the Vanilla version, as well as a reduction in the three error metrics.

To answer RQ3.1 and to assess how students perceived the gamified experience we made use of the TAM [58] and GAMEX [59] questionnaires.

The technology acceptance model (TAM) is composed of questions that evaluate four high-level constructs related to users' perception of a system: perceived usefulness (PU), how much an individual believes that using the system would improve their performance; perceived ease of use (PE), how much an individual believes that using the system would require little mental and physical effort; behavioral intention to use (BI), the measure of how much an individual would be inclined to behave in a specific way; attitude toward usage (ATU), the measure of how much a user would desire using the system.

The GAMEX questionnaire is used to evaluate six different aspects of a gamified experience: enjoyment, absorption, creative thinking, activation, possible negative effects, and dominance.

To answer RQ3.2 we defined a questionnaire that asked, for each gamified mechanic, a set of five questions that aimed at assessing the students' opinions on the mechanic's perceived usefulness, influence, satisfaction, motivation to use, and provided distraction.

The questionnaires used for RQ3.1 and RQ3.2 all accepted answers in the form of a Likert scale ranging from 1 (strongly disagree) to 5 (strongly agree); we performed a qualitative assessment of the distribution of the answers to each questionnaire to identify the general student opinions on the tool usability, on the gamified experience, and on the mechanics.

Lastly, we included one open question to answer RQ3.3: students could use this question to report any issue they had encountered, as well as suggestions on how UMLegend could be improved.

Appendix B lists the full questionnaire in the same order as it was provided to the students.

4.4 Participants

The study involved 280 participants, recruited through convenience sampling among the students enrolled in a Master's level *Information Systems* course taught at Politecnico di Torino in the 2023-2024 academic year.

Participation in the experiment was optional, and to encourage students to take part in the study we awarded them two additional points to their final exam grade. Out of the 280 participants, we had 163 male and 117 female students.

4.5 Instrumentation

For the experiment, we implemented a modified version of UMLegend with two exercises and two modes of execution (Vanilla and Gamified). We provide an online Docker image that contains the tool used for the experimentation for replication purposes[3].

The experiment was conducted during lecture hours at Politecnico di Torino in January 2024: the tool was deployed on a website to make it easily accessible for the students.

The tool was also set up to automatically collect the exercise metrics and diagrams after every check performed by the students; this data was then retrieved after the experiment for our analysis. The data are available as an online resource[4].

[3] https://hub.docker.com/r/giacomogaraccione/umlegend_replication

[4] https://doi.org/10.6084/m9.figshare.25968082.v2

4.6 Hypotheses

To answer the research questions derived from RQ1, we have formulated the following null hypotheses:

- H_{s_0}: The application of gamification has no statistically significant impact on the size of the diagrams produced by the students
- H_{as_0}: The application of gamification has no statistically significant impact on the number of attempts (diagram evaluations) made by the students

The null hypotheses we have formulated for answering RQ2 and its sub-questions are:

- H_{pr_0}: The application of gamification has no statistically significant impact on the completeness of the diagrams produced by the students
- H_{syn_0}: The application of gamification has no statistically significant impact on the number of syntax errors made by students
- H_{sem_0}: The application of gamification has no statistically significant impact on the number of semantic errors made by students
- H_{prag_0}: The application of gamification has no statistically significant impact on the number of pragmatic warnings raised by students

We chose a nonparametric approach to test our hypotheses, anticipating that most variables would not follow a normal distribution. All statistical analyses were conducted using the R statistical tool [60].

The analysis approach we adopted follows the guidelines defined by Vegas et al. [61] for analyzing full factorial crossover experiments: we used a repeated measures linear mixed model (ANOVA) test which compared the metrics depending on the treatment (Vanilla or Gamified version of the tool). Exercise and group assignments have been considered as possible confounding factors that may threaten the validity of our experiment.

We also adjusted the significance level through Bonferroni correction to account for the multiple comparisons we performed in our analysis and to counteract the risk of multiple ($n = 6$) statistical tests increasing the chance of false positives (Type I errors); the p-value threshold we consider as statistically significant is $\alpha_C = \alpha/n = 0.0083$.

Regarding RQ3, which was evaluated using a questionnaire with answers that are not easily analyzable using statistical hypothesis testing, we opted instead for descriptive statistics, reporting the results using stacked diverging bar charts [62] for the Likert-scale questions related to RQ3.1 and RQ3.2.

Lastly, to answer RQ3.3, we analyzed the answers to the final open question according to the Straussian Grounded Theory approach [63], more specifically, the *open coding* approach.

With our analysis, we aimed to identify the topics in the students' answers, and then group the answers depending on their topics. We describe the process and the distribution of roles below, following the guidelines by Stol et al. [64] for applying grounded theory to software engineering.

Our open coding process consisted of analyzing each answer singularly, searching among the already identified topics for one that was compatible with the answer; if there was a compatible one then it would be assigned to the answer, while if there was no compatible topic we would define a new topic and assign it to the answer. This led to the creation of an incrementally growing set of topics.

The second step of our analysis, performed after assigning a topic to each answer and finalizing the set of topics, consisted of a second review of the answers where we assessed whether the assigned topic was still appropriate or if there was a more suitable one.

Lastly, after confirming the most suitable topic for each answer, we reviewed the entire pool of answers for a third time to identify answers that could have more than one suitable topic, leading to some answers having more than one assigned topic.

The open coding process described above was performed by one of the authors: the set of topics, and the assignment of topics to each question were then reviewed by all the authors to reach a final consensus.

4.7 Threats to validity

We discuss the potential threats to the study's validity according to the four categories defined by Wohlin et al. [65].

Threats to Internal Validity concern internal factors that may affect a dependent variable that our study did not consider.

The crossover design can introduce fatigue and learning effects due to consecutive task execution. Carryover effects are possible as the second task might be influenced by the first. Using tools with a common base may result in learning effects that impact the second task. Different exercise difficulties could yield different results, and one exercise might be easier. The students' motivation due to extra course points is not considered a threat.

As a prominent internal validity threat, we must also consider the limited amount of correct alternatives to the reference solution employed in the automated assessment. These limited alternatives may introduce a (negative) bias in the evaluation of the analyzed responses to the modeling tasks. The threat has been mitigated, as of now, by manually defining valid synonyms for each considered solution,

and by allowing basic syntax and grammar differences in the submitted exercises.

Additionally, the choice to consider the feedback a game mechanic and distinguish between generic and specific indications for the two versions of the tool may have impacted the results, reducing the effect in scores between the vanilla and gamified versions.

Threats to External Validity concern whether the study results can be generalized rather than be applicable only to the specific sample of participants involved.

The results might not generalize beyond the specific sample of participants, as students with different UML modeling skill levels could produce different outcomes. The benefits observed in the university course context may not apply to real-world modeling situations.

Threats to Construct Validity concern the extent to which the measures selected for the study represent the observed construct.

The selected metrics for answering research questions may not be the most effective. For productivity, we could have measured the time spent on each exercise, but it does not necessarily correlate with correctness. For correctness, the rules selected might not be the most effective, and different methods of measuring semantic correctness could yield different results.

Threats to Conclusion Validity concern the ability to conclude from the study results.

We used nonparametric statistical tests with minimal prerequisites, and all measures were collected automatically to avoid human errors. Random allocation of participants to different groups might have introduced confounding factors, but the distribution of skill levels makes this threat improbable, though it cannot be entirely ignored.

5 Results

5.1 RQ1-Productivity

In Table 4, we report the minimum, maximum, mean, median, and standard deviation for the metrics selected to answer RQ1, e.g., the size of the final diagram submitted by students for each exercise and the number of attempts made by the students. Metrics have been computed for both possible treatments (Vanilla tool vs Gamified tool).

Figure 5 shows combined box plots and violin plots for the distribution of values of the two metrics depending on the treatment.

