Towards Automated Grading of UML Class Diagrams with Machine Learning

Dave R. Stikkolorum[1][0000-0001-8935-1605], Peter van der Putten[1][0000-0002-6507-6896], Caroline Sperandio[2], and Michel R.V. Chaudron[3][0000-0001-7517-6666]

[1] LIACS, Leiden University, The Netherlands
[2] ESIEE-Amiens, France
[3] Chalmers and University of Gothenburg, Sweden

d.r.stikkolorum@liacs.leidenuniv.nl,
p.w.h.van.der.putten@liacs.leidenuniv.nl,
caroline.sperandio@esiee.org, chaudron@chalmers.se

Abstract. This paper describes an exploratory study on the application of machine learning for the grading of UML class diagrams. Bachelor student pairs performed a software design task for learning software design with the use of UML class diagrams. After experts had manually graded the diagrams, we trained a regression model and multiple classification models. Based on the results we conclude that prediction of a 10 point grading scale can't be done reliably. Classifying with trained data using expert consensus and a rubric comes closer to accuracy, but is still not good enough (a precision of 69%). Future work should include larger training sets and an exploration for other features.

Keywords: automated grading · software design · class diagrams · UML

1 Introduction

Many university programs, including ours, support a learning by doing approach for learning software design. Software design is known to be a discipline that needs to facilitate students with a lot of of exercise time. For large groups of students, lecturers face an enormous amount of assignments to grade. In classical lectures, accompanied with practice labs, 200 students or more are not an exception. Grading a single class diagram can easily take 10 to 15 minutes per student. Hence, the grading of practical assignments and running practice labs has a high demand on the presence of lecturers and teaching assistants.

In addition, it is a challenge for teachers to provide students with immediate feedback when the need arises. Moreover, most software design assignments do not have a single right answer. There a multiple solutions that solve a particular problem. Novice software designers lack the experience to recognise the solution space. Therefore it is difficult for students to perform self-evaluations of their solutions. This is a pity, because self-evaluation, supported by automated evaluation, would enable students to practice at home more often in addition to their university hours, at their own pace. Further, it enables students to take small steps while practising their task. In summary, a system that grades students' design assignments would decrease the demand on the lecturers' time and potentially offer faster feedback.

This paper reports on an early exploration of the possibility of using machine learning for the automatic grading of students' work. The main research question is: How can machine learning contribute to the automated grading of software design class diagrams tasks?

By answering the research question we aim to improve our (online) learning approaches. Integrated automated evaluation can enrich our learning environments. Future learning environments would be mature environments for practice and testing. In addition to grading, machine learning could improve feedback mechanisms such as our feedback agent [1]. These rich learning environments enable students to reflect on their tasks more often. Moreover, in cases of massive online courses, manual grading is not an option anymore. Overall automated evaluation can decrease the workload of lecturers.

We approached our research as follows. Student pairs were given a UML class design task, and in total 99 pairs were graded by 3 experts. Features were extracted from the UML models, and models were built to predict the grades. Rule based tools for UML tasks exist, as well as machine learning based approach for grading programming assignments, but to our knowledge our paper is the first to study how a machine learning based could be used to grade UML modeling assignments. Note that our work is exploratory, and with that in mind our preference is to present results for a simple approach first over using more technically sophisticated approaches.

The remainder of this paper is structured as follows: Section 2 discusses related work. Section 3 presents the research method we used. The results are presented in Section 4 and discussed in Section 5. Section 6 concludes the paper.

2 Related Work

For the automated evaluation of software analysis and design assignments with the use of UML modelling there are several approaches that use an example solution for comparison with the student's solution.

Hasker et al. present their tool: *UMLGrader* [4]. It compares student solutions against a standard solution. They supplied the students with a very constrained assignment and therefore minimize the solution space. By repeating their submissions students are guided towards an acceptable solution. Despite the name, the systems doesn't grade but only produces a binary pass-fail decision. The grading is theory based, i.e. based on predefined criteria, not machine learning model based. *UMLGrader* is based on UMLint, a UML diagram defect identifier

[5], especially developed for students to learn the UML notation and not software design.[4]

Striewe et al. present a static analysis approach for automated checks on UML diagrams with the help of rules that are executed on graphs that represent the content of a diagram [8]. Also, they focus on behavioural diagrams and present their approach for evaluating activity diagrams [9].

