Evaluating the layout quality of UML class diagrams using machine learning

Gustav Bergström[a], Fadhl Hujainah[a], Truong Ho-Quang[a], Rodi Jolak[a,b], Satrio Adi Rukmono[c], Arif Nurwidyantoro[d], Michel R.V. Chaudron[a,c]

[a] *Chalmers | Gothenburg University, Gothenburg, Sweden*
[b] *Volvo Car Corporation, Sweden*
[c] *Eindhoven University of Technology, The Netherlands*
[d] *Monash University, Melbourne, Australia*

Abstract

UML is the de facto standard notation for graphically representing software. UML diagrams are used in the analysis, construction, and maintenance of software systems. Mostly, UML diagrams capture an abstract view of a (piece of a) software system. A key purpose of UML diagrams is to share knowledge about the system among developers. The quality of the layout of UML diagrams plays a crucial role in their comprehension.

In this paper, we present an automated method for evaluating the layout quality of UML class diagrams. We use machine learning based on features extracted from the class diagram images using image processing. Such an automated evaluator has several uses: 1) From an industrial perspective, this tool could be used for automated quality assurance for class diagrams (e.g., as part of a quality monitor integrated into a DevOps toolchain). For example, automated feedback can be generated once a UML diagram is checked in the project repository. 2) In an educational setting, the evaluator can grade the layout aspect of student assignments in courses on software modeling, analysis, and design. 3) In the field of algorithm design for graph layouts, our evaluator can assess the layouts generated by such algorithms. In this way, this evaluator opens up the road for using machine learning to learn good layouting algorithms.

Approach. We use machine learning techniques to build (linear) regression models based on features extracted from the class diagram images using image processing. As ground truth, we use a dataset of 600+ UML Class

Diagrams for which experts manually label the quality of the layout.

Contributions. This paper makes the following contributions:

1) We show the feasibility of the automatic evaluation of the layout quality of UML class diagrams.
2) We analyze which features of UML class diagrams are most strongly related to the quality of their layout.
3) We evaluate the performance of our layout evaluator.
4) We offer a dataset of labeled UML class diagrams. In this dataset, we supply for every diagram the following information: a) a manually established ground truth of the quality of the layout, b) an automatically established value for the layout-quality of the diagram (produced by our classifier), and c) the values of key features of the layout of the diagram (obtained by image processing). This dataset can be used for replication of our study and others to build on and improve on this work.

Keywords: Quality of Layout, Machine Learning, Quality of UML class diagrams

---

1. Introduction

The Unified Modeling Language (UML) is an industry-standard to represent designs of software systems using diagrams. The UML consists of various types of diagrams that all serve different purposes and have different areas of use. In this study, we focus on one of the most commonly used diagram types: the class diagram [1, 2, 3]. Class diagrams belong to the diagram types that capture aspects of the structure of software/systems: they show the building blocks of a system and how they are related to each other. In class diagrams, classes are drawn as rectangles, and relationships are drawn as lines between the rectangles. Sometimes lines have an arrow indicating the direction of the relationship.

One of the primary purposes of UML diagrams is to provide an (abstract) overview of a system's structure. UML class diagrams are important aid for understanding software systems [4, 5] and for sharing the understanding of a system across development teams. Moreover, the human comprehension of systems is greatly affected by the quality of the diagram's layout: In the

study of the Störrle [6], participants were asked to perform certain tasks using diagrams which were rated with either "good layout" or "bad layout". The participants performed significantly better using the "good" diagrams. Layout quality can be about how aesthetically appealing a user finds the diagram and how easily the user can comprehend the diagram. Several specific layout aesthetics have been shown in research to affect the layout quality. An example is the number of crossing lines in a diagram: fewer crossing lines contribute to a higher quality, but there are quite a few more aesthetics.

Class diagrams are often created in one of two ways: 1) either as a design prior to and in order to guide the programming, or 2) after a system is developed in order to document the design and share/communicate knowledge about a design to a team of developers. Diagrams created before the development are said to be forward-engineered and can serve as a guiding reference during the development. Forward-engineered diagrams are usually drawn by humans, who tend to have an intuitive feeling for layout. In addition, humans are very good at recognizing high-quality UML diagrams. Nevertheless, at the same time, we found that human-made diagrams frequently ignore some straightforward aesthetical layout guidelines. Diagrams created after the development are said to be reverse-engineered and can serve as documentation of the system. The diagrams are usually created using an automatic generator that generates a diagram based on a (collection of) source code. A wide range of algorithms for creating reverse-engineered diagrams exist [7, 8, 9, 10, 11, 12]. It is crucial for such algorithms that the quality of the diagrams they produce is high. Hence, there is a need to assess the quality of UML diagrams. For manual and reverse-engineered diagrams, automated assessment of diagrams can indicate if the layout quality is good or whether it needs improvements. We foresee that an automated quality assessment tool could be integrated into a DevOps toolchain and be invoked when a UML diagram is added to the project files.

When constructing a diagram layout, there are many aesthetic criteria to consider. Some of them might even be conflicting: when trying to optimize one of the criteria, another might suffer. Therefore, it is essential to know which aesthetics are most important for the layout quality to be focused on when constructing layouts. By creating an extensive dataset of features representing aesthetics and labels representing the perceived quality, we can discover which aesthetics seem most important.

The majority of existing studies related to the assessment of the layout quality mainly focus on finding the most important rules for layout aesthet-

ics [13, 14]. These existing studies have focused on finding the (relative) importance of layout aesthetics and estimating the layout quality of the diagrams. These studies have not looked into the automatic evaluation of the layout quality of class diagrams. To evaluate the layout quality of diagrams, time and user-intensive studies are often conducted to see if the users find diagrams with one layout easier to comprehend than diagrams with other layouts. An automatic evaluator could quickly indicate how good a layout is. Therefore, the primary purpose of this study is to create an automatic evaluator of the layout quality of UML class diagrams using machine learning approaches. The tool could also suggest which aspects of a diagram are good and which could be improved.

The main research question of this study is:

RQ_{main}: How can machine learning be used to automatically evaluate the layout quality of UML class diagrams?

To answer the main research question, the following sub-questions are investigated:

RQ1: How well does the automatic evaluator of the layout quality of class diagrams perform?

RQ2: Which features of class diagram layout are the most important for evaluating their layout quality?

Given that we use a machine learning approach, the basis for creating the evaluator will be a set of ground truth data of diagrams, with their respective image features and layout quality.

The contributions of this work can be summarised as follows:

1) We offer a new automatic evaluator of the layout quality of UML class diagrams. The proposed evaluator is the first automatic evaluator of the layout quality for UML class diagrams to the best of our knowledge. The proposed evaluator can be a valuable tool for assessing the quality of class diagrams in industrial and academic settings.
2) We reveal the most important aesthetics for the layout quality of class diagrams. Furthermore, the results of such identification can assist in enhancing the performance of automated layout algorithms.
3) We evaluate the proposed automatic evaluator to affirm its capability to assess the layout quality of previously unseen class diagrams.

4) We offer a dataset of class diagrams together with extracted features and manually labeled quality, which can be used for further studies in the quality of diagram layouts.

The focus of this paper is on the automatic evaluation of the layout of class diagrams. We have argued above that the layout of class diagrams is important for communication and comprehension of designs. While evaluating layout is useful in its own right, it can contribute to evaluating designs that are represented by UML class diagrams. Indeed, there are obvious uses for evaluating the quality of UML designs, both in industrial settings (e.g. as part of quality assurance) and also in educational settings (e.g. for evaluating student's assignments). However, for evaluating the quality of a design, an additional complication is that also the semantics of the design and the domain need to be taken into account – amongst others: is there a clear allocation of responsibilities, do relations between classes make sense? A broader discussion of evaluating the quality of UML designs can be found in the PhD thesis of Christian Lange [15].

The remainder of this paper is structured as follows: Section 2 presents an overview of related work. Section 3 describes the proposed automatic evaluator. In Section 4, we evaluate the performance of the proposed layout evaluator. Section 5 discusses the results. Section 6 elaborates on threats to validity. Finally, section 7 concludes this study and provides recommendations for future work.

2. Background on Layout Aesthetics of Diagrams

This section describes background knowledge regarding the aesthetics of the layout of diagrams. In particular, we present a selection of characteristics of diagrams that have been proposed in the literature as being relevant to the layout quality of diagrams.

Layout aesthetics are properties of a diagram layout that can have a relationship to the subjective perception of the diagram. Since UML class diagrams can be seen as a type of mathematical graphs, aesthetics from the field of general graph layout are relevant. Research has been done both into general graph layout aesthetics, as well as aesthetics that are specific to UML class diagrams.

According to Störrle [6], four levels of design principles govern the layout of UML diagrams.

First, general principles apply to all kinds of diagrams. For example, elements should be aligned and not obscure each other.

Second, some principles apply to mathematical/abstract graphs. For example, the number of crossings and bends of lines should be minimized.

Third, some principles apply mostly to UML diagrams, for example, that similar elements should be grouped.

The fourth principle level is support for better addressing the audience, for example, by highlighting items through color or size.

Most research on UML diagrams focuses on principles from the second level. This is probably because these principles are, to a greater extent, quantifiable and measurable than principles from other levels. This is important for this study because the aesthetics need to be found through image processing and then converted into numerical features for the machine learning part to work. We scope this study to the graphical properties of diagrams. Hence metrics that relate to the semantics of class diagrams will not be considered.

