UML Class Diagram Feedback Tool

Euan Chalmers

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2024

Abstract

UML (Unified Modeling Language) is a modelling language that aims to provide a standardised and expressive way to represent software systems. However, many undergraduate students struggle to learn modelling due to a lack of compilers or syntax checkers compared to textual languages like Python or Java. This project proposes and implements a learning tool for automated UML class diagram evaluation, utilising rules-based model evaluation language to give valuable feedback while exploring advanced language feature and their application to my learning tool. The project proposes and implements a solution to loose syntax modelling tools. The tool displays this feedback within a front-end interface using the proposed method of feedback templates to suit different learning scenarios. I assessed the tool, finding an ease-of-use score of 60% within the target group. Furthermore, the tool found 40% of the errors compared to a human marker. The tool shows potential for a learning environment with possible future work involving model layout handling, UX/UI improvements and rule feature development.

Research Ethics Approval

Ethics for the evaluation of this project was approved under informatics ethics number 302345. My participant consent form and participant information sheet are in the appendices E.2 and E.1.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Euan Chalmers*)

Acknowledgements

I want to thank my supervisor, Perdita Stevens, for all the time she has dedicated to my project and invaluable insight on all aspects of my work. I would also like to thank my family, friends and partner for their support throughout my university journey.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

UML (Unified Modeling Language) is a highly expressive modelling language that can represent nearly any software system. However, many students struggle to master UML modelling due to the need to turn natural language requirements into models while complying with the chosen modelling language syntax.

1.2 Contributions

I will create a customisable and cloud-based model evaluation system to allow educators to clear up ambiguity in model evaluation. Secondly, I will use these evaluations to give granular formative feedback to students in my online tool, applying literature for feedback frameworks.

Chapter 2

Background

2.1 State of the Art

The software development industry is ever-changing with the growing complexity of systems and the proliferation of microservices and computer-generated code. As such, modelling software systems is a more important step than ever to help us understand and communicate designs as the complexity grows. It is critical that software engineers can communicate and design systems in a standardised way. Unified Modeling Language (UML) provides a standard for presenting system architecture and interactions in several structural and behavioural diagrams. The current UML standard is version 2.5.1 [4], maintained by the Object Management Group (OMG). UML is defined in a four-layer meta-model that helps to determine the structure and semantics of the UML language elements that can be used. These meta-models are defined using the Meta Object Facility [53].

After two decades, UML is still regarded as one of the best ways to model complex systems [5, 55] and is used by various industry practitioners due to the flexibility to express nearly any scenario. UML models can be used at many stages of the software engineering lifecycle, with each step requiring different amounts of detail and formality. UML is taught in many undergraduate software engineering courses to varying extents. After I reviewed course websites for 152 universities, it was found that 52% deliver UML-specific courses or courses containing UML (Appendix B).

Educators must now be seen as more than simply lecturers; they are facilitators creating a suitable learning environment where students can receive personalised feedback. These requirements can be difficult to achieve with the ever-increasing workloads placed on academic staff and the growth of undergraduate cohorts. Giving summative feedback to a whole class when class sizes exceed 300 students is no longer possible.

A high-level overview of UML class diagrams can be summarised as a static structural UML model. More detail can be found in Appendix A, which defines many concepts discussed in the following chapters.

2.2 Evaluation of Models

The basis for the proposed tool requires a method to take a model and output practical and actionable evaluations. This section will discuss how model evaluation can be used for model-driven engineering (MDE), education, design pattern identification and safety-critical systems. Furthermore, I will review methods used in any setting as long as they meet the requirement of giving feedback from a UML model.

A model can be evaluated by comparing it to a valid model or perfect solution. A tool will compare diagram elements to those in a perfect solution, for example, matching class and operations. Like the evaluation proposed by Al-Rhman et al. [3], this uses a range of similarity metrics to match the similarity of two class diagrams. Metrics applied included:

1. Class name (lexical analysis).
2. Internal analysis of attributes and operations.
3. Surrounding elements and their relations to the class in question. This information is used to give a similarity score.

The similarity of UML models also plays a factor in MDE use cases. Two categories of inconsistencies are highlighted in Engels' Classification [13]: intra-model (models at the same level of abstraction conflicting, e.g. class and sequence diagram) and Inter-model (different levels of abstraction).

A model can also be evaluated using rules that are checked against models using languages such as Epsilon Validation Language (EVL) [1], OCL, and MetricView [71], which take in a list of rule declarations and compares them against the model under evaluation. Rule-based systems focus on a set of rules to verify that a model meets two requirements:

1. Compliance with the syntax of UML;
2. The question or model-specific requirements, usually set by the educator [38].

EVL rule-based approach can also be used to detect inconsistencies between class and sequence diagrams, which can represent the same software system statically and dynamically as proposed by Allaki et al. [14]. The authors add EVL constraints at the meta-model level, mapping similarly between diagram meta-models. Another study by Arendt et al. [15] created EMF Smell tool based on the EMF language system that can detect UML diagram smells (common indicators of a more significant problem) using Java code and OCL, another rule-based declarative language. Their tool, EMF Refactor, attempts to refactor the input model. They address a main body of work around detecting model smells but do not develop the role of anti-patterns as smells. This is explored by Reischmann et al. [59], who propose the use of error-correcting subgraph isomorphism, a method to detect overlaps between two graph-based structures while allowing for minor discrepancies, to detect design patterns in UML. They then attempt to match these to the expected or preferred patterns for the modelling task. Furthermore, the study effectively created a limited modelling editor, which is more

manageable for education and limits the types of models that participants can create to those that can be handled by the tool.

Abstract State Machines can be employed to model the semantics of the UML models, assuming the model is well-formed when validated against the UML specification meta-model proposed by Compton et al. [63]. Their system uses the XMI format (XML Metadata Interchange) [6], a standard file format for UML diagrams based on XML, but does not evaluate the tool in the paper. Transformation is another evaluation method presented with tools such as vUML [45] converting UML models to PROMELA, a verification language used by the SPIN model verification tool [39]. Mokhtari also transforms UML class diagrams into observational transition systems (OTS) language and uses CafeOBJ, an OTS verification language, to reason in formal algebra about the graph under evaluation [49].

A model can also be grouped into letter grades using UML heuristics and machine learning (ML) training on previous student models [21]. However, there is no formalisation of the technique or proof of the correctness of a model, only that it is similar to models the ML system has seen before and lacking determinism, providing no assurance to the user that the model meets the syntax of the modelling language or the context of the question.

2.2.1 Model Semantics

The semantics of a UML model is what a diagram represents and conveys. Semantics are more complex to evaluate due to the real-world requirements and ambiguous nature of natural language. In addition, the specification for UML 2.0 is defined in a mix of meta-model diagrams, OCL constraints and natural language, leading to different interpretations from tool makers [32]. Furthermore, UML abstracts real-world systems into a model that sometimes removes unimportant details and, therefore, could have many possible implementations, making it hard to test models simply by executing them. Unhelkar's book [73] synthesises the different elements of model evaluation into three main aspects. syntax, semantics, and aesthetics. Each model evaluation method has shortcomings, requiring many tools, which are discussed in Section 2.2, to mix and match different evaluation methods to reach a threshold of correctness or completeness of response on the three aspects of evaluation covered by Unhelkar. An aspect of UML's appeal is the flexibility and ability to represent nearly any system, even itself. As such, attempts to formalise the semantics using Petri nets and similar methods can lead to limiting the modelling language expressiveness [34, 16].

2.2.2 Model Pragmatics and Aesthetics

A UML model can represent the same system with different layouts (Figure 2.1), but looking at this motivating example, which layout is easier to understand and learn from? This is supported by some suggesting that layout can significantly affect a modeller's understanding [58]. Model pragmatics can refer to what modellers usually do, the collective intelligence. Exploring the layout aspect, many modellers will try to reduce the number of times relations cross. Studies have been conducted into preferences

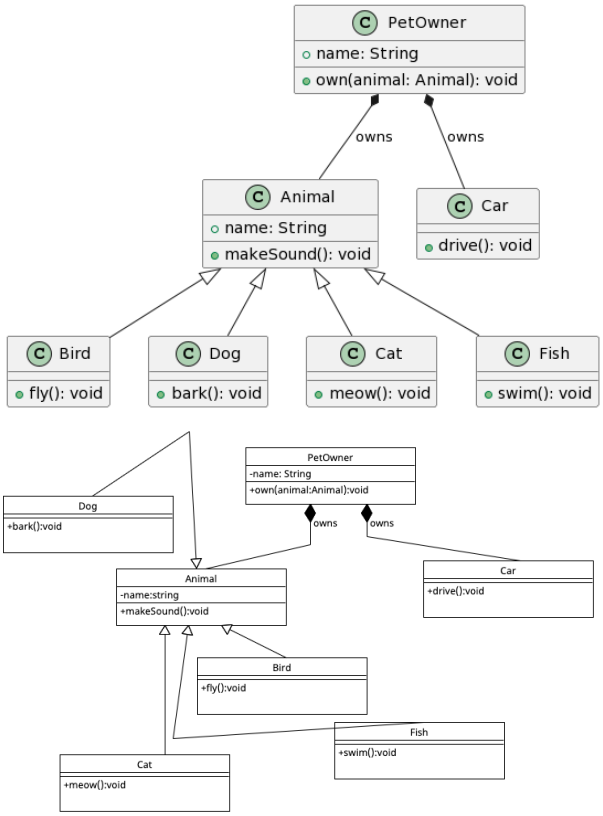

Figure 2.1: Layout motivating example - comparing 'bad' layout (Bottom) with auto layout plantUML (Top)

around layout, which have motivated different layout algorithms [31]. These can motivate individual modeller styling choices or formalised styling guides. While many novice modellers may think of a model as the representation they see in front of them, it is essential to distinguish the underlying model and the visual representation (rendering), which can be represented in many different ways.

Firstly, some modelling tools like UMLetino support manual drag-and-drop layouts, while others perform automatic layout engines. For example, PlantUML uses the GraphViz algorithm to visualise a model. Table 2.1 highlights a small subsection of different tools and their layout methods.

Table 2.1: Modelling tools and layout handling

Modelling tool	**Layout handling**
UMLetino	within .uxf file (coordinates)
Papyrus	.notation files (coordinates)
PlantUML (text-based)	auto layout (GraphViz)
Nomnoml (text-based)	auto layout
Mermaid (text-based)	auto layout

2.3 Gamification and Feedback Tools

The proposed tool will use model evaluation techniques to create insights for the target user group. Several systems have been proposed that are similar to this project's scope and try to bring together model evaluation and teaching. Bucchiarone et al. [22] propose PapyGame, a plugin to Papyrus, an open-source modelling tool [75], to gamify the modelling learning experience (e.g. hangman and level-based modelling questions). Their system allows students to create UML models conforming to a given problem and then receive a score and XP (experience points) for correct and incorrect elements. Their second paper [23] strongly focuses on gamification and presents the most complete study of gamification in modelling. Models are evaluated in two ways by the system:

1. Direct comparison of modelling artefacts to a model solution;
2. Rule-based evaluation (declared in JSON) to check the diagram for specific features, such as the minimum number of classes or attributes.

The main limitation of PapyGame, as currently presented, is the restriction of the Papyrus platform, which requires local installation with no cloud version available, raising the barrier for use. Furthermore, the system lacks the granularity of feedback, which is limited by their evaluation methods, allowing for false, correct answers and catch-all error rules.

The challenges of evaluating the UML models for education that an automated grading system aims to address include the lack of clear model criteria, the time-consuming nature of marking [68] and the difficulty of giving personalised feedback as class sizes grow [9]. These problems are a jumping-off point for Modi et al. [48], as they create a grading system for UML class diagrams. The proposed tool takes in XMI and returns a feedback text file with a score and list of missing or incorrect elements. The presented system uses a model comparison algorithm implemented in Java, which attempts to mimic a human marker by awarding points for elements of a model. However, this method has the same limitation as the model comparison methods highlighted in Section 2.2.

Ali et al. [12] propose the UML Class Diagram Assessor (UCDA), which takes petal files (the export output of the Rational Rose UML modelling tool [7]) and performs two distinct evaluation types on the student model:

1. Analysis of the structure, focusing on the number of classes, relationships, and the correctness of relationships;
2. Naming evaluation checks that the name of a class is accurate to a specification or is a noun.

The system does not give the student a mark. Instead, feedback is focused on the type of error made and includes a comment to provide more context. The system is limited by a proprietary file format input, which restricts the modelling tools that can be used. Furthermore, this paper is outdated, making the practical implementation unlikely to be applicable, but the feedback strategy applies to my tool.

Model heuristics and ML can be employed to give a student submission a binary thresh-

old (e.g. pass/fail) and a letter grade as shown by Boubekeur et al. [21] who use a dataset from student submissions marked by TouchCore [2] to train their heuristics. They then evaluate their heuristic method against TouchCore and a human marker. As the authors highlight, rule-based systems such as TouchCore can be more cumbersome, having to define a ruleset for every question. However, the paper's proposed solution faces the same drawback of non-determinism. This work is further explored by Stikkolorum et al. [68], who use important diagram features such as associations and class names (Appendix A). The authors suggest the benefit of the system is when the assessment rubric is not well defined, but it can be clarified by the human marker as they mark it (contributing to the training set). This method avoids handling the semantics of a model directly and leverages the collective decision of a group of markers on the semantics of models.

Tools have been proposed that create a UML class diagram grading system that focuses on four elements:

1. Lexical matching of class and attribute names using the Levenshtein distance (the number of modifications to transform one sentence to another) [76];
2. Class similarity matching: for example, the number of attributes are present in both classes;
3. Matching classes based on relationships to other classes.
4. Class split and merger: whether a class has been split into two or combined compared to the model solution. This was a more common mistake for functional programmers learning object-oriented programming.

This was proposed by Bian et al. [18], who evaluated their tool and found a 14 % difference between the mark of the human assessor and the grading system. They present a detailed solution to catch many of the common UML mistakes outlined in 2.4.1. However, the authors do not discuss the standard deviation of the results as the outlier results are significant to a grading system (e.g. Most students may receive a score a few percentage points different to a human marker, but one student model received a score of 50% different between the human and system markers.)

A commonality between reviewed literature emphasises the difficulty in automated grading/marking of UML diagrams due to each problem having an ample solution space (e.g. many appropriate class names and possible relationship patterns).