Bian et al. propose a grading algorithm that uses a syntactic, semantic and structural matching against a template solution [2]. Their approach uses meta-models to map the teachers solution with the student solutions and to grade model elements. In a case study 20 students were automatically graded with a 14% difference from the lecturer's grades.

In our own research with our educational UML tool WebUML [7] we introduced a pedagogical feedback agent [1]. The feedback agent also compares the solution with a standard solution. We use synonyms for providing some flexibility. We did not use the feedback agent for grading but for guiding feedback during software design tasks.

For the automated evaluation of student solutions for programming assignments there are examples of the application of machine learning. A big advantage of using source code is that it can be tested. This is not the case for software designs when using a non domain specific language, such as UML.

Srikant et al. demonstrated an approach for grading practical assignments for programming assignments [6]. Manual grading was done based on a 5 points scale rubric. The precision range of their research results is 55%-77%, based on experiments with a sample size of 84-294 students. Based on the findings of Srikant et al., Boudewijn explores the use of machine learning for predicting the manual grading of Java assignments in relation to the automation of hiring processes [3]. In addition to Srikant et al. semantic features were added to the test set. She did not find this to be significant better that without the semantic features. However, it seems that the features have a positive influence on the test case accuracy.

To our knowledge, tools for the automated grading of software design assignments are scarce and there are no tools that use machine learning for predicting the grades.

3 Method

This section describes the method we used for our research. First we present the overall research approach. Second, we present the participants, explain the students' design task and the manual grading process of the assignments. Subsequently we explain the features that were extracted from the data we use. We close with explaining how we build and analyse our models.

3.1 Overall Approach

We performed two types of grading approaches: one in which the graders divided the grading work (experiment 1) and one in which the graders individually looked

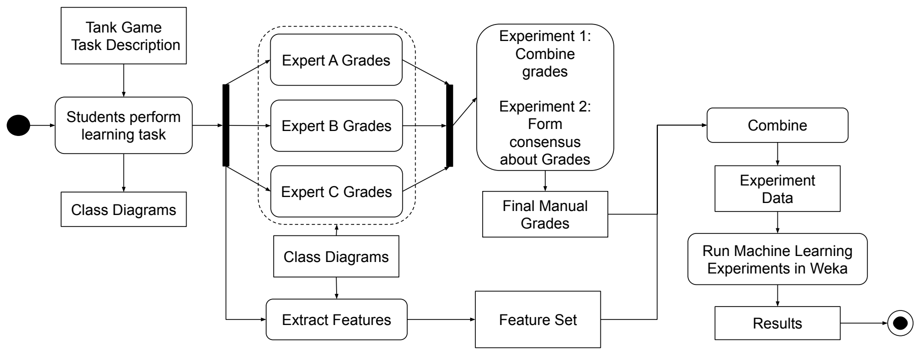
Tank Game Task Description
Students perform learning task
Class Diagrams
Expert A Grades
Expert B Grades
Expert C Grades
Class Diagrams
Extract Features
Experiment 1: Combine grades
Experiment 2: Form consensus about Grades
Final Manual Grades
Feature Set
Combine
Experiment Data
Run Machine Learning Experiments in Weka
Results

Fig. 1: Overall Research Approach

at all the work based on a rubric and had to come to a consensus (experiment 2) Figure 1 shows the overall approach which consisted of the following steps:

1. Task Performance – Students performed a practical learning task, designing a class diagram for a game.
2. Expert Grading – three experts manually graded the produced UML class diagrams. For experiment 1 the experts divided the workload by grading their own part of the student submissions. For experiment 2 all experts graded all submissions and a final grade list was composed based on the consensus of the experts.
3. Defining and extracting the features – Next to an image representation, the class diagrams' actual values (e.g. class name, attribute name, multiplicity value etc.) and the UML element type themselves (e.g. class, operation, association etc.) were recorded. A selection from these characteristics was made and used as features for conducting the machine learning experiments.
4. Experiments with various machine learning algorithms – In Weka's Experimenter we ran classification and regression experiments with the use of several well known methods and algorithms, such as the random forest algorithm.