Purchase [16] presents seven common aesthetic criteria for graph drawings and defines metrics to assess the presence of each one of them. In other research, Purchase et al. [13, 17] studies graph layout aesthetics with a focus on UML, where some additional aesthetics are presented. The aesthetics are evaluated empirically to find their relationships to user preferences. Ware et al. [18] studies how eight different graph layout aesthetics affect the cognitive load of finding the shortest path between two nodes in a graph. Eichelberger [19] describes and orders 14 aesthetic for class diagrams in a priority list. Eichelberger [14] extends on this in his Ph.D.-thesis and presents a large number of aesthetics that are grouped into different categories. In later work, Eichelberger and Schmid [20] make an extensive summary of guidelines for the aesthetic quality of UML diagrams on different levels based on prior work. Sun and Wong [21] present 14 criteria for UML class diagram layout, where some are more general and apply to all graph drawings, and some specifically target the semantics of the UML diagram. Coleman and Parker [22] present a list of 19 graph layout aesthetics that were derived from literature and common sense.

We have compiled the following categories of layout aesthetics mentioned in prior works.

A1 – Line crossings. A line crossing is a point in the diagram where two lines intersect. If more than two lines intersect in the same point, all pairwise intersections are considered [16]. Minimizing the number of line crossings is one of the most commonly referenced aesthetics. Crossings make the lines harder to follow [21], and it is harder to see which classes are connected [14]. In addition, crossings at small angles are more likely to cause visual confusion than crossings with an angle close to 90° [18].

A2 – Line bends. A line bend is a point on a line that does not lie on a straight line between the two endpoints of the line [16]. Minimizing the number of line bends is also a very commonly referenced aesthetic. Straight lines are more continuous [21] and easier to follow for users [14].

A3 – Orthogonality. Orthogonality applies both to lines and the orientation of nodes: The orthogonality of a line represents how far (as an angle) the direction of a line deviates from a horizontal or vertical line [16]. To improve layout quality, lines should be drawn horizontally or vertically on an orthogonal grid [14], [21]. This is because horizontal and vertical orientations are more likely to be perceived as figures than other orientations [20].

Orthogonality for nodes means that the nodes should be placed on an orthogonal grid, i.e., nodes should be—as much as possible—aligned both horizontally and vertically [16], [14], [21].

A4 – Line lengths. The length of a line is the distance from the start to the end of the line through all points on the line. Lines should not be too long or too short because long lines make grouping and separation hard [19], [21]. Line lengths should also be kept as uniform as possible in a diagram [14].

A5 – Diagram drawing size. The actual size of the diagram drawing should be minimized to support a homogenous node and line distribution and to reduce the need for scrolling [20]. Naturally, the drawing still has to fit all the diagram elements, and making it too small would probably create conflicts with other aesthetics. Some research only focuses on that it is the width of the drawing that should be minimized [22].

The aspect ratio is the relationship between a drawing's height and width. An optimal aspect ratio could be either minimized, which means that the diagram is quadratic [14] or fixed to a specific rectangular proportion [19], e.g., such that it fits on modern display monitors or fits nicely in an electronically typeset document (often portrait A4 or letter-format).

A6 – Symmetry. Existing research frequently suggests that increasing the symmetry of a diagram leads to increased understandability. Symmetric diagrams are usually seen as a good figure [21], [20]. However, it is an aesthetic that can be hard to define as it is best considered perceptually rather than computationally [17]. Symmetric lines could be drawn arbitrarily in diagrams (e.g., non-orthogonal), and there could be both local and global symmetry. According to Eichelberger [14], all kinds of symmetry should be maximized.

A7 – Line angular distance. If multiple lines are going from a node, the minimum angle between two lines should be maximized [16], [22]. The lines should be far apart so that it's easy to distinguish between them, which is of extra high importance if the resolution of the rendering of that the diagram is low [14].

A8 – Class placement. Much research on graph layout brings up the placement of nodes as an important aesthetic. Given that classes in UML diagrams can be seen as nodes in a graph, this research is also relevant for class diagrams.

Nodes should be distributed uniformly within the drawing area [14], [20], [22]. This helps provide a uniform appearance of the drawing, which supports similarity and homogeneity [20]. Dishomogeneity leads to double observation, a discontinuity in the visual perception process [14].

Nodes should not be too close together or too far apart [22]. Nodes that are connected should be as close as possible to each other [14]. Nodes should also not be too close to edges that they are not connected to [19], [22]. Classes with a high degree should be placed near the center of the diagram [14].

A9 – Overlapping. Nodes should not overlap other nodes [19], [14], [20], [21]. When nodes overlap, part of one node is not visible and the entire diagram cannot be read by the user. This is usually disliked by users [20].

Nodes and edges should not overlap each other [19], [14], [20], [21]. If a node overlaps an edge, it might look like the edge enters and exits the node instead of going past it. If an edge overlaps a node, it is not as problematic. This is usually tolerated by users but should, however, be avoided [20].

Edges that are not intended to be joined should not overlap each other, which

means that every edge should be readable as an individual [19], [14], [20]. Edge overlapping can be differentiated from edge crossing by defining overlapping as two edges having a path segment in common, rather than just a crossing point [20]. Edge overlapping can also be considered as edge crossing [21]. Edge overlapping is similar to node overlapping. At least a segment of one edge is not visible as an individual path, which means the entire diagram is not readable [20].

A10 – Class sizes. The difference among sizes of the rectangles representing classes should be minimized [14]. The class sizes should also be as small as possible [14], [19].

3. Related work and Limitations of our Study

First, in Section 3.1, we discuss related work on evaluating layout-quality of UML diagrams. This leads us to formulate some limitations on the scoping of our approach in Section 3.2.

3.1. Related Work

In Section 2, we described the concepts relevant to defining the quality of layout of (UML) diagrams and the aesthetic criteria that relate to it. Regarding the *automatic evaluation of the layout* of (class) diagrams, no prior work is known to us. Indeed, the extensive collection of UML diagrams used for this study is relatively new and unique. The creation of this dataset has opened up the opportunity for this study. However, some works are related to different parts of this study: this includes estimating layout quality and finding the most important layout aesthetics. The related work on these topics is described in the remainder of this subsection.

3.1.1. Classifying layout quality

As seen in Section 2, work has been done to analyze individual layout aesthetics and how they contribute to the perceived quality of a diagram. However, no work is found where the goal is to use those aesthetics to estimate the overall quality somehow. Störrle [6] classifies diagrams as "good" or "bad" to conduct his study, but mentions that not much emphasis is put on this classification. He classifies diagrams that conform to positive aesthetics and do not violate negative aesthetics as "good" diagrams.

3.1.2. Importance of aesthetics

When it comes to finding the importance of different layout aesthetics for the overall quality, some work has been done, which was also touched upon in Section 2. However, no solid empirical evidence was found for many of the aesthetics we reported earlier.

Purchase et al. [23] performed a user study to validate the correlation between some aesthetics and the understandability of graphs. The participants had to carry out tasks to test how well they could understand graphs with different conformance levels to the investigated aesthetics. The tasks had to be completed within a time limit, and the measured variable was whether the participants could find the correct answers to the tasks or not. The study showed that minimizing line crossings (A1) and line bends (A2) had a significant correlation with the understandability of graphs, while the hypothesis that increasing local symmetry (A6) increases understandability is unconfirmed.

Purchase [24] performed a similar study, with the difference that some additional aesthetics were investigated and that both the taken time and the correctness of the answers to the tasks were measured. The study showed that the effect of minimizing line crossings (A1) was significant for both time and correctness, the effect of minimizing line bends (A2) was significant for correctness but only approaches significance for time, and the effect of increasing symmetry (A6) was significant for time but not for correctness. However, the effect of increasing orthogonality (A3) and maximizing the minimum angle between edges leaving the same node (A7) was non-significant for both correctness and time.

Purchase et al. [13] performed another kind of user study to test the user preference of diagrams with different values of different aesthetics. The participants were given pairs of diagrams: one with a high value of a certain aesthetic and one with a low value of the same aesthetic. They then had to choose which of the diagrams they preferred. The data were analyzed by calculating a percentage preference for each aesthetics. The percentage preference was 93% for fewer line crossings (A1), 91% for fewer line bends (A2), 73% for narrower diagrams (A5), and 61% for increased orthogonality (A3). All results were statistically significant.

Purchase et al. [17] also performed a user study where participants were given a text specification and an example diagram modeling the specification. The example diagram had a medium-high value of all of the investigated aesthetics. The participants were then presented with diagrams with higher and lower aesthetics values than the example diagram, where some diagrams modeled the same specification as the given one and some did not. They were to answer if the presented diagrams modeled the given specification or not, and both the accuracy and time of the answers were measured. After the study, the authors concluded that only the aesthetic of minimizing line bends (A2) seemed to matter, although only a little. None of the other investigated aesthetics, including line length variation (A4), orthogonality (A3), symmetry (A6), node distribution (A8), and having short but not too short lines (A4), had a significant impact.

Ware et al. [18] performed a user study where the time needed for participants to perceive the shortest path between two specified nodes in a graph was measured. In the graphs, the values of several aesthetics were varied. The study showed that the continuity of the shortest path (related to A2) and line crossings on the shortest path (A1) were significant. However, other aesthetics, including the total number of line crossings in the graph (A1), line crossing angles on the shortest path (A1), average line length on the shortest path (A4), and total line length on the shortest path (A4), were not significant.