2.4 Teaching and Giving Feedback

Teaching UML can be a complex endeavour that requires a balance of many elements of teaching, including the need to teach the syntax of a modelling language, abstract system design (e.g. abstraction and encapsulation), and sometimes introducing a new modelling tool, which can sometimes contradict the UML specification. Many modelling educators propose and apply different strategies to pedagogy. A study by Akayama et al. [13] discusses their experiences teaching software modelling and the related tool use. A common theme runs through many of the paper's observations,

including the tendency of students to focus on learning the modelling tool instead of the modelling language, which is pertinent to any feedback tool, ensuring the focus is on the students' diagramming. This idea is supported by a student survey in a study, which described modelling tools as "very hard to learn in the beginning" and confusing [44]. The authors emphasise how learning a modelling tool is a non-trivial part of many software modelling courses, further supported by others [54, 69].

Many teaching strategies can be applied to modelling education. An empirical study by Silva et al. [65] proposes using active learning, problem-solving, discussion and real-world examples to engage students in modelling education. Active learning is a method of teaching that encourages learners to think, create, and discuss. This can be applied to modelling teaching through problem-based learning, as suggested by Paige et al. [54].

2.4.1 Common Student Mistakes

Undergraduate students can struggle when modelling in UML due to the difficulty in translating partially defined or unclear specifications into successful models [64]. Students can also face a plethora of issues as novice modellers. Chren et al. [29] created a catalogue of 146 common mistakes across various structural and behaviour models derived from literature and modelling practitioners. They used this catalogue to evaluate over 2700 student models, finding improvement over time through iterative feedback from course tutors. Similar common mistakes can be consistently found across the literature, focusing on class diagrams. Some of the common errors detected in models are listed below as motivating examples of my focus when implementing the tool:

- Inappropriate class names - 36%
- Missing attributes - 58.75%
- Wrong associations - 58%
- Missing associations - 77.4%

These results are an evenly weighted average of results from [43, 60]. However, it is important to consider why students are making these errors, not just that they have made an error. There is a range of reasons for mistakes, whether it is a lack of understanding of context, modelling language syntax, or the modelling tool they are using. Detecting the type of mistake made is only part of the learning process.

2.4.2 Feedback

Feedback is information from a source (e.g. peer, self, educator) on one's performance [37]. A study by Bogdanova et al. [20] discusses how feedback is effective in both traditional learning environments [57] and hybrid/flipped classrooms [36]. The authors also highlights aspects of feedback that are important to students:

1. Timeliness
2. Detail over purely corrective feedback

3. Promote autonomy and reflection.

They propose using the Hattie and Timperley Feedback Model [37] in Modelling Education, using a three-question model of 'Where am I going?', 'How am I going?' and 'Where to next?' to create personalised feedback reports on submitted UML models. However, this study needs to include an analysis of the effectiveness of their feedback reports over time (a longitudinal study), not just reporting on perceived usefulness at a point in time.

Disconnection between the ability of learners to receive instant feedback when learning a modelling language when compared with a programming language, which has near real-time feedback from their compilers and IDEs (integrated development environment), is highlighted by Paige et al. [54]. This motivates the need for real-time feedback on models, such as a basic syntax checker within modelling tools or a standalone tool. This common theme is identified throughout the literature to bring the feedback closer to the error and the learner.

UML learning ontology applications show how modelling educators could link errors, feedback items, learning content and learning outcomes to improve the student experience, as discussed by Bogdanova et al. [19]. The author suggests the use of assessment automation and personalised learning experiences. This system could be expanded to build on the suggestions of Paige et al. to link the learning outcomes and features of a modelling language to reveal different features of a language over time, teaching the complexity of a general-purpose modelling language like UML slowly so as not to overwhelm or dissuade novice modellers.

Another motivational point for an automated feedback solution is the size of growing cohorts in undergraduate computing science degrees [9]. These cohort sizes limit educators' ability to facilitate personalised learning and to deliver the aspects of feedback highlighted by Bogdanova et al.

Chapter 3

Design and Implementation

3.1 Research Questions

I propose three research questions that the project attempts to answer and will identify them as (RQ).

1. Which model evaluation techniques are most suitable for educational tools?
2. How can the evaluation of loose syntax UML modelling tools be implemented?
3. Is it feasible to create a tool that employs the separation of an evaluation method and the evaluation context?

3.2 Design

3.2.1 Class Diagram Evaluation

Class diagrams are static structural UML models that describe the structure of a system by showing the system's classes, operations, properties and relationships (Appendix A, Figure 3.1). I chose to focus on the evaluation of the class diagram for several reasons:

1. The requirements to handle types as part of the UML metamodel added extra conceptual problems to the tool. This development would allow for a more straightforward extension by using sub-parts of the created tool for state diagrams with similar base rules implemented.
2. Discussion with my supervisor about the types of errors and how evaluation of some models can focus on mistakes not likely made by students learning the modelling language, for example, deadlocks in state diagrams. This could have led to the tool being less focused on pedagogy.
3. Test set availability was a key consideration, as access was available to 20 previous real exam submissions from the 2021 Software Design and Modelling (SDM) course that could be used to benchmark my tool.

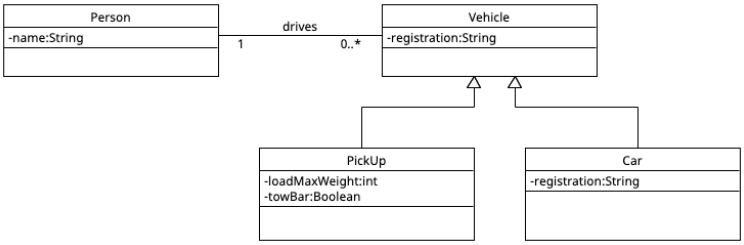

Figure 3.1: Simple UML class diagram

3.2.2 High-Level Design and Stakeholders

The tool must provide valuable insight and utilise the knowledge of each stakeholder to ensure the learner's success. As such, a model of stakeholder involvement was designed (Figure 3.2). This diagram conveys the expected contributions to the tool, with modelling experts create definitions of base rules. Question setters create question contexts. Educators create feedback templates, and students complete the questions. It is expected that the same person may play multiple stakeholder roles; for example, a student may want to set questions for another student, such as on a student-created question platform such as Peerwise [10] or Quizlet [11], or an educator may also be the modelling expert. Furthermore, the division of the question setter and the modelling experts allows for a modelling expert to write a range of rules that could be selected based on the strictness of marking wanted by the question setter. For example, if a student is revising the relation type, they may not be interested in other feedback like the camel case naming convention (e.g. where the second element is capitalised, such as firstWord).

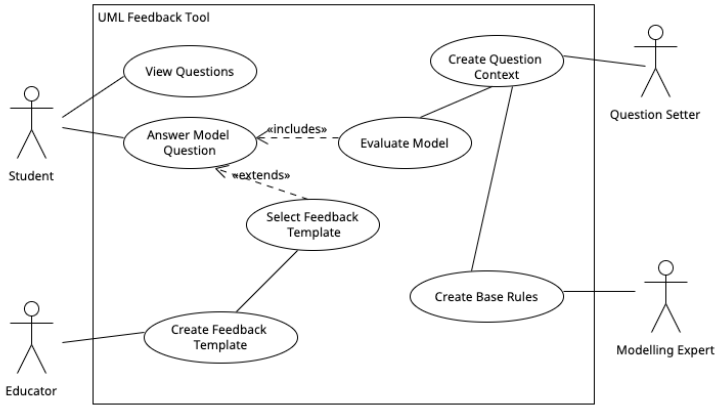

Figure 3.2: Stakeholders use case diagram

3.2.3 Learning Framework

One of the conceptual problems the tool faced was the need to be able to give different granularity of feedback depending on the question, for example, the requirement to only hint to the student the type of mistake they might have made. The learning framework was initially proposed by Bogdanova et al. [19], which I used as a jumping-off point for

the learning framework design within the tool. The learning framework was chosen as it allows a flexible approach where a student can be given meaningful feedback about the errors they have made alongside a suggestion to continue learning in the form of learning resources relevant to the errors made. The connection between the errors and the learning resources helps to bring the feedback closer to the learner. The learning platform, as seen in Figure 3.3, is also a standardised model allowing educators to include as much information as they want to reveal within the feedback templates.

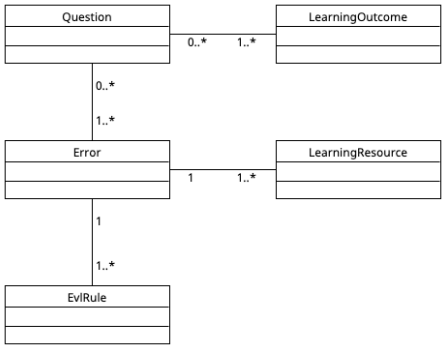

Figure 3.3: Learning framework

3.2.4 Epsilon Platform

The Eclipse Epsilon platform is a family of scripting languages designed for MDE operations [42]. Epsilon has clear benefits for this project, supporting the work in answering research questions 1 and 3. Epsilon has many sublanguages for model-related operations that are utilised in my project and are defined below as:

- **EVL** (Epsilon Validation Language): A model validation language that evaluates models using a set of constraints. If an error is triggered, it produces error messages or fixes.
- **EGL** (Epsilon Generation Language): A generation language that can create code and textual representations from models.
- **ETL** (Epsilon Transformation Language): A model transformation language based on a series of rules.

Each of the aforementioned sublanguages supports expressing code blocks in Epsilon Object Language (EOL), the core language. This allows code to be reused and operations in EOL to be imported into any language. Furthermore, a well-developed UML 2 package was available which is implemented closely in line with the UML specification and was developed for using UML models with any of these sublanguages [52]. The Epsilon Model Connectivity (EMC) layer is a unified interface that allows a model to be imported into any language; for example, if a UML EMC layer is built, then UML models will be compatible with all sublanguages. Moreover, Epsilon supports in-memory models that can be transferred between steps without the need to be stored in model files, which is important for my serverless architecture. Additionally, Epsilon has an active contribution community, meaning many extensions could aid the development

of the tool. The final design can be seen in Figure3.4 and shows my utilisation of each Epsilon scripting language. The process is covered further in the tutorial in Section 3.2.5.

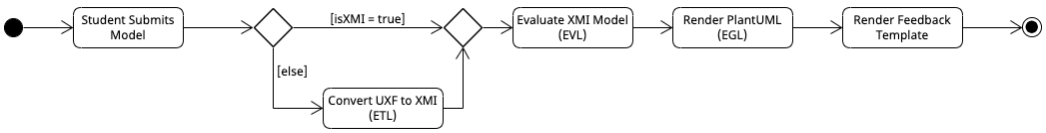

Figure 3.4: High-Level final design of epsilon language uses

3.2.5 Student Tutorial

A full tutorial can be viewed in a demo video [25], but the high-level steps are:

1. Student requests an account token.
2. Student selects a question they would like to attempt.
3. Student models the answer in the modelling software, either Papyrus[30] or UMLetino [72] (embedded into the tool with the question being answered).
4. Student submits their model for evaluation.
5. Student reviews feedback on the model.
6. Students can change the feedback template, retry, or start another question.

3.2.6 Question Setting

The implemented solution to support the Question-setter stakeholder is a web interface for the tool, where a question-setter can create the context of their question, defining expected classes, operations and other modelling elements [27]. The front end implements the jsonForm package under the MIT licence [66], allowing the page to render input forms for different modelling elements from a form schema stored on the server. This then enables the form to be dynamically generated with the information at runtime. This is necessary as if you are defining an operation; you may want to assign a parent class to it (which will have been declared at runtime) in addition to the return type, which could be a class. This solution considers how the modelling elements will interact with each other by amending the forms as modelling elements are added. In addition, storing the JSON schema on the server allows for minimal changes when adding more features, such as abstract classes or more relationship types. The JSON representation of a question context can then be uploaded to the server and added to the question list or downloaded for future use. The server currently attempts to attach all applicable rules to a model context, but this could be changed to allow a question setter to choose which base rules to apply or the strictness level. The JSON representation of questions could even support AI-generated questions using a LLM (large language model) with custom JSON prompting to grow the number of questions on the platform, increasing the value proposition.

3.3 Evaluation of Models

3.3.1 Context-Driven Evaluation Approach (EVL)

A rule-based language was chosen over other model evaluation methods for several key reasons. Rules have strong determinism, making testing of the tool achievable within the constraints of the project. A rule-based language allows different stakeholders to create rules and contexts, unlike comparing models against a perfect solution. Rules allow for multiple correct solutions, which a perfect solution may not be able to capture elegantly. Additionally, the power of Epsilon is highlighted in the list of possible rules-based languages. This is why EVL was chosen over other languages, such as OCL [61] or Schematron [56]. EVL rules are defined into code blocks which contain: a *guard* that will check if the rule should be executed; a *check* which defines the rule to be evaluated; and *a message* that returns a message if the rules fail. Each block can accept EOL, the parent language [42].

```
Task {
    guard: self.name = "MyToDos"
    constraint ValidStartMonth {
        check: self.startMonth > 0
        message: "Start month must be positive for MyToDos"
    }
}
```

Figure 3.5: Example EVL rule

Fourteen base rules have been implemented that cover the highlighted methods in the meta-model (Figure 3.7) and can be used as often as a context requires [26]. This is not enough to say with certainty that a model is correct, but the tool's design supports additional elements added with relative ease by creating new base rules in DB (database) and with a few changes to the tool code. Implementation of the evaluation rules based on my simplified meta-model can be compared with the real meta-model (Figure 3.6). The model also shows all base rules as operations within the element. The design of the EVL rules was considered carefully, using features such as guards to only check elements in the correct context; for example, if the return parameter of an operation in class X is being checked, we do not want to check if any or a different operation has the expected return type. Furthermore, while most rules are defined within the context of their modelling element with a guard as seen in Figure 3.5, to be able to check existence, we need to check with the model context in case there are no elements of that kind as context blocks will only execute if one or more of that element exists (Appendix D.1).