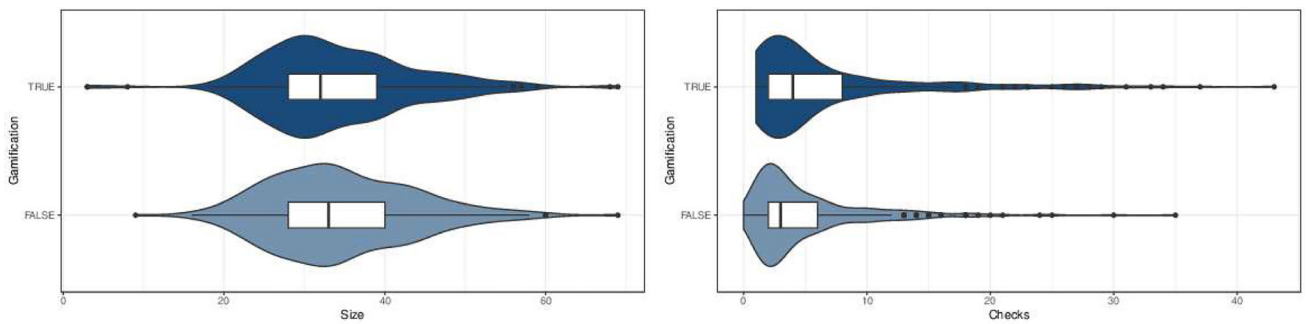
By observing the two plots, we see that the average number of submissions is higher for the Gamified version of the tool, meaning that students are more encouraged to evaluate their diagrams when there are gamification mechanisms connected to such an action; regarding the average size, however, the mean value is higher for the Vanilla version, meaning that students produce smaller diagrams when adopting gamification.

We performed an ANOVA test on the mean values of the two Productivity metrics to assess whether there is a statistically significant impact of the treatment. The results of each test, with the corresponding null hypothesis, are presented in Table 5.

We observe that the p-value for hypothesis H_{s_0} is not statistically significant, meaning that we cannot reject the null

Table 4 Maximum, minimum, average, and median scores for productivity metrics

	Min	Max	Mean (SD)	Median
Size (V)	9	64	34.61 (9.09)	33
Size (G)	3	69	34.18 (9.40)	32
Attempts (V)	0	35	5.68 (5.00)	3
Attempts (G)	1	43	6.66 (7.16)	4



(a) Box and violin plot for the distribution of the Size metric (b) Box and violin plot for the distribution of the Attempts metric

Fig. 5 Box and violin plots for productivity metrics.

Table 5 Null hypotheses for RQ1 and p-values of the applied ANOVA univariate tests ($\alpha = 0.0083$)

Hypothesis	p-value	Decision
H_{s_0}	0.577	accept
H_{as_0}	2.45e-03	reject

hypothesis: applying gamification to UML modeling does not affect the size of the diagrams produced by the students.

The p-value related to the number of attempts, however, is statistically significant: the higher mean number for the metric associated with the Gamified version of the tool means that adopting gamification encourages students to evaluate their solutions more frequently; we can thus reject hypothesis H_{as_0} and assert that gamification has a statistically significant positive impact on the number of attempts made by the students, leading to an increased commitment.

The only statistically significant result that comes from analyzing the confounding factors is the Exercise affecting the diagram size (p-value = $<$ **2e-16**), with Exercise B having a much lower average size compared to the other (30.23 against 38.57); we calculated the sizes of the two reference solutions and found a disparity between the two sizes, with Exercise A having a total of 43 elements and Exercise B a total of 32.

The difference in the average sizes obtained by the students is in line with the difference between the sizes of the two reference solutions; however, the fact that one exercise requires a significantly higher number of elements in comparison with the other may imply that the two exercises, whose difficulty we expected to be similar, may have different difficulty levels, with Exercise B being easier by requiring fewer elements.

Summary (*RQ1*): The statistical analysis found that gamification has no statistically significant impact on the size of the diagrams made by the students. There is, however, a statistically significant increase in the number of attempts made by the students with the gamified version, meaning that they tend to evaluate their solutions more frequently. It can be assumed that gamification increases student commitment. The statistically significant impact of the Exercise variable on diagram size may imply that one exercise was easier than the other, and this may affect the validity of our conclusions.

5.2 RQ2–Correctness

In Table 6, we report the minimum, maximum, mean, median, and standard deviation for the metrics selected to answer RQ2, that is, the Completeness obtained by the students when performing the two exercises and the errors made by the students, divided into three separate categories (syntax, semantic, and pragmatic). Similarly to how it was done for RQ1, metrics are computed depending on the different treatments.

Table 6 Maximum, minimum, average, and median scores for correctness metrics

	Min	Max	Mean (SD)	Median
Completeness (V)	0	98	42.54 (13.09)	44
Completeness (G)	9	95	47.21 (13.18)	46
Syntax errors (V)	0	31	5.4 (6.48)	3
Syntax errors (G)	0	32	3.79 (5.51)	1
Semantic errors (V)	2	18	9.64 (2.91)	9
Semantic errors (G)	2	18	8.71 (2.69)	8
Pragmatic warnings (V)	0	8	2.05 (1.54)	2
Pragmatic warnings (G)	0	10	2.20 (1.61)	2

Figure 6 shows combined box plots and violin plots for the distribution of values of the two metrics depending on the treatment.

At first glance, the distribution of metrics displays better results for what concerns the Gamified version of the tool: the average completeness is around 5% higher compared to the Vanilla version, indicating a better student performance, on average.

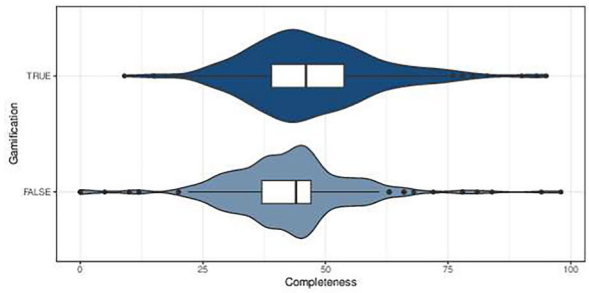
The distribution of errors shows similar results: the average number of both syntax and semantic errors is higher for the Vanilla version, meaning that the use of gamification helped students make fewer errors; the same cannot be said for pragmatic warnings, which show a higher mean number for the Gamified version.

We performed an ANOVA test on the mean values of the four Correctness metrics to assess whether there is a statistically significant impact of the treatment. The results of each test, with the corresponding null hypothesis, are presented in Table 7.

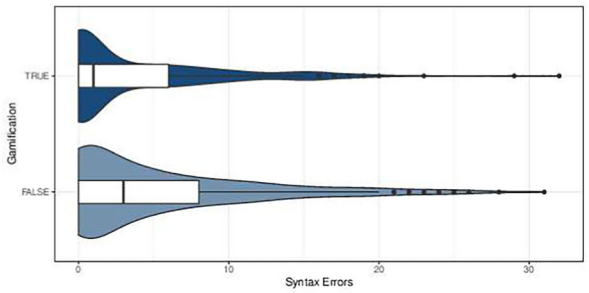
We observe a statistically significant effect of gamification on the average completeness obtained by the students: this result, combined with the higher mean score obtained with the Gamified version of UMLegend, implies that gamification has a positive statistically significant effect on the exercise completeness, meaning that we can reject the null hypothesis H_{pr_0}.

The ANOVA test on syntax errors yielded a significant p-value, meaning that gamification has a statistically significant impact on the metric; the lower mean value for the Gamified version implies that this effect is positive and that gamification can reduce the number of syntax errors made by students. We can thus reject H_{syn_0}.

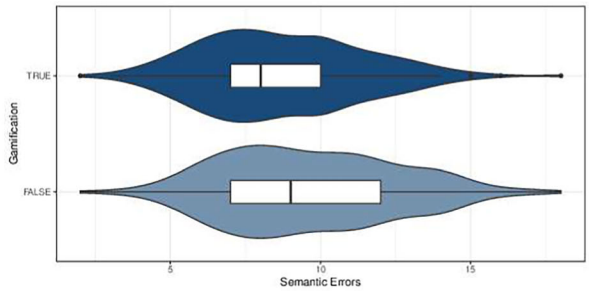
A statistically significant p-value can also be observed when comparing semantic errors between the two versions: the fact that semantic errors made with the Gamified version are, on average, fewer than the ones made with the Vanilla



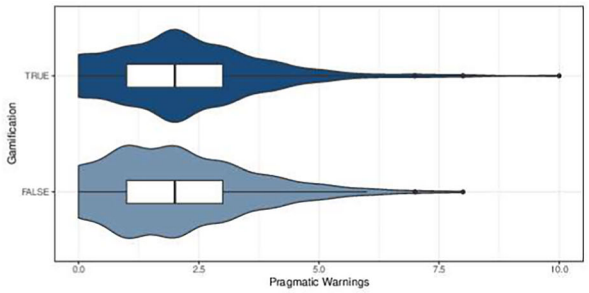
(a) Box and violin plot for the distribution of the Completeness metric



(b) Box and violin plot for the distribution of the Syntax Errors metric



(c) Box and violin plot for the distribution of the Semantic Errors metric



(d) Box and violin plot for the distribution of the Pragmatic Warnings metric

Fig. 6 Box and violin plots for the four Correctness metrics.

version means that gamification has a positive statistically significant impact, and hypothesis H_{sem_0} can be rejected.