As mentioned in Section 2, Eichelberger [19] orders 14 aesthetics for class diagrams in a priority list. The importance of aesthetics was validated through discussions in software engineering courses, evaluations of CASE tools, and the author's work on a domain-specific layout algorithm. However, no evaluation by making user studies was done. In the list, general constraints on nodes, including distances between nodes (A8), avoiding overlapping (A9), and minimizing class sizes (A10), are in third place. Avoiding line crossings (A1) is in fifth place. General edge constraints, including line lengths (A4) and line bends (A2), as well as not placing nodes too close to edges (A8), is in sixth place. Graph drawing constraints, including aspect ratio (A5), drawing size (A5), symmetry (A6), line angles (A7), and, again, line bends (A2), are in fourteenth place. Many of the other aesthetics on the list relate to semantics in diagrams.

In the field of Business Process Models (BPM), there is a considerable number of studies that research the quality, layout, and aesthetics of diagrams that depict business process models [25], [26]. On the one hand, both BPM and class diagrams are abstract descriptions of (software) systems. However, diagrams for BPM are quite different from class diagrams because business process models describe behavioral aspects of systems, while class diagrams describe the structural aspects of systems. Still, some general heuristics appear applicable to both, such as line-crossings, orthogonality, and symmetry.

Most of the approaches in the BPM area are aimed at finding individual 'flaws' of diagrams (suggesting opportunities for improvement), rather than being aimed at the grading of diagrams as a whole.

3.1.3. Prior work

On the topic of using image processing to find features of UML diagrams, earlier work was done by Karasneh et al. [27], [28]. They have developed a system that reads an image of a UML diagram and extracts its semantic meaning, i.e., the classes and relationships. It then creates an XMI file of the UML model from this information.

Moreover, Ho-Quang et al. [29] have developed an automatic classifier that recognizes whether an arbitrary image is a UML class diagram. That work was done using similar methods as used in our study: It uses image processing to find features in images and then applies machine learning to train a model that can distinguish if an image is a class diagram or not. The difference with this study is that we have to recognize more aspects and details of the layout of the UML diagrams for our problem. Furthermore, their approach delivers a binary classifier (UML-CD or not). Instead, in our approach, we aim for an evaluator that can estimate the layout quality as a numerical value (on a 5-point scale) for any UML class diagram.

In [30], Nikiforova et al. create a list of layout principles and then analyze how different layouting algorithms that are used by some popular UML modeling tools comply with these criteria. They look at layouts of both class and sequence diagrams.

3.2. Limitations & Scoping of our study

This section describes aspects of UML diagrams that we leave *out of scope* of our study for assessing layout quality.

1. *Uniformity* : A generally accepted guideline is to follow *uniformity*, i.e., apply the same layout principle in the same way to all elements in the diagram. This applies to many of the aesthetics above. For example: to use the orthogonal placement for all classes in the diagram—not only for some, or to use only straight angles in the elbows of lines—not a mix of straight and curved arrows. In a way, uniformity is a meta-principle, and we have not considered any features based on uniformity in the remainder of our study.
2. *UML Conventions* : Several layout conventions specific to UML class diagrams may not hold for arbitrary (structure) diagrams. For example, for generalization (inheritance) relations, the more general class is conventionally drawn (vertically) above the specialized class. Another example is that dependencies are preferably drawn pointing downward, possibly horizontal, but preferably not upward. While we imagine that including such conventions could be useful, we leave these aspects out of the scope of our current study.
3. *Text* : UML diagrams contain all types of text: as names of classes, attributes, operations, labels describing relations, or labels near the start/end of arrows to indicate multiplicities. The quality of diagrams is affected by many factors related to these texts, such as, e.g., the font and the distance of text to rectangles/lines. However, we leave factors related to the shape and placement of text in diagrams out of the scope of this study.
4. *Highlighting/Coloring* : In practice, UML diagrams frequently contain visual cues for highlighting, such as the use of color and sometimes the use of underlining or bold-facing or using thicker line-widths. While these are relevant for the cognition of diagrams, we leave out of scope all these aspects related to the highlighting/coloring in UML diagrams.
5. *Semantical aspects of the layout of diagrams* : There are also many 'semantical aspects' that relate to the quality of a diagram. For example, for inheritance relations, there is a convention that places the general class above its specialized class(es). However, this is not required by the UML-notation, and also some developers deviate from this. We point out that this property can only be checked by taking the meaning of lines (i.e. line represents inheritance) into account. In our study, we have opted to leave such aspects out of scope.
   For future reference, other examples of semantical aspects relate to: appropriate naming of classes/relations, appropriate directions of de-

pendencies, appropriate meaning of aggregation/composition relations.

6. *Design Principles* : Design principles apply to the quality of a design. Some design principles can be detected in a diagrams of a design, such as for example (high) coupling, by looking at the number of lines that is connected to a rectangle. Other principles that are mostly syntactical are: (avoid) cyclic dependencies, and use of proper layering (no jumping, no cycles). However, many other design principles can not be detected by looking at the layout of the diagram only (e.g. allocation of single responsibility). We imagine that other techniques (such as source code analysis) are better ways to detect quality of the design. In our study, we chose to leave checking design principles out of scope.
7. *Types of Class Diagrams* : Sometimes, the UML class diagram notation is used for depicting (relational) data models and hence represents ER-diagrams[1]. While our approach can, in principle, also work for ER diagrams, our focus in this paper is only on diagrams that represent software designs. Moreover, there are several pragmatic aspects that we needed to consider when creating the dataset we used for our study. More details on this are described in Section 4.1.

4. Research Method

This section describes the methods used for performing the research in this study. Figure 1 shows a high-level overview of our research. We create a curated set of UML Class diagrams out of an extensive collection of UML diagrams, establish a ground truth for the quality of the layout through manual labeling of the diagrams by a group of UML experts, apply image processing to extract various features of the layout of the diagram mentioned in the literature as potentially relevant to the layout quality of diagrams, and apply supervised machine learning on the set of images, ground truth, and features. Next, we explain these steps in more detail.

4.1. Construction of the Dataset of Images

The starting point for our research is the Lindholmen-dataset of images created by Hebig et al. [31]. This dataset consists of almost $100,000$ UML diagrams. This dataset was assembled through mining open source repositories on GitHub for images. The images found there were then classified

[1]Formally, UML is a superset of ER-diagrams

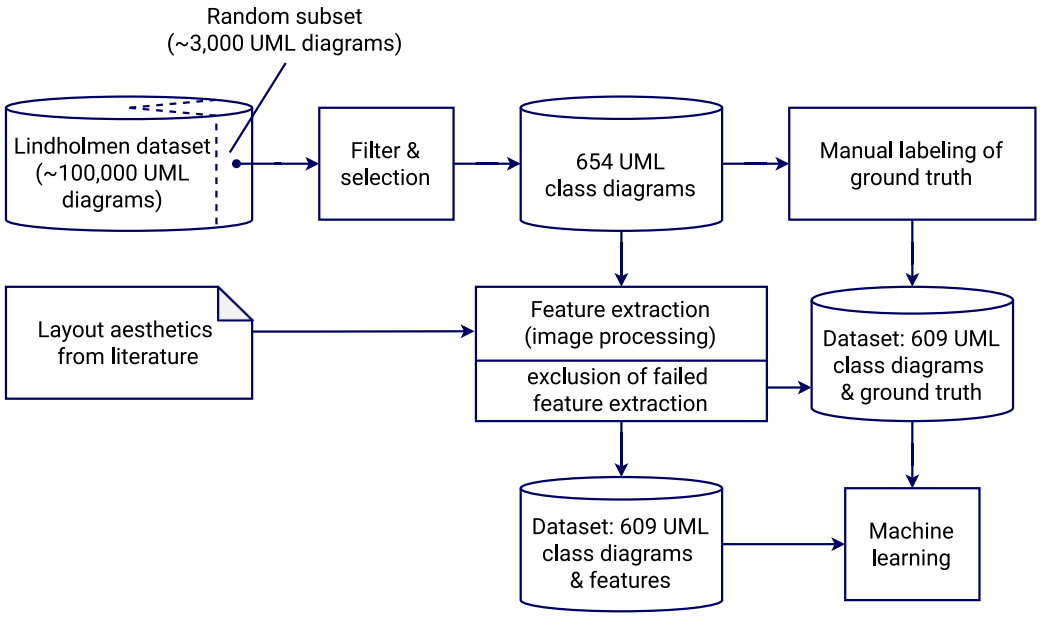


Figure 1: Overview of Research Method

as being either UML diagrams or not, using the work done by Ho-Quang et al. [29]. The Lindholmen dataset includes a variety of types of UML diagrams: most diagrams are class diagrams, but there are also sequence diagrams, component diagrams, and some others included. Moreover, due to being constructed through mining many GitHub projects, the Lindholmen dataset may (and does) contain some diagrams that are duplicates of each other (although sometimes under different file names).

Because we do manual labeling of the diagrams to obtain the ground truth, we scaled down to an initial subset of $3,000$ diagrams selected randomly from the Lindholmen dataset. From this subset, we further selected diagrams that satisfy the following criteria:

C1 – The image should be of the type class diagram. This is a scoping decision: our focus is on diagrams that represent a system's structure. For this effort, we have left out other diagrams that describe structure (such as component or package diagrams) to obtain a mostly homogeneous dataset. Extending our study to these other structural diagram types is an option for future work.