3.2 Participants

We invited 120 student pairs to perform a class design task. After cleaning the data 99 student pairs remain. The cleaning consisted of incomplete assignments and removing outliers. The students followed the third year of a bachelors program in Computer Science. They were familiar with UML, software design principles and programming. The lecturers of the university embedded our experiment in the practical part of their course. The students were used to work in pairs.

Three experts, one lecturer and two PhD students, performed the grading of the students' work. A rubric support their manual grading. The grading scales that were used, were a 10 points (experiment 1) and a 5 points (experiment 2) scale. Consensus about the grade was discussed live or by (video) conference calls.

3.3 Assignment

All participants worked on the same assignment. The assignment was to model a class diagram for a tank game (Appendix A). The solution contains 10-12 classes using attributes, operations, named associations, inheritance and multiplicity. The assignment is chosen because it is representative for the students' study level.

The class diagram is created online with our WebUML editor [7]. The editor is capable of saving the diagram to a XMI file, a standard file format for UML diagrams, and an image file in png[4] format.

Table 1: Conversion table grading

10 points	1	2	3	4	5	6	7	8	9	10
5 points	1		2		3		4		5	
3 points	fail				pass			good		

3.4 Grading

The experts graded the students on a 10-point scale. For experiment 1 an expert graded a part of the students' work. The three expert grades were combined later. In addition, the grading of experiment 1 was done with two extra scales: a 5-point – and a 3-point scale. The extra scales were derived from the 10-point grades, and were not the result of an independent grading activity. For all the scales, see the overview in table 1.

For experiment 2 the experts graded based on a rubric that uses a 5 point scale (Appendix B) and formed a consensus grade later. There is no 3 points scale translation, because in experiment 1 this is derived from the 10 points scale, which is not used in experiment 2.

3.5 Feature Extraction

The features that are used for training our model are extracted from the XMI file that contains the student's solution. Together with the experts' grades the features are stored in a database. In this research we used two type of features: i) a generic set, which focused on the presence of important elements and the use

4 https://fileinfo.com/extension/png

Table 2: List of features

Attribute code	**Description**
GENERIC FEATURES	
ICC	Important Class Count
IATC	Important Attribute Count
IOC	Important Operation Count
IASC	Important Association Count
IIHC	Important Inheritance Count
IAGRC	Important Aggregation Count
ICOC	Important Composition Count
IINC	Important Realisation Involving Inhertance Count
SPECIFIC FEATURES	
TANK_C	Class Tank Present
BULLET_C	Class Bullet Present
SHELL_C	Class Shell Present
WORLD_LEVEL_C	Class World/Level Present
SCORE_C	Class Score Present
PLAYER_USER_C	Class Player Present
TANK_AT	Attribute in Tank Present
BULLET_AT	Attribute in Bullet Present
WORLD_LEVEL_AT	Attribute in World/Level Present
SCORE_AT	Attribute in Score Present
TANK_O	Operation in Tank Present
BULLET_O	Operation in Bullet Present
WORLD_O	Operation in Bullet Present
SCORE_O	Operation in Bullet Present
PLAYER_O	Operation in Bullet Present
TANK_BULLET_AS	Association between Tank and Bullet
TANK_TANK_AS	Self-association of Tank
TANK_PLAYER_AS	Association between Tank and Player
GAME_LEVEL_AS	Association between Game and Level
TANK_SCORE_AS	Association between Tank and Score
TANK_IH	Inherits from Tank
AMMO_IH	Inherits from Ammo
BULLET_IH	Inherits from Bullet
TANK_WORLD	Aggregation - Tank part of World
SCORE_WORLD	Aggregation - Score part of World

of specific structures (such as e.g. inheritance or aggregation). And ii), a specific set that focuses on the specific use of elements that carry a specific name (or variation thereof).

