RESEARCH TOPICS 2026-02-09

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 UML diagrams. The grading of these diagrams is mainly done by humans, resulting in a costly, lengthy, and error-prone process. In this paper, we investigate the theoretical feasibility of automatically grading UML diagrams, focusing on the UTML variant developed at the University of Twente. We find that graph isomorphism algorithms that account for synonyms and spelling mistakes provide the best results. Furthermore, we propose *Seshat*, an algorithmic autograder that combines the aforementioned techniques and adapts them for UTML. In the final thesis, we compare *Seshat* 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 [2], as humans are inherently unreliable graders [3]. M. Meadows *et al.* [3] pose two possible solutions to this problem: 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 a rubric, or even 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*).

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 implementation papers starting to appear around the 80s, primarily focused on grading the writing style of computer programs [7]. Interest in specifically diagram grading seems to have started around the early 2000s [8] [9].

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

[1] From personal experience.

[2] Given that the process is deterministic

flows etc. [14], and UTML - an in-house standard developed at the University of Twente for representing UML diagrams [15] [16].

The degree to which automated grading is implemented can vary as well. We divide autograding into the following categories: non-automated (manual), 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 examin the feasibility of automatically grading UTML UML diagrams, we provide a main research question (**MRQ**):

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

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

RQ1: What existing work can be found for automatically analysing and/or grading UML diagrams?

- **RQ1a**: What correction models are employed by existing works?

RQ2: To what extent are existing solutions suitable for use in autograding UTML diagrams with regards to (1) accuracy, (2) consistency, (3) transparency, (4) availability of source code, (5) extent of linking ILOs to grading instructions, (6) ease of integration into the grading process, and (7) UTML support?

RQ3: To what extent can a suitable autograder be constructed from previous work to be able to grade UTML UML diagrams?

RQ4: To what extent does the suitable autograder compare to human grading in the context of grading first-year UML exam questions?

RQ1 and **RQ2** are answered in Section 2 and Table 1, by analysing these works for suitability of grading. Section 3 explains the plan for building the autograder and Section 4 outlines the planning. **RQ3** and **RQ4** 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** and **RQ2**, we conduct a small-scale literature study covering roughly 40 works. It aims to provide an exploratory view into the world of autograders, which means that formal inclusion and exclusion criteria are not set up. Works are collected from Google Scholar[3] and ResearchGate[4], using terms including but not limited to “automatically grading UML diagrams”, “autograder diagram”, “UML diagram assessment”, “machine learning diagrams”, and “diagram evaluation assessment AI”. For ILO research, terms were used such as “learning outcomes include in rubric”, “learning objectives in rubrics”, and similar.

2.1. Autograders

Multiple methods and types of diagrams are researched, including proposed frameworks for autograders, purely algorithmic implementations, and Machine Learning (ML) / Generative AI (GenAI) / Large Language Model (LLM)-based methods. Additionally, work on integrating ILOs is researched.

2.1.1. Frameworks / Theoretical

Some autograder frameworks were found. These dictate a certain design or methodology for building an autograder.

N. Smith *et al.* [8] provide a five-step framework for assessing “possibly ill-formed or inaccurate diagrams” that include (1) segmentation, (2) assimilation, (3) identification, (4) aggregation, and (5) interpretation. While the first two steps are aimed at translating images or other “raster-based input” into diagrammatic primitives, the latter stages provide a foundation to grade diagrams used by other papers [17].

F. Batmaz [18] takes a broader look at the process of grading, identifying and developing techniques to reduce repetitive actions, focusing

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

on database Entity Relation diagrams. The paper suggests a semi-automatic grading system which identifies 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 [18, 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.* [19] propose a UML use case assessment architecture, providing a useful catalogue about edge cases related to (use case) diagram assessment, such as the chance of misspellings, synonyms, abbreviations, directionality of relationships, and more.

W. Bian *et al.* [20] establish a metamodel to map submissions to example solutions and present a metamodel to grade submissions. It suggests using syntactic matching, semantic matching, and structural matching, with the goal to optimally match parts of a student submission with those of a teacher, considering spelling mistakes, synonyms and related words, and neighbours / inheritance, respectively.

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. Unfortunately, the strategies do not account for integrating ILOs into the grading process explicitly.

