RESEARCH TOPICS 2026-02-20

AUTOMATIC ANALYSIS AND GRADING OF UTML UML DIAGRAMS

Douwe Osinga
d.r.osinga@student.utwente.nl

Supervisor
dr. ir. Vadim Zaytsev
v.zaytsev@utwente.nl

Supervisor
dr. Nacir Bouali
n.bouali@utwente.nl

Faculty of Electrical Engineering,
Mathematics, and Computer Science

UNIVERSITY OF TWENTE.

ABSTRACT

During computer science studies, students are often required to submit diagrams. The grading of these diagrams is currently done by humans, resulting in a costly, lengthy, and error-prone process. In this paper, we investigate the theoretical feasibility of automatically grading diagrams, focusing on Unified Modelling Lanuage diagrams and the UTML file format used by the University of Twente. Existing work shows that graph isomorphism algorithms which account for the use of synonyms and the presence of spelling mistakes provide the best results, but that existing autograders are not adaptable UTML format. Based on these findings, we propose *Seshat*, an autograder that combines the aforementioned techniques into a generic autograder capable of supporting arbitrary diagrams and file formats, including UML diagrams or Entity-Relation diagrams stored in arbitrary formats, with built-in support for UTML. In the final thesis, we realise *Seshat* and compare it to human grading for multiple UTML exam submission datasets.

1. INTRODUCTION

Unified Modelling Language (UML) diagrams, introduced by the Object Management Group [1], play a significant role in computer science, as they allow for communicating software designs in a standardised format. During technical studies, students are often required to make UML diagrams for graded assignments or exams.

However, the grading of these diagrams is often a costly and lengthy process, involving multiple paid members of staff [2][1]. Additionally, this process is prone to grading inconsistencies due to various reasons [2], but mainly the inconsistency of human graders [2] [3]. M. Meadows *et al.* [3] pose two possible solutions to the problem of human grading: either "report the level of reliability associated with marks/grades, or find alternatives to [grading]." We propose a third alternative: finding alternatives to the grading *process*.

The (partial) automatisation of grading diagrams ('autograding') provides a grading paradigm that can both reduce the cost and time required for institutions and reduce the inherently present inconsistencies in human grading[2] [4] [5]. This could result in similar or superior performance compared to human grading in terms of **accuracy**, **grading transparency**, and **consistency**.

With *accuracy*, we mean the percentage of points assigned to a submission that are prescribed by the rubric for a particular excercise. With *consistency*, we mean both the extent to which similar grades are given to similar submissions and the difference between consecutive runs (i.e. determinism). With *grading transparency*, we mean the extent to which the reasoning for a particular grade is explained with regards to the rubric for the exercise or to the Intended Learning Objectives (ILOs) of a module. These properties are desirable in the grading process, as it means that students are graded in a way that reflects their performance (*accuracy*), allows them to see which parts they could improve for future assignments (*grading transparency*), and is minimally unfair (*consistency*).

Specifically for the implementation, we desire an autograder that can *link its grading to ILOs*, as described by the previous paragraph. Furthermore, built-in *UTML support* is a must, as it is the main file format the University of Twente uses. Finally, an *easily extensible* autograder would be ideal, since we might want to extend support to other file formats or alter the behaviour of the autograder.

1.1. Background

The idea of letting a computer program (partially) grade tests has been discussed in papers since the 70s [6, p.13], with some papers with implementations starting to appear around the 80s, primarily focused on grading the writing style of computer programs [7]. Interest in grading diagrams specifically seems to have started around the early 2000s [8] [9].

[1] From personal experience.

[2] Given that the process is deterministic

Diagrams themselves have multiple variants for different purposes. UML diagrams, for example, mainly serve to visualise and document software [1], while Entity-Relation diagrams focus on the relations between different components, making it ideal for visualising database designs [10].

Different formats exist for storing these diagrams. Examples include XMI - the standard diagram interchange format for UML, most commonly used by the Eclipse Modelling Framework [11], the Rose Petal format - used by the UML development program IBM Rational Rose [12], PlantUML - an open-source textual standard for representing various diagrams including UML and ER diagrams [13], Visual Paradigm files - software that allows for modelling UML, architecture diagrams, business flows etc. [14], and UTML - an in-house developed program and file format used at the University of Twente for representing UML and various other types of diagrams, building on the JSON standard [15] [16].

The degree to which automated grading is implemented can vary as well. We divide autograding into the following categories: *non-automated* (everything must be done by a human), *automated* (part of the process requires no human input), and (fully) *automatic* (no human input is required). In this paper, we only consider autograders that fall into the categories *automated* and *automatic*.

1.2. Research Questions

In order to examine the feasibility of automatically grading UML diagrams saved in the UTML format, we provide a main research question (**MRQ**):

To what extent can UML diagrams be graded automatically while maintaining or improving the accuracy, consistency, and grading transparency of human grading?

We aim to answer the main research question with the following research questions:

RQ1: To what extent are existing solutions suitable for use in autograding UTML diagrams with regards to (1) accuracy, (2) consistency, (3) grading transparency, (4) extent of linking ILOs to grading instructions, (5) UTML support, and (6) ease of extending the source code to alter functionality?