The data sets that were used for building prediction models consisted of a feature set in combination with one of the grading scales used as classifier. A feature set consisted of all features (generic + specific) or only the generic features. The features are listed in table 2.

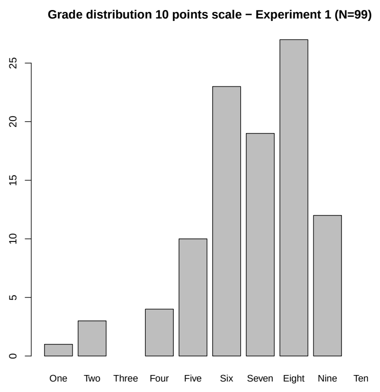

Fig. 2: Grade distribution 10 points scale - Experiment 1

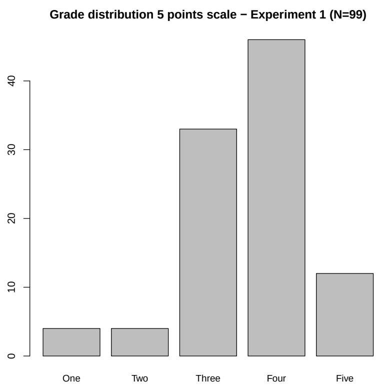

Fig. 3: Grade distribution 5 points scale - Experiment 1

3.6 Modeling

For analysing the data and running the machine learning experiments we used Weka[5] (version 3.8.3). We built classification and regression models using a range of algorithms. We evaluated the classification models using accuracy and AUC, through ten runs of five and ten fold cross validation (five fold results omitted for brevity). The regression models were evaluated by analysing the mean absolute error (again, five fold results omitted for brevity).

4 Results

First, we present the distribution of the grades for the various scales. Next, we will present the results for the various classification and regression models. For both the classification model-table as well as the regression models, the results are compared with majority vote and predicting the overall average (ZeroR), as a baseline benchmark for the other models.

4.1 Grade distribution

Figures 2, 3 and 4 show the distribution of the grades for experiment 1 using a 3-, 5- and 10-point scale. Figure 5 shows the distribution for experiment 2. For the second experiment, the experts were able to come to a consensus for 85 student submissions. As can be seen, the distribution is non-uniform and skewed, and this will be taken into account when interpreting the model results.

[5] https://www.cs.waikato.ac.nz/ml/weka/

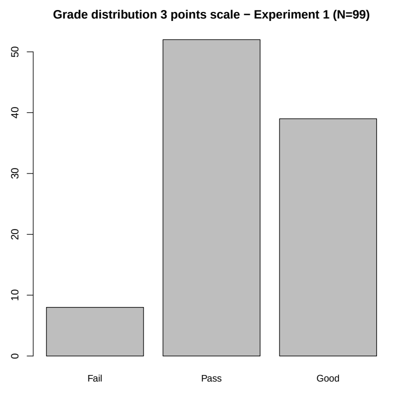

Fig. 4: Grade distribution 3-pnts scale - Experiment 1

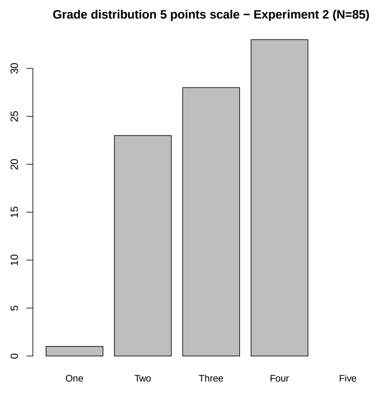

Fig. 5: Grade distribution 5-pnts scale - Experiment 2

4.2 Classification models

For the classification models, we evaluate the performance of the models by two measures: accuracy (percentage correct) and AUC (area under the ROC curve). To take the skewedness and non uniformity of the outcome (grade distribution) into account, accuracies are compared against majority vote, and it was also one of the reasons to include AUC.