C2 – Classes should be represented as rectangular boxes in the image. It is part of the UML standard to draw class diagrams as rectangles. This did not exclude many diagrams. However, one category of class diagrams excluded were those where classes were drawn using rectangles with rounded corners. These were left out to get better performance of the image recognition of rectangles.

C3 – The image must be drawn by tool, not by hand. This is a scoping decision: opening up for hand-drawn diagrams would require different approaches for image recognition.

C4 – The image must not be a screenshot of some UML modeling tool where a UML diagram is shown inside some application, i.e., the diagram also shows toolbars and menus of the drawing tool. The reason for this requirement is to avoid that menu bars and toolbars could be recognized as rectangles.

C5 – The image should display the entire diagram. We think this makes sense for most applications. We did not want to deal with this as a corner case.

C6 – The image should only contain diagram elements as defined by the UML notation (i.e. no additional annotations or icons - except UML-comment-boxes). In creating software documentation, it is not uncommon that developers include domain-specific icons or pictures of actors. We scoped out dealing with such aspects. In practice, we did not exclude many diagrams from the dataset for this reason.

C7 – The image should not be too simple. The criteria we applied for this are that a diagram should contain at least four classes and at least three relations.

C8 – No duplicates of diagrams are included. The original dataset was found to contain some duplicates. This does not contribute to our study.

The conformance to some criteria (C1, C2, C3) is ensured automatically through the use of the existing classifier and image-processing tools [29]. For the remaining criteria, we checked the conformance manually.

After applying these above criteria as filters, a total of 654 diagrams (out of around 3.000 in the starting set) remained to be useful.

An additional practical constraint is that our image processing should extract the important features of the diagram. However, our image processing could not detect any lines between classes for some diagrams. This can happen because lines are drawn in a dotted style with too much space between the dots, or if the lines are drawn in a curved style or with multiple angles along a single line. As the last step, we filtered out an additional 45 diagrams for which the image processing could not detect enough features from the image. This left us with a total of 609 diagrams in the final dataset for our study.[2]

We give some descriptive statistics for the final dataset: The distribution of the number of classes is shown in Figure 2. This distribution is similar to the distribution of diagram sizes of the overall Lindholmen dataset. Given that the Lindholmen dataset is collected from 'the real world' open source projects, we believe this set is fairly representative for software engineering practice. Some characteristics of this distribution are: the largest group of diagrams contains between 8 and 12 classes. The second-largest group of diagrams contains fewer than eight classes per diagram. Diagrams with more than 32 classes are pretty rare.

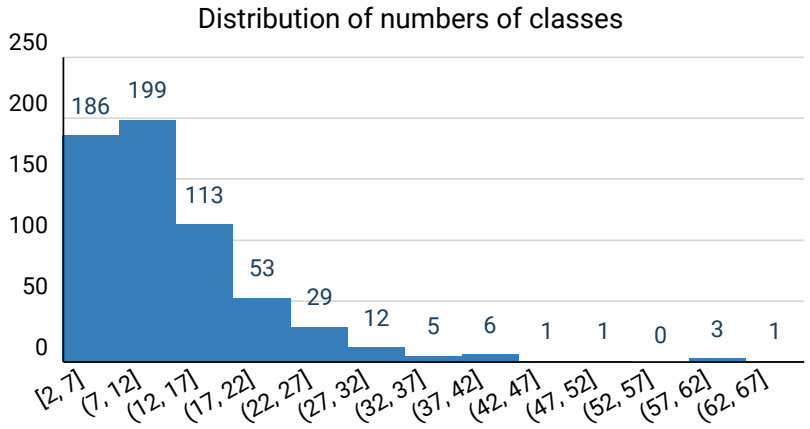


Figure 2: Distribution of the number of classes per diagram.

[2]The resulting dataset can be accessed at this link: `http://zenodo.org/record/5037744#.YO1lUagzaUk`.

4.2. Establishing a Ground Truth of Layout Quality

The original dataset contains diagram images without any information about their layout quality. To be able to train and evaluate a supervised machine learning model, the diagrams need to be labeled with a layout quality. The labeling was done manually by six experts in the areas of software modeling and HCI: four of the authors (one master's student, two PhD students, and a professor), and in addition, one more PhD student and an associate professor.

4.2.1. Labeling strategy

This section describes the steps of this labeling process: we chose a 5-point Likert scale [32] for labeling diagrams. This scale is symmetric around a neutral value and is intended to have equal distances between the options: 1: Very Bad, 2: Bad, 3: OK, 4: Good, and 5: Very Good. A Likert scale is an ordinal scale, but if the distances between the options are the same and the scale is symmetric around a middle point, it can also be treated as an interval scale. This is of interest regarding machine learning because many regression algorithms assume an interval scale for the dependent variable. We discuss our machine learning approaches further in Section 4.5.

To streamline the labeling, a software tool was developed to support this. The primary function of this tool was to automatically show (in sequence one after each other) the UML diagrams to be labeled and allow the easy entry of the assessment of the layout of the diagrams. This tool showed images on a computer screen and gave the labelers the following instructions: *"Please assess the diagrams with your perceived layout quality of it between (1) Very bad and (5) Very good. Consider how aesthetically appealing you find the diagram layout and how well you think it would help to understand the system it represents"* Thus, labeling data was entered digitally and linked to the images.

Next, we describe the process of establishing ground truth labels. The labeling was done in several rounds. From the start, guidelines were given on how to perform the labeling. These took the form of i) a list of typical flaws in layouts of class diagrams and ii) guidelines that indicate which amount of flaws roughly corresponds to which label on the Likert scale. In any case, these guidelines still left some freedom for labelers on how to weigh the magnitude/impact of layout flaws on the understandability of the diagram.

The initial round involved four of the experts. Each were given 30 diagrams selected randomly from the dataset to label. From this initial set,

diagrams in which labels differed significantly were discussed by the labelers. The discussions led to two slight refinements of the guidelines for labeling:

> Guideline 1: *"Assess only the layout of the diagrams without considering the semantic content of them."*.

This guideline was added because some labelers took some semantics of the classes into account when assessing the quality while others did not. For example, diagrams with a badly drawn hierarchy were assessed with low quality in some cases.

> Guideline 2: *"Think about how many different issues you find with the diagrams that could be improved. A tiny diagram without any issues could, for example, have a good layout albeit very simplistic."*

The diagrams with the highest variation in quality labels appeared to be very simple, i.e., diagrams with few classes and relations. Such diagrams were, in some cases, not considered of high quality even if there was no room for improvement. It was agreed that diagrams could have an outstanding layout quality even if the complexity of the diagram is low.

For these initial expert-labels, the Intra-Class-Coefficient (ICC [33]) was calculated to check the agreement amongst the raters. ICC gives a value between 0 and 1, where a higher value indicates a higher agreement. There was an inter-rater agreement with an ICC of 0.50, which is considered fair by Cicchetti [34], and (just) moderate by Koo and Li [35]. It was decided that this level of agreement was acceptable since there is no absolute truth of what is a right or wrong layout quality assessment. Perceived quality is somewhat subjective, and it must therefore be accepted to differ between different people.

The refined guidelines were then used by all six experts for the labeling of the remaining diagrams. Because the guidelines now led to more agreement (ICC of 0.64, interpreted as good [34] or moderate [35]), it was considered enough for each diagram to be labeled by precisely two raters, and take the average of those ratings as the final label for the ground truth of the diagram. Hence, the ground truth is no longer an integer but can be a fraction—specifically, halves.

Figure 4 shows the distribution of the final labels. It has a skewness of -0.4948, i.e., slightly skewed towards higher quality values. This is reasonable because diagrams are desired to have a high layout quality. The kurtosis of the quality labels are -0.3360. It shows that our dataset is neither flat nor peaked [36]. Combined with the skewness category of *approximately symmetric* [37], the distribution of our dataset is fairly close to normal [36]; this normal-like distribution is suitable for machine learning. The fact that some diagrams are labeled with extreme values of 1 and 5 allows machine learning algorithms to learn from both good and bad diagrams, which is beneficial.

4.3. Feature Extraction by Image Processing

This section describes the extraction of features from the images that can be used by machine learning. Our feature extraction includes three major parts: Processing the images to find key elements in the diagrams, calculating basic features that represent relevant layout aesthetics, and finally calculating some complex features based on more basic image features.

The approach we use in image processing is based on the work of Ho-Quang et al. [29]. We extend this tool, written in C/C++, with our feature calculation from the elements detected by the image processing. This section briefly describes the image processing algorithms used. We also validate how well these methods work with the class diagram dataset used in this study.

4.3.1. Image processing algorithms

Four common image processing algorithms are used for finding elements[3] in the diagram images. Those are *Canny edge detection*, *Hough Transformation*, *Suzuki 85* and *Ramer-Douglas-Peucker*. The basics of these algorithms are described in the following paragraphs.

Canny edge detection. Canny edge detection [38] is an algorithm for detecting edges in images. First, the algorithm removes noise in the image by blurring it using a Gaussian filter [39]. The blurred image is then used to find the edge gradient and direction for each pixel in the image. Then, the algorithm removes any pixels not part of an edge. This happens if the pixel is not a local maximum in the direction of the gradient. Finally, the edges are thresholded to only keep those with a high-intensity gradient or connected to such an edge. This removes all shorter lines that are considered noise.