RQ2: To what extent can a suitable autograder be constructed from previous work to be able to grade UTML diagrams?

RQ3: To what extent does the suitable autograder compare to human grading in the context of grading first-year UML exam questions?

RQ1 is answered in Section 2, by analysing these works for suitability of grading. Existing works are categorised according to the requirements outlined by **RQ1** in Table 1. Section 3 explains the plan for building the autograder and Section 4 outlines the planning for research questions **RQ2** and **RQ3**, which are to be answered in the final thesis, where we grade UTML diagrams using an implementation based on related work and compare it to human grading.

2. RELATED WORK

In order to answer research questions **RQ1**, we conduct a small-scale literature study covering roughly 40 works. It aims to provide an exploratory view into the world of autograders and ILOs, which is why we omit formal inclusion and exclusion criteria. Works are collected using the search engines Google Scholar[3] and ResearchGate[4], with related work for autograders being searched with terms including but not limited to “automatically grading UML diagrams”, “autograder diagram”, “UML diagram assessment”, “machine learning diagrams”, and “diagram evaluation assessment AI”. For papers on ILO and ILO integration into rubrics, terms were used such as “learning outcomes include in rubric”, “learning objectives in rubrics”, and similar. Snowballing (the practice of looking at sources of sources) was used to a depth of 1.

[3] https://scholar.google.com
[4] https://www.researchgate.net

2.1. Autograders

Multiple types of papers on autogrades are researched, which we categorise into frameworks for autograders, purely algorithmic autograders, and AI-driven autograders (using Machine Learning (ML) / Generative AI (GenAI) techniques). Findings on autograder implementations are summarised in Table 1.

2.1.1. Frameworks / Theoretical

Autograder frameworks dictate certain designs or methodologies for building autograders. We present summaries of the explored frameworks and provide a general summary of all frameworks.

N. Smith *et al.* [8] provide a five-step framework for assessing "possibly ill-formed or inaccurate diagrams" that include the steps (1) segmentation, (2) assimilation, (3) identification, (4) aggregation, and (5) interpretation. While the first two steps are meant for translating images or other "raster-based input" into diagrammatic primitives, which is not useful for us, the latter stages provide a solid conceptual foundation to grade diagrams.

F. Batmaz [17] takes a broader look at the process of grading, identifying and developing techniques to reduce repetitive actions, focusing on database ER diagrams. The paper suggests a semi-automatic grading system, including automatic grading based on identifying identical segments between a submission and the solution. Assuming multiple submission revisions are available, it suggests to "not only [use] the reference text but also the intermediate diagrams" for identifying semantic matches [17, p.40]. While multiple solutions are not useful for the purpose of grading only a single submitted diagram after an exam, this might be useful for live feedback.

V. Vachharajani *et al.* [18] propose a UML use case assessment architecture, providing a useful catalogue about edge cases related to use case diagram assessment which are also applicable to other types of diagrams, such as the chance of misspellings, synonyms, abbreviations, directionality of relationships, and more.

W. Bian *et al.* [19] establish a model to map submissions to example solutions and one to grade submissions. It recommends syntactic matching to help with spelling mistakes, semantic matching to match related words, and structural matching to match neighbouring elements and/or inheritance, with the goal to optimally match parts of a student submission with a sample solution.

In conclusion, most autograder strategies recommend structural matching (to identify similar segments of graphs), often in combination with syntactic matching that accounts for misspellings and semantic matching to account for synonyms or related words. Unfortunately, the strategies do not account for integrating ILOs into the grading process explicitly.

2.1.2. Algorithmic

Next to frameworks for autograders, implementations were also discovered during the literature review, of which a subset used purely algorithmic methods. Summaries of these sources are discussed in this section, along with a general summary on algorithmic autograders.

W. Bian *et al.* [5] implements their previous framework [19] and validates it with a case study, using the Levenstein distance between words for syntactic matching, several metrics for semantic matching, and structural matching based on similar attributes and operations within classes. They find that multiple teacher solutions result in more accurate grades with an average accuracy over 95% [5, p.10]. Additionally, they find that grading configurations need to change per exam if the goal is to produce the most similar scores to manual grading, likely because the focus of each exam or exercise is on a different aspect of diagram creation (associations, inheritance, etc.). Lastly, they state that autograding "has shown to be more consistent and able to ensure fairness in the grading process" compared to manual grading [5, p.11]. Additionally, their visual feedback system seems to provide a clear visualisation of the grading results, which might feel more intuitive for students. An example is shown in Figure 1.