Table 3 shows the analysis of the accuracy of the different prediction models. What can be observed is that the coarses the grading scale gets in the first experiment, the better the performance of the model is – which is to be expected. The best accuracy is found in the second experiment. This experiment delivers a significant accuracy of 69.42. Compared to the majority vote baseline (ZeroR) of 38.82 this gives the best result.

In addition to the accuracy, Table 4 shows the AUC. We did to double check our findings for the accuracy. The AUC values support our finding that the random forest algorithm has the best performance to build the model.

4.3 Regression models

Table 5 shows the results for the regression models. We analysed the mean absolute error (MAE). Which means that if we use regression as a prediction model, using the linear or random forest algorithm, the best models will be between 0.47-0.56 point off at best. Although significant, in comparison with just giving everyone the average grade (ZeroR, MAE = 0.71) this is not a large difference.

Table 3: Classification experiment 1 and 2 - accuracy (10 runs, 10 fold). Values in italics are significantly better than ZeroR at the 0.05 level.

Dataset	ZeroR	logistic	simple-log	1-nn	dec. table	dec. tree	rnd forest	rnd tree	naive bayes
all 10 1st	27.30	30.89	33.57	32.98	*37.31*	29.07	37.44	32.71	34.92
generic 10 pts 1st	27.30	32.2	32.88	*42.30*	28.31	33.32	*42.76*	*39.30*	*39.02*
all 5 1st	46.47	45.78	*59.66*	54	*58.41*	52.73	*58.02*	47.38	40.51
generic 5 pts 1st	46.47	49.61	53.34	51.06	54.64	51.89	53.97	46.62	51.52
all 3 pts 1st	52.58	57.06	62.03	*62.82*	*68.79*	59.76	*66.30*	59.89	*64.46*
generic 3 pts 1st	52.58	*64.14*	*62.33*	*64.66*	60.46	60.57	*65.24*	*64.30*	*65.20*
all 5 pts 2nd	38.82	*51.49*	*56.12*	*58.88*	*50.92*	*60.33*	*64.00*	*55.19*	*57.89*
generic 5 pts 2nd	38.82	*61.33*	*59.08*	*54.49*	*46.76*	*66.10*	*69.42*	*59.01*	*55.76*

Table 4: Classification experiment 1 and 2 - AUC (10 runs, 10 fold). Values in italics are significantly better than ZeroR at the 0.05 level.

Dataset	ZeroR	logistic	simple-log	1-nn	dec. table	dec. tree	rnd forest	rnd tree	naive bayes
all 10 1st	0.50	0.62	*0.66*	0.52	*0.72*	0.59	*0.73*	0.56	*0.64*
generic 10 pts 1st	0.50	*0.63*	*0.63*	*0.68*	*0.57*	*0.67*	*0.76*	*0.65*	*0.69*
all 5 1st	0.50	0.61	*0.65*	*0.64*	*0.65*	0.59	*0.74*	*0.60*	0.59
generic 5 pts 1st	0.50	0.61	*0.62*	0.59	*0.62*	0.62	*0.67*	0.57	*0.63*
all 3 pts 1st	0.50	*0.70*	*0.79*	*0.71*	*0.80*	*0.70*	*0.84*	*0.67*	*0.85*
generic 3 pts 1st	0.50	*0.81*	*0.81*	*0.71*	*0.72*	*0.73*	*0.84*	*0.71*	*0.83*
all 5 pts 2nd	0.50	*0.72*	*0.82*	*0.77*	*0.70*	*0.75*	*0.87*	*0.69*	*0.83*
generic 5 pts 2nd	0.50	*0.86*	*0.85*	*0.74*	*0.58*	*0.76*	*0.86*	*0.67*	*0.84*

Table 5: Regression models - MAE (10 runs, 10 fold). Values in italics are significantly better than ZeroR at the 0.05 level.