3.3.2 Multiple Correct Solutions

Another conceptual problem is the element naming problem. We may have a class that should be called 'Student', but a modeller may choose to call it 'undergraduate', 'learner' or 'Studnt'. We should not treat all of these naming differences the same, so

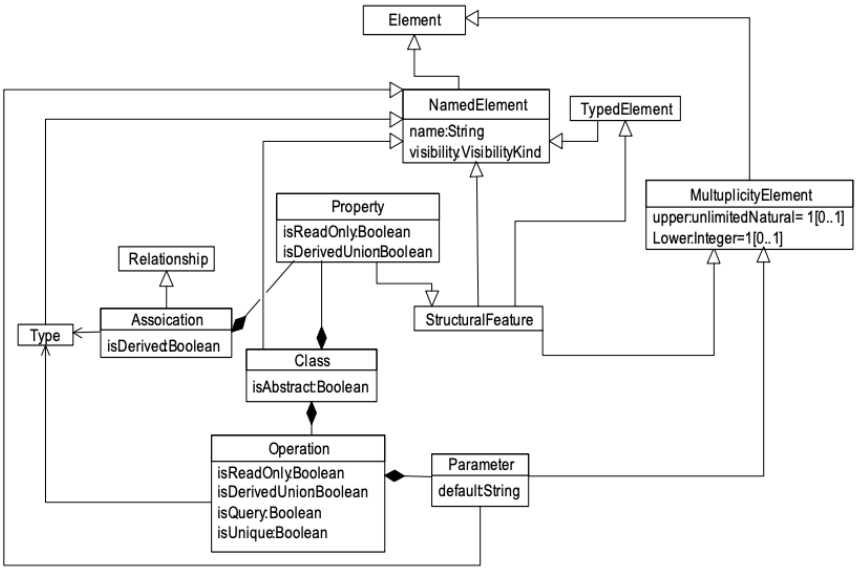

Figure 3.6: Simple UML class diagram meta-model - credit: Solberg et al.[67]

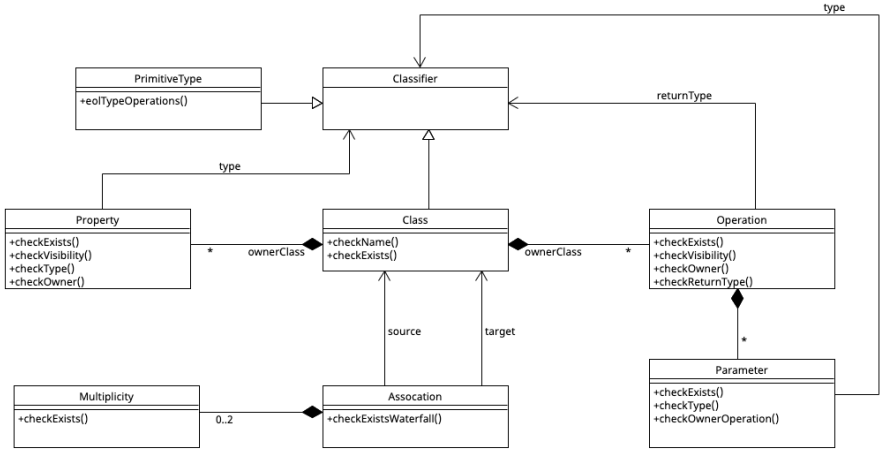

Figure 3.7: Pseudo meta-model of the implemented rules

the tool implements a solution in the question creator and EVL base rules to accept an array of acceptable names for classes, parameters and operations. This is also a jumping-off point for receiving and evaluating multiple correct solutions.

3.3.3 Waterfall Rule Validation Method

The waterfall validation method is the proposed solution for the common near-miss mistakes that can occur in class diagram modelling. For example, a student is showing a connection between an abstract parent class and a concrete subclass using an association when an ideal solution would look for a generalization. When giving feedback on this error, it is important to have information about where the learner has gone wrong to add detail to the feedback. My relationship EVL base rules use the waterfall method, where the rule will try to validate each type of relation until the one used is found, as seen in Figure 3.8. This allows the tool to make more insightful comments like No generalization found, but an association does exist between X and Y. Returning to the learning framework, the more detailed information can be gathered, the better the learning resources can be targeted and the more flexibility educators have when creating feedback templates.

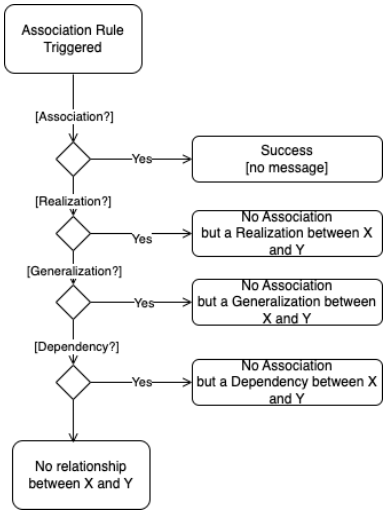

Figure 3.8: Relationship EVL rule waterfall design

3.3.4 Base Rule Development

Base rules are the underlying rules that support the context-driven EVL approach that can be viewed on the tool base rule webpage [26]. During my studies, I entered the MDEnet education platform competition [17], where participants had to create MDE activities. To contribute, I made EVL activities tested against the UML model and developed my Epsilon skills. I propose this implementation could be used as a testing environment for new base rules to be tested in a user-friendly way before they are deployed to the tool. Furthermore, I have also implemented a modelling activity with some model heuristics EVL as discussed in future work in Section 5.3.

3.4 Tool Implementation

3.4.1 Tool Architecture

The tool was designed to be extendable and web-based to remove the download barrier for entry that many similar tools have faced, as discussed in Section 2.4.2. The tool is written in Java, employing the Model-view-Controller Pattern (MVC) chosen due to its strong separation of concern. This was deployed in an AWS Lambda serverless function, allowing it to stay deployed with little ongoing cost. This is then supported by hosting the front-end site on S3 and storing all tool data in MongoDB NoSQL tables (Figure 3.9). The implementation also leads to the tool utilising advanced features of Epsilon, such as in-memory models, to remove the need to load models into files or write to a database, allowing the final implementation to simultaneously have multiple models in flight to increase performance and concurrency. Another feature of the tool's architecture is anonymous user IDs, which are generated by the server and stored as a cookie on the user's device. These allow student submissions to be linked to accounts, which could be extended and logins added to allow users to view previously completed submissions and progress over time.

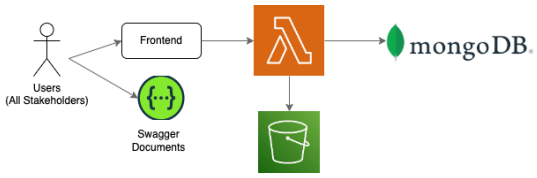

Figure 3.9: Tool AWS deployment

3.4.2 Type System

As can be seen in the pseudo-meta-model (Figure 3.7), the types of modelling elements are key to their successful evaluation. UML has two main groups: primitive types (e.g., String, Boolean, Real, UnlimitedNatural) or a class element within the model. When handling types in these scenarios, this must be taken into account. To address this complex conceptual problem, I built a unified UML-type handling function in EOL that is imported into my ETL and EVL implementations. This brought the benefit of a consistent type system across the tool and the ability to easily add support for different UML-type systems supported by various tools. This was difficult to implement due to the necessity of understanding the primitive types and the current context (e.g., all classes in a model). The main uses are converting from human-readable types (within a UMLetino model) to correct types in XMI form and comparing human-expected types from the question context evaluated against UML types. One clear limitation is the creation of types within ETL when translating a type such as the property -carType: Car (Figure 3.10, Line 9). Where Car is a class, my XMI output will be a primitive type within UML named Car (Figure 3.10, Line 5), which will pass evaluation when evaluating the type but may limit future base rule implementations.

```
1  <?xml version="1.0" encoding="ASCII"?>
2  <xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.
       org/XMI" xmlns:xsi="http://www.w3.org/2001/
       XMLSchema-instance" xmlns:uml="http://www.eclipse.
       org/uml2/5.0.0/UML">
3    <uml:Model>
4      <packagedElement xsi:type="uml:PrimitiveType"
           name="String"/>
5      <packagedElement xsi:type="uml:PrimitiveType"
           name="Car"/>
6    </uml:Model>
7    <uml:Class name="Person">
8      <!--Type field references the primitive type-->
9      <ownedAttribute name="carType" visibility="
           private" type="/0/Car"/>
10   </uml:Class>
11   <uml:Class name="Car"/>
12 </xmi:XMI>
```

Figure 3.10: ETL transformation XMI output for UXF

3.4.3 Logging System

The tool implements a logging system that can be used within any Epsilon language, allowing the programmer to log into a Java array and then use this data outside of the Epsilon languages. This is implemented using native EOL types, allowing Java methods to be imported into Epsilon languages (Figure 3.11). This solution utilises a simple Java logger to log loose syntax errors, as discussed in Section 3.5.3. This could be used to improve the imprecise logging of Epsilon or unify logging between a Java application and the Epsilon used within large systems.

```
//create an instance with etlTestRun being the name
var logger = new Native("org.inf.src.Services.
    EpsilonLogging")("etlTestRun");

//log example
logger.log("This text with be logged out to the java
    logger")
```

Figure 3.11: Epsilon logging example

3.5 Loose Syntax

3.5.1 Modelling Tools

The choice of modelling tool is a personal one, with each tool having trade-offs, whether an industrial modeller looking for a strict tool that enforces good modelling or a drag-and-drop modelling tool for use in education, with many tools being best suited to different UML model creations, as discussed by Lu and Alexandru [46]. My design philosophy was to build a base tool that handles the strictest enforcement of UML specifications and creates an XMI export. To meet these requirements, Papyrus was initially chosen [30] as the first tool to add support for. The tool can then be extended to accept different modelling tools by building adapters to match the range of export types and converting syntax to match the XMI format. The more adapters that can be created, the more possible users the tool will have.

Therefore, UMLetino was selected due to the XML-based export feature and web-based application, which will aid the evaluation of the tool, allowing participants to model in a web browser. I created an adapter to take a UMLetino export (UXF file) to attempt to solve the conceptual problem of translating a loosely modelled diagram into XMI. The design was to take source model S from UMLetino and create a target UML model T. Creating an adapter adds a layer of complexity as a student can now either make an error in their understanding of the UML syntax or a tool error, where they may intend to do something, but the tool does not act as expected. My final tool has the UMLetino web modelling tool embedded into the student user flow, allowing them to view the question and model in the same window. This could also allow for the extension of the project to evaluate the model in real-time and give live feedback on syntax through diagram highlighting or error alerts within the UMLetino window.

3.5.2 Initial Approach

The initial approach to handling input from a loose syntax tool was to build an extension to my Java application that recreated a simplified meta-model of a UML class diagram that could be used to parse the file and create the UML document output. This solution had drawbacks of increased code lines, complexity and reduced maintainability (the initial partial solution had 476 lines). Furthermore, there is difficulty in extending this if, for example, I wanted to change the input modelling tool; this would require another new adapter to be written. Furthermore, the solution did not utilise the fact that the output source S is UXF (a type of XML model), which can interact similarly to any other model using Epsilons' EMC layer.

3.5.3 Epsilon Translation Language (ETL)

At the MDEnet symposium 2023[8], I attended talks showing the use of ETL, where I was inspired to reassess my approach to the loose syntax problem. Using ETL could allow for more features to be implemented and make the tool more extendable if, for example, a developer wanted to add another modelling tool input. When considering the new input syntax, they would only have to define another ETL rule set. Furthermore,

the implementation can use the platform's power again by implementing my Epsilon type and logging systems.

ETL rules are defined within a context, with many rules within this, as seen in this example, rule (Figure 3.12) where *s* = source UXF model and *t* = target UML model. We are transforming any relation in UXF (that is guarded with type Realization (Figure 3.12, Line 3), meaning only those of the Realization type will trigger this translation rule. Then, the check sets the supplier and client classes based on the inferred class connection (see Section 3.5.4).

```
1  rule ElementToRealization
2      transform s : Source!t_element
3      to t : Target!Realization {
4          guard : s.c_id.text[0] = "Relation" and (s.
               c_panel_attributes[0].text.split("\n")[0].
               getRelationship() == "Realization")
5
6          var relation = s;
7          var panel_attributes = relation.
               c_panel_attributes[0].text;
8          var arr = panel_attributes.split("\n");
9
10         //FORMAT x,y,w,h
11         var coords = relation.c_coordinates.children[0].
               text;
12         var ends =  coords.getTheConnections();
13
14         if(ends.size() == 2){
15             t.getSuppliers().add(ends[1]);
16             t.getClients().add(ends[0]);
17         }
18         else{
19             logger.log("Error A4 : Unconnected Relation
                   "+"Realization");
20         }
```

Figure 3.12: ETL relations complex example

In scenarios where a model cannot be correctly transformed, errors will be recorded using the epsilon logging (see Section 3.4.3); the initial implementation of loose syntax errors is below. The trigger can be found in the ETL file.

- Error A1: Unconnected relation for an association
- Error A2: Unconnected relation for a generalization
- Error A3: Relationship not correctly defined (e.g., does not match expected above)

- Error A4: Unconnected relation for a realization
- Error A5: Visibility not correctly defined (can currently only be +,-,)
- Error A6: Functionally/element currently not implemented
- Error A7: You are missing a name or type for the parameter, likely due to the lack of ":" to define type and name.
- Error A8: You have not defined a type for this property

There are many important things to be considered when writing the ETL. The ability for users to represent types and names in different orders while still conveying the exact information required defensive programming, building rules that could be flipped to handle these cases regardless of the represented text. Furthermore, the need for the tool to understand the context of the model when translating means we must know the list of all class names as a return type could be of another class object. This required the use of the ETL pre-function, a code block executed before any other code, to gather this information first. ETL also supports advanced model transformation features such as accepting user prompts, which could be used to develop the tool to prompt the modeller to ask them to clarify while the model is being transformed, for example, to prompt what classes they meant to connect.

3.5.4 Inferring Connections

The ability of a user to connect two classes in the modelling area within a loose syntax tool is not well represented by many loose syntax tools. In the case of UMLetino, coordinates are used for each object but do not relate classes to objects. Therefore, a closeness system has been implemented to detect if a relation is close to a class or intercept it and infer these relations in XMI form (Figure 3.13). This was implemented in Java due to the available libraries for the coordinates handling and was then imported into ETL using the naive type, as seen in the example. Furthermore, it supports the extendibility of the tool through the use of the inferring connection for other types of UML models, such as activity and use case.

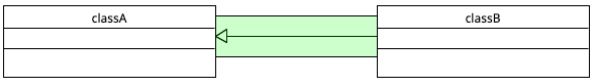

Figure 3.13: Bounding box system

One clear limitation of how the relationships were handled was the inability to cover the relationship direction, as shown in Figure 3.14. Two associations are going in the same direction, but looking at the UXF output (Figure 3.15), they have different properties for the relation type and, therefore, would be represented differently in XMI. This could be rectified with research and utilisation of the additional properties that convey where the arrowhead is, building new functionality to handle this within my tools handling within ETL (Section 5.3). Furthermore, reflexive associations (classes connecting to themselves) have also not been addressed in this implementation.