No significant effect was found on the pragmatic warnings obtained by the students, meaning that it is not possible to reject H_{prag_0}.

For what concerns confounding factors, we observe that the resulting p-values are statistically significant for all dependent variables when considering the Exercise as independent variable, with p-values being $<$ **2e-16**, **2.76e-06**, $<$ **2e-16**, and **6.82e-10** for Completeness, Syntax Errors, Semantic Errors and Pragmatic Warnings, respectively.

The average completeness is around 10% higher for Exercise B compared to Exercise A, while syntax, semantic, and pragmatic errors are, on average, lower for Exercise B. This reinforces the idea that Exercise B might be easier than Exercise A, leading to a possible influence on the results of our experiment.

None of the metrics have obtained a significant p-value for what concerns the Group, so we can assert that the treatment combination had no impact on the student performance and that any learning effect had no significant effect.

Summary (*RQ2*): The statistical analysis identified an increase in exercise completeness with the Gamified version of the tool, compared to non-gamified UML modeling. The usage of gamification also reduced the number of syntax and semantic errors made by students, implying an overall positive effect of gamification on the correctness of student diagrams. The Exercise had a statistically significant impact on all four dependent variables, with Exercise B obtaining better performance, on average. This further reinforces the hypothesis that the two exercises had different difficulties, possibly impacting the validity of our findings.

5.3 RQ3-Student perception of the gamified experience

5.3.1 RQ3.1-Perceived usability

We evaluated the perceived usability of the UMLegend tool with the TAM questionnaire and the general perception of the gamified experience with the GAMEX questionnaire.

The analysis of the answers given by the students was done by computing the mean value of the answers that made up each questionnaire construct for each student; this mean value was then considered as the general opinion of the student for the construct.

Table 7 Null hypotheses for RQ2 and p-values of the applied ANOVA univariate tests ($\alpha = 0.0083$)

Hypothesis	p-value	Decision
H_{pr_0}	2.96e-05	**reject**
H_{syn_0}	1.62e-03	**reject**
H_{sem_0}	9.80e-05	**reject**
H_{prag_0}	2.59e-01	accept

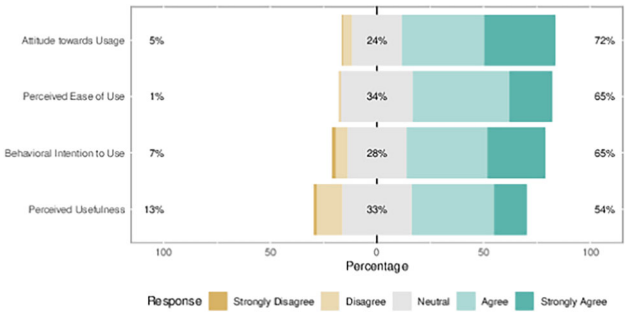


Fig. 7 Distribution of the answers to the TAM questionnaire.

We present the distribution of answers to the TAM questionnaire in Fig. 7: the distribution of opinions is encouraging, with at least half of the participants answering at least *Agree* to all four constructs.

Moreover, negative opinions seem to be uncommon: the *Perceived Usefulness* is the one with the highest negative percentage, and the remaining three constructs have negative answers in the range 1–7%.

The high amount of positive answers, and the low distribution of negative ones, let us affirm that the UMLegend tool has been perceived successfully from the usability point of view.

The answers to the GAMEX questionnaire, which we report in Fig. 8, are more neutral: four out of the six constructs have answers with positive answers ranging from 14 to 24%, with negative answers being more common.

The only construct with a high amount of positive answers is *Enjoyment* with a 49% positive distribution: this can be interpreted as UMLegend being perceived as a fun and interesting tool to use for modeling; additionally, the high amount of negative answers for *Possible Negative Effects* is a positive result, as it means that a small percentage of students (6%) perceived the tool as frustrating or detrimental.

The remaining four constructs highlight the perceived limitations of the gamified experience: freedom of action and creative thinking are not completely supported, and students do not tend to feel completely immersed in the activity.

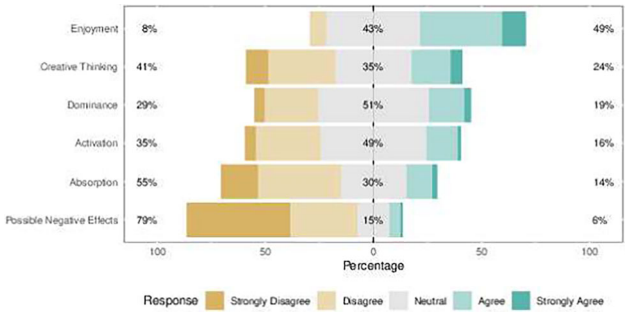


Fig. 8 Distribution of the answers to the GAMEX questionnaire.

Summary (*RQ3.1*): The answers given to the TAM questionnaire show an overall positive appreciation of UMLegend: all four constructs have obtained positive answers by at least half of the participants, and negative answers are quite rare, with 13% being the highest number; the Perceived Ease of Use is particularly encouraging, with only 1% of the participants finding the tool hard to use. The answers given to the GAMEX questionnaire, instead, are more toward the neutral side, showing that the gamification side of the UMLegend tool still needs improvements. The high number of positive answers to *Enjoyment*, and of negative answers to *Possible Negative Effects*, however, imply that the experience, while not complete, is perceived as fun, not frustrating, and enjoyable.

5.3.2 RQ3.2-Appreciation of gamified mechanics

To assess the appreciation of gamified mechanics, we have defined a specific questionnaire that focuses, for each mechanic, on five separate aspects: perceived usefulness, influence, satisfaction, motivation to use, and provided distractions.

We computed, for each mechanic, the mean value of the answers given by each student to obtain the distribution of answers and assess the general feeling toward the mechanics; however, we considered the opposite of the score assigned to the *provided distraction* question when computing the average score (e.g., in case of a *Strongly Disagree* answer, we considered a *Strongly Agree* value instead, to offset the negative connotation of the question).

The distribution of answers for the gamified mechanics is reported in Fig. 9: we observe that the three most appreciated mechanics, all with at least 50% positive answers and less than 10% negative answers, are those that are most helpful when performing the exercise (Error Lists, Indicators, and Diagram Feedback).

The remaining mechanics also show positive distributions, with the least appreciated one being the *Exercise Leaderboard* with 34% positive answers: we can deduce that such a

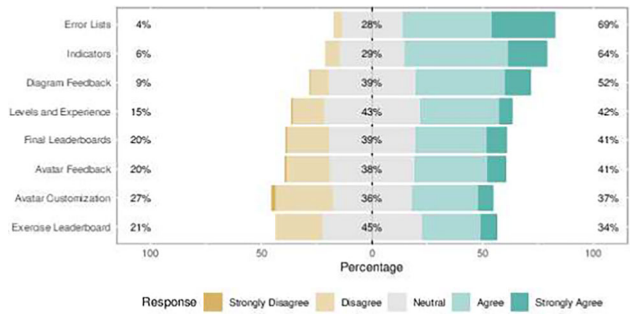


Fig. 9 Distribution of answers related to the different game mechanics.

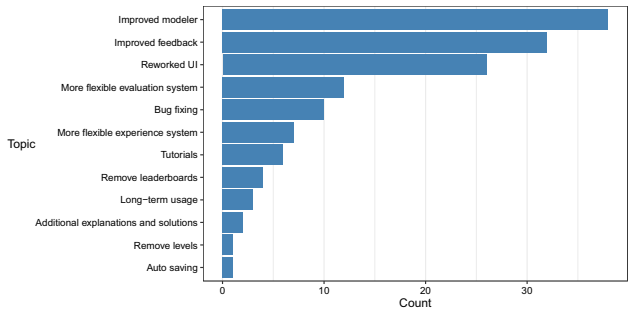


Fig. 10 Topics found in the open-ended question after performing open coding on the answers.

feature was not considered too relevant by the students, and that offering a live view of their standing does not provide additional benefits.