M. Hosseinibaghdadabadi *et al.* [20] also implements the framework by W. Bian *et al.* [19],

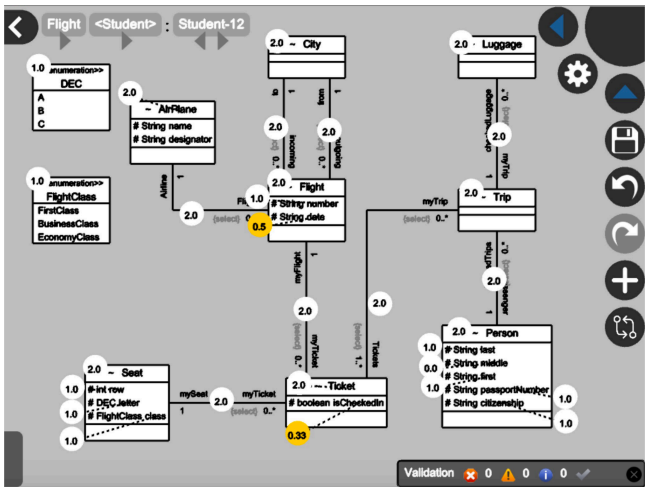

Figure 1: Visual feedback module from W. Bian *et al.* [5, Fig.9]

comparing UML use case diagrams to one or multiple example solutions and preferring the maximum grade. It uses a graph similarity algorithm which matches nodes based on structural matching, along with syntactic matching using the Levenstein distance between words and semantic matching using WordNet similarity score (HSO, WUP, LIN metrics). It achieves a similarity percentage of 93.31% [20, p.114].

O. Anas *et al.* [21] compares UML class diagram submissions to an example solution. It uses graph similarity scores based on structural matching along with syntactic and semantic matching. Syntactic matching is done with substring matching and semantic matching is done with neighbour similarity ("the comparison of the neighboring classes" [21, p.1585]), relationship name, type, multiplicity, and inheritance. It achieves a respectable correlation with human grading, with more than 80% identical grades, more than 90% achieving a corelation larger than 0.85, and no grading achieving a correlation lower than 70%.

Multiple papers mention the use of XMI [22] [23], the object notation standard by OMG [11], or Rose Petal files [24] [25], the standard of IBM Rational Rose [12], but fail to mention specifics about matching algorithms or results.

H. AlRawashdeh *et al.* [26] provides an interesting alternative way of grading submissions: by means of combining many UML diagram validators, model checkers, and even LTL properties given by instructors. However, a clear purpose, scope, and results are missing from the paper.

M. Striewe *et al.* [27] also attemps property checking akin to H. AlRawashdeh *et al.* [26]'s work by focusing on graph queries for evaluation, providing a domain-specific language that looks similar to SQL. While it offers an interesting alternative approach, the fact that teachers would have to learn a query language and transform their existing rubrics/example solutions into this format could be a real hurdle, especially given the performance of graph-isomorphism-based solutions [5] [20] [21]. Additionally, the paper does not account for misspellings or the use of synonyms.

S. Foss *et al.* provide multiple papers on AutoER, a database diagram generator and evaluator that provides direct interaction with a description text [28] [29] [30]. It is more geared towards interactive use, intended for providing intermediate results and hints during the diagram creation process, before handing in the final submission. Unfortunately, concrete comparisons to manual grading and its source code could not be found.

P. Thomas, like S. Foss *et al.*, also provides a selection of papers on the automatic grading of ER diagrams [9] [31] [32] [33] [34]. Unlike S. Foss *et al.*, these papers are focused on a *single* assessment point and provide a grading strategy that accounts in its basis for imprecise diagrams (diagrams containing misspellings, duplicate entities, etc.). They base their analysis on comparing ever increasing subsets of the graph ((Minimal) Meaningful Units) based on the work of N. Smith *et al.* [8]. By 2009, P. Thomas *et al.* manage to achieve a correlation to human grading of 92%, along with statistically proving that the autograder grades more consistently than human grading. The grading results can be viewed in Figure 2.

In 2011, P. Thomas *et al.* provide an online platform for both students and teachers to ease the process of automatic grading further, also used by N. Smith *et al.* [35], which further mathematically specifies P. Thomas *et al.*'s work. Unfortunately, we were not able to locate the source code of this grader.

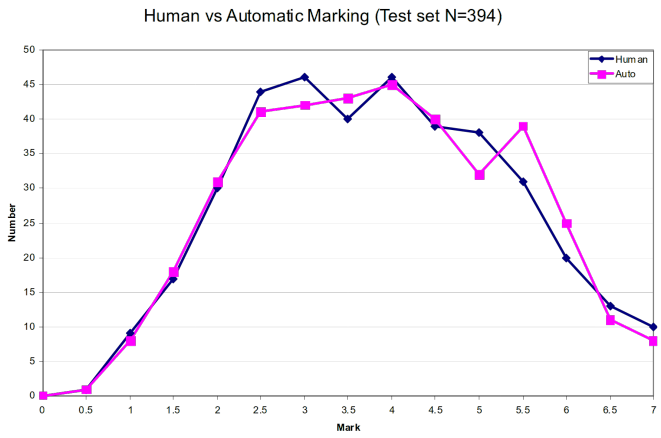

Figure 2: P. Thomas *et al.* [33, Fig. 3]: Human vs. automatic grading in database ER diagrams.

In conclusion, most existing implementations of autotgraders use some graph isomorphism algorithm with a combination of structural, semantic, and syntactic matching, as suggested by most frameworks. Some solutions attempt to autograde using property or formula checking, but fail to mention a detailed enough methodology or results to warrant further investigation. No autograders provide methods on integrating ILOs into the grading process.

2.1.3. ML- / GenAI-driven

Next to using purely algorithmic methods, some papers experiment with Machine Learning / Generative AI (collectively: 'AI-driven solutions') to automatically grade submissions, and even mention some hybrid AI / algorithmic solutions. We provide summaries of the explored sources below, along with a general conclusion on AI-driven autograders.