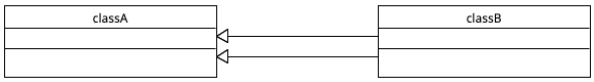

Figure 3.14: Example of UXF class diagram

```
<element>
    <id>Relation</id>
    <coordinates>
        <x>370</x><y>250</y><w>190</w><h>30</h>
    </coordinates>
    <panel_attributes>lt=&lt;&lt;&lt;-</
        panel_attributes>
    <additional_attributes>10;10;170;10</
        additional_attributes>
</element>
<element>
    <id>Relation</id>
    <coordinates>
        <x>370</x><y>290</y><w>190</w><h>30</h>
    </coordinates>
    <panel_attributes>lt=-&gt;&gt;&gt;</
        panel_attributes>
    <additional_attributes>170;10;10;10</
        additional_attributes>
</element>
```

Figure 3.15: UXF UMLetino Output

3.6 Feedback

3.6.1 Feedback Templates

As part of supporting the educator stakeholder, the tool accepts any correctly structured HTML file as a feedback template, allowing for various styles and information to be revealed to best support learners (Figure 3.16). For example, an educator may want to give as detailed feedback as possible or only indicate the learning resources as hints to try to get the student to find their own mistakes as part of an active learning process (see Section 2.4.2), or the student could view the Hattie feedback template I have created [37]. This is all powered by the learning framework and utilises Thymeleaf, a Java template engine which was chosen due to its support for both HTML and JavaScript templating [70]. After evaluating the model, this engine will insert the learning framework's relevant aspects into each template (Figure 3.17). The current implementation allows the student to view their feedback in any of the implemented templates, but with minor tweaks to the code, the system could fix each question to a feedback type.

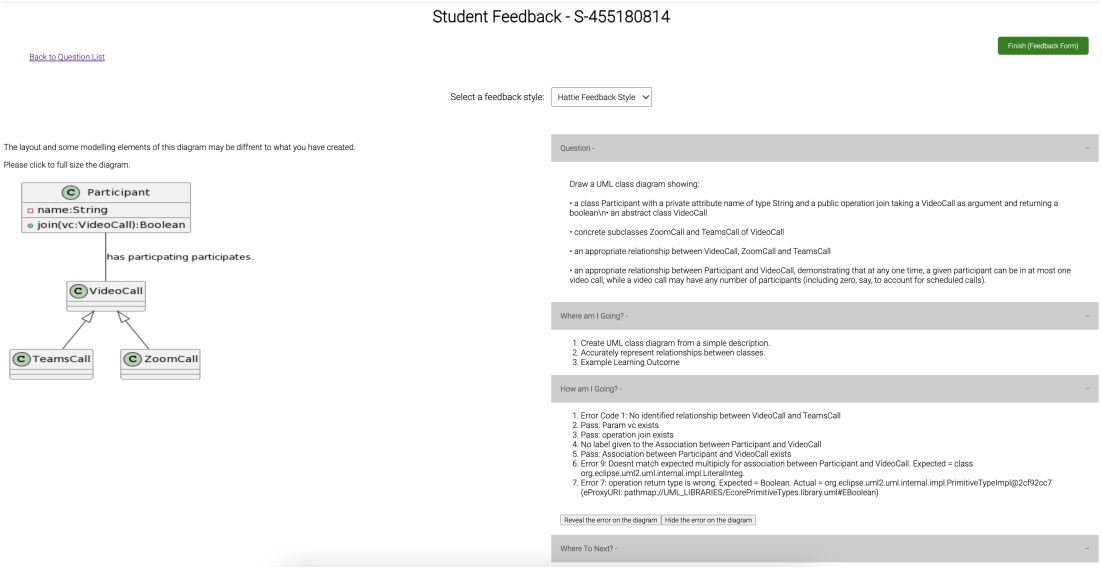

Figure 3.16: Hattie feedback example

3.6.2 Diagram Rendering (EGL)

When giving feedback, it is essential to relate the feedback given to the student's model to increase the effectiveness. As such, one of the conceptual problems was creating a visual representation of the student's work connected to evaluating the completed models.

The current options of the web diagram tool are limited to a few applications that support UML online implementation, mainly plantUML, mermaid and nominal. These are rendered from textual representations as shown in the plantUML example (Figures 3.18, 3.19). The proposed solution to this is Epsilon Generation Language (EGL), which is a model-to-text transformation (M2T) language that would allow the tool to take the UML representations of the models and transform these into textual plantUML representation

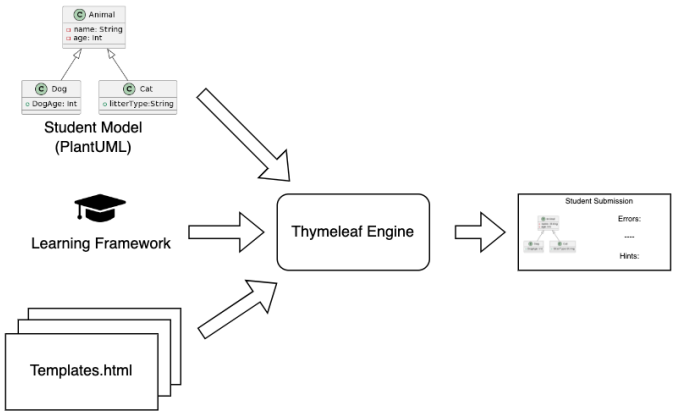

Figure 3.17: Feedback template process

and can be rendered as seen in Figure 3.16. An example of EGL and plantUML usage can be seen in the Epsilon playground [41], where the creators have used EGL to render Flexmi and Ecore into plantUML. EGL allows the utilisation of the shared type system operations similar to ETL usage (see Section 3.4.2). My solution implements the main features of class diagrams, visibility, types and relationships. However, it has a limited scope, lacking more features, such as abstract classes and multiplicities, because of the limited time of the project.

One explicit limitation of the visualisation is the layout of model components due to the lack of layout standards in the XMI format. This leads to each modelling tool representing the layout of the diagrams in different ways, such as the translations and generation, for example, between UXF and XMI then to the plantUML form layout can be lost. This could lead to increased cognitive load as learners struggle to map between their original and rendered models. This has apparent drawbacks as layout can be an essential part of a student's understanding of a model and the ability of the design to express the intention of a model. Additionally, this limits the evaluation of model heuristics, for example, if relations cross as discussed further in Section 5.3).

3.7 Testing and Improving

3.7.1 Testing

Behaviour-Driven Development (BDD): The tool has been tested by applying BDD test scenarios for the main flows of each stakeholder to ensure the implementation and designs discussed in Section 3.2.2, are functioning correctly. These requirements were created from informal discussions with my supervisor (a modelling and education expert) and students (with various experiences in UML modelling). These tests can also be used as good documentation for the system to understand the services used by each stakeholder in a form closer to natural language.

Frontend Testing: While not evaluating the design of the front end for design heuristics due to the limited scope of the ug4 project, some testing was conducted to ensure the

```
@startuml
class Animal {
  -name: String
  -age: Int
}

class Dog {
  +DogAge: Int
}

class Cat {
  +litterType: String
}

Animal <|-- Dog
Animal <|-- Cat
@enduml
```

Figure 3.18: plantUML example

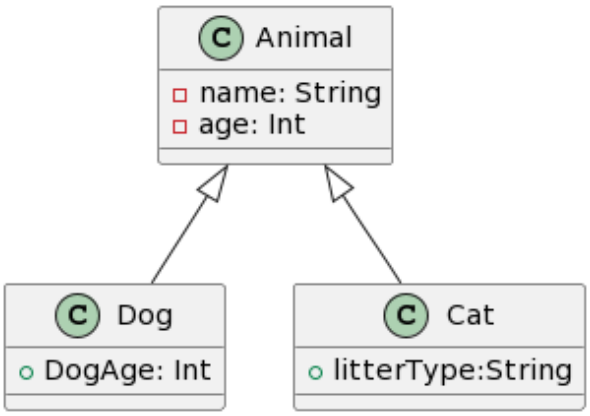

Figure 3.19: Pet example plantUML rendering

tool is compelling enough so as not to affect the evaluation of the tool. Analysis was completed on all pages using Google Lighthouse, which checks accessibility, performance and best practices. The pages got an average score of 86.47% with a standard deviation of 9.76 (Figure 3.20). Furthermore, I have created a multi-browser driver that allows the upload model page to be tested in Chrome, Safari, and Firefox, after which the system will hand over to manual tests to ensure all items in the feedback template are rendered as expected. With a combined market share of 93% [40], these tests attempt to cover a large amount of the possible target group use of the tool.

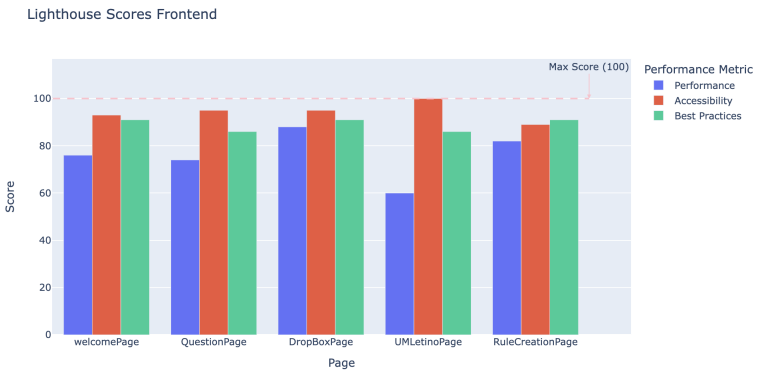

Figure 3.20: Lighthouse front-end evaluation results

Stress and API Testing: A Python API test script was created that tests each endpoint's responses once deployed on the AWS Lambda server, including a stress test with 20 concurrent requests. This is not anticipated as a problem as AWS serverless functions should scale automatically far beyond the need for the evaluation, and the MongoDB

cluster has enough bandwidth deployed. These tests run each time the system is deployed to AWS to ensure minor tweaks create no breaking changes in a CI/CD (continuous integration/continuous delivery) pipeline.

Testing Platform: Finally, I have created a testing website for each test example that displays the model in each state directly from the tool, after ETL translation and the plantUML is generated (Figure 3.21). Each page also shows the XMI code in each state. The testing system allows troubleshooting of scenarios to quickly identify where an error has occurred and where there is a deviation from the expected model. The website visually shows the layout inconsistencies between models during testing. The testing page was further developed to support evaluation by extracting unique EVL error codes for each model to allow for easy evaluation of the rules triggered.

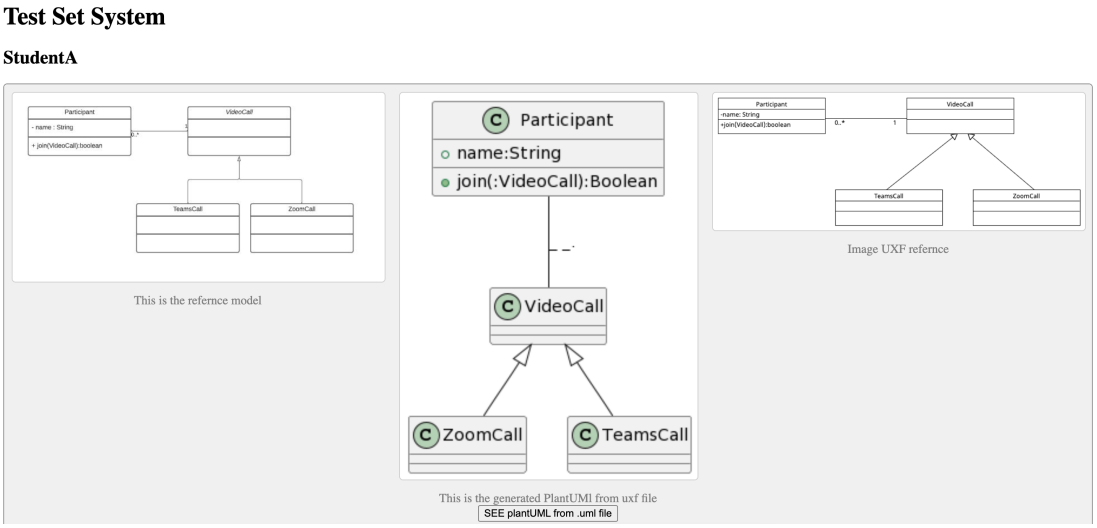

Figure 3.21: Testing platform

3.7.2 Refining the Tool and Documentation

I have attempted to refactor my tool's backend by moving the Epsilon function into a standard class, which simplifies additional uses of the Epsilon languages; similar refactoring was done to my CRUD (create, read, update and delete) operations, creating a consistent way to access resources which support the system to store data to both the MongoDB cloud database or local JSON files (for local development). Swagger open API docs have been created that can be used for further development, possibly for more frontend-focused projects using the same base development of the tool [33]. Moreover, a base rule syntax checker has been created that can be used instead of or in parallel to the MDEnet tool suggested in Section 3.3.4. Furthermore, several how-to documents have been produced for setting up and creating similar standalone implementations of the Epsilon languages within Java; these how-to documents were written to apply to other standalone Epsilon Java projects in addition to the current project [24].

Chapter 4

Evaluation

4.1 Student Evaluation Study

4.1.1 Ethics

Ethics for the evaluation workshop was approved under Informatics ethics number 302345. The participant consent form and participant information sheet are in the appendices E.2 and E.1.

4.1.2 Aim

The following study aims to gain feedback from the students learning UML modelling, the tool's target group. This evaluation will attempt to analyse the tool's effectiveness in a teaching context and research the most valuable features for the aforementioned target group.

4.1.3 Method

The participants were given a brief introduction to the goals and key features of the tool and informed of the steps highlighted in the tutorial flow (Section 3.2.5). The participants could also ask questions about unclear aspects. Finally, the participants were directed to the feedback form to gather the feedback discussed in the results section. Workshops were chosen as the preferred evaluation method due to the tool's development stage, allowing problems to be resolved quickly and participants to be better supported compared to a remote survey-only study.

4.1.4 Pilot Study

As part of the workshop planning process, a pilot study was conducted with a small number of participants who carried out the workshop. These participants were recruited from year 4 undergraduate students who had previously taken a course teaching UML and had experience participating in evaluation workshops. The goals were to find

any problems with the workshop design and any small suggestions that could be implemented in the final pre-evaluation iteration of the tool.