[3] In our paper, diagram elements are lines and rectangles

Hough Transformation (HT). Hough Transformation (HT) [40] is a procedure for detecting straight lines in images. It is applied to an image where edges are already detected. The edges are represented as a direction and distance from the center of the image. Edges with the same direction and distance are considered to form a line.

Suzuki 85 (S85). Suzuki 85 (S85) [41] is a border-following algorithm that is used to find contours in an image. First, the pixels in the image are scanned, and when a pixel that satisfies a condition for being a starting point of a border is found, that border is followed and all pixels on it are marked.

Ramer-Douglas-Peucker (RDP). The Ramer–Douglas–Peucker (RDP) algorithm [42] is used to approximate a polygon with few points from a curve with arbitrarily many points. The algorithm calls itself recursively and removes the points that are least important for representing the curve. RDP is a suitable algorithm for finding shapes. For example, a curve approximated with four points could potentially be a rectangle.

4.3.2. Defining Image Features for Layout Aesthetics

Based on the literature (described in Section 3.1.2), we selected a set of candidate diagram features that appeared most important for layout quality. Mostly, these features depend on recognizing rectangles and lines in the diagrams. Below, we describe a set of aspects of quality of layout (labeled A1 to A10) and propose a set of image features (labeled F1 to F16) as the basis of which the aspects can be computed/approximated. Table 5 summarizes the image features and their relation to aesthetics.

A1 – Line crossings. The number of crossing lines should be minimized, and the angles of the line crossings should be maximized. The number of lines crossing each other and their angle can be computed from image features F1 *Line crossings* and F2 *Crossing angles*.

A2 – Line bends. The number of line bends should be minimized. The lines extracted in the image processing are divided into straight segments. For example, if a line has one bend, it is represented as two straight line segments that connect at the point of the line bend. Thanks to this representation, a feature F3 *Line bends* can easily be computed.

A3 – Orthogonality. One aspect of aesthetics is the orthogonal orientation of classes. For the image processing, this translates into rectangles that should be found on an orthogonal grid and lines drawn horizontally or vertically. The angle of the lines and their deviation from orthogonality is extracted by a feature F4 *Line angles*. This image feature can also be used to classify lines as orthogonal or not orthogonal and thus the feature *Line orthogonality*. The positions of the found rectangles can be checked to see how well they conform to an orthogonal grid, which gives rise to the feature *Rectangle orthogonality*.

A4 – Line lengths. Lines should not be too long or too short, and the line lengths should be as uniform as possible. The length of the lines can easily be found and can be used to compute the features F7 *Average Line length*, F8 *Line length variation*, F9 *Longest line* and F10 *Shortest line*.

A5 – Diagram drawing size. The size of a diagram should be minimized while still fitting all of the diagram elements and without conflicting other aesthetics by squeezing elements too close together. The feature F11 *Rectangle coverage* is proposed to address this aesthetic. F11 indicates how much of the total diagram area is covered by rectangles. Furthermore, the aspect ratio of a diagram could also impact its layout quality. Therefore, we also define the feature F12 *Aspect ratio* using the height and width of the diagram.

A6 – Symmetry. Symmetry in diagrams should be maximized to increase understandability. However, according to Purchase et al. [17], a computational algorithm to evaluate the symmetry of a diagram would be complex and is only a very rough model of how humans perceive symmetry. It was therefore chosen not to compute any image features regarding symmetry.

A7 – Line angular distances. If multiple lines are going from a rectangle, the minimum angle between two of those lines should be maximized. Unfortunately, the image processing used only finds standalone rectangles and lines and no connections between the elements. Since the lines are not related to rectangles, no feature was computed for this aesthetic.

A8 – Class placement. Rectangles should be distributed uniformly within the diagram, and they should not be too close or too far apart. The positions of the rectangles can be computed using the features *Rectangle distribution* and *Rectangle proximity*. Other aspects of this aesthetic include that connected rectangles should be close to each other, rectangles should not be too close to

lines that they are not connected to, and rectangles with a high degree should be placed near the center of the diagram. All of these require connections between diagram elements, Establishing such connections between different diagrams is a high level of analysis not supported by the image processing that we use. Therefore, we have not defined any image features to represent these latter aesthetics.

A9 – Overlapping. Rectangles and lines should not overlap each other. The image processing can identify rectangles that represent classes, but it gets confused by overlapping elements and can not make sense of what is represented in those cases. Therefore we do not define any image feature for this aesthetic.

A10 – Node sizes. Rectangles should be as small as possible, and the difference among their sizes should be minimized. For these aesthetics. we define the image sizes the features F15 *Rectangle size* and F16 *Rectangle size variation*.

4.4. Definitions of Image Features

This section provides definitions of the features based on image processing. These features will be the input to machine learning. These features assume that the image processing provides a set of rectangles (representing classes) and lines (representing relations between classes). Other than features related to diagram aesthetics (F1–F16), we include two features that describe the diagram itself (F17 and F18).

F1 – Line crossings. The number of line crossings in a diagram can be found by counting the intersections between the found lines. However, simply using the number of line crossings as a feature would favor diagrams with fewer relationships, as they would more likely have fewer line crossings. A relative value was used to counter this: the ratio of the number of line crossings to the maximum possible number of line crossings. The maximum number of line crossings occurs when every line in the diagram crosses every other line. In this case, there would be $\frac{\#lines \times (\#lines-1)}{2}$ crossings. Equation 1 shows the calculation of F1.

$$\text{F1} \;=\; \frac{\#crossings \times 2}{\#lines \times (\#lines - 1)} \tag{1}$$

F2 – Crossing angles. For each line crossing, the crossing angle is calculated - this is a value in the range from 0° to 90°. The average crossing angle is then calculated as the feature F2. Equation 2 shows the definition.

$$\text{F2} \;=\; \frac{\sum_{i=1}^{\#\text{crossings}} \text{angle}(\text{crossing}_i)}{\#\text{crossings}} \tag{2}$$

F3 – Line Bends. Lines are represented as straight segments, and the number of bends on a line is one less than its number of segments. The average number of bends per line is calculated as feature F3. Equation 3 shows the definition.

$$\text{F3} \;=\; \frac{\sum_{i=1}^{\#\text{lines}} \text{bends}(\text{line}_i)}{\#\text{lines}} \tag{3}$$

F4 – Line angles. The deviation angle from orthogonality, i.e., how far from being horizontal or vertical, is calculated for each line. If a line is more than 45° degrees from being horizontal, it is less than 45° from being vertical and vice versa, which makes the deviation from orthogonality a value between 0 and 45. The average line angle is calculated and used as the feature F4. Equation 4 shows the definition.

$$\text{F4} \;=\; \frac{\sum_{i=1}^{\#\text{lines}} \text{angle}(\text{line}_i)}{\#\text{lines}} \tag{4}$$

F5 – Line orthogonality. A line is orthogonal if it is exactly horizontal or vertical. A margin of error of 1° is used to allow for small deviations due to the quality of the image and the image processing. The number of orthogonal lines is divided by the total number of lines to get a ratio between 0 and 1 as a feature F5. Equation 5 shows the definition.

$$\text{F5} \;=\; \frac{\sum_{i=1}^{\#\text{lines}} \begin{cases} 1 & \textit{if } \text{angle}(\text{line}_i) < 1°, \\ 0 & \textit{otherwise} \end{cases}}{\#\text{lines}} \tag{5}$$

F6 – Rectangle orthogonality. The orthogonality of rectangles is calculated by counting the number of distinct rectangle positions on the $x-$ and $y-$axis, respectively. For this purpose, the center of a rectangle is considered its position. A set of occupied positions is kept for both axes, and all rectangles

are looped through and examined to populate this set. If the currently examined rectangle has a position that is not in the set, its position is added to the set. If the rectangle's position is (within a small error margin) in the set, then no position is added to the set. The ratio between the number of distinct rectangle positions and the number of rectangles is calculated for the horizontal and vertical directions. The values for these directions are then added together. Equation 6 shows the definition.

$$\begin{aligned} \text{F6} = &\left(1 - \begin{cases} 0 & \text{if \#xPositions} = 1 \\ \frac{\text{\#xPositions}}{\#rectangles} & \text{otherwise} \end{cases}\right) \\ &+ \left(1 - \begin{cases} 0 & \text{if \#yPositions} = 1 \\ \frac{\text{\#yPositions}}{\#rectangles} & \text{otherwise} \end{cases}\right) \end{aligned} \tag{6}$$

F7 – Average Line length. This feature is the average length of all found lines. Equation 7 shows the definition.

$$\text{F7} = \frac{\sum_{i=1}^{\#\text{lines}} \text{length}(\text{line}_i)}{\#\text{lines}} \tag{7}$$

F8 – Line length variation. This feature measures how much the line lengths vary by calculating the standard deviation of the lengths. Equation 8 shows the definition.

$$\text{F8} = \text{StDev}(\{\text{length}(\text{line}_i) | 1 \leq i \leq \#\text{lines}\}) \tag{8}$$

F9 – Longest line. This feature is the length of the longest found line. Equation 9 shows the definition.

$$\text{F9} = \max_{i=1}^{\#\text{lines}} \text{length}(\text{line}_i) \tag{9}$$

F10 – Shortest line. This feature is the length of the shortest found line. Equation 10 shows the definition.