D. R. Stikkolorum *et al.* [36] is one of the earliest papers found that attempts Machine Learning-based autograding, using several machine learning algorithms to compare submissions to expert grades. Unfortunately, the grading only reaches a maximum accuracy of 42.76%, while rounding off scores to a 10-point integer scale. Exact methods and algorithms are not mentioned.

C. Wang *et al.* [37] evaluate the feasibility of LLM-based grading with the LLM model ChatGPT-4o, focusing on student reports containing multiple types of UML diagrams. They feed pictures of student-submitted UML diagrams directly into the model along with an explanatory prompt that aims to trigger a Chain-of-Thought process (which should help LLMs "tackle complex arithmetic, commonsense, and symbolic reasoning tasks" [38]), and runs the model once per submission, with a temperature of 0.1. They find that score differences range from −0.25 to +3.75 points, with with the LLM handing out significantly lower average scores compared to humans. Additionally, they note many occurrences of incorrect grading (wrong identifications, overstrictness, and misunderstandings) [37, p.18], which means that, while the authors claim that their solution "demonstrates particular proficiency in the automated evaluation of UML use case diagrams", the grading is not internally consistent and contains hallucinations. In the words of the authors: "In the evaluation based on UC4, GPT deducts points for missing relationships between specified actors and use cases, but theses relationships existed in the UML use case" [37, p.13]. Furthermore, the paper does not express a strong correlation between LLM grading and human grading, at least compared to papers utilising graph matching algorithms [33] [20], nor does it recognise the inherent bias of LLMs [39] or their inherent non-determinism (even with a zeroed temperature) [40] [41].

N. Bouali *et al.* [42] uses various LLMs (Llama 3.2B, ChatGPT-o1 mini, and Claude Sonnet) to grade diagrams, first translating the diagrams into text instead of giving the LLM images directly like C. Wang *et al.* [37]. While they achieve a Pearson correlation to human grading of 0.76 with both ChatGPT and Claude, they run into the same inconsistency issues as C. Wang *et al.*: "while the models would provide a final score as requested in the prompt's response format, this score often did not match the actual sum of points awarded in their criterion-by-criterion assessment", and "'One ChargingPort is associated with One Vehicle' was matched with 'One ChargingPort is associated with One ChargingStation' with a similarity of 0.92, despite describing different domain relationships" [42, p.164].

N. Bouali *et al.* identify the problem with grading with LLMs perfectly, stating that "This discrepancy can be attributed to the

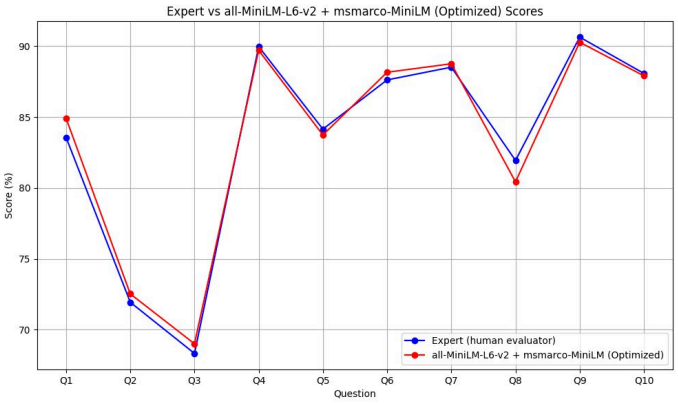

Figure 3: R. Ramachandran *et al.* [44, p.13]: Comparison of expert scores and CodeLLama scores using a combination of `all-MiniLM-L6-v2` and `msmarc-MiniLM` as word similarity models.

autoregressive nature of LLMs, where they generate responses token by token" [42, p.164]. Because these models are in their very essence based on predicting tokens [43], there is no formal guarantee that the grades are internally consistent and that grades are produced accurately with respect to the rubric. The fact that LLMs produce grades that correlate with human grading does not mean that this grading is done in a fair, consistent, or reliable manner. While N. Bouali *et al.* try to reduce the non-determinism of LLMs by setting the temperature to zero, this does not necesssarily remove non-determinism [40] [41], nor does it account for training biases [39], as mentioned before.

R. Ramachandran *et al.* [44], unlike the previous papers, use a human-in-the-loop design in combination with both purely algorithmic steps, using LLMs only for semantic and syntactic matching. Using structural matching algorithms similar to papers presented in Section 2.1.2, it achieves a Mean Average Error of only 0.611, aligning very closely to human grading (see Figure 3). Unfortunately, the data set contains only ten self-procured images, which negatively impacts the significance of these results, not to mention that the nondeterminism introduced by the LLMs will impact the consistency of grading, although it is unclear to what extent.

In conclusion, while AI-based grading has been attempted in recent years, purely AI-driven solutions produce lacking similarity to human grading compared to graph isomorphism-based solutions as well as introducing non-deterministic and biased behaviour, while providing no consistency guarantees. This makes these types of solutions inferior to graph isomorphism solutions in terms of *accuracy*, *consistency*, and *grading transparency*. When used only for semantic and/or syntactic matching, it can provide similar accuracy to algorithmic solutions, although it still introduces nondetermism in grading which negatively affects *consistency*.