4.1.5 Resulting Changes

This pilot study observed that participants struggled with the UMLetino Modelling Page, where users view the question and an iframe (embedding of another HTML page) of the UMLetino Modelling interface. This resulted in the following changes being implemented:

1. Creating additional JavaScript for the UMLetino iframe to remove the Dropbox export option to clarify the required export option for participants.
2. A help button was added in the properties tab to guide users struggling with the modelling tool. The feedback on UMLetino also made it clear the evaluation of results had to attempt to exclude any feedback on the modelling tool itself as this was not the project's goal.
3. ETL transformations from the modelling tool were strengthened to reduce the number of unhandled modelling elements that would cause the system to crash.

4.1.6 Participants

Participants were recruited from the target group of modelling learners at the university. Those with some experience in UML or had participated in a course that contained UML, while complying with the ethics requirements. The participants are incentivised to practice UML as part of their respective courses, increasing the evaluation proposition for my workshops.

- **Software Design and Modelling**: A level 10 course usually taken in year 3 or 4. I attended a workshop as part of the course to test my tool with the students.
- **Software Engineering and Professional Practice**: A level 8 course (a required course in year 2). The students were contacted via an email advertising the workshops I was hosting.

4.1.7 Data Collection and Cleaning

Data was collected using a Microsoft form completed by participants after using the tool. After the workshop, the data was exported into a CSV file, where the ethics consent section was checked, and the name of the consent giver was removed. All open-ended questions were evaluated for personally identifiable information and sanitised when required. This data was then stored securely on the university-provided OneDrive, ready for evaluation.

4.1.8 Results

Net promoter score (NPS) is a one-question metric asking, 'How likely are you to recommend this tool?'. It was gathered from the participants to gauge the likelihood of

engagement and usefulness of the tool alongside the possibility of continuing to use it [35]. The NPS is calculated by taking the percentage of promoters minus the percentage of detractors, resulting in the tool scoring -30. This shows that the tool does not have many promoters within the evaluation group, suggesting it is either not fit for purpose or does not have product market fit with the user group, as seen in Figure 4.1. This metric was chosen because it allows the tool to be benchmarked against other successful software tools with an NPS of 40-70, but at this stage, the most important aspect of the results is the detractors, evaluating if most participants respond net positively to the first iteration of the tool.

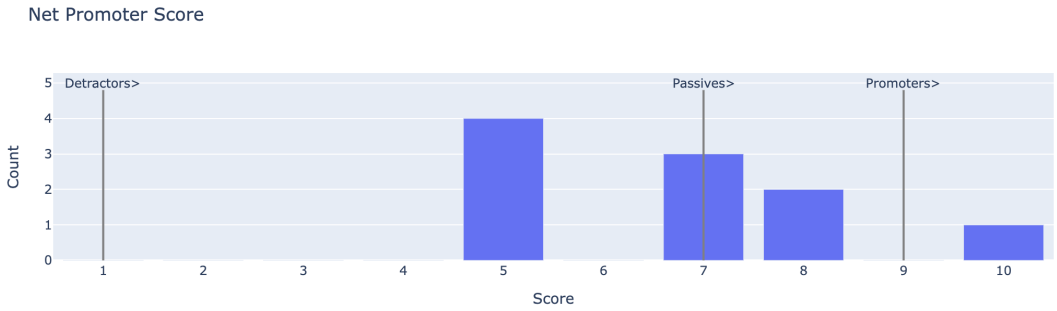

Figure 4.1: Net promoter score

The importance of requirements was also gathered from participants by asking, 'Please order these four requirements from which would be most important to you to the least important?'. These results found in Figure 4.2 show that ease of use was a key aspect for many users in addition to the detailed feedback. These results support the hypothesis of the project's motivation of providing detailed feedback. However, consideration should be given to the validity of this question as this question was asked at the end of the workshop survey, which could have influenced participants' answers as by then, they had already used the tool.

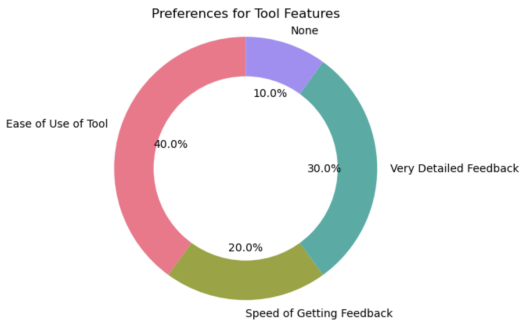

Figure 4.2: Most important feature requests for the tool

Analysing the answers to the open-ended questions, we can see the tool has poor UI (user interface) and structure of the feedback, supported by comments such as:

'The feedback was a bit confusing - the structure of the feedback was unclear.' - Participant F

Furthermore, the problems with the tool crashing when translating from a loose syntax from UMLetino to XMI for evaluation created issues for some participants:

'I had trouble getting my diagram to get marked. I tried the whole procedure of downloading and then uploading the file several times. Every time, however, I was faced with a failed-to-render error, and I couldn't really figure out why.' - Participant D

From the results of the two questions 'How easy is the tool to use?', and 'How would you rate your experience level with UML (Unified Modeling Language)?'. It can be seen that there is little deviation between a participant's ease-of-use score when compared with their modelling experience (Figure 4.3). This conveys the modelling experience may be a confounding variable in the other workshop results. However, the validity of these results could be questioned due to social desirability bias, which is participants' tendency to report views desirable to others [50]. For example, participants may overestimate their modelling experience or rate a score around the centre of the scale if they struggle to understand the question but want to give a helpful response.

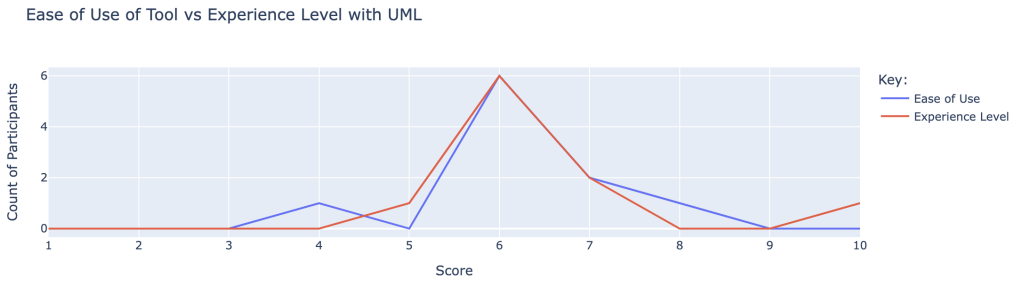

Figure 4.3: Comparison of ease of use and experience level

4.1.9 Thematic Analysis & Resulting Requirements

As part of the evaluation workshops, participants were asked an open-ended question: 'Do you have any suggestion to improve the tool?'. Braun and Clarke's thematics analysis [47] was carried out on this data as part of the requirements refinement for further tool development. Phase 1 extracted open-ended codings from the data, and phase 2 then refined these codings into themes, as seen in Table 4.1. Some of the themes reiterate the limitations of the evaluation, such as the design choices of the embedded UMLetino tool, which was outside of the project's scope. However, some themes, such as understanding the task and the limitations of the tool, could be addressed with a clear tutorial. Furthermore, priorities were assigned to each theme based on the number of mentions.

4.1.10 Limitations of Evaluation Workshops

Some limitations of this evaluation are:

1. The lack of participants means it cannot definitively be concluded the data gathered was reliable. The evaluation should have either focused on an in-depth user study, such as think-aloud with 5-10 participants, or a similar workshop with 20-30 participants. The participation was likely limited due to the busy schedule of students during semester 2 of the university year and the lack of compensation for the workshop.

Table 4.1: Thematic analysis - phase 2

Theme	**Extracted Codings**	**Priority (1-5)**
Frontend design	• FE flow & ease of use • FE Menu Layout • Visibility of FE elements	5
Modelling tool & UMLetino	• FE modelling & need to learn a new tool • UMLetino tool limitations	3
Understanding of tool limitations and design	• Lack of workshop guidance • Understanding what the tool can do	2
Tool feature improvement	• ETL crashes • Error code failed to render in the diagram • Ability to run evaluation within UMLetino	3

2. The design of the evaluation workshop led some participants to need help completing the required tasks. This was clear as 20 participants started the workshop, but only ten made it to the end of the workshop survey. This could have been due to the need for more tutorials, hints, or tips due to the front-end design needing improvements. It was also noted that no participants attempted to create their questions.

3. Also, due to a lack of continuous evaluation, it cannot be concluded if the tool helps increase a user's modelling skills. This was not done due to the time limitations of the UG4 project and the possible ethical implications of a longitudinal study having a control group and a test group of participants using the tool; for example, if the tool proved effective, it would give one group an advantage.

4. The embedding of UMLetino within the tool led some participants to need help learning a new modelling tool they had not used before; this could have affected the feedback on the tool. Many feedback points highlighted design problems with UMLetino, which were outside the project's scope. However, this supports the motivation for developing loose syntax adapters architecture.

4.2 Seeded Errors

4.2.1 Aims

In this section, the model evaluation is compared to a human marker and attempts to answer RQ3: Is it feasible to create a tool that employs the separation of an evaluation method and the evaluation context?

4.2.2 Method

I injected the error into the UXF model files (e.g. a class diagram with unconnected relations, or a use case diagram element)(Figure 4.4). Through this process, 18 seeded errors were created. For this evaluation, I took inspiration from seeding errors within a system from mutation testing [74]. These seeded errors were then run against all the rules or a specific rule being tested, such as property or operation parameters.

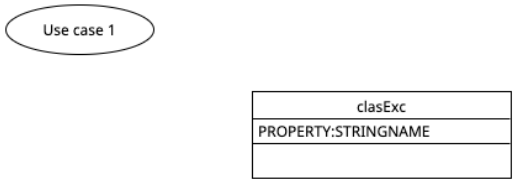

Figure 4.4: Example of a seeded UXF model with a seeded use case diagram element

4.2.3 Data Collection and Cleaning

The seeded errors were created in the UMLetino modelling tool, the same as the tool users, and then downloaded as PNG and UXF files. A Python script then moves the models to the corresponding part of the repository and accepts user input to match the expected error code and context that should be triggered.

4.2.4 Results

The results were categorised into three groups: Successfully returned the expected error code; failed to detect any error; and failed to transform the UXF model to XMI for evaluation. The results are highlighted in Figure 4.5.

These results suggest an area of weakness within the EVL ruleset as it does not detect a number of these errors. This could be due to only a subset of the UML syntax being covered by the base rules (Figure 3.7), meaning many edge cases seeded will not be detected. However, the results also suggest a robust ETL model transformation with only 5.56% failing to create a valid XMI model or flag these exceptions as loose syntax errors. Reflecting on these negative results, the tool requires more work to be of value to users, but this project aimed to create a platform that can be open-source and built upon. For example, context-based rules support more personalised rule changes for self-hosted users due to the separation of method and context, as mentioned in RQ3.

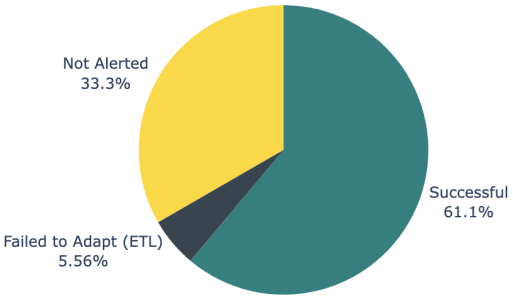

Figure 4.5: Seeded evaluation results

4.3 Human Marker Compared Against Tool

4.3.1 Aims

This evaluation aimed to benchmark the tool against a human marker's ideal solution. This would allow for conclusions about discrepancies between the tool and the human marker. This evaluation also aimed to conclude areas of particular weakness within the tool. This method has been employed in many related projects evaluation methodologies.

4.3.2 Method

This evaluation method used the 2021 SDM (Software Design and Modelling Course) lab assessment submissions test set to evaluate these models against a question created using the question setting platform. The test set was in the form of PNG, which has to be converted into both UXF and XMI to evaluate both the effectiveness of the rules set and the model transformation between UXF and XMI. This was then run against the complete set of generated rules for the question.

4.3.3 Data Collection and Cleaning

All data was collected and stored in JSON files as part of the testing. The tests extract the error codes flagged by the system. These JSON results files were then parsed, and Python scripts generated graphs. The test set also needed to be tagged using the testing platform with the error codes each submission was expected to trigger.

4.3.4 Results

The above graph shows that out of 22 possible errors, nine were caught across 15 students. An identification rate of 41% conveys the need for more reliable base rules. It is also important to note the differences between the UXF results and the XMI-only evaluation, which conveys ETL's failure to correctly represent a model in XMI. Error code one (waterfall connection checking) is the most common error that fails to be identified. It was also discovered a false flag rate of 40% of all the error codes flagged. This is too high for a viable tool, but this result was mainly due to one error, making up

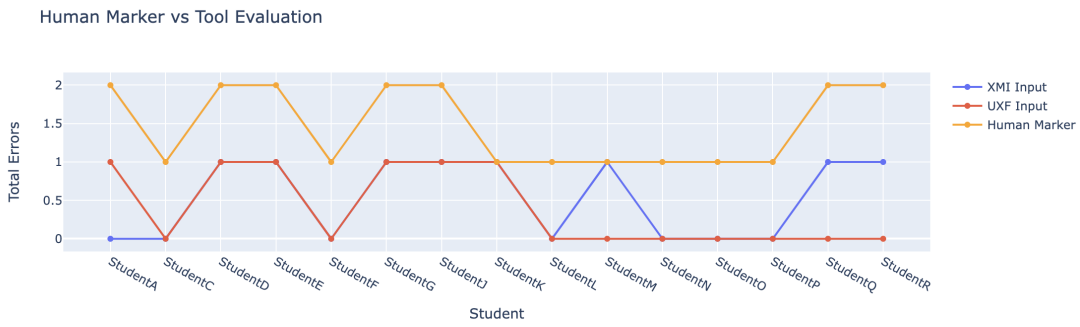

Figure 4.6: Human compared with tool evaluation results

80% of all the false flags. Error code 8, is the multiplicities check due to the lack of implementation of ETL multiplicity handling.

4.3.5 Limitations

One limitation of this evaluation was the decision not to gather the models created by the participants, which removed the need for pre- and post-workshop surveys and ethics forms. If this had been done, the model evaluation similar to that performed in Section 4.3 could have also been conducted on these student-submitted models. The tool can record submissions and connect to an account, but it was turned off for the evaluation workshops.