$$\text{F10} = \min_{i=1}^{\#\text{lines}} \text{length}(\text{line}_i) \tag{10}$$

F11 – Rectangle coverage. This feature represents how much of the total diagram area is covered by rectangles (which represent classes). This is defined as the ratio between i) the sum of all rectangle areas and ii) the total area of the diagram. This gives a ratio between 0 and 1. Equation 11 shows the definition.

$$\text{F11} \;=\; \frac{\sum_{i=1}^{\#\text{rectangles}} \text{size}(\text{rectangle}_i)}{\text{diagram}_{\text{width}} \times \text{diagram}_{\text{height}}} \tag{11}$$

F12 – Aspect ratio. This feature is calculated by dividing the width of the diagram by the height of the diagram. Equation 12 shows the definition.

$$\text{F12} \;=\; \frac{\text{diagram}_{\text{width}}}{\text{diagram}_{\text{height}}} \tag{12}$$

F13 – Rectangle distribution. For this feature, the image is divided into four equal quadrants (obtained by drawing horizontal and vertical lines through the image's center point). For each rectangle, the area of each of the four sections that it covers is calculated. These rectangle areas are summed up for the four sections, respectively, which gives each a value for how much of it is covered by rectangles. The variance of these values indicates whether all areas of the image are used. Equation 13 shows the definition of this feature.

$$\text{F13} \;=\; \text{var}(\{\text{rectangleCoverage}(\text{quadrant}_i) | 1 \leq i \leq 4\}) \tag{13}$$

F14 – Rectangle proximity. The distance from its center to the center of the other rectangles is calculated for each rectangle. The average of this distance over all rectangles is used as the feature F14. Equation 14 shows the definition.

$$\text{F14} \;=\; \frac{\sum_{i=1}^{\#\text{rectangles}} \text{shortestDistanceToOtherRectangle}(\text{rectangle}_i)}{\#\text{rectangles}} \tag{14}$$

F15 – Rectangle size. This feature is the average area of all found rectangles. Equation 15 shows the definition.

$$\text{F15} \;=\; \frac{\sum_{i=1}^{\#\text{rectangles}} \text{size}(\text{rectangle}_i)}{\#\text{rectangles}} \tag{15}$$

F16 – Rectangle size variation. This feature measures how much the rectangle sizes vary by calculating the standard deviation of the sizes. Equation 16 shows the definition.

$$\text{F16} \;=\; \text{StDev}(\{\text{size}(\text{rectangle}_i) | 1 \leq i \leq \#\text{rectangles}\}) \tag{16}$$

F17 – Number of rectangles. This feature reflects the number of rectangles detected by image processing. This corresponds directly to the number of classes.

F18 – Number of lines. This feature reflects the number of lines detected by image processing. This corresponds with the number of relationships between classes.

4.5. Considerations on Selecting Machine Learning Approaches

We mentioned that we treat our layout-quality label as an interval scale. The use of interval scale instead of categorical is important in our pursuit for a prediction model because there is a meaningful notion of distance, e.g., for a data point with an actual label of 4, a prediction of 3.5 is better ("closer to the actual label") than a prediction of 2. This distance information is lost if we treat the label as categorical type.

The use of interval scale eliminates certain machine learning approaches. The problem can now be considered a regression problem. The fact that there are multiple features to take into account also limits the types of regression approaches. Furthermore, the limited size of the training set prevents an optimal use of neural-network techniques.

With the aforementioned considerations, we selected 12 algorithms to experiment with that covers a diversity of machine learning techniques. This selection includes traditional regression approaches as well as support vector, rule-based, and tree-based approaches:[4]

1. Gaussian Processes [43]
2. Linear Regression

[4]Consult the Weka documentation at `https://weka.sourceforge.io/doc.stable/`, specifically the subpackages of `weka.classifiers`, for more information on the algorithms.

3. Multi-layer Perceptron
4. Simple Linear Regression
5. Support Vector Regression (SMOreg) [44, 45]
6. Decision Table [46]
7. M5 Rules [47, 48, 49]
8. Decision Stump
9. M5P [48, 49]
10. Random Forest [50]
11. Random Tree
12. REP Tree

5. Machine Learning Results and Evaluation

This section describes the results of the machine learning algorithms and their evaluation. The software tool *Weka*[5] was used for the machine learning parts of this research.

5.1. Performance of the Machine Learning

In our study, we trained different machine learning algorithms. In all cases, we used 10-fold cross-validation: the entire dataset is split randomly into ten folds ('parts'). Then, ten runs are performed: one fold is held out as a validation set in each of these runs, and the other nine folds are used as a training set. Finally, when all ten folds have served as a validation set, the average of the ten validation results is taken as the final result.

In an attempt to increase the performance of our regression models, we also trained the models using preprocessed dataset. The preprocessing scales and translates each feature individually such that it is in the range between zero and one.

The transformation for each feature achieves a normalization and is given by the following formulation.

$$x'_i = (x_i - min(\mathbf{X}))/(max(\mathbf{X}) - min(\mathbf{X})) \tag{17}$$

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

where x_i denotes the value of the feature for data point i and $\mathbf{X}$ contains all values of x_i.

Table 1 shows the evaluation results for the different machine learning algorithms. The table includes three evaluation metrics: Pearson's correlation coefficient (column *PCC*), Mean Absolute Error (*MAE*), and Relative Absolute Error (*RAE*). Pearson's correlation coefficient is the ratio between the covariance of two variables and the product of their standard deviations; thus the result always has a value between -1 and 1. The further away the coefficient is from zero, the stronger correlation there is between the two variables. MAE is the "average difference" between actual and predicted quality labels, i.e., an arithmetic average of the absolute errors from each data point. RAE is expressed as a ratio, comparing a mean error (residual) to errors produced by a trivial or naive model. A model which produces results that are better than a trivial model will result in a ratio of less than one. The table is sorted by MAE and grouped by dataset preprocessing kind.

The best-performing machine-learning approach is the *RandomForest* algorithm with dataset preprocessing. It achieves a correlation of 0.709, which means our regression model can account for 70.9% of the quality value of a diagram layout. An MAE of 0.533 means that on average, the predicted quality label is ± 0.533 points away from the actual label. It has an RAE of 68.7%, which means that our prediction is better than a trivial guess, e.g., stamping a quality label of "3" for all diagrams.

We performed gird search on the random-forest model to tweak the parameters with the hope of producing better performance. From this search, we found that the default parameters supplied by Weka already gave us the best MAE value.

As an alternative view on the performance of the evaluator, Table 2 visualizes the results of our 10-fold validation using the best performing algorithm in a confusion matrix. A confusion matrix is usually used to visualize predictions in classification problems and shows the distributions of predictions for each class. In order to use this visualization for our regression-based approach, we defined categories of the size of 0.5 on a 5-point scale (with the lowest value being 1.0). The evaluations made for layout quality by our evaluator are rounded to the nearest .5, which gives nine classes between 1.0 and 5.0. The rows in the table represent the ground-truth values of the images, and the column shows the predicted value for diagrams. We can, for example, see that 52 diagrams labeled with a ground truth-layout quality of 3.5 were classified to have a layout quality of 3.0. In this table, darker cell

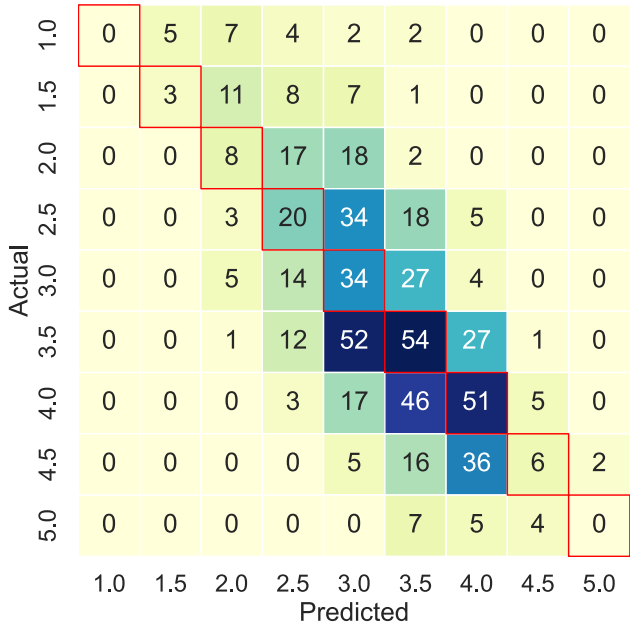
colors indicate a larger number of diagrams in that cell of the table. The table shows that the predicted values are often within an interval of -0.5 to $+0.5$ of the ground truth value. This shows as a linear diagonal pattern from the top-left to the bottom-right.

Figure 3 illustrates the prediction errors in a higher level. For 459 out of 609 diagrams (75.4%), the predicted quality labels fall within ± 0.5 points from the actual labels. Negative values in the y-axis means that the predicted label is lower than the actual label, and vice versa. For example, a diagram

Table 1: Machine learning approach evaluation.