2.1.2. Algorithmic

Implementations of autograders 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] expand their previous work [20] (see Section 2.1.1) with a case study. Their main findings are that multiple teacher solutions result in more accurate grades with an average accuracy of more than 95% [5, p.10], that grading configurations change per exam if you want similar grades to the teacher, and that their autograding “has shown to be more consistent and able to ensure fairness in the grading process” [5, p.11]. Additionally, their visual feedback system seems to be a nice addition for easily seeing where marks were awarded / taken away (see Figure 1).

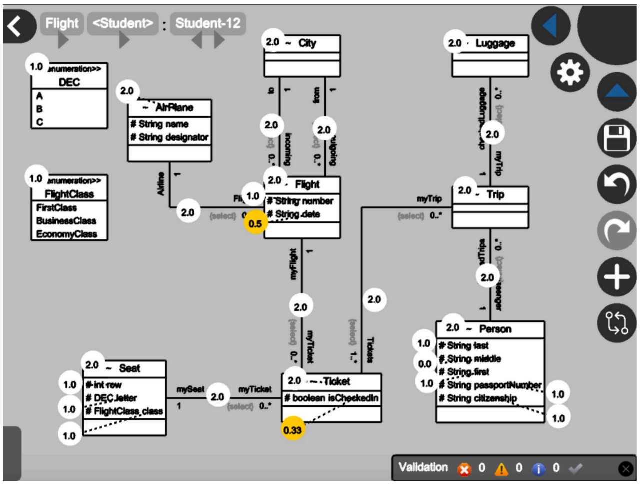

Figure 1: Visual feedback module from W. Bian *et al.* [5, Fig.9]

M. Hosseinibaghdadabadi *et al.* [21] also implements the framework by W. Bian *et al.* [20] by comparing UML use case diagrams to one or multiple example solutions, preferring the maximum grade. It uses a graph similarity strategy which matches nodes based on structural matching, along with syntactic and semantic word matching. Syntactic matching with Levenshtein distance, semantic matching with WordNet similarity score (uses HSO, WUP, LIN metrics). It achieves a similarity percentage of 93.31% [21, p.114].

O. Anas *et al.* [22] 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, semantic matching is done with neighbour similarity (“the comparison of the neighboring classes” [22, p.1585]), relationship name, type, multiplicity, and inheritance. It achieves a respectable correlation with human grading, with more than 80% is perfectly similar, over 90% >0.85 correlated, and no correlations lower than 70%.

Multiple papers mention the use of XMI [23] [24], the object notation standard by OMG [11], or Rose Petal files [25] [26], the standard of IBM

Rational Rose [12], but fail to mention specifics about matching algorithms or results.

H. AlRawashdeh *et al.* [27] 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 lacking from the paper.

M. Striewe *et al.* [28] continues H. AlRawashdeh *et al.* [27]'s property checking trend by focusing on graph queries for evaluation, providing a Domain-Specific Language that looks similar to SQL. While it looks promising, 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 high similarity to existing grading of graph-isomorphism-based solutions [5] [21] [22]. Additionally, the paper does not provide approximate matching that would account for misspelling or synonyms.

S. Foss *et al.* provide multiple papers on AutoER, a database diagram generator and evaluator that provides direct interaction with a description text [29] [30] [31]. It is more geared towards interactive use, intended for multiple checks during the diagram creation process before the final submission. Unfortunately, concrete comparisons to manual grading and 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] [32] [33] [17] [34]. However, 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.

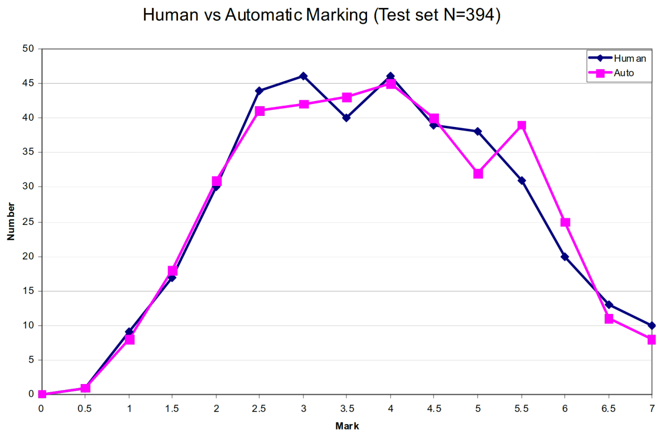

Figure 2: P. Thomas *et al.* [17, Fig. 3]: Human vs. automatic grading in database ER diagrams.