Lastly, *Avatar Customization*, the second least appreciated mechanic, is also the one with the highest percentage of negative answers (27%): this result may be connected with the fact that the experiment was conducted in a single session, and also the fact that some avatar props need to be unlocked by leveling up, which not all students managed to do. We have imagined avatar customization as more of a long-term feature, so it makes sense that it was not appreciated in the context of a single session. In future work, we might as well investigate the possibility that the presence of cartoon avatars—even if widespread in related works in the literature—might make the students consider the learning activity trivial or even childish.

Summary (*RQ3.2*): The distribution of answers related to the game mechanics shows that students appreciate the most the mechanics that guide them toward successful exercise completion by providing clear and direct feedback on what they need to improve. The mechanics conceived to be used longitudinally have shown to be less appreciated in a single session.

5.3.3 RQ3.3—Issues and improvement areas

We present in Fig. 10 the results of the open coding we performed on the answers given to the open-ended question asking for the students' opinions. Out of the 280 total answers, we identified 142 answers that did not contain any meaningful issue or suggestion; these answers were thus not considered during the analysis, leading to a total of 138 considered answers.

The most common topic found in the open question (38 answers) pertains to the need for an improved UML modeler: the modeler was implemented with the open-source Apollon library, which is constantly being updated[5]; the implementation lacked relevant features that students were used to such as a detailed class editor with which they could easily specify attributes and methods, and association management was also thought of as complex and unfriendly.

Other common topics include improving the feedback mechanism to explain errors more in detail and a general rework of the user interface to have the exercise text visible together with the modeler, or having the possibility to select text from the exercise and create classes directly from the selection.

Less common topics that are worth mentioning include the need to rework the experience assignment, either by reducing the penalty for errors or by having errors reduce experience only the first time they are made, and a change in the evaluation engine so that there is more variance in the allowed solutions.

Summary (*RQ3.3*): The open coding analysis showed that UMLegend, while appreciated in its general usability, needs improvements in its user interface and, most importantly, in the way the modeler is implemented, as the current implementation is perceived as unfriendly and hard to manage. Other relevant topics that emerge are the need for an improved evaluation system that offers more correct alternatives, the need for more detailed and precise feedback, and a rework of the experience system so that it is more forgiving and less frustrating.

6 Discussion

For this study, we assessed whether using gamification can bring benefits in terms of student commitment in UML modeling, and measured several key takeaways regarding the size of the generated diagrams, the number of attempts performed by the students, the correctness—in terms of syntactic, semantic and pragmatic errors.

[5] https://github.com/ls1intum/Apollon_standalone, last accessed on 31/05/2024

6.1 Key takeaways of the study

Regarding diagram size, we found that gamification had no statistically significant impact, meaning that there was no benefit brought by the game mechanics on that front: the detailed feedback and the presence of rewarding elements for correctly modeled elements were not enough to bring significant changes in terms of elements added or removed.

There was, however, a statistically significant impact on the number of attempts as students performed, on average, more checks when adopting gamification: we assume that this came as a consequence of the two indicators (exercise completeness and experience rewards) being directly connected to submitting a correct diagram; the presence of mechanics that change after new submissions leads to increased student commitment.

We evaluated correctness by considering four dependent variables: the exercise completeness, computed as the number of correct elements in a student's diagram over the total number of expected elements, the number of syntax and semantic errors found in a diagram, and the number of pragmatic warnings raised.

The results of the ANOVA test applied to exercise completeness showed a statistically significant increase when applying gamification: we consider this result a beneficial effect of the various mechanics on student performance, as they managed to produce more complete diagrams thanks to the assistance of detailed feedback, indicators, and visible effects of their changes such as avatar feedback.

A statistically significant effect was also found for both syntax and semantic errors: for both, the gamified version led to fewer errors, suggesting that detailed error explanations and mechanics tied to making and correcting errors lead to better performance.

Regarding pragmatic warnings, we found that there was no statistically significant difference between the vanilla and gamified approaches: the fact that pragmatic warnings are not considered errors, but instead suggestions on how to better model certain concepts, may have influenced this result, as students may have felt less compelled to fix their pragmatic warnings, as there was no benefit in doing so.

We considered, as possible confounding factors for the analysis, the two exercises solved by the students and the combination of exercise and treatment, which translated to the group in which students were randomly assigned.

The analysis based on the exercise as an independent variable showed significant differences in diagram size, exercise completeness, syntax and semantic errors, and pragmatic warnings: Exercise B obtained, on average, a smaller diagram size; we assume that an average smaller size may imply that the two exercises, which we assumed had a similar level of difficulty, actually had different levels, and this may influence the conclusions we can draw.

This assumption is further reinforced by the analysis made on the four Correctness metrics: the mean values showed a generally better performance obtained on Exercise B, with average completeness being higher and the three error metrics being lower for said exercise; the lower difficulty of Exercise B appears evident by observing the generally better results obtained by the students.

The cross-factorial structure of the experiment does counterbalance this disparity in difficulty: as students were randomly assigned to one out of four combinations, we did not have a situation where the easier exercise was performed with just one treatment, balancing the effect of the Exercise.

The analysis focused on the group showed no statistically significant impact on any of the six metrics we analyzed, which means that there was no learning bias, as well as no fatigue effect influencing the results.

We gauged the students' perception of the gamified experience by analyzing three constructs: the perceived usability of UMLegend, the appreciation of the gamified mechanics, and the issues and improvement areas identified in the students' opinions.

For the perceived usability, we adopted the TAM and GAMEX questionnaires and analyzed the answers by considering the mean answers for each questionnaire construct.

The answers to the TAM questionnaire were overall positive, as at least half of the students agreed on the four constructs. Perceived ease of use and Attitude toward usage were the most appreciated constructs, showing that the tool was well received by the students, with a high perceived usability.

The answers given to the GAMEX questionnaire, however, were more toward the neutral side: a high amount of the participants agreed that using the tool was enjoyable and free of negative effects, but other constructs did not achieve similar results. The UMLegend tool was generally perceived as something that does not offer full freedom of action, is not immersive, and does not allow enough creative thinking.

To assess the appreciation of the gamified mechanics, we defined a questionnaire that asked, for each mechanic, the students' opinion on its perceived usefulness, influence, satisfaction, motivation to use, and provided distraction. The most appreciated mechanics were those that offered direct feedback after the students' actions and helped guide them toward improving their execution of the exercises.

The mechanics that were imagined for long-term usage, as a consequence of the experiment consisting of a single session, were not appreciated in the same way; we expect that future longitudinal use of UMLegend would see an increased appreciation of leaderboards, levels, and avatar customization.

Lastly, the students' opinions showed a need to improve the user interface of the tool and, most importantly, to change the class modeler itself to make it easier to use and in line with

standard editors: changes on how classes are represented and edited, as well as the option to select text from the exercise and directly create classes were among the most common requests.

Other topics of interest come from how the evaluation engine is implemented: currently, only a single reference solution is allowed, with enough variation in the synonyms, but different combinations of classes and attributes may be possible for the same problem, and this issue was raised by students. Improvements in the feedback, more forgiving experience management, and changes in the mechanics were also requested, although not as commonly.

Overall, we deem that the described results suggest that gamification is beneficial in the context of UML class diagrams and motivation. The students' positive answers to the GAMEX questionnaire testify to a favorable opinion of the gamified approach, an increased motivation, and a higher commitment to completing exercises and diagrams. This increased motivation is paired with improvements in semantic quality, which suggests a better adherence to modeling standards for UML class diagrams. The educational improvements are paired, in the context of the described gamified tool, with an automated evaluation system capable of easing the workload for instructors, by handling diagram validation and providing detailed feedback for the learners.

6.2 Current limitations and research directions

In the current version of the tool, the generation of the exercises (both the textual requirements and the reference solution) is performed as a completely manual task by the instructors. This aspect limits a possible large-scale utilization of the tool for two reasons: (i) the generation of new exercises requires the manual definition of requirements; and (ii) at the present time, only a single reference solution is considered correct when the students' responses are analyzed. The definition of the solutions is a task that has to be carried out by the instructor as described in Section 3.3 of the present manuscript. This necessary manual intervention can be seen as an important limitation for the scalability of the tool, especially if it is required to generate multiple diverse exercises to provide variability among different (sets of) students in the same class.