The size of the dataset was a limitation of the study, with only 25 student submissions tested. Unfortunately, I was unsuccessful in my search for an open-source dataset to use for my evaluation with dataset such as [62], as having a wide range of diagrams created in a range of tools means there would be wide variations from unsupported tools that real users would not see of the tool.

Benchmarking the tool's effectiveness against similar and more mature tools, such as PapyGame, was not achievable within the project's scope, with the need to manually create an identical model in the accepted input form for each tool. This task would be challenging to automate, making the time commitment too large and possibly increasing the chance of inconsistencies. Finally, it should be noted that there is a clear challenge to validity as I created the error scenarios as well as, designing and implementing the tool.

4.4 Question-Setting Evaluation

4.4.1 Aims

This section aimed to evaluate the effectiveness of the question-setting platform and present an initial iteration of the heuristic evaluation of the user interface, with the hope of this being applied more broadly as part of future work.

4.4.2 Heuristic Evaluation

A heuristic evaluation was carried out where an evaluator (myself) reviewed the UI of the question setter page and completed a UAR (usability aspect report) focusing on particular heuristics. For the following review, Nielsen's heuristics [51] were chosen as the preferred set of heuristics due to their high industry adoption. Table 4.2 shows one of the UARs created, while the others can be found in appendix C.

Table 4.2: Evaluation UAR #1

UAR No. 1	**Problem**/Good Aspect
Name	Navigation Menu Unclear
Evidence	Heuristic(s): User Control & Freedom(N-3), Consistency & Standards(N-4) The menu to return to the home screen help page needs clarification, and the tool has a small font size. Ref:4.7
Explanation	The feature was highlighted due to the critical requirement for users to navigate to and from this page, especially for question setters and students. The lack of navigation options means that only returning to the home page is possible. This may also slow users down significantly due to the difficulty of finding it. Finally, this return button has a different colour and style to all other pages, which may increase the cognitive load on the user.
Severity	Impact: Medium - exiting the page can still be performed but is highly degraded. Frequency: High - it is a common task to exit this page after a successful question upload to return to the question page, affecting most users. Persistence: High - this constant feature affects all users regardless of their actions.
Possible Solutions	A possible solution could be creating a consistent navigation bar similar to many standard websites. This could also be adapted to show the current page the user is on and highlight it with an on-hover JavaScript.
Relationships	N/A

4.4.3 User Study

A further user study was conducted on UG4 Poster Day with six undergraduate students to rectify the lack of question-setting evaluation in the student workshops. The survey was carried out under the same ethical approval as the workshop study, as the survey covered the same context. The students were given the following guidance:
Please attempt to create the question using the question-setting platform. Create a question with one class called owner with a public attribute name of type string, and make another class called a car. Connect these two classes with an association. No labels or multiplies are needed.

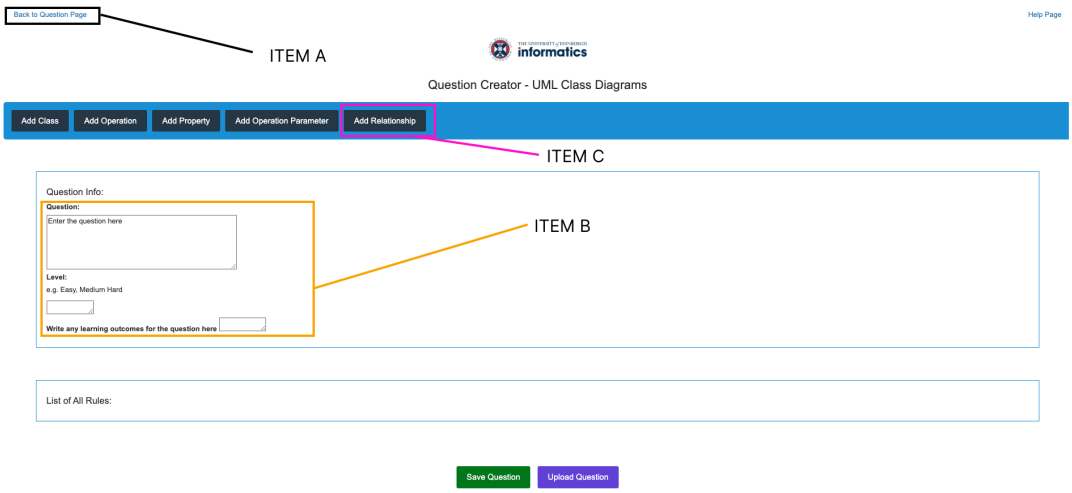

Figure 4.7: Heuristic evaluation supporting screenshot with annotation

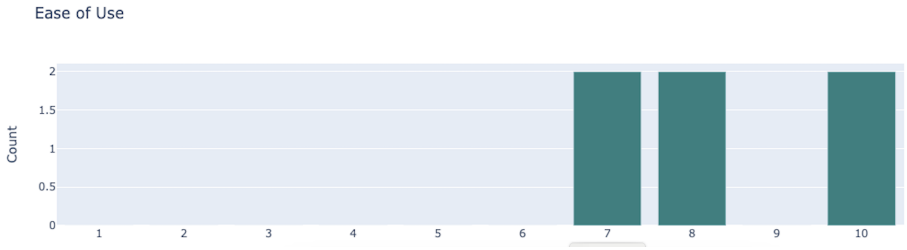

Figure 4.8: Poster day user study ease of use results

The ease of use received an average score of 8.34, which was a positive result, showing the ease of use of the page was high, and the separation of the context was understandable to the participants (Figure 4.8).

4.4.4 Limitations

Many limitations of the user study overlap with the evaluation workshops discussed in Section 4.1.10. Furthermore, another potential factor affecting validity is the framing effect, as each participant had a short discussion with the creator (myself) before being asked to carry out the evaluation, leading to the task being framed in a more positive light. There are also further limitations, such as the heuristic evaluation usually performed by an HCI (human-computer interaction) expert. Instead, I completed these evaluations, using my experience from a HCI university course. However, there are further challenges to validity, such as the same individual carrying out the evaluation and designing the page.

Chapter 5

Conclusions

5.1 Contributions and Research Questions

I have designed and implemented a UML class diagrams feedback tool. I researched related work and possible technologies that could be employed within the tool. I then implemented and developed a rule-based evaluation method that separates the rules and the question context to support my separation of stakeholders. These rules were then implemented into my cloud-based tool supported by my frontend interface. This then led me to develop the project further, designing and implementing a proposed solution to handling a loose syntax tool and transforming these models into XMI so they can also be evaluated. I then conducted evaluation workshops to gather feedback on the tool and refine future requirements. The response to the tool's usefulness was positive, but the ease of use and implementation required further work for the tool to be viable. The project also aimed to answer the following research questions:

RQ1: Which model evaluation techniques are most suitable for educational tools?

Each model evaluation method brings different advantages that suit each usage. My background research found many projects utilising the EVL rule-based evaluation methodology (Section 3.3.1). Rules-based evaluation allows a model to be evaluated for context and conformance with UML syntax. The rule-based evaluation method also has strong determinism. Therefore, rule-based evaluation was selected as the choice for implementation. During my background research, the Epsilon language family was also chosen to utilise the platform's power as laid out in Section 3.2.4.

RQ2: How can the evaluation of loose syntax UML modelling tools be implemented?

I have proposed a possible design for connecting a loose syntax modelling tool to XMI form evaluation in my tool. The translation creates loose syntax rules for when the tool detects uncertain or incorrect modelling elements (Section 3.5). I have implemented a translation between UMLetino and XMI form to allow my modeller to create a model in UMLetino and evaluate these within the tool. This method utilised ETL model transformations and attempted to create an extendable solution to this conceptual problem.

RQ3: Is it feasible to create a tool that employs the separation of an evaluation method and the evaluation context?

I have created a tool that employs the separation of the model context and the evaluation method. The evaluation method has been designed and implemented to allow a modelling expert to create base rules in EVL. Then, a question setter can create the context of a question about the class names, operations and attributes it will have. This method allows for the experience of a modelling expert without the need to code large numbers of rules for each context.

5.2 Limitations of the Tool

One of the limitations of the tool is the frontend design and guidance on how to use the tool. While not the direct focus of the research goal, the user interface prevents practical evaluation and likely reduces the tool's ease of use (Section 4.4.2). The tool's reliability is another limitation, as many participants found that when adding non-UML elements, the system crashes with little feedback. They needed to re-export the diagram, and hope they found the error. This process directly contradicts the project goal of improving the learning experience for modelling.

Another limitation of the tool was the limited base rule coverage of the UML class diagram meta-model, meaning some rules, such as multiplicities, were not implemented at the time of evaluation. Part of this limitation is the need for more ability to handle relationship directions from a loose syntax tool and the layout problems and inconsistencies between each step.

5.3 Future Work

Live Evaluation

During my informal requirements gathering, an ideal solution emerged where the modellers could see live evaluation within their modelling tool, similar to a compiler or syntax checker in an IDE. Furthermore, this feedback should be included in the diagram, building on the work done within the tool. As such, future development could utilise UMLetino to automatically evaluate the models in real-time using the same rules and backend services by intermittently sending the current state of the diagram and the question to the tool. A mock-up of the possible design can be seen in Figure 5.1.

Rule Feature Development

As discussed in Section 3.3.1, the tool only has partial meta-model coverage. As such, future developments could expand the base rule coverage of the class diagram meta-model. Furthermore, the base rules could be developed to support other UML models (sequence, state and activity diagrams). The type of evaluation within the tool could be expanded; for example, new base rules could be defined to evaluate Chidamber and Kemerer's metrics [28], such as the number of methods in a class or the number of classes an object is coupled to. Rules could also focus on different model heuristics

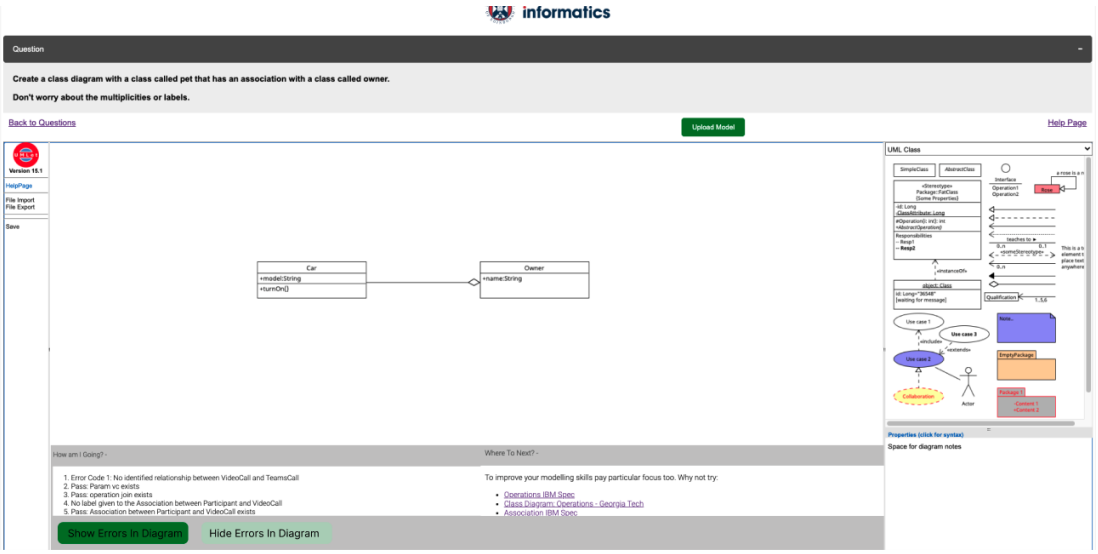

Figure 5.1: Mockup of live evaluation and feedback

and pragmatics, such as number relationships that cross each other and detecting the diamond problem (multiple inheritance). This was initially explored in my entry to the MDEnet competition (Section 3.3.4).

Meta-Model Design

The structure and semantics of UML are represented in MOF (meta object facility). As such, the ideal architecture of the tool would be a representation of the UML class meta-model that transforms into the schema for the question-setting platform, the base rules and the EGL diagram generations. Using one meta-model as the single source of truth for each aspect of the tool would allow for easy updates to a model's semantics and extendibility to support more diagrams by implementing their meta-models.

Evaluation Study

As identified within the evaluation of the tool, continuous evaluation (a longitudinal study) would be required to say confidently if the tool improves learning outcomes for modelling students. The tool could also be further evaluated during a study of this kind, focusing on the types of feedback templates that best suit learners. This would require research into the feasibility of the ethical approval for creating a tool and control group, as discussed further in Section 4.1.10. More evaluation could be conducted with modelling educators to gather additional requirements and feedback on the usefulness of the created tool from these important stakeholders.

UI Improvements

User interfaces are a vital aspect of any tool; great interfaces should be able to guide beginner users and support expert users with additional functionality. As discussed in the evaluation, the frontend design was a primary barrier to the tool's ease of use and the effectiveness of the evaluation workshops. As such, further user-centred design work should be conducted with a redesign of the front end attempting to apply Neilson Heuristics [51] and evaluating with both users and experts, possibly performing

cognitive walkthroughs. This could be supported by the current swagger documentation covering all APIs for the backend, making a redesign of the frontend more streamlined [33].

Layout and Bidirectional ETL

A possible solution to the inconsistent layout (Section 3.6.2) could be to create a standardised specification for the layout and then use ETL to translate each tool's format to the standard layout format, which can then be used to visualise models in a similar way they were created will still be supporting in diagram errors. This would allow the tool to move towards making observations on more pragmatics of modelling, for example, how many relations cross one another. The implementation still needs to address this due to the large undertaking that requires defining a standard and finding a new online diagram render as PlantUML uses GraphViz, an auto layout engine incompatible with the proposed solution. Another possible route involves creating bidirectional transformations from UXF to UML, which could then be used to output the model with the error added to it without the requirement to write two separate bodies of code for input and output. This is similar to other MDE languages, such as QVT-Relations (query, view, transformation), which supports bidirectional transformations.

To conclude, despite mixed evaluation results, the project has succeeded in achieving its primary goals, including proposing and implementing feedback templates, separating the question context and evaluation methodology, and transforming loose syntax models to XMI form. These contributions are utilised in the presented cloud-based tool.

Bibliography

[1] Model Validation (EVL) - Epsilon version 2.4. Available from: https://eclipse.dev/epsilon/doc/evl/.