In conclusion, most existing implementations of autotgraders use some form of graph isomorphism algorithms 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- / LLM-driven

Next to using algorithmic methods, there has also been work on using Machine Learning / Generative AI / Large Language Models (collectively: 'AI-driven solutions') to automatically grade submissions, and even 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 first papers that was found that attempts Machine Learning-based autograding, using several machine learning algorithms to compare it to expert grades. Unfortunately, the grading reaches only a maximum accuracy of 42.76% using 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 model ChatGPT-4o, specifically for entire 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 helps LLMs "tackle complex arithmetic, commonsense, and symbolic reasoning tasks" [38]), and runs the model one time per student, with a temperature of 0.1. It finds that score differences range from −0.25 to +3.75 points, with significantly lower average scores given by the LLM compared to humans. Additionally, there are many occurrences of incorrect grading (wrong identifications, overstrictness, 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", hallucinations are present in the grading. An example by 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 [17] [21], nor does it recognise the inherent bias of LLMs [39] or their inherent non-determinism (even with a zeroed temperature) [40] [41], which make it a sub-optimal solution for consistent, fair grading.

N. Bouali *et al.* [42] uses various LLMs (Llama, GPT-o1 mini, Claude) to grade, translating the models into text instead of giving the LLM images directly such as 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 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 results are internally consistent and thus grades are produced with accuracy. 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 remove non-determinism necesssarily, nor does it correct training biases, as mentioned before [40] [41].

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 similarity 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 sample size was a self-procured test set of only ten 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 how much.

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.

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 fundamental non-deterministic behaviour, and a lacking internal consistency. This makes these types of solutions inferior to graph isomorphism solutions in terms of accuracy and consistency. When used only for semantic and/or syntactic matching, it may provide similar accuracy to algorithmic solutions, although it still introduces nondetermism in grading.

2.2. Intended Learning Objectives and examination

D. Osinga [4] mentions that the top-down approach of defining Intended Learning Objectives, translating them into exercises, and then constructing grading rubrics for these exercises seems to produce exercises and grading rubrics that relate most to a module's ILOs. In the name of transparency, one could encode these ILOs into the grading rubric and show these in the final grade, to indicate to students how well they achieved the learning goals of the module.

While we could find little research on the inclusion of ILOs in grading rubrics, A. Dinur *et al.* [45] mention that "rubrics are a way of explictly stating the criteria of student work", allowing for a more analytical style of grading, which provides more details than a more global, holistic rubric [46]. If ILOs are more explicitly integrated into the grading rubric, and this rubric is of the analytical kind, this would allow students to receive more detailed feedback about their competency with respect to the ILOs of the course, which helps with grading transparency.

2.3. Conclusion

In the explored related work, existing frameworks primarily recommend structural matching in combination with syntactic and semantic matching to be able to effectively grade solutions containing 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, albeit at the cost of the teacher having to produce one or more sample solutions. Purely AI-driven methods require less effort from teachers, since they do not need to produce sample solution(s), but produce noticeably inferior results to graph matching algorithms. Using hybrid methods, specifically using ML-/LLM-based classification algorithms for semantic/syntactic matching and graph isomorphism for structural matching, seems to produce similar results to 'pure' graph matching algorithms, but does not provide accuracy gains over algorithmic solutions and can additionally introduce nondeterminism in otherwise determinstic solutions, which reduces consistency.

3. TOOLS AND TECHNIQUES

Given existing works, the best approach for maximising accuracy, consistency, and transparency seems to be to use graph isomorphism algorithms akin to those suggested by N. Smith *et al.* [35], implemented by W. Bian *et al.* [5] and P. Thomas *et al.* [17]. Using a visual representation such as 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 enough time.

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 incur extra work for a teacher to add metrics on how much a certain ILO is tested. Therefore, we aim to add support for ILO weights either per part 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 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.

To maximise the flexibility of each individual component, we aim to implement a query framework, akin to that of the Rust compiler [47]. This encourages decoupling the input parsing, the algorithm, and output formatting. This additionally supports transparency internally in the grading process, as one can easily query intermediate solutions from the grading process.

Additionally, 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 refetch some parsed input, or if we need to grade a solution we have already seen before.