2.2. Intended Learning Objectives and examination

A. Dinur *et al.* [45] states that analytical rubrics (those which mention explicit criteria) provide more details than global, holistic rubrics. These types of rubrics (and their exercises) can be constructed directly from the ILOs of a course or module, which would provide a detailed grading rubric that aligns closely to its ILOs D. Osinga [4]. Given that rubrics and exercises are defined in such a way, one could link these ILOs to the grading rubric and provide functionality for an autograder to show these in the final grade This would indicate to students how well they achieved the learning goals of the module in order to improve *grading transparency*.

2.3. Conclusion

In the explored related work, existing frameworks primarily recommend structural matching in combination with syntactic and semantic matching to account for spelling mistakes and the use of synonyms. Existing implementations mostly use the methods recommended by the frameworks, with the best results stemming from determinstic graph isomorphism algorithms. Purely AI-driven methods may require less effort from teachers, since teachers do not need to produce sample diagrams but can describe their rubric in words, but produce noticeably inferior results to graph matching algorithms. Using hybrid methods, specifically using AI-driven classification algorithms only for semantic/syntactic matching, seems to produce similar results to 'pure' graph matching algorithms, but does not necessarily provide accuracy gains over algorithmic solutions and can additionally introduce nondeterminism in otherwise determinstic solutions, reducing consistency.

3. TOOLS AND TECHNIQUES

Given the existing works mentioned in Section 2 and Table 1, the best approach for maximising accuracy, consistency, and grading transparency seems to be to use graph isomorphism algorithms akin to those suggested by N. Smith *et al.* [35], implemented by papers such as W. Bian *et al.* [5] and P. Thomas *et al.* [33]. Visualisations similar to Figure 1 could prove to be a nice addition, so architectural support for visualisations will be taken into account, which can be implemented should there be time left.

Unfortunately, no solutions seem to support the integration of ILOs into their grading rubric inputs. While we believe that this is a vital point to consider when making rubrics or example solutons [4], we realise that it may require extra work for a teacher to add metrics on how much a certain ILO is tested, and may not be compatible with existing rubrics. Therefore, we aim to add support for optional ILO weights, either per diagram feature or for specific parts of an example solution. For example, a teacher would ideally be able to mark that the presence of certain classes, certain associations, the multiplicity of associations, or the connection between (certain) classes satisfies a certain set of ILOs.

Since existing solutions do not provide the features necessary, nor their source code (see Table 1), we develop our own autograder, named *Seshat*[5].

3.1. Architecture

Seshat needs to take input (from either exam exports, a list of files, or via some other format), transform the input into an internal graph representation, run comparison algorithms on it defined in Section 2.1.2 which produces a set of scores (a 'grade'), and format this grade in a certain way. The exact methodology, algorithms, and visualisations are likely to change, which is

Author	Di	Ac	Co	Tr	F	A	I	R	ILO	UTML
W. Bian *et al.* [5]	UML Class	H	H	H	-	-	i	r	N	N
M. Hosseinibaghdadabadi *et al.* [20]	UML Use Case	H	H	H	-	-	i	r	N	N
O. Anas *et al.* [21]	UML Class	M	H	H	-	-	i	r	N	N
S. Modi *et al.* [22]	UML Class	?	H	H	-	-	-	-	N	N
R. Jebli *et al.* [23]	UML Class	?	H	H	-	-	-	-	N	N
N. H. Ali *et al.* [24] [25]	UML Class	?	H	L	-	-	-	-	N	N
H. AlRawashdeh *et al.* [26]	UML State/Sequence	?	H	?	-	-	I	-	N	N
M. Striewe *et al.* [27]	UML Class	?	H	H	-	-	I	-	N	N
S. Foss *et al.* [28] [29] [30]	ER	?	H	?	-	-	I	r	N	N
P. Thomas *et al.* [9] [31] [32] [33] [34]	ER	H	H	H	-	-	I	r	N	N
D. R. Stikkolorum *et al.* [36]	UML Class	L	L	L	-	-	-	-	N	N
C. Wang *et al.* [37]	UML	M	L	M	F	a	I	r	N	N
N. Bouali *et al.* [42]	UML Class	M	M	M	F	a	I	r	N	N
R. Ramachandran *et al.* [44]	ER	H	M	H	F	a	i	r	N	N

Table 1: Autograders and their suitability scores.

What* **Di*agram types are supported*, *how high is the* **Ac***(curacy)*, **Co***(nsistency)*, and *Grading* **Tr***(ansparency)*, *how* **F***indable,* **A***ccessible,* **I***nteroperable, and* **R***eproduable is the tool*, *can the tool link* **ILO***s to grading*, *and how well is* **UTML** *supported*?