In future versions of the tool, we intend to explore the possibility of including an automated mutation-based approach (like those discussed by Gomez-Abajo et al. [52] or by Foss et al. [53]) for the generation of variations of the exercises, and for the generation of additional acceptable solutions. It is worth underlining, however, that the majority of research effort in mutating class diagrams is aimed at generating *incorrect* variations starting from a correct diagram, and not *equivalent* ones (as in [53]). The identification and generation of diagram equivalents can therefore be considered a promising research alley in software modeling gamification.

AI-based systems, like large language models, can also be considered as a valid source for the generation of alternatives to the exercises manually defined by the instructors. The inclusion of an automated generation of variants would constitute a significant improvement because it would increase the efficiency in making new exercises available for students, and the fairness in evaluating their responses. Both aspects would make the tool more usable for educators and more dependable in a context where the students are numerous and the automated assessment is taken as a component of the students' grades (e.g., in the context of a final exam of a modeling course). Finally, word and sentence-embedding approaches can be used to provide a more dependable evaluation of the similarity of a provided solution to the reference one, which is by now limited to exact text comparisons. LLMs have shown to be particularly effective at semantical analysis [66], so we imagine that integrating them with the tool should vastly improve its overall performance.

A possible contribution that can be performed by researchers in the field is the establishment of an open-source repository of software engineering exercises—not necessarily related to class diagram modeling. Such a repository would be extremely valuable to the community since—to date—it is considerably difficult to find in textbooks and related literature sets of exercises covering all the aspects of class diagram modeling, or multiple variations of exercises with different semantic requirements but with comparable difficulty.

Finally, researchers in the field of gamification of software modeling may build upon the results of the present study—and similar ones—to explore additional gamification mechanics that are less covered in the related literature (such as social influence, which is typically more covered in serious games rather in gamified solutions, and scarcity and avoidance, that are typically neglected in favor of white-hat gamification mechanics), and/or conduct experiments in different contexts than a controlled academic setting with graduate students.

7 Conclusion and future work

In this study, we have investigated whether gamification can be effectively used as a way to improve student performance in UML class modeling. For this purpose, we developed UMLegend, a gamified modeling tool with diagram evaluation capabilities, and performed an experiment with students of a Masters' Degree Information Systems course; the experiment aimed to assess whether gamification can lead to improvements in the quality of the produced diagrams, as well as to gauge the reception of a gamified modeling tool for future classroom usage.

The experiment showed that gamification leads to increased student commitment, as students felt more inclined to use the evaluation functionality of the tool with the gamified version of the tool, which tied successful modeling with in-game rewards and offered direct feedback on the errors.

Moreover, adopting gamification also led to an increase in the correctness of the produced diagrams, with the gamified version having, on average, a higher completeness percentage (number of correct elements over the total expected ones) and fewer syntax and semantic errors.

The students' perception of the experience was, overall, positive: UMLegend was seen as a highly usable and enjoyable tool, while also being judged as not frustrating to use; the gamified experience, however, was not seen as particularly complete, as students did not feel fully in charge and able to express their creativity completely.

The gamified mechanics were well received, with the ones directly tied to successful exercise completion and giving direct feedback being the most appreciated; mechanics conceived for long-term usage were not appreciated, due to the experiment consisting of a single session.

The students' opinions showed that UMLegend needs important changes in its modeling interface, which was considered unfriendly and detrimental to the overall enjoyable experience, in its experience management system, considered at times to be excessively unforgiving, and in its evaluation engine, limited to a single reference solution.

Future plans regarding UMLegend involve:

- Reworking the evaluation engine so that multiple reference solutions can be allowed, instead of just a single one, with an extended degree of variations in classes, attributes, and alternative modeling choices. Allowing multiple semantical interpretations will make the experience less frustrating for students.
- Implementing interactions with large language model agents for the definition of reference solutions, the evaluation of diagrams through text analysis, and the creation of expanded diagram-specific feedback.
- Improving the current implementation of the user interface, following the most common comments: a friendlier modeler, as well as the ability to act directly on the exercise text, should make the modeling less frustrating and more engaging for the students.
- Applying UMLegend for the entire length of the new edition of the course, to assess the reception of the long-term mechanics, as well as to identify whether gamification applied through an entire course can improve the students' modeling capabilities. The development of similar tools for the other course topics (process modeling, use case diagrams, use case narratives, cost estimation) is something we have also considered, as we imagine that the gamification of an entire course should yield benefits to commitment and performance.

Appendix A Evaluation criteria

A. 1 Syntax rules

A. 1.1 Class constraints

- A class must have a non-empty name
- A class must have a unique name, two classes cannot have a duplicate name
- A class must have at least one valid attribute
- A class must have at least one valid association

A. 1.2 Attribute constraints

- An attribute must have a non-empty name
- An attribute must be expressed in an explicit format where its name is followed by a colon, and then by its type (*name: type*)
- An attribute cannot have a type that is not among the allowed ones (int, float, number, double, string, boolean, date, time, datetime, integer, latlong, currency, enum)
- An attribute cannot represent a foreign key of another class (e.g., the identifier of another class, or the name of another class in plural form)

A. 1.3 Association constraints

- An association must have a non-empty name
- An association must have its multiplicity values specified on both ends
- An association cannot have as multiplicity values that are not numbers, or *star* to indicate multiple and unspecified values
- A recursive association must have its roles specified on both ends

A. 1.4 Intermediate class constraints

- An intermediate class must have exactly two associations
- An intermediate class must have 1 as multiplicity values toward the two classes it connects
- An intermediate class must have its multiplicity values specified on both ends of both of its associations
- An intermediate class cannot have as multiplicity values from the two classes it connects values that are not numbers, or *star* to indicate multiple and unspecified values

A. 2 Semantic rules

A. 2.1 Class constraints

- A reference class whose weight is STRONG must have a matching class in the diagram
- A diagram class cannot have a name that is not allowed

A. 2.2 Attribute constraints

The following attribute constraints are verified if and only if there is a diagram class that match the diagram class that contains the specific attribute.

- An attribute in a reference class that has a matching diagram class must have a matching attribute in the same class
- An attribute in a diagram class that matches an attribute in the corresponding reference class must have a type that is among the allowed ones
- It is not possible to have a diagram class that matches an attribute of a reference class that cannot be represented as a class
- A diagram attribute cannot have a name that is not allowed

A. 2.3 Association constraints

The following association constraints are verified if and only if there are diagram classes that match the source and destination classes of the reference association:

- A reference association must have a matching diagram association between two diagram classes that both have a STRONG weight
- A diagram association that matches a reference association must have its multiplicity values be among the allowed values in the reference association, for both source and destination

The following association constraints are applied to all diagram associations that do not match a solution association:

- A diagram association cannot have a name that is not allowed
- A diagram association cannot connect two classes that are not allowed to have an association

A. 2.4 Intermediate class constraints

- Both classes connected by a reference intermediate class must have a matching diagram class
- The diagram classes that match two classes connected by a reference intermediate class must be connected by an intermediate class
- Each diagram intermediate classes must match a reference intermediate class

Appendix B Questionnaire

B. 1 TAM questionnaire

B. 1.1 Perceived usefulness

- Using UMLegend for conceptual modeling would enable me to accomplish tasks more quickly
- Using UMLegend for conceptual modeling would improve my performance
- Using UMLegend for conceptual modeling would improve my productivity
- Using UMLegend for conceptual modeling would make it easier to perform modeling tasks
- Using UMLegend for conceptual modeling would be useful

B. 1.2 Perceived ease of use

- Learning to operate UMLegend was very easy for me
- I found it easy to get UMLegend to do what I wanted it to do
- My interactions with UMLegend were clear and understandable
- I found that the gamified interface elements were clear and understandable
- It would be easy for me to become skillful at using UMLegend
- I generally found UMLegend easy to use

B. 1.3 Behavioral intention to use

- If I had to perform conceptual modeling in the future, I would consider using gamified techniques
- If I had to perform conceptual modeling in the future, I would use gamified techniques if I had access to them

B. 1.4 Attitude toward usage

- I believe it is a good idea to use a gamified tool for conceptual modeling
- I like the idea of using a conceptual tool for process modeling

B. 2 GAMEX questionnaire

B. 2.1 Enjoyment

- Playing UMLegend was fun
- I liked playing UMLegend
- I enjoyed playing UMLegend very much
- My experience with UMLegend was pleasurable
- I think playing UMLegend is very entertaining
- I would use UMLegend for its own sake, not only when being asked to

B. 2.2 Absorption

- Playing UMLegend made me forget where I was
- I forgot about my immediate surroundings while I played UMLegend
- After playing UMLegend, I felt like coming back to the "real world" after a journey
- Playing UMLegend "got me away from it all."
- While playing UMLegend I was completely oblivious to everything around me.
- While playing UMLegend I lost track of time.