3.2. Language(s)

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, it is not Object-Oriented, but still allows for attaching methods to certain data structures, thereby allowing us to express the diagrams as pure data while still allowing for the familiar dot-syntax (`object.property` or `object.method()`) of Object-Oriented languages. Finally, we opt for Go as it does not enforce strict object lifetimes and memory safety unlike languages such as Rust, which should make it faster to develop *Seshat*.

Author	Di*	Ac	Co	Tr	OSS	ILO	UTML
W. Bian *et al.* [5]	UML Class	H	H	H	M	N	N
M. Hosseinibaghdadabadi *et al.* [21]	UML Use Case	H	H	H	M	N	N
O. Anas *et al.* [22]	UML Class	M	H	H	M	N	N
S. Modi *et al.* [23]	UML Class	?	H	H	M	N	N
R. Jebli *et al.* [24]	UML Class	?	H	H	M	N	N
N. H. Ali *et al.* [25] [26]	UML Class	?	H	L	L	N	N
H. AlRawashdeh *et al.* [27]	UML State/ Sequence	?	H	?	M	N	N
M. Striewe *et al.* [28]	UML Class	?	H	H	L	N	N
S. Foss *et al.* [29] [30] [31]	ER	?	H	?	M	N	N
P. Thomas *et al.* [9] [32] [33] [17] [34]	ER	H	H	H	M	N	N
D. R. Stikkolorum *et al.* [36]	UML Class	L	L	L	L	N	N
C. Wang *et al.* [37]	UML	M	L	M	M	N	N
N. Bouali *et al.* [42]	UML Class	M	M	M	M	N	N
R. Ramachandran *et al.* [44]	ER	H	M	H	M	N	N

Table 1: Autograders and their suitability scores.

***Di**(*agram type*), **Ac**(*curacy*), **Co**(*nsistency*), **Tr**(*ansparency*), **OSS** = *how much of the solution is open-source*, **ILO** = *support for linking ILOs to grading*, **UTML** *support*.

Scoring is divided into "N" (*No Support*), "L" (*Low*), "M" (*Medium*), "H" (*High*), and "?" (*Unknown*), which gives an indication of suitability w.r.t. that particular criterium. The scoring is done in a comparative way, with the lowest-scoring solution receiving a "L" or "N" and the highest scoring receiving a "H". High **consistency** is awarded for determinstic solutions. High **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)). High **OSS** is given to solutions that completely open-source their work, with lower scores indicating that partial algorithms/methods are available. For AI-driven solutions, we also takes into account the open-source nature of the models used.

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

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, in addition to keeping a daily journal of our progress to be able to more effectively reflect on the process, which should aid in planning efficiency.

The increments are initially structured in the way defined in Table 2. These are subject to change, as it might turn out there is more research needed to complete certain algorithms or architecture.

Increment	Task
Wk. 6 - 7	*Seshat* - set up prototype architecture
Wk. 8 - 9	*Seshat* - UTML input parsing
Wk. 10 - 11	*Seshat* - internal graph representations
Wk. 12 - 13	*Seshat* - implement algorithms [5] [17] [35]
Wk. 14 - 15	*Seshat* - implement algorithms [5] [17] [35]
Wk. 16 - 17	*Seshat* - implement algorithms [5] [17] [35]
Wk. 18 - 19	*Seshat* - implement algorithms [5] [17] [35]
Wk. 20 - 21	*Seshat* - implement algorithms [5] [17] [35], compare to manual grading
Wk. 22 - 23	*Seshat* - compare to manual grading
Wk. 24 - 25	Finalise paper / buffer time
Wk. 26 - 27	Finalise paper / 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*.

BIBLIOGRAPHY

[1] "Object Management Group 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." [Online]. Available: https://www.visual-paradigm.com/

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

[16] "utml.apps.utwente.nl." [Online]. Available: https://utml.apps.utwente.nl/

[17] 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

[18] 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

[19] 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

[20] 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.

[21] 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

[22] 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

[23] 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

[24] 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

[25] 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

[26] 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.

[27] 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

[28] 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.

[29] 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.

[30] 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.

[31] 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

[32] 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.

[33] 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.

[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] D. Allen and K. Tanner, “Rubrics: Tools for making learning goals and evaluation criteria explicit for both teachers and learners,” *CBE—Life Sciences Education*, vol. 5, no. 3, pp. 197–203, 2006, [Online]. Available: https://www.lifescied.org/doi/full/10.1187/cbe.06-06-0168

[47] R. 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/