Dataset preprocessing	**Algorithm**	**PCC**	**MAE**	**RAE**
none	RandomForest	0.699	0.539	69.4%
	GaussianProcesses	0.659	0.562	72.4%
	LinearRegression	0.655	0.566	73.0%
	SMOreg	0.646	0.569	73.4%
	REPTree	0.574	0.629	81.0%
	DecisionTable	0.535	0.630	81.3%
	DecisionStump	0.494	0.660	85.1%
	RandomTree	0.477	0.743	95.8%
	SimpleLinearRegression	0.470	0.665	85.7%
	MultilayerPerceptron	0.367	0.832	107.3%
	M5P	0.152	0.640	82.5%
	M5Rules	-0.091	3.815	491.8%
feature scaling	RandomForest	0.709	0.533	68.7%
	GaussianProcesses	0.659	0.562	72.4%
	LinearRegression	0.655	0.566	73.0%
	SMOreg	0.646	0.569	73.3%
	REPTree	0.572	0.630	81.2%
	DecisionTable	0.535	0.630	81.3%
	RandomTree	0.531	0.700	90.2%
	DecisionStump	0.494	0.660	85.1%
	SimpleLinearRegression	0.470	0.665	85.7%
	MultilayerPerceptron	0.367	0.832	107.3%
	M5P	-0.004	0.771	99.3%
	M5Rules	-0.091	3.823	492.8%

Table 2: Confusion matrix for the random forest regressor with normalized features.



labeled 4 but predicted as 3.5 belongs to $y = -0.5$. There are more diagrams with negative y-values (226, compared to 207 for positive y-values), but the distances between actual and predicted labels are larger in the diagrams with positive y-values.

We can also compare the actual and evaluated values more abstractly by looking at the overall distribution of values of images. Figure 4 shows the distributions of the actual quality labels (obtained by manual labeling) and the predicted quality labels (obtained as output from the machine learner). The x-axis denotes the value of the layout quality assigned to a diagram. For any x-value, The y-axis denotes the number of images classified into that 'bucket' as a value for layout quality. From this, one could see that the machine-learning is a little more conservative in assigning extreme values to layouts. Instead, the classifier tends to labels diagrams a bit more towards the average of the distribution. This is a common phenomenon in machine learning, especially given that the average values occur more frequently and direct the classifier in this direction.

Our analysis uses 'buckets' of 'size' 0.5 (ranging from 1.0 to 5.0 in steps of 0.5). The number of falsely classified diagrams (off-diagonals in the confusion matrix) decreases/increases when using larger/smaller buckets. While

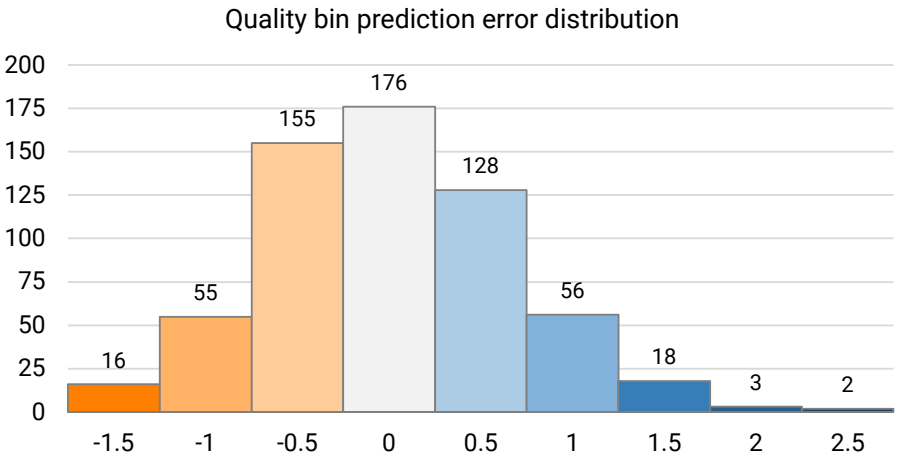


Figure 3: Distribution of prediction errors.

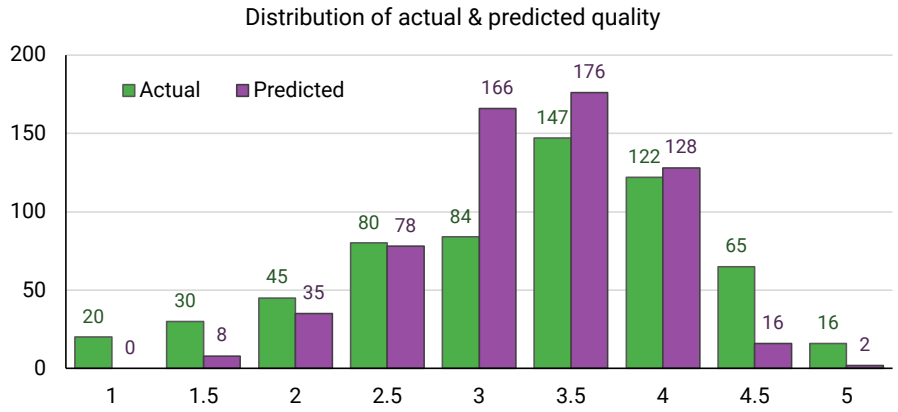


Figure 4: Distribution of actual (green) and predicted (purple) quality labels.

the choice of bucket size is somewhat arbitrary, we find that a 10-point scale provides significant differentiation between the quality of good and bad layouts. For many practical purposes, possibly a scale with five buckets (1-5) could already offer enough differentiation on layout quality. Furthermore, when used for grading purposes, a 10-point scale nicely aligns with exam grading scales used internationally. At the same time, the scale has not been 'normalized'/calibrated to have 5.0 as a pass/fail-threshold.

In order to understand the impact of the number of classes in a diagram on

the performance of our models, we grouped our dataset into three categories: *small*, *medium*, and *large* diagrams, each containing 1 to 7, 8 to 12, and more than 12 classes, respectively. The boundary numbers 7 and 12 were selected according to the percentiles of our labels. In effect, the three categories contain roughly the same number of diagrams. Table 3 shows the mean absolute error values for the categories. We can see that our regression model tends to perform better with smaller diagrams.

Table 3: Regression performance by diagram size.

Diagram size Category	**Number of diagrams**	**MAE**
Small	186	0.466
Medium	199	0.534
Large	224	0.587

5.2. Analysis of Importance of Features

In this section, we explore whether some layout features have more importance than others in determining the quality of the layout. Table 4 shows a ranking of the features based on the absolute value of their correlation with the ground truth layout quality. Features with a correlation score closer to 1 are the ones that have a more significant contribution to the layout score of the diagram. This table includes the value of the correlation and the absolute correlation. The sign indicates whether this feature positively or negatively contributes to the layout quality.

The table shows that features related to line lengths (A4), F9, F7, and F8, are important. Other essential features are *rectangle orthogonality* (F6). These features are all related to the 'regularity' and uniformity of the spacing of classes across a diagram. The third category of important features is related to the shape, positioning, and orientation of lines between classes (features F2 Crossing Lines and F3 Line Bends). It turns out that F10 Shortest Line and F5 Line Orthogonality are not very significant in determining layout quality.

The distributions of the six most important features are shown in Figure 5. At a glance, we can see that higher-quality diagrams have shorter lines (features *longest line* length and *average line length*) and less varied line lengths (feature *line length variation*). They also have a smaller number of diagram elements (features *number of lines* and *number of rectangles*),

Table 4: Feature importance ranking based on absolute correlation coefficient.

Rank		Image feature	Aesthetic	PCC	Abs. PCC
1	F9	Longest line	A4	-0.494	0.494
2	F18	Number of lines	–	-0.483	0.483
3	F17	Number of rectangles	–	-0.425	0.425
4	F7	Average line length	A4	-0.363	0.363
5	F6	Rectangle orthogonality	A3	0.357	0.357
6	F8	Line length variation	A4	-0.307	0.307
7	F2	Crossing angles	A1	0.255	0.255
8	F3	Line bends	A2	-0.224	0.224
9	F14	Rectangle proximity	A8	-0.200	0.200
10	F15	Rectangle size	A10	-0.194	0.194
11	F4	Line angles	A3	-0.171	0.171
12	F1	Line crossings	A1	-0.158	0.158
13	F16	Rectangle size variation	A10	-0.121	0.121
14	F13	Rectangle distribution	A8	0.092	0.092
15	F11	Rectangle coverage	A5	0.078	0.078
16	F12	Aspect ratio	A5	0.053	0.053
17	F5	Line orthogonality	A3	0.036	0.036
18	F10	Shortest line	A4	0.002	0.002

and more orthogonal rectangles positioning (feature *rectangle orthogonality*). Furthermore, in most features, we can also see that the variance for higher quality bins tend to be smaller than those in lower quality bins. This makes them appropriate to serve as heuristics in guiding the creation of diagram layout, e.g., in an automated diagram-layouting algorithm or as a user-feedback in diagramming software.

6. Discussion

This section discusses the results of the experiments' interpretation and comparisons concerning the proposed estimator performance. We reveal the essential aesthetics for indicating layout quality.