B. 2.3 Creative thinking

- Playing UMLegend sparked my imagination
- While playing UMLegend I felt creative
- While playing UMLegend I felt that I could explore things
- While playing UMLegend I felt adventurous

B. 2.4 Activation

- While playing UMLegend I felt activated
- While playing UMLegend I felt nervous
- While playing UMLegend I felt frenetic
- While playing UMLegend I felt excited

B. 2.5 Possible negative effects

- While playing UMLegend I felt upset
- While playing UMLegend I felt hostile
- While playing UMLegend I felt frustrated

B. 2.6 Dominance

- While playing UMLegend I had the feeling of being in charge
- While playing UMLegend I felt influential
- While playing UMLegend I felt autonomous
- While playing UMLegend I felt confident

B. 3 Game mechanics questions

B. 3.1 Levels and experience

- I think that Levels and Experience were useful features
- I think that Levels and Experience were influential features in my usage of UMLegend
- I am satisfied with the use I made of the Levels and Experience features
- Levels and Experience motivated me in using UMLegend more
- Levels and Experience distracted me during my usage of UMLegend

B. 3.2 Avatar customization

- I think that the Avatar Customization was a useful feature
- I think that the Avatar Customization was an influential feature in my usage of UMLegend
- I am satisfied with the use I made of the Avatar Customization feature
- Avatar Customization motivated me in using UMLegend more
- Avatar Customization distracted me during my usage of UMLegend

B. 3.3 Final leaderboards

- I think that the Final Leaderboards were a useful feature
- I think that the Final Leaderboards were an influential feature in my usage of UMLegend
- I am satisfied with the use I made of the Final Leaderboards feature
- The Final Leaderboards motivated me in using UMLegend more
- The Final Leaderboards distracted me during my usage of UMLegend

B. 3.4 Exercise leaderboard

- I think that the Exercise Leaderboard was a useful feature
- I think that the Exercise Leaderboard was an influential feature in my usage of UMLegend
- I am satisfied with the use I made of the Exercise Leaderboard feature
- The Exercise Leaderboard motivated me in using UMLegend more
- The Exercise Leaderboard distracted me during my usage of UMLegend

B. 3.5 Avatar feedback

- I think that Avatar Feedback was a useful feature
- I think that Avatar Feedback was an influential feature in my usage of UMLegend
- I am satisfied with the use I made of the Avatar Feedback feature
- Avatar Feedback motivated me in using UMLegend more
- Avatar Feedback distracted me during my usage of UMLegend

B. 3.6 Indicators

- I think that Completeness and Experience Indicators were a useful feature
- I think that Completeness and Experience Indicators were influential features in my usage of UMLegend
- I am satisfied with the use I made of the Completeness and Experience Indicators features
- Completeness and Experience Indicators motivated me in using UMLegend more
- Completeness and Experience Indicators distracted me during my usage of UMLegend

B. 3.7 Error lists

- I think that Error Lists were a useful feature
- I think that Error Lists were influential features in my usage of UMLegend
- I am satisfied with the use I made of the Error Lists features
- Error Lists motivated me in using UMLegend more
- Error Lists distracted me during my usage of UMLegend

B. 3.8 Diagram feedback

- I think that Diagram Feedback was a useful feature
- I think that Diagram Feedback was an influential feature in my usage of UMLegend
- I am satisfied with the use I made of the Diagram Feedback features
- Diagram Feedback motivated me in using UMLegend more
- Diagram Feedback distracted me during my usage of UMLegend

B. 4 Open question

Did you encounter any issue when using UMLegend? Do you have suggestions on how we could improve the tool?

Appendix C Exercises

C. 1 Exercise A

A convenience store wants to provide a delivery service to its customers. To do so, it sets up a distributed system based on mobile applications and web interface. The system can be used by the customer (to order products, via mobile app), by the deliveryman (to deliver the goods, via mobile app), and by the manager (to upload available products and quantities, via web interface).

It is assumed that each customer has an account and is logged into the system. In addition to personal data (first name, last name, tax code), the customer's address is known in text form and as latitude and longitude.

Next he will receive a list of products with photograph, title and price. By clicking on the product he will be able to view the detail and place it in the cart indicating the quantities. Payment will be made exclusively via cash or credit card with the rider. Once the order is completed it will be possible to proceed to checkout. At that point the customer's app will search for an available delivery driver. Once the deliveryman has accepted the assignment and left the convenience store, the user will receive a notification and from that point on the customer can enter the delivery details (e.g., intercom, phone number, etc.).

After delivery, the user can rate the service (goods quality, delivery speed, deliveryman courtesy, commentary). Ratings can be assigned on a numerical scale from 1 to 5 (inclusive). At any time the user can view a list of all past orders.

It is assumed that each deliveryman has an account and is logged into the system. The deliveryman, at any time, sets his status (available for deliveries, or unavailable). If available it can be selected by the manager.

When a user completes an order an available deliveryman will be notified for delivery and can accept the assignment or decline it. Once accepted, he will scan the delivery barcode (or, in case of problems, enter it manually) and display the route to the customer's address on the map.

Once the customer is reached, it will report that payment has been made and the delivery has ended. In case of payment problems or absent customer, the delivery failure will be reported and the deliveryman will bring the products back to the store. At the end of the delivery, the deliveryman can evaluate the customer (courtesy, presence at home, comment).

Each deliveryman should be able to view the history of assignments with their outcome (accepted and delivered, accepted and delivery failed, assignment refused).

It is assumed that the manager has an account and is logged into the system. The manager will be able to upload available items. A photo, title, description, price can be uploaded. It is also possible to edit products already available by updating photos, description, etc.

C. 2 Exercise B

A long-term car rental company wants to launch contracts in the market with rates that can be adjusted according to the miles driven and the type of vehicle use. For this, it needs to monitor the use of cars to charge for any negligent behavior in the use of the rented car by customers or to reward the most attentive customers. This will be done with penalties to be added to the annual fee or discounts on the annual fee, respectively.

Each contract is in the name of a company (whose VAT number, company name, registered office address, iban on which to charge fees and an e-mail address are known), a vehicle is associated with it (whose license plate, chassis number, make, model, power in kW and registration date are known), and it can last from 1 to 3 years. When the rental contract is signed, an electronic device (identified by a serial number and managed by an outside company) is installed on the vehicle. Each client company can rent several vehicles, for each of which it will have to enter into a dedicated contract. An employee of the company will enter the contract data into the system.

The company initially defines a set of price bands based on the category of the rented car. There are five possible price ranges in which all rented cars must necessarily belong. The categories of cars and their prices are: citycar (3000 euros per year), small (3600 euros per year), medium (4200 euros per year), large (4800 euros per year), and executive (6,000 euros per year).

Each rental rate includes 15,000 km per year regardless of the category of car rented. Every additional 5,000 km driven will result in a 5% penalty on the annual fee. (So if you drive 12400 km more on an average car, the fee will become 4830 euros). If less than 10,000 km are driven, the customer will get a 5% discount on the annual fee.

In addition, through the API made available by the company that manages the electronic device and for each contract year, it is possible to retrieve the total kilometers driven, and for each car a type of use can be associated, which can be attentive (which allows a 15% discount on the fee), normal (which leaves the fee unchanged), and negligent (which carries a penalty of 10% of the fee).

At the end of the contractual year, through the data received from the control unit, the company's employee performs the annual contractual assessment by calculating, through the system, the penalty or discount according to the rules indicated above.

Once this amount (which can be either positive or negative) is calculated, it is automatically charged (or credited) using the customer's bank data and notified via email. After two consecutive years of negligent use, the contract will be automatically terminated and the car must be returned within one working week of receiving the email notification.

Customers can, at any time, view their situation in their personal area and view, if available, the discount or penalty for the contract year just ended.

Funding: Open access funding provided by Politecnico di Torino within the CRUI-CARE Agreement. This study was carried out within the "EndGame - Improving End-to-End Testing of Web and Mobile Apps through Gamification" project (2022PCCMLF) - funded by European Union - Next Generation EU within the PRIN 2022 program (D.D.104 - 02/02/2022 Ministero dell'Università e della Ricerca). This manuscript reflects only the authors' views and opinions and the Ministry cannot be considered responsible for them.