Dataset	ZeroR	linear	simple linear	random tree
all 10 1st num	1.34	*1.00*	1.28	1.08
generic 10 pts 1st num	1.34	*0.99*	1.24	1.09
all 5 1st num	0.74	0.62	*0.63*	0.62
generic 5 pts 1st num	0.74	*0.57*	*0.62*	0.62
all 5 pts 2nd num	0.71	*0.55*	*0.56*	0.58
generic 5 pts 2nd num	0.71	*0.48*	*0.56*	*0.47*

5 Discussion

In this section we answer our main research question: How can machine learning contribute to the automated grading of software design class diagram tasks?). In order to do this we reflect on the models of Section 4. In addition we discuss threats to validity and discuss future work.

5.1 Reflection on models

In general, based on the results, we can say that for now it is not reliable enough to apply the model to grade students class diagrams automatically on a 10 point scale. The highest accuracy we have achieved is 42.76%. What we can say is that the rougher the grading scale gets, the higher the accuracy, which is to be expected. In the first experiment this leads to a top accuracy of 59.66% using a 5 point (46.47% ZeroR) and 68.79% (52.58% ZeroR) using a 3 point scale.

But how applicable would such a coarse scale grading then be? It could be used in online training systems where the emphasis lies on getting s rough indication of quality of the solution. Also, often at the universities, practical parts of SE courses only require a Pass-Fail. That said, it would lack feedback about the errors made for the students. In addition, compared to the majority vote we can say that the models do not perform that much better.

In the second experiment we achieved a accuracy of 69% using a 5 point scale and trained by a manual 5 point scale grading. Also, the graders looked at all the models and came to a consensus. In this case not only the accuracy is higher, but also the delta with ZeroR (39%) is larger.

From the results it is not clear that adding specific features is a better approach than just using the generic features. Looking at the random forest column we observe that the generic features perform better in the 10 points scale of experiment 1 and in experiment 2. However, this is not consistent throughout all results. Even if there is a significant difference, it is not a big difference.

5.2 Threats to validity

We are aware of the validity threats in this research. We discuss the following: The experts graded a part of the submitted assignments twice in the second experiment. This could introduce a bias. We expect that the use of the rubric and the consensus of the experts lead to a high accuracy and that it did not negatively influenced the experiment.

The features in this case are specific to the assignment. On one hand one could argue we cannot generalize the approach presented is this paper. On the other hand the process that delivers the features could be repeated in other cases with other assignments. Future work should confirm the generalizability of the approach.

In general we are aware that a larger number of subjects would yield a more accurate model when using machine learning.

5.3 Future work

The goal of this work was not to build the best model possible, it was to explore the feasibility of this use case. If the grading task is very well defined, for instance through a rubric, there would be no need to use a machine learning approach as the rubric could simply be automated. However, it becomes more interesting if the rubric is not well known, elements may be missing, or the weighting of various elements is not well defined.

One angle to future work would be to repeat these experiments with more instances, and different features. Also grading tasks that are less well defined, such as the quality of the layout of the diagram, could lend itself better to a machine learning based approach. What will remain challenging is to develop models that would generalize and validate across different exercises, this could be evaluated with a separate validation exercise.

In this paper we constrained the evaluation of students' work to grading. Future research should explore the application of machine learning for providing feedback to students during the performance of software design tasks.

6 Conclusion

In this paper we presented our approach for using machine learning for building models for the prediction of grades for software design assignments. We collected the data of submitted software design assignments of bachelor student pairs. With this data we performed two experiments in which we built models for prediction with machine learning software.

Based on the results of the first experiment we conclude that using machine learning for a classification that uses the same precision as a 10 point grading scale is not accurate. Classifying in a 3 points (fail, pass, good) or maybe even using a 2 points (fail, pass) comes closer to accuracy, but is still not good enough (an accuracy of 65.24 for 3 points). The models of the second experiment performed better. The best model had an accuracy of 69.42 using a 5 points grading scale. Again, this model is not accurate enough, but heads towards an acceptable value.

Currently we could integrate a prediction model in online training software to give users a rough indication of the quality of their models. Future work should include larger training sets and the exploration of the application within e-learning environments and for other features such as providing feedback.