[2] TouchCORE Software Introduction [software]. Accessed on: 11th Dec 2023, Available from: https://touchcore.cs.mcgill.ca/.

[3] UML Class Diagrams : Similarity Aspects and Matching Lecture.

[4] Unified Modeling Language Specification Version 2.5.1.

[5] Does UML make the grade? Insights from the software development community. *Information and Software Technology*, 47(6):383–397, April 2005.

[6] XML Metadata Interchange Specification Version 2.5.1, 2015. Accessed on: 10th Oct 2023, Available from: https://www.omg.org/spec/XMI/.

[7] Rational Rose [software] version 7.0.0.4, March 2021. Accessed on: 24th Oct 2023, Available from: https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=migration-rational-rose-model.

[8] Mdenet annual symposium 2023. In *Proceedings of the Annual Symposium on Model-Driven Engineering*, 2023.

[9] UK 18-year-olds make record number of applications for computing courses, July 2023. Accessed on: 14th Nov 2023.

[10] Peerwise web application [software], 2024. Accessed on: 12th Feb 2024, Available from: https://peerwise.cs.auckland.ac.nz/.

[11] Quizlet app [software], 2024. Accessed on: 12th Feb 2024, Available from: https://quizlet.com/gb.

[12] Noraida Haji Ali, Zarina Shukur, and Sufian Idris. A Design of an Assessment System for UML Class Diagram. In *2007 International Conference on Computational Science and its Applications (ICCSA 2007)*, pages 539–546, August 2007.

[13] Driss Allaki, Mohamed Dahchour, and A. En-Nouaary. A New Taxonomy of Inconsistencies in UML Models : with their Detection Methods for Better MDE. *Int. J. Comput. Sci. Appl.*, 2015.

[14] Driss Allaki, Mohamed Dahchour, and Abdeslam En-Nouaary. A Constraint-based Approach for Checking Vertical Inconsistencies between Class and Sequence UML

Diagrams. In *Proceedings of the 18th International Conference on Enterprise Information Systems*, ICEIS 2016, pages 441–447, Setubal, PRT, April 2016. SCITEPRESS - Science and Technology Publications, Lda.

[15] Thorsten Arendt and Gabriele Taentzer. Integration of smells and refactorings within the Eclipse modeling framework. In *Proceedings of the Fifth Workshop on Refactoring Tools*, WRT '12, pages 8–15, New York, NY, USA, June 2012. Association for Computing Machinery.

[16] Luciano Baresi and Mauro Pezzè. Improving UML with Petri nets. *Electronic Notes in Theoretical Computer Science*, 44(4):107–119, July 2001.

[17] Will Barnett, Steffen Zschaler, Artur Boronat, Antonio Garcia-Dominguez, and Dimitris Kolovos. An online education platform for teaching mde. In *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 114–121, 2023.

[18] Weiyi Bian, Omar Alam, and Jörg Kienzle. Automated Grading of Class Diagrams. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 700–709, September 2019.

[19] D. Bogdanova and M. Snoeck. Let's Automate! Making Use of a Learning Ontology for Conceptual Data Modelling. 2019.

[20] Daria Bogdanova and Monique Snoeck. Use of Personalized Feedback Reports in a Blended Conceptual Modelling Course. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 672–679, September 2019.

[21] Younes Boubekeur, Gunter Mussbacher, and Shane McIntosh. Automatic assessment of students' software models using a simple heuristic and machine learning. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '20, pages 1–10, New York, NY, USA, October 2020. Association for Computing Machinery.

[22] Antonio Bucchiarone, Maxime Savary-Leblanc, Xavier Le Pallec, Jean-Michel Bruel, Antonio Cicchetti, Jordi Cabot, and Sébastien Gérard. Gamifying model-based engineering: The PapyGame tool. *Science of Computer Programming*, 230:102974, August 2023.

[23] Antonio Bucchiarone, Maxime Savary-Leblanc, Xavier Le Pallec, Antonio Cicchetti, Sébastien Gérard, Simone Bassanelli, Federica Gini, and Annapaola Marconi. Gamifying model-based engineering: the PapyGame experience. *Software and Systems Modeling*, 22(4):1369–1389, August 2023.

[24] Euan Chalmers. How to's epsilon, 2024. Accessed: 12th Mar 2024, Available from: http://euanchalmers.s3-website-eu-west-1.amazonaws.com/.

[25] Euan Chalmers. Uml tool demo video [internet], available from: https://www.youtube.com/watch?v=5sxyglw9-y0, 2024.

[26] Euan Chalmers. Evl tool base rules, 2024, February 12. Accessed: 12th Mar 2024, Available from: https://euanchalmers.s3.eu-west-1.amazonaws.com/FE/BaseRulesPage.html.

[27] Euan Chalmers. Uml feedback tool - question setting platform, 2024, February 12. Accessed on 12th Feb, Available from: https://euanchalmers.s3.eu-west-1.amazonaws.com/FE/RuleCreationPage.html.

[28] S.R. Chidamber and C.F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

[29] Stanislav Chren, Barbora Buhnova, Martin Macak, Lukas Daubner, and Bruno Rossi. Mistakes in UML Diagrams: Analysis of Student Projects in a Software Engineering Course. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 100–109, May 2019.

[30] Eclipse Foundation. *Eclipse Papyrus [software] Version 6.6.0*. Eclipse Foundation, 2022. Accessed on: 21th Nov 2023, Available from: https://www.eclipse.org/papyrus/.

[31] Holger Eichelberger. *Aesthetics and Automatic Layout of UML Class Diagrams*. PhD thesis, Bayerischen Julius-Maximilians-Universität Würzburg, Würzburg, 2005.

[32] John Erickson. A decade and more of uml: An overview of uml semantic and structural issues and uml field use. *Journal of Database Management*, 19(3), 2008.

[33] Euan Chalmers. Uml feedback tool swagger docs, 2024. Accessed: April 1, 2024, Available from: https://euanchalmers.s3.eu-west-1.amazonaws.com/FE/swagger.html.

[34] Andy Evans and Stuart Kent. Core Meta-Modelling Semantics of UML: The pUML Approach. In Robert France and Bernhard Rumpe, editors, *UML'99 — The Unified Modeling Language*, Lecture Notes in Computer Science, pages 140–155, Berlin, Heidelberg, 1999. Springer.

[35] Therese Fessenden. Net promoter score: What a customer-relations metric can tell you about your user experience, 2016.

[36] J. W. Gikandi, D. Morrow, and N. E. Davis. Online formative assessment in higher education: A review of the literature. *Computers & Education*, 57(4):2333–2351, December 2011.

[37] John Hattie and Helen Timperley. The Power of Feedback. *Review of Educational Research*, 77(1):81–112, March 2007.

[38] Pavel Herout and Premek Brada. UML-Test Application for Automated Validation of Students' UML Class Diagram. pages 222–226, April 2016. ISSN: 2377-570X.

[39] Gerard Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 1st edition, April 2011.

[40] Kinsta. Browser market share report, 2023. Accessed on: 2th Feb 2024, Available from: https://kinsta.com/browser-market-share/.

[41] Dimitris Kolovos and Antonio Garcia-Dominguez. The epsilon playground. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 131–137, 2022.

[42] Dimitris Kolovos, Louis Rose, Antonio García-Domínguez, and Richard Paige. *The Epsilon Book*. Epsilon, July 2018.

[43] Helena Kruus, Tarmo Robal, and Gert Jervan. Teaching modeling in SysML/UML and problems encountered. In *2014 25th EAEEIE Annual Conference (EAEEIE)*, pages 33–36, May 2014.

[44] Grischa Liebel, Rogardt Heldal, and Jan-Philipp Steghöfer. Impact of the Use of Industrial Modelling Tools on Modelling Education. In *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*, pages 18–27, April 2016. ISSN: 2377-570X.

[45] J. Lilius and I.P. Paltor. vUML: a tool for verifying UML models. In *14th IEEE International Conference on Automated Software Engineering*, pages 255–258, October 1999.

[46] Yuting Lu and Cristina Adriana Alexandru. Systematic review of uml diagramming software tools for higher education software engineering courses. In *Proceedings of the 2023 Conference on United Kingdom & Ireland Computing Education Research*, UKICER '23, New York, NY, USA, 2023. Association for Computing Machinery.

[47] Paul Mihas. Qualitative research methods: approaches to qualitative data analysis. In Robert J Tierney, Fazal Rizvi, and Kadriye Ercikan, editors, *International Encyclopedia of Education (Fourth Edition)*, pages 302–313. Elsevier, Oxford, fourth edition edition, 2023.

[48] Salisu Modi, Hanan Abdulrahman Taher, and Hoger Mahmud. A Tool to Automate Student UML diagram Evaluation. *Academic Journal of Nawroz University*, 10(2):189–198, June 2021.

[49] Rabah Mokhtari. Validation of UML Class Diagram and OCL pre-and post-conditions using OTS/CafeOBJ proof scores. In *2020 4th International Symposium on Informatics and its Applications (ISIA)*, pages 1–4, December 2020.

[50] Anton J Nederhof. Methods of coping with social desirability bias: A review. *European journal of social psychology*, 15(3):263–280, 1985.

[51] Jakob Nielsen and Rolf Molich. Heuristic evaluation of user interfaces. In *Proc. ACM CHI'90 Conf.*, pages 249–256, Seattle, WA, April 1-5 1990.

[52] Object Management Group. *Unified Modeling Language Specification*. OMG, version 2.5.1. edition, 2024. Accessed on: April 1, 2024, Available from: https://www.omg.org/spec/UML/.

[53] OMG. Omg meta object facility (mof) core specification, version 2.4. 1, 2013.

[54] R. Paige, F. Polack, D. Kolovos, Louis M. Rose, N. Matragkas, and James R. Williams. Bad Modelling Teaching Practices. 2014.

[55] Marian Petre. UML in practice. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 722–731, May 2013. ISSN: 1558-1225.

[56] Wendell Piez and Debbie Lapeyre. Introduction to schematron. *IDEAlliance*, 90(1.0):Pages, 2008. Version 90-1.0 (November 2008).

[57] Ann Poulos and Mary Jane Mahony. Effectiveness of feedback: the students' perspective. *Assessment & Evaluation in Higher Education*, 33(2):143–154, April 2008.

[58] Helen C. Purchase, Jo-Anne Allder, and David Carrington. Graph layout aesthetics in uml diagrams: User preferences. *Journal of Graph Algorithms and Applications*, 6(3):255–279, 2002.

[59] Tobias Reischmann and Herbert Kuchen. A web-based e-assessment tool for design patterns in UML class diagrams. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, pages 2435–2444, New York, NY, USA, April 2019. Association for Computing Machinery.

[60] Rebecca Reuter, Theresa Stark, Yvonne Sedelmaier, Dieter Landes, Jürgen Mottok, and Christian Wolff. Insights in Students' Problems during UML Modeling. In *2020 IEEE Global Engineering Education Conference (EDUCON)*, pages 592–600, April 2020. ISSN: 2165-9567.

[61] Mark Richters and Martin Gogolla. *OCL: Syntax, Semantics, and Tools*. Springer, 2024.

[62] Gregorio Robles, Truong Ho-Quang, Regina Hebig, Michel R.V. Chaudron, and Miguel Angel Fernandez. An extensive dataset of uml models in github. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 519–522, 2017.

[63] Wuwei Shen, K. Compton, and J. Huggins. A toolset for supporting UML static and dynamic model checking. In *Proceedings 26th Annual International Computer Software and Applications*, pages 147–152, August 2002. ISSN: 0730-3157.

[64] Ven Yu Sien. An investigation of difficulties experienced by students developing unified modelling language (UML) class and sequence diagrams. *Computer Science Education*, 21(4):317–342, December 2011.

[65] Williamson Silva, Bruno Gadelha, Igor Steinmacher, and Tayana Conte. Towards an Open Repository for Teaching Software Modeling applying Active Learning Strategies. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 162–172, October 2020.

[66] JSONForm Contributors [software] Version 2.2.5. Jsonform, Jan 2024. Accessed on: 11th Oct 2023, Available from: https://github.com/jsonform/jsonform.

[67] Arnor Solberg, Robert France, and Raghu Reddy. Navigating the metamuddle.

[68] Dave R. Stikkolorum, P. V. D. Putten, Caroline Sperandio, and M. Chaudron. Towards Automated Grading of UML Class Diagrams with Machine Learning. 2019.

[69] D.R. Stikkolorum, Truong Ho-Quang, and M.R.V. Chaudron. Revealing Students' UML Class Diagram Modelling Strategies with WebUML and LogViz. In *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, pages 275–279, August 2015. ISSN: 2376-9505.

[70] Thymeleaf Team. *Thymeleaf: Modern server-side Java template engine for web and standalone environments [software]*. Thymeleaf, 2024. Accessed on: 24th Feb 2024, Available from: https://www.thymeleaf.org/.

[71] M. Termeer, C.F.J. Lange, A. Telea, and M.R.V. Chaudron. Visual exploration of combined architectural and metric information. In *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 1–6, 2005.

[72] Umlet. Umletino: Uml modeling tool [software]. Last commit 4 months ago, Available from: https://umletino.com/.

[73] Bhuvan Unhelkar. *Verification and Validation for Quality of UML 2.0 Models*. Wiley, 1 edition, July 2005.

[74] Filip Van Laenen. Mutation testing. *Overload Journal*, (108):16–22, 2012.

[75] Eclipse Web. Eclipse Papyrus™ [software], January 2013. Available from: https://projects.eclipse.org/projects/modeling.mdt.papyrus.

[76] Shengnan Zhang, Yan Hu, and Guangrong Bian. Research on string similarity algorithm based on Levenshtein Distance. In *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 2247–2251, March 2017.

Appendix A

Class Diagram

A.1 Overview

UML class diagrams are important in object-oriented software development. A class diagram is a static structural UML model, as seen in the model below, and may contain each of the following:

1. Class - can represent an object in OOP (object-oriented programming) (Person, Vehicle, Car, Truck).
2. Attributes - are data within that object and found in the middle compartment of a class (e.g. name, registration, etc.).
3. Operations - sit within the third compartment of a class and convey any logic within that class.
4. Relation - any logical connection between classes.