Declarations

Competing interests. The authors declare they have no conflicts of interest.

References

1. Siau, K., Loo, P.: Identifying difficulties in learning UML. Inf. Syst. Manag. **23**, 43–51 (2006)
2. Boustedt, J.: Students' different understandings of class diagrams. Comput. Sci. Educ. **22**, 29–62 (2012). https://doi.org/10.1080/08993408.2012.665210
3. Stettina, C., Heijstek, W., Faegri, T.: Documentation work in agile teams: the role of documentation formalism in achieving a sustainable practice. In: 2012 Agile Conference. pp. 31–40 (2012). https://api.semanticscholar.org/CorpusID:10335773
4. Bogdanova, D., Snoeck, M.: Learning from errors: error-based exercises in domain modelling pedagogy. In: The Practice of Enterprise Modeling: 11th IFIP WG 8.1. Working Conference, PoEM 2018, Vienna, Austria, October 31-November 2, 2018, Proceedings 11. Springer International Publishing (2018)
5. Deterding, S., Dixon, D., Khaled, R., Nacke, L.: From Game design elements to gamefulness: defining "gamification". In: Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments. pp. 9–15 (2011). https://doi.org/10.1145/2181037.2181040
6. Marín, B.: Lessons learned about gamification in software engineering education. In: Research Anthology on Developments in Gamification and Game-Based Learning. pp. 1473–1496. IGI Global (2022)
7. Jurgelaitis, M., Ceponiene, L., Ceponis, J., Drungilas, V.: Implementing gamification in a university-level UML modeling course: a case study. Comput. Appl. Eng. Educ. (2018). https://doi.org/10.1002/cae.22077

8. Fraser, G.: Gamification of software testing. In: 2017 IEEE/ACM 12th International Workshop On Automation of Software Testing (AST). pp. 2–7 (2017). https://doi.org/10.1109/AST.2017.20
9. Porto, D., Jesus, G., Ferrari, F., Fabbri, S.: Initiatives and challenges of using gamification in software engineering: a systematic mapping. J. Syst. Softw. **173**, 110870 (2021). https://doi.org/10.1016/j.jss.2020.110870
10. Pedreira, O., García, F., Brisaboa, N., Piattini, M.: Gamification in software engineering—a systematic mapping. Inf. Softw. Technol. **57**, 157–168 (2015)
11. Cagnazzo, C., Garaccione, G., Coppola, R., Ardito, L., Torchiano, M.: UMLegend: a gamified learning tool for conceptual modeling with UML class diagrams. In: Proceedings of the 2nd International Workshop On Gamification in Software Development, Verification, And Validation. pp. 2–5 (2023). https://doi.org/10.1145/3617553.3617883
12. Chou, Y.: Actionable Gamification: Beyond Points, Badges, and Leaderboards. Createspace Independent Publishing Platform, USA (2015)
13. Robson, K., Plangger, K., Kietzmann, J., McCarthy, I., Pitt, L.: Is it all a game? Understanding the principles of gamification. Bus. Horiz. **58**, 411–420 (2015)
14. Marczewski, A.: Gamification: a simple introduction. Andrzej Marczewski (2013)
15. Hunicke, R., LeBlanc, M., Zubek, R.: MDA: a formal approach to game design and game research. In: Proceedings of the AAAI Workshop On Challenges in Game AI. vol. 4. pp. 1722 (2004)
16. Werbach, K., Hunter, D.: For the Win. Wharton Digital Press, Philadelphia (2012)
17. Alhammad, M., Moreno, A.: Gamification in software engineering education: a systematic mapping. J. Syst. Softw. **141**, 131–150 (2018). https://doi.org/10.1016/j.jss.2018.03.065
18. Gasca-Hurtado, G., Gómez-Álvarez, M., Manrique-Losada, B.: Using gamification in software engineering teaching: study case for software design. New Knowl. Inf. Syst. Technol. **3**, 244–255 (2019)
19. Djaouti, D., Alvarez, J., Jessel, J.: Classifying serious games: the G/P/S model. In: Handbook of Research On Improving Learning and Motivation through Educational Games: Multidisciplinary Approaches. pp. 118–136 (2011)
20. Garousi, V., Felderer, M., Mäntylä, M.: Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. Inf. Softw. Technol. **106**, 101–121 (2019)
21. Júnior, E., Farias, K.: ModelGame: a quality model for gamified software modeling learning. In: Proceedings of the 15th Brazilian Symposium On Software Components, Architectures, And Reuse. pp. 100–109 (2021). https://doi.org/10.1145/3483899.3483910
22. Bucchiarone, A., Savary-Leblanc, M., Le Pallec, X., Bruel, J., Cicchetti, A., Cabot, J., Gérard, S.: Gamifying model-based engineering: the PapyGame tool. Sci. Comput. Program. **230**, 102974 (2023). https://doi.org/10.1016/j.scico.2023.102974
23. Bucchiarone, A., Savary-Leblanc, M., Le Pallec, X., Cicchetti, A., Gérard, S., Bassanelli, S., Gini, F., Marconi, A.: Gamifying model-based engineering: the PapyGame experience. Softw. Syst. Model. **22**, 1369–1389 (2023). https://doi.org/10.1007/s10270-023-01091-8
24. Livovský, J., Porubän, J.: Open computer. Science. **4**, 171–182 (2014). https://doi.org/10.2478/s13537-014-0209-2
25. Marín, B., Larenas, F., Giachetti, G.: Learning conceptual modeling design through the Classutopia serious game. Int. J. Softw. Eng. Knowl. Eng. **28**, 1679–1699 (2018)
26. Larenas, F., Marín, B., Giachetti, G.: Classutopia: a serious game for conceptual modeling design. In: International Conference On Software Engineering and Knowledge Engineering (2018). https://api.semanticscholar.org/CorpusID:53240926
27. Denden, M., Tlili, A., Salha, S., Abed, M.: Opening up the gamification black box: effects of students' personality traits and perception of game elements on their engaged behaviors in a gamified course. Technol. Knowl. Learn. **29**, 921–940 (2024). https://doi.org/10.1007/s10758-023-09701-6
28. Yigitbas, E., Schmidt, M., Bucchiarone, A., Gottschalk, S., Engels, G.: GaMoVR: gamification-based UML learning environment in virtual reality. Sci. Comput. Programm. **231**, 103029 (2024)
29. Yigitbas, E., Schmidt, M., Bucchiarone, A., Bassanelli, S., Engels, G.: Gamification- and virtual reality-based learning environment for UML class diagram modeling. In: 2024 IEEE Global Engineering Education Conference (EDUCON). pp. 1–10 (2024)
30. Yigitbas, E., Gorissen, S., Weidmann, N., Engels, G.: Design and evaluation of a collaborative UML modeling environment in virtual reality. Softw. Syst. Model. **22**, 1397–1425 (2023). https://doi.org/10.1007/s10270-022-01065-2
31. Yigitbas, E., Schmidt, M., Bucchiarone, A., Gottschalk, S., Engels, G.: Gamification-based UML learning environment in virtual reality. In: Proceedings of the 25th International Conference On Model Driven Engineering Languages And Systems: Companion Proceedings. pp. 27–31 (2022). https://doi.org/10.1145/3550356.3559088
32. Boughzala, I., Chourabi, O., Lang, D., Feki, M.: Feedback on the integration of a serious game in the data modeling learning (2017)
33. Daehli, O., Kristoffersen, B., Sandnes, T.: Exploring feedback and gamification in a data modeling learning tool. Electron. J. E-Learn. (2021). https://api.semanticscholar.org/CorpusID:245152803
34. Krugel, J., Hubwieser, P.: Computational thinking as springboard for learning object-oriented programming in an interactive MOOC. In: 2017 IEEE Global Engineering Education Conference (EDUCON). pp. 1709–1712 (2017)
35. Cammaerts, F., Snoeck, M.: Towards leveraging gamified code-testing for effective model validation. In: Advances in Conceptual Modeling. pp. 233–248 (2025)
36. Cammaerts, F., Snoeck, M.: ModelDefenders: a novel gamified mutation testing game for model-driven engineering. PoEM Companion. (2023). https://api.semanticscholar.org/CorpusID:268946614
37. Dupuy-Chessa, S., Céret, E., Blanco-Lainé, G.: Lego, Pictionary and poker: lessons learnt from gamification for information system teaching at university. In: 7th International Conference On Education And New Learning Technologies (Edulearn'2015). (2015)
38. Bucchiarone, A., Guidolin, T., Fasol, L., Schiavo, G., Kienzle, J., Gerard, S., Négrier, D., Martorella, T.: PolyGloT-UML: a gamified framework for enhancing UML learning paths. In: Proceedings Of The ACM/IEEE 27th International Conference On Model Driven Engineering Languages And Systems. pp. 31–35 (2024), https://doi.org/10.1145/3652620.3687785
39. Sanjaya, L.: Ferdianto & titan development of gamification mobile application for students. In: 2020 International Conference On Information Management And Technology (ICIMTech). pp. 605–608 (2020)
40. Lindland, O., Sindre, G., Solvberg, A.: Understanding quality in conceptual modeling. IEEE Softw. **11**, 42–49 (1994). https://doi.org/10.1109/52.268955
41. Anas, O., Mariam, T., Lyhyaoui, A.: New method for summative evaluation of UML class diagrams based on graph similarities. Int. J. Electr. Comput. Eng. **11**, 1578 (2021)
42. Sun, D., Wong, K.: On evaluating the layout of UML class diagrams for program comprehension. In: 13th International Workshop On Program Comprehension (IWPC'05). pp. 317–326 (2005) https://doi.org/10.1109/WPC.2005.26
43. Bolloju, N., Leung, F.: Assisting novice analysts in developing quality conceptual models with UML. Commun. ACM. **49**, 108–112 (2006). https://doi.org/10.1145/1139922.1139926