References

1. Anckar, H.: Providing Automated Feedback on Software Design for Novice Designers. BSc thesis, Goteborg University (2015)
2. Bian, W., Alam, O., Kienzle, J.: Automated grading of class diagrams. In: Proceedings of the Educators' Symposium at the 22nd International Conference on Model Driven Engineering Languages and Systems (2019)

3. Boudewijn, N.: Automated Grading of Java Assignments. Master's thesis, Utrecht University, The Netherlands (2016)
4. Hasker, R.W.: UMLGrader: an automated class diagram grader. Journal of Computing Sciences in Colleges pp. 47–54 (2011)
5. Hasker, R.W., Rowe, M.: UMLint: Identifying defects in uml diagrams. In: American Society for Engineering Education. American Society for Engineering Education (2011)
6. Srikant, S., Aggarwal, V.: A system to grade computer programming skills using machine learning. Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14 pp. 1887–1896 (2014)
7. Stikkolorum, D.R., Ho-Quang, T., Chaudron, M.R.V.: Revealing students' uml class diagram modelling strategies with webuml and logviz. In: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications. pp. 275–279. IEEE (2015)
8. Striewe, M., Goedicke, M.: Automated checks on uml diagrams. In: Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education. pp. 38–42. ITiCSE '11, ACM, New York, NY, USA (2011)
9. Striewe, M., Goedicke, M.: Automated assessment of uml activity diagrams. In: ITiCSE. p. 336 (2014)

A Appendix: Tank Assignment

The Modeling Task – a Tank Game[6]

In this task you will design a game with use of the UML class diagram. You do not need to use packages in this assignment. The description of the game is as follows: A player (user) controls a certain tank. This tank is a Panzer Tank, a Centurion Tank or a Sherman Tank. They fire bullets and Tank shells. Bullets can be Metal, Silver or Gold bullets.

A tank moves around a world (level). The aim is to destroy all other tanks in the world. After a world has been completed the tank advances to the next world. A list of all the worlds visited is kept.

An entire game consists of 8 levels. A world contains a maximum of 20 tanks that compete for victory. Each tank remembers which tanks it has destroyed in the past. The score for each level is kept by a scoreboard that gets notified by the individual tanks each time an opponent is shot. The players control their tanks through an interface allowing for steering, driving (reverse / forward), switching ammo and firing.

[6] text: B. Karasneh

B Appendix: Grading Rubric

Table 6: Class Diagram Rubric for Grading Design Modelling

Grade	**Judgement, criteria description**
1	The student does not succeed to produce a UML diagram related to the task. He/she is not able to identify the important concepts from the problem domain (or only a small number of them) and name them in the solution/diagram. The diagram is poor and not/poorly related to problem description with a lot of errors: high number of wrong uses of UML elements mostly no detail in the form of attributes or operations
2	The student is not able to capture the majority of the task using the UML notation. Most of the concepts from the problem domain are not identified. The detail, in the form of attributes or operations, linked to the problem domain is low. Some elements of the diagram link to the assignment, but too much errors are made: misplaced operation / attributes non cohesive classes few operation or attributes are used
3	The student is able to understand the assignment task and to use UML notions to partly solve the problem. The student does not succeed to identify the most important concepts. A number of logical mistakes could have been made. Most of the problem is captured (not completely clear) with some errors: missing labels on associations missing a couple important classes / operations / attributes Logical mistakes that could have been made: wrong use of different types of relationships wrong (non logical) association of classes
4	The student captures the assignment requirements well and is able to use UML notations in order to solve the problem. Almost all important concepts from the problem are identified. Some (trivial) mistakes have been made: Just one or two important classes / operations / attributes are missing Design could have been somewhat better (e.g. structure, detail) if the richness of the UML (e.g. inheritance) was used.
5	Student efficiently and effectively used the richness of UML to solve the assignment. The problem is clearly captured from the description. concepts from the domain / task are identified and properly named The elements of the problem are represented by cohesive, separate classes (supports modularity) with a single responsibility In the problem domain needed attributes and operations are present Multiplicity is used when appropriate Naming is well done (consistent and according to UML standard) Aggregation / Composition / Inheritance is well used No unnecessary relationships (high coupling) are included