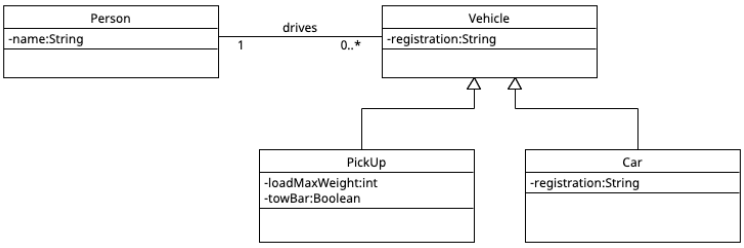

Figure A.1: A simple UML class diagram

A.2 Visibility

Visibility coveys how the attribute or operation can be accessed can be:

1. Private (-)
2. Public (+)

3. Protected (#)

A.3 Relations

Relations are connections between class elements; they can be of the following types:

1. **(Simple) Association** - can be named or labelled and have ends, which can also be named or have multiplicities attached. They can also be navigable in one or more directions. (with an arrowhead).
2. **Dependency** - exists where changes to a target element will affect the changes to the source.
3. **Aggregation** - aggregation is a complex type of association that can represent a ”part of” a relationship with the unfilled diamond. Meanwhile, composition is a stronger form of association usually used to signify an element that should not exist alone, for example, duck and pond. If the container (pond) is destroyed, the contents (duck) are also destroyed. These are not discussed in the dissertation.
4. **Generalization/Inheritance** - used to represent objects that are a subclass of a superclass. Think, for example, a superclass of pet and of a subclass of dog and cat.
5. **Realization/Implementation** - one class/object (client) implements the behaviour of the other model element (the supplier).

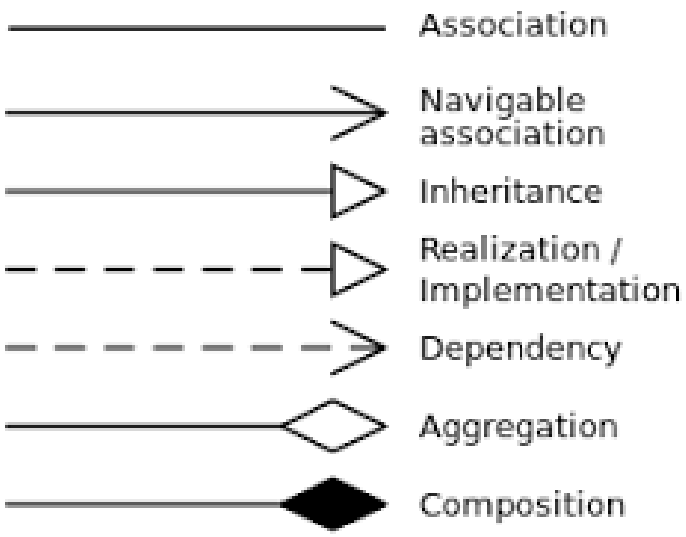

Figure A.2: Types of relations In UML class diagrams

A.4 Multiplicities

Multiplicities add detail to relations and are an optional notion that can be placed at each end of an association:

- 0..1 — no instances, or one instance
- 1 — exactly one
- 0..* or * — zero or more
- 1..* — one or more

This is a subsection of the UML class diagram elements with many further diagram elements omitted, but it covers the required knowledge for this dissertation.

Appendix B

Research into Courses

When researching background research, I wanted to understand the prevalence of UML within undergraduate courses. As such, I conducted my own research. Firstly, I used an open data set of 152 universities in the UK. I then used a Python script to search for possible web pages for UML courses. This was then inputted into OpenAI GPT 3.5 to evaluate if this page would be likely to indicate a UML course. Out of 152 scanned 46 were positively identified by the AI. At the same time, I evaluated the rest manually, leading to the results of any university that had slides, readings, or clear public course pages referring to UML. Manually, 33 were found to have UML courses, meaning 79/152 (52%) of UK universities have UML courses.

Furthermore, I also randomly selected 10% of the AI identification to check they were valid with an error rate of 6.6%.

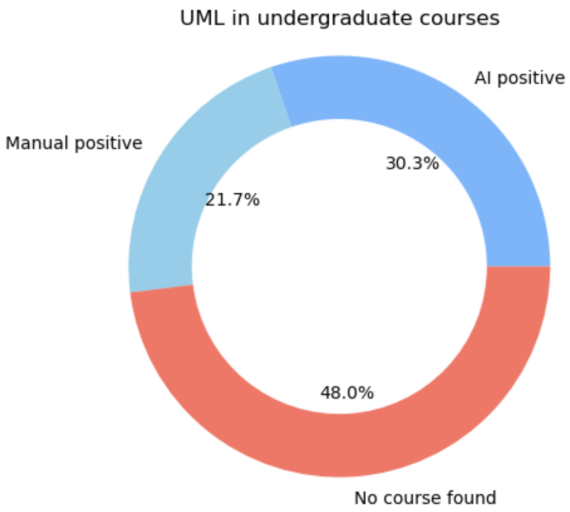

Figure B.1: Results of research into UML prevalence in undergraduate courses in the UK

Appendix C

Question Setting UAR Evaluation

Table C.1: Evaluation UAR #2

UAR No. 2	**Problem**/Good Aspect
Name	Inconsistent input box styling and layout
Evidence	Heuristic(s): Consistency & Standards(N-4), Aesthetic & Minimalist (N-8) Ref: 4.7
Explanation	‘Consistency & Standards are key aspect for any website as consistent interfaces reduce cognitive load and increase the speed of use for expert users. This feature was highlighted due to the inconsistencies on the page. As seen in the screenshot, there are some input boxes that have on hover highlighting while others do not. Furthermore, some input boxes are on the same line as the label, while others are below. This could confuse users and lead to increased input error rate.
Severity	Impact: Medium - this is a medium impact problem as it will affect most users due to its prevalence on the main form highlighted and the popup forms discussed in the dissertation. Frequency: Low - this is low severity as it does not prevent the completion of any tasks. Persistence: High - this will affect all users as it affects all input boxes, including required ones such as the description of the question etc.
Possible Solutions	Standardise the input boxes to ensure a clean and consistent design for users. This could be done with a standardised component with external CSS (similar to react components).
Relationships	N/A

Table C.2: Evaluation UAR #3

UAR No. 3	**Problem**/Good Aspect
Name	Lack of Useful Error Feedback To User
Evidence	Heuristic(s): Recognise and Recover from errors(N-5), Help and Documentation(N-10) Ref:4.7
Explanation	The webpage does not provide error messages to the user at any point, excluding submission. As such, some users may need help understanding that they first need to create a class to be able to add a property within a class. A clear human-readable error message could help guide this user and help them recover from errors.
Severity	Impact: Medium - this is a problem as some user may fail to understand the situation at all and abandon the task. In contrast, others with more modelling experience may overcome this more quickly. This means on average, the severity is medium. Frequency: Medium - this will occur when a user tries to create any element when a class has not yet been designed, meaning the frequency could be high. Persistence: Low- this can be overcome and will become less of an issue for repeat users as they will recall the need to define classes before properties, etc.
Possible Solutions	The solution to this could be adding alert error codes when a task is impossible. Additionally, help guidance could be added to the areas where users struggle most. Furthermore, the tool could have a tutorial mode where first-time users receive additional guidance. This could also be implemented across other pages for consistency.
Relationships	N/A

Appendix D

Complex ETL Code Example

Displayed on next page due to formatting.

```
Model {
    constraint checkTypeofConnection {
    check {
        var classOneNames : Sequence = Sequence{%%listClassNamesOne
            %%};
        var classTwoNames : Sequence = Sequence{%%listClassNamesTwo
            %%};
        for (classTwo in classTwoNames) {
            for (classOne in classOneNames) {
                if (Association.all.select(elem | elem.
                    getRelatedElements().exists(elem | elem.name =
                    classOne)
                     and elem.getRelatedElements().exists(elem | elem
                         .name = classTwo)).size() == 1) {
                     return true;
                }
            }
        }
        return false;
        }
    message {
        var msg: String;
        for (classTwo in classTwoNames) {
            for (classOne in classOneNames) {
                if (Generalization.all.select(elem | elem.getTargets
                    ().exists(elem | elem.name = classOne) and
                elem.getSources().exists(elem | elem.name = classTwo
                    )).size() == 1 or
                Generalization.all.select(elem | elem.getTargets().
                    exists(elem | elem.name = classTwo)
                and elem.getSources().exists(elem | elem.name =
                    classOne)).size() == 1) {

                     return " Error X: No Association exists between
                         " + classOne + " and " + classTwo + " but a
                         Generalization does";
                } else if (Realization.all.select(elem | elem.
                    getRelatedElements().exists(elem | elem.name =
                    classOne) and elem.getRelatedElements().exists(
                    elem | elem.name = classTwo)).size() == 1) {
                     return "Error 5: No Generalization but a
                         Realization exists " + Realization.all;
                } else {
                     msg = "Error 1: No relationship between " +
                         classOneNames[0].toString() + " and " +
                         classTwoNames[0].toString() + "";
                }
            }
        }
        return msg;
    }
    }
}
```

Figure D.1: An example of an EVL rule

Appendix E

Ethics Forms

E.1 Participants' Information Sheet

This study was certified according to the Informatics Research Ethics Process, reference number 302345. Please take time to read the following information carefully. You should keep this page for your records.

Who are the researchers?

This research is conducted by Euan Chalmers as part of his undergraduate project. This project is supervised by Perdita Stevens. Contact details can be found above.

What is the purpose of the study?

The purpose of this study is to gather feedback from current undergraduate students with a range of software modelling experiences through an interactive workshop. The results of the study will be used to shape the requirements for my ug4 project, which is a feedback tool for student learning UML modelling. The results of further workshops will help evaluate a prototype of the web tool I have created.

Why have I been asked to take part?

You have been asked to take part in this study as a current undergraduate student studying an informatics degree.

Do I have to take part?

No – participation in this study is entirely up to you. You can withdraw from the study at any time, up until 5th March 2024 without giving a reason. After this point, personal data will be deleted, and anonymized data will be combined, making it impossible to remove individual information from the analysis. Your rights will not be affected. If you wish to withdraw, contact the PI. We will keep copies of your original consent and your withdrawal request.

What will happen if I decide to take part?

You will take part in a short workshop (30min approx.) where you will:

Requirements Gathering Workshop:

- Introduction & Signing of Consent forms.
- Short introduction survey on students' current knowledge of UML.
- A brain teaser UML question will be given to the student (this will be aggregated, anonymized, and may be used to make assumptions about common mistakes made by students and as a test set for the tool) (this will be made aware to students).
- The students will then be shown a low-fidelity paper design of my web tool, and they will be asked to stick post-it notes on the design of what they like and don't like. (This will then be anonymized and used by the to perform affinity diagramming task to extract the most important requirements for my system).
- Thank the participant & remind them of the right to withdraw.

Evaluation Semi-Structured Workshop:

- Introduction & Signing of Consent forms.
- Short introduction survey (same as above).
- A brain teaser UML question will be given to the student (This is used to remind the student about UML modeling & will also be used to pass into the prototype tool to generate personalized feedback).
- Students will then be encouraged to use the web prototype further, exploring freely or following the tutorial.
- They will then receive a survey asking them about the experience (e.g., how helpful different elements of the feedback were how confident they feel at different UML tasks).
- Thank participant & remind of the right to withdraw.

Further Info (applies to both workshop types):

- It will be made clear in any communication and at the start of each workshop which of the two workshops you will be taking part in.
- Workshops will not be video or audio recorded.
- The workshop(s) will take place in bookable classrooms at the University of Edinburgh Campus.
- Workshops will be held in 2 main 'waves' as split out above:
  - To gather requirements (Nov 2023).
  - To evaluate the prototype created (Jan or Feb 2024).
- Compensation: No monetary compensation will be provided for taking part in this study, but tea and coffee may be provided at some workshop sessions.

Are there any risks associated with taking part?

There are no significant risks associated with participation. There are no significant risks identified with the study.

Are there any benefits associated with taking part?

Compensation (Tea and Coffee may be provided classroom dependent).

What will happen to the results of this study?

The results of this study may be summarized in published articles, reports, and presentations. Quotes or key findings will be anonymized: We will remove any information that could, in our assessment, allow anyone to identify you. With your consent, information can also be used for future research. Your data may be archived for a maximum of 4 years. All potentially identifiable data will be deleted within this timeframe if it has not already been deleted as part of anonymization.

Data protection and confidentiality.

Your data will be processed in accordance with Data Protection Law. All information collected about you will be kept strictly confidential. Your data will be referred to by a unique participant number rather than by name. Your data will only be viewed by the researcher/research team: Euan Chalmers (lead researcher) & Perdita Stevens (Project Supervisor).

All electronic data will be stored on a password-protected encrypted computer, on the School of Informatics' secure file servers, or on the University's secure encrypted cloud storage services (DataShare, ownCloud, or Sharepoint), and all paper records will be stored in a locked filing cabinet in the PI's office. Your consent information will be kept separate from your responses to minimise risk.

What are my data protection rights?

The University of Edinburgh is a Data Controller for the information you provide. You have the right to access information held about you. Your right of access can be exercised in accordance with Data Protection Law. You also have other rights, including rights of correction, erasure, and objection. For more details, including the right to lodge a complaint with the Information Commissioner's Office, please visit `www.ico.org.uk`. Questions, comments, and requests about your personal data can also be sent to the University Data Protection Officer at `dpo@ed.ac.uk`.

Who can I contact?

If you have any further questions about the study, please contact the lead researcher, Euan Chalmers at [email].

If you wish to make a complaint about the study, please contact `inf-ethics@inf.ed.ac.uk`. When you contact us, please provide the study title and detail the nature of your complaint.

Updated information

If the research project changes in any way, an updated Participant Information Sheet will be made available on `http://web.inf.ed.ac.uk/infweb/research/study-updates`.

Alternative formats

To request this document in an alternative format, such as large print or on coloured paper, please contact Euan Chalmers at [email].

General information

For general information about how we use your data, go to: `edin.ac/privacy-research`.

E.2 Participants' Consent Form

The following was the participant consent form used when conducting the evaluation workshops.

By participating in the study you agree that:

- I have read and understood the Participant Information Sheet for the above study, that I have had the opportunity to ask questions, and that any questions I had were answered to my satisfaction.
- My participation is voluntary, and I can withdraw at any time without giving a reason. Withdrawing will not affect any of my rights.
- I consent to my anonymised data being used in academic publications and presentations.
- I understand that my anonymised data will be stored for the duration outlined in the Participant Information Sheet.

Please tick yes or no for each of these statements.

1. I allow my data to be used in future ethically approved research.
2. I agree to take part in this study.