Scoring (except FAIR) is divided into "N" (*No Support*), "L" (*Low*), "M" (*Medium*), "H" (*High*), and "?" (*Unknown*), which gives an indication of each autograder's suitability w.r.t. that particular criterium. The scoring for these rubrics is done in a comparative way, with the lowest-scoring solution receiving a "L" or "N" and the highest scoring receiving a "H". High **accuracy** is awarded for deterministic solutions, with lower values given to nondeterministic programs. High **consistency** is awarded for determinstic solutions. High **grading transparency** is awarded for solutions that explain the final grade in terms of rubrics (medium for full rubrics that might not match (i.e. AI-driven solutions)). **ILO** and **UTML** support is given a "H" or "N" based on inclusion of these features. **FAIR** scoring is done by checking the Findability, Accessibility, Interoperability, and Reusability, inspired by M. D. Wilkinson [46, Box 2, p.4]. We focus purely on the autograder solutions for this rubric. For example, if the code is findable with a fixed ID or link, the project is available but only through a paywall, the algorithms in the paper are designed to be interoperable with only one diagram format (for example XMI) and only one type of diagram (for example ER diagrams), and the work is partially reproducible (deriving parts of the source code using algorithms in the paper), it gets a score of 'Fa_r'.

[5]The Egyptian record-keeping godess and daughter of *Thoth*, the name of D. Osinga [4]'s autograder.

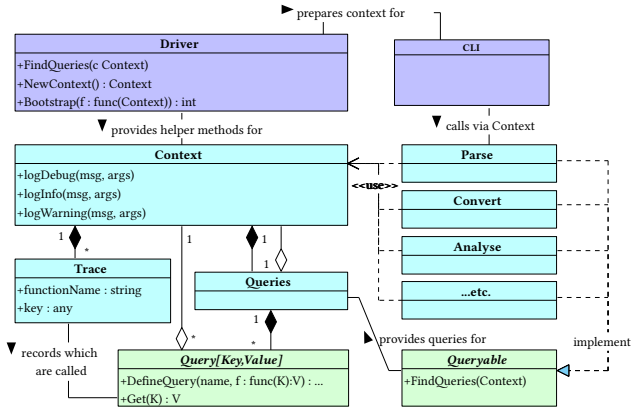

Figure 4: Query architecture for *Seshat*. (UTML was not used to make this diagram)

why we want to maximally decouple these parts.

In order to achieve this, we aim to implement a query-based architecture akin to that of the Rust compiler [47], of which a draft architecture can be seen in Figure 4. This encourages decoupling each stage of the process, and additionally increases transparency internally in the grading process, as one can easily query intermediate solutions from the grading process. Testing components is also inherently made easier due to the split-up functionality.

Increment	Task
Wk. 6 - 7	set up prototype architecture
Wk. 8 - 9	UTML input parsing
Wk. 10 - 11	internal graph representations
Wk. 12 - 13	implement/verify **syntactic** matching algorithm(s) [5] [33] [35]
Wk. 14 - 15	implement/verify **semantic** matching algorithm(s) [5] [33] [35]
Wk. 16 - 17	implement/verify **structural** matching algorithm(s) [5] [33] [35]
Wk. 18 - 19	combine/verify algorithms [5] [33] [35]
Wk. 20 - 21	combine/verify/document algorithms [5] [33] [35], compare to manual grading
Wk. 22 - 23	compare to manual grading, document results in paper
Wk. 24 - 25	Finalise paper / buffer time / peer reviews by colleagues
Wk. 26 - 27	Finalise paper / buffer time / peer reviews by colleagues

Table 2: Increment planning of the final thesis. Note that paper development is done in parallel to the development of *Seshat*.

Finally, a query-based architecture allows for caching all stages of the process, which is possible since we make the explicit choice to use only deterministic algorithms. This allows for efficiency improvements if we need to fetch some query output multiple times, or if we need to grade a solution we have already seen before.

3.2. Language

For this project, we opt for Go, as it is a multi-pardigm, statically typed, and compiled language. This should allow us to leverage both imperative and functional paradigms, create a robust architecture, and allow for fast grading. Additionally, the author is familiar with it, which should decrease the time it takes to implement features.

4. PLANNING

We plan to develop *Seshat* according to the Agile methodology [48]. This means that we divide the work up into increments, and aim to show new deliverables frequently. This prioritises prototyping and frequent feedback, allowing the supervisors to steer the direction of the project effectively.

We divide these increments up into two weeks. This should allow for enough time inbetween to make significant progress on *Seshat* and the final paper, while keeping increments small enough to be able to reflect on the progress made often enough and make adjustments to the plan if necessary. We meet with the main supervisor every increment. We invite the co-supervisor to every meeting: they are free to attend when they wish to see progress and/or give advice. When in doubt, we explicitly ask advice of both supervisors to get a view that spans multiple perspectives.

During the development of *Seshat*, we add to the paper in parallel, documenting design decisions and progress.

The increments are initially structured in the way defined in Table 2. Note that these are subject to change, and there is a bit of buffer time planned in, as it might turn out there is more research needed to complete certain algorithms .

BIBLIOGRAPHY

[1] Object Management Group, "OMG website." [Online]. Available: https://www.omg.org/

[2] F. Ahmed, N. Bouali, and M. Gerhold, "Teaching Assistants as Assessors: An Experience Based Narrative," 2024. [Online]. Available: https://research.utwente.nl/files/457355611/126242.pdf

[3] M. Meadows and L. Billington, "A Review Of THe Literature On Marking Reliability." [Online]. Available: https://assets.publishing.service.gov.uk/media/5a820a57e5274a2e87dc0d5a/0505_Meadows_and_Billington_CERP_RP.pdf

[4] D. Osinga, "Combining Dynamic and Static Analysis for the Automation of Grading Programming Exams," Bachelor thesis, 2024. [Online]. Available: https://github.com/osingaatje/ut-bachelor-thesis

[5] W. Bian, O. Alam, and J. Kienzle, "Is automated grading of models effective?: assessing automated grading of class diagrams," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, in MODELS '20. ACM, Oct. 2020, pp. 365–376. doi: 10.1145/3365438.3410944.