44. Nikiforova, O., Gusarovs, K., Kozacenko, L., Ahilcenoka, D., Ungurs, D.: An approach to compare UML class diagrams based on semantical features of their elements (2015). https://doi.org/10.13140/RG.2.1.3104.4889
45. Farias, K., Silva, B.: What's the grade of your diagram? Towards a streamlined approach for grading UML diagrams. In: Proceedings of the 23rd ACM/IEEE International Conference On Model Driven Engineering Languages and Systems: Companion Proceedings (2020). https://doi.org/10.1145/3417990.3420052
46. Modi, S., Taher, H., Mahmud, H.: A tool to automate student UML diagram evaluation. Acad. J. Nawroz Univ. **10**, 189–198 (2021)
47. Bian, W., Alam, O., Kienzle, J.: Automated grading of class diagrams. In: 2019 ACM/IEEE 22nd International Conference On Model Driven Engineering Languages And Systems Companion (MODELS-C). pp. 700–709 (2019)
48. Gómez-Abajo, P., Guerra, E., Lara, J.: A domain-specific language for model mutation and its application to the automated generation of exercises. Comput. Languages Syst. Struct. **49**, 152–173 (2017)
49. Gómez-Abajo, P., Guerra, E., Lara, J.: Wodel: a domain-specific language for model mutation. In: Proceedings of the 31st Annual ACM Symposium On Applied Computing. pp. 1968–1973 (2016). https://doi.org/10.1145/2851613.2851751
50. Fauzan, R., Siahaan, D., Rochimah, S., Triandini, E.: Class diagram similarity measurement: a different approach. In 2018 3rd International Conference On Information Technology, Information System and Electrical Engineering (ICITISEE). pp. 215–219 (2018)
51. Fauzan, R., Siahaan, D., Rochimah, S., Triandini, E.: Automated class diagram assessment using semantic and structural similarities. Int. J. Intell. Eng. Syst. **14** (2021)
52. Gómez-Abajo, P., Guerra, E., Lara, J.: Automated generation and correction of diagram-based exercises for Moodle. Comput. Appl. Eng. Educ. **31**, 1845–1866 (2023)
53. Foss, S., Urazova, T., Lawrence, R.: Automatic generation and marking of UML database design diagrams. In: Proceedings of the 53rd ACM Technical Symposium On Computer Science Education-Volume 1. pp. 626–632 (2022)
54. Sadigh, D., Seshia, S., Gupta, M.: Automating exercise generation: a step towards meeting the MOOC challenge for embedded systems. In: Proceedings of the Workshop On Embedded and Cyber-physical Systems Education. pp. 1–8 (2012)
55. Papasalouros, A.: Automatic exercise generation in Euclidean geometry. In: Artificial Intelligence Applications And Innovations: 9th IFIP WG 12.5 International Conference, AIAI 2013, Paphos, Cyprus, September 30-October 2, 2013, Proceedings 9. pp. 141–150 (2013)
56. Jedlitschka, A., Pfahl, D.: Reporting guidelines for controlled experiments in software engineering. In: 2005 International Symposium On Empirical Software Engineering, 2005. pp. 10 (2005). https://doi.org/10.1109/ISESE.2005.1541818
57. Garaccione, G., Coppola, R., Ardito, L., Torchiano, M.: Gamification of business process modeling education: an experimental analysis. Softw. Syst. Model. **23**, 1569–1594 (2024). https://doi.org/10.1007/s10270-024-01171-3
58. Lee, Y., Kozar, K., Larsen, K.: The technology acceptance model: past, present, and future. Technol. (2003). https://doi.org/10.17705/1CAIS.01250
59. Eppmann, R., Bekk, M., Klein, K.: Gameful experience in gamification: construction and validation of a gameful experience scale [GAMEX]. J. Interact. Market. **43**, 98–115 (2018). https://doi.org/10.1016/j.intmar.2018.03.002
60. The R Foundation R: The R Project for Statistical Computing https://www.r-project.org/ Accessed on 05/06/2024
61. Vegas, S., Apa, C., Juristo, N.: Crossover designs in software engineering experiments: benefits and perils. IEEE Trans. Softw. Eng. **42**, 120–135 (2016). https://doi.org/10.1109/TSE.2015.2467378
62. The R Foundation CRAN—Package Likert M. Package 'Likert'. https://cran.r-project.org/web/packages/likert/ Accessed on 05/06/2024
63. Thai, M., Chong, L., Agrawal, N.: Straussian grounded theory method: an illustration. Qualitative Rep. (2011). https://doi.org/10.46743/2160-3715/2012.1758
64. Stol, K., Ralph, P., Fitzgerald, B.: Grounded theory in software engineering research: a critical review and guidelines. In: 2016 IEEE/ACM 38th International Conference On Software Engineering (ICSE). pp. 120–131 (2016). https://doi.org/10.1145/2884781.2884833
65. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A.: Experimentation in software engineering. Springer Science & Business Media (2012). https://doi.org/10.1007/978-3-642-29044-2
66. Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., Sutton, C.: Program synthesis with large language models (2021). https://doi.org/10.48550/arXiv.2108.07732

Giacomo Garaccione received a M.Sc. degree in Computer Engineering, Software branch, from Politecnico di Torino, Turin, Italy, where he is currently working towards a PhD degree. His research interests include gamification applied to Software Engineering, particularly to software modeling, and software modeling evaluation. He is currently working on building a gamified environment for process and software modeling education.

Riccardo Coppola received the M.Sc. degree in Computer Engineering and the PhD in Control and Computer Engineering from Politecnico di Torino, Turin, Italy, where he is currently working as an Assistant Professor with time contract. His research interests include automated GUI testing for web and mobile applications, and the evaluation of non-functional properties of software projects.

Luca Ardito is an Assistant Professor at Dept. of Control and Computer Engineering of Polytechnic of Turin, where he works in the Software Engineering research group. He received BSc, MSc, and PhD in Computer Engineering from Politecnico di Torino. His current research interests are mobile development and testing, gamification in software engineering, green software and empirical software engineering methodologies.

Marco Torchiano is full professor of Software Engineering at the Control and Computer Engineering Dept. of Politecnico di Torino, Italy; he has been postdoctoral research fellow at Norwegian University of Science and Technology (NTNU), Norway. He received an MSc and a PhD in Computer Engineering from Politecnico di Torino. He is Senior Member of the IEEE, member of the software engineering committee of UNINFO (part of ISO/IEC JTC 1), faculty fellow of the Nexa Center on Internet & Society. He is author or co-author of over 170 research papers published in international journals and conferences, of the book Software Development Case studies in Java from Addison-Wesley, and co-editor of the book Developing Services for the Wireless Internet from Springer. He recently was a visiting professor at Polytechnique Montral studying software energy consumption. His current research interests are: green software, UI testing methods, open-data quality, and software modeling notations.The methodological approach he adopts is that of empirical software engineering.