[6] I. G. Pirie, *The measurement of programming ability*. University of Glasgow (United Kingdom), 1975. [Online]. Available: https://search.proquest.com/openview/3afb41488c34283f38dec7f13a3ea744/1

[7] M. J. Rees, "Automatic assessment aids for Pascal programs," *ACM SIGPLAN Notices*, vol. 17, no. 10, pp. 33–42, Oct. 1982, doi: 10.1145/948086.948088.

[8] N. Smith, P. Thomas, and K. Waugh, "Interpreting imprecise diagrams," in *Diagrammatic Representation and Inference*, in International Conference on Theory and Application of Diagrams. Mar. 2004, pp. 239–241. doi: 10.1007/978-3-540-25931-2_24.

[9] P. Thomas, "Grading Diagrams Automatically," 2004. [Online]. Available: https://oro.open.ac.uk/90155/1/2004_01.pdf

[10] S. Bagui and R. Earp, *Database design using entity-relationship diagrams*. Auerbach Publications, 2003. [Online]. Available: https://www.taylorfrancis.com/books/mono/10.1201/9780203486054/database-design-using-entity-relationship-diagrams-sikha-bagui-richard-earp

[11] OMG, "XML Metadata Interchange format." [Online]. Available: https://www.omg.org/spec/XMI

[12] IBM, "Rational Rose." [Online]. Available: https://www.ibm.com/support/pages/ibm-rational-rose-enterprise-7004-fix-pack-4-7000

[13] PlantUML, "PlantUML." [Online]. Available: https://plantuml.com/

[14] Visual Paradigm, "Visual Paradigm." [Online]. Available: https://www.visual-paradigm.com/

[15] D. Huistra, "UTML - internal GitLab repository." [Online]. Available: https://gitlab.utwente.nl/ewi/eduapps/UTML/

[16] University of Twente, "utml.apps.utwente.nl." [Online]. Available: https://utml.apps.utwente.nl/

[17] F. Batmaz, "Semi-Automatic Assessment ofStudents' Graph-Based Diagrams," 2010. [Online]. Available: https://www.academia.edu/download/66135135/70_22270_EM_26aug_20feb_L.pdf

[18] V. Vachharajani and J. Pareek, "A Proposed Architecture for Automated Assessment of Use Case Diagrams," in *International Journal of Computer Applications (0975 – 8887)*, 2014. [Online]. Available: https://www.academia.edu/download/67672696/pxc3900193.pdf

[19] W. Bian, O. Alam, and J. Kienzle, "Automated Grading of Class Diagrams," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, IEEE, Sept. 2019, pp. 700–709. doi: 10.1109/models-c.2019.00106.

[20] M. Hosseinibaghdadabadi, O. A. N. Almerge, and J. Kienzle, "Automated Grading of Use Cases," in *2023 ACM/IEEE 26th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, IEEE, 2023. [Online]. Available: https://ieeexplore.ieee.org/iel7/10343461/10343549/10343598.pdf

[21] O. Anas, T. Mariam, and L. Abdelouahid, "New method for summative evaluation of UML class diagrams based on graph similarities," 2021. [Online]. Available: https://www.academia.edu/download/66135135/70_22270_EM_26aug_20feb_L.pdf

[22] S. Modi, H. A. Taher, and H. Mahmud, "A Tool to Automate Student UML diagram Evaluation," 2021. [Online]. Available: https://www.academia.edu/download/72488756/575.pdf

[23] R. Jebli, J. E. Bouhdidi, and M. Y. Chkouri, "Assessing Students' UML Class Diagrams: a NewAutomated Solution," in *2023 7th IEEE Congress on Information Science and Technology (CiSt)* |, IEEE, 2023. [Online]. Available: https://ieeexplore.ieee.org/iel7/10409867/10409868/10409936.pdf

[24] N. H. Ali, Z. Shukur, and S. Idris, "Assessment System For UML Class DiagramUsing Notations Extraction," 2007. [Online]. Available: https://www.researchgate.net/profile/Zarina-Shukur/publication/253243639_Assessment_System_For_UML_Class_Diagram_Using_Notations_Extraction/links/55487af30cf2b0cf7acec2e4/Assessment-System-For-UML-Class-Diagram-Using-Notations-Extraction.pdf

[25] N. H. Ali, Z. Shukur, and S. Idris, "A Design of an Assessment System for UML Class Diagram," in *2007 International Conference on Computational Science and its Applications (ICCSA 2007)*, IEEE, Aug. 2007, pp. 539–546. doi: 10.1109/iccsa.2007.2.

[26] H. AlRawashdeh, S. Idris, and A. M. Zin, "Using Model Checking Approach for Grading the Semantics of UML Models," 2014. [Online]. Available: https://iieng.org/images/proceedings_pdf/8684E0114567.pdf

[27] M. Striewe and M. Goedicke, "Automated Checks on UML Diagrams," in *ITiCSE'11*, in ITiCSE '11. ACM, June 2011, pp. 38–42. doi: 10.1145/1999747.1999761.

[28] S. Foss, T. Urazova, and R. Lawrence, "Learning UML database design and modeling with AutoER," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, in MODELS '22. ACM, Oct. 2022, pp. 42–45. doi: 10.1145/3550356.3559091.

[29] S. Foss, T. Urazova, and R. Lawrence, "Automatic Generation and Marking of UML Database Design Diagrams," in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education*, in SIGCSE 2022. ACM, Feb. 2022, pp. 626–632. doi: 10.1145/3478431.3499376.

[30] S. Foss, "AutoER: A System for the Automatic Generation and Evaluation of UML Database Design Diagrams," 2022. [Online]. Available: https://open.library.ubc.ca/media/download/pdf/24/1.0421624/4

[31] P. Thomas, N. Smith, and K. Waugh, "An approach to the automatic grading of imprecise diagrams," technical report, 2006. doi: .org/10.21954/ou.ro.00016046.

[32] P. Thomas, N. Smith, and K. Waugh, "Automatically assessing graph-based diagrams," *Learning, Media and Technology*, vol. 33, no. 3, pp. 249–267, 2008, doi: 10.1080/17439880802324251.

[33] P. Thomas, N. Smith, and K. Waugh, "Automatically Assessing Diagrams," in *Proceedings of the IADIS International Conference on e-Learning*, 2009. [Online]. Available: https://www.

researchgate.net/profile/Pete-Thomas/publication/42799920_Automatically_assessing_diagrams/links/0fcfd5060076dd8ba2000000/Automatically-assessing-diagrams.pdf

[34] P. Thomas, K. Waugh, and N. Smith, "Generalised Diagramming Tools with Automatic Marking," in *Innovation in Teaching and Learning in Information and Computer Sciences*, 2011. doi: 10.11120/ital.2011.10010022.

[35] N. Smith, P. Thomas, and K. Waugh, "Automatic Grading of Free-Form Diagrams with Label Hypernymy," in *2013 Learning and Teaching in Computing and Engineering*, IEEE, Mar. 2013, pp. 136–142. doi: 10.1109/latice.2013.33.

[36] D. R. Stikkolorum, P. van der Putten, C. Sperandio, and M. R. Chaudron, "Towards Automated Grading of UML Class Diagrams with Machine Learning," 2019. [Online]. Available: https://ceur-ws.org/Vol-2491/paper80.pdf

[37] C. Wang, B. Wang, P. Liang, and J. Liang;, "Assessing UML Diagrams by GPT: Implications for Education," technical report, 2025. [Online]. Available: https://www.researchgate.net/publication/397720325_Assessing_UML_Diagrams_by_GPT_Implications_for_Education

[38] J. Wei *et al.*, "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," 2023. [Online]. Available: https://arxiv.org/abs/2201.11903

[39] R. Ranjan, S. Gupta, and S. N. Singh, "A Comprehensive Survey of Bias in LLMs: Current Landscape and Future Directions," 2024. doi: 10.48550/arXiv.2409.16430.

[40] M. Brenndoerfer, "Why Temperature=0 Doesn't Guarantee Determinism in LLMs," 2025, [Online]. Available: https://mbrenndoerfer.com/writing/why-llms-are-not-deterministic

[41] B. Atil *et al.*, "Non-Determinism of "Deterministic" LLM Settings," 2025. [Online]. Available: https://arxiv.org/pdf/2408.04667

[42] N. Bouali, M. Gerhold, T. U. Rehman, and F. Ahmed, "Toward Automated UML Diagram Assessment: Comparing LLM-Generated Scores with Teaching Assistants," 2025. [Online]. Available: https://research.utwente.nl/files/496461589/134819.pdf

[43] A. F. Ferraris, D. Audrito, L. D. Caro, and C. Poncibò, "The architecture of language: Understanding the mechanics behind LLMs," *Cambridge Forum on AI: Law and Governance*, vol. 1, pp. 1–19, 2025, doi: 10.1017/cfl.2024.16.

[44] R. Ramachandran, P. Vijayan, A. Anilkumar, and V. Gandadharan, "AI Assisted System for Automated Evaluation of Entity-Relationship Diagram and Schema Diagram Using Large Language Models," technical report, Dec. 2025. doi: 10.3390/bdcc10010002.

[45] A. Dinur and H. Sherman, "Incorporating outcomes assessment and rubrics into case instruction," *Journal of Behavioral and Applied Management*, vol. 10, no. 2, pp. 291–311, 2009, [Online]. Available: https://jbam.scholasticahq.com/article/17258.pdf

[46] M. D. Wilkinson, "The FAIR Guiding Principles for scientific datamanagement and stewardship," 2016. [Online]. Available: https://www.nature.com/articles/sdata201618

[47] Rust Team, "Overview of the Rust Compiler." [Online]. Available: https://rustc-dev-guide.rust-lang.org/overview.html

[48] K. Beck *et al.*, "Manifesto for Agile Software Development." [Online]. Available: https://agilemanifesto.org/