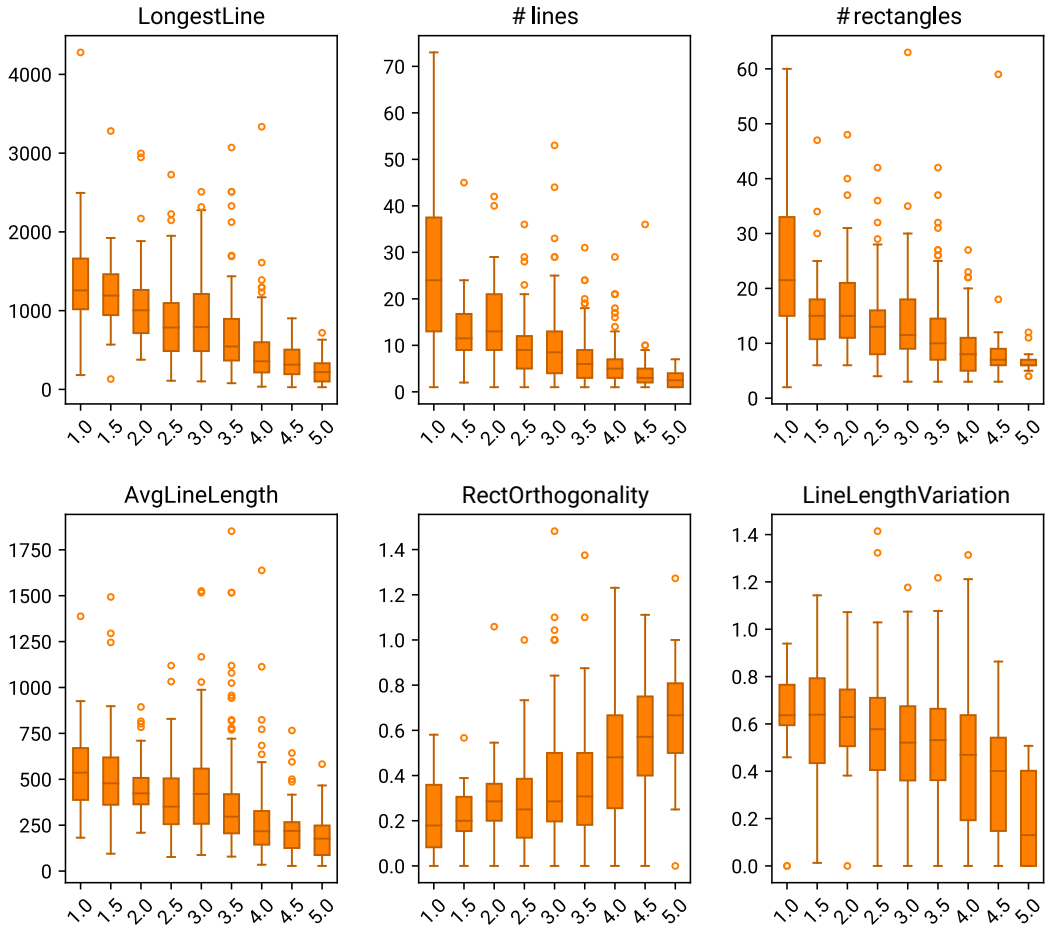


Figure 5: Distribution of key diagram features in each quality bin.

6.1. The Estimator performance

The best performing machine learning approach (Section 5) has an MAE of 0.56. This means that the predicted layout quality of a previously unseen diagram differs by 0.56 on average from the labeled layout quality of the diagram. This is proportional to a Relative Absolute Error of 68.7% for this dataset, meaning that the absolute error of a prediction of a diagram is on average 68.7% of the absolute error between the mean labeled layout quality of the dataset and the labeled layout quality of the diagram. In other words, the estimator's performance is significantly better than simply predicting the sample mean for every diagram. The correlation coefficient was 0.709, indicating that 70.9% of the variance in layout quality can be

explained by the trained model. This is a strong correlation according to Evans' guidelines [51].

6.2. Image Processing

The image processing has some flaws that can lead to incorrect calculation of features, which can mislead the machine learning algorithms. We illustrate these impacts by discussing some examples from our dataset.

Below, we show the three diagrams with the highest prediction error. The left part of the figures shows the original diagram image, and the right part shows the representing elements extracted by the image processing module. Rectangles are drawn in white, and lines are drawn in green.

Figure 6 shows the diagram that has the highest prediction error. It was labeled with a layout quality of 1.0, and the machine learning model produced a quality of 3.5, which gives a prediction error of 2.5. From the right-hand side, we can see that many of the lines from the class diagram on the left-hand side were not found by the image processing. Many undetected lines are long, have many bends, and partially overlap with the rectangles that represent classes. Those as mentioned above are probably the reasons for the ground truth layout quality being labeled low. These are features that are negatively correlated with layout quality. The machine learning algorithm does not get the correct information for these features from the image processing. Hence, it is misled to predict a higher layout quality than it should.

Figure 7 shows the diagram that had the second-highest prediction error. It was labeled with a ground truth layout quality of 1.0, and the machine learning model predicted a quality of 3.4, which gives a prediction error of 2.4. In this case, the difficulty for the image processing to find lines is even more clear. Only one line is found, and that one line is not correctly detected. Many lines are crossing each other, which would otherwise negatively affect layout quality. Another possible reason for the low layout quality label is that many lines overlap rectangles. This is a common aspect that is recommended against in literature However, due to the limitations of our image processing (described in Section 4.3.1), no feature regarding this was created. This means that the machine learning algorithms do not get this information and thus cannot take it into account when making their prediction.

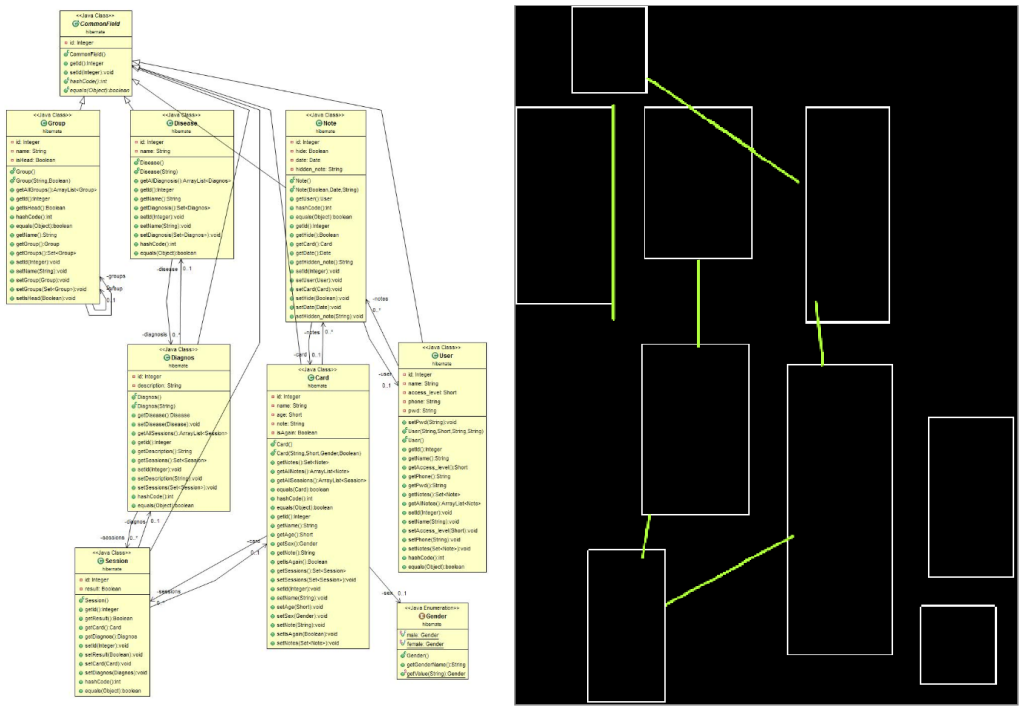


Figure 6: The diagram with the highest prediction error.

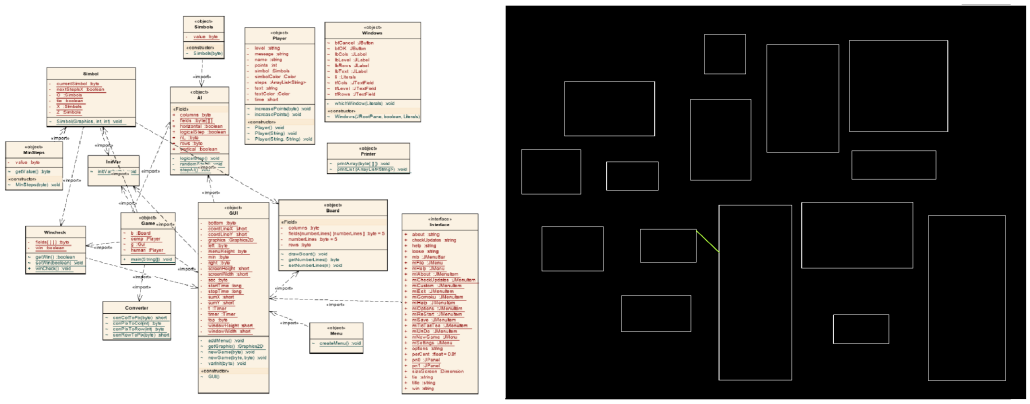


Figure 7: The diagram with the second highest prediction error.

Figure 8 shows the diagram that has the third-highest prediction error. It was labeled with a ground truth layout quality of 1.5, and the machine learning model predicted a quality of 3.8, which corresponds with a predic-

tion error of 2.3. Here the image processing finds no lines at all, and some of the rectangles are also undetected. In the diagram, many long lines, line crossings, and line bends are not detected. In this case, the original diagram uses dotted lines and grey-toning of lines which probably affect the inability to detect the lines in this diagram. The machine-learning algorithm predicts too high a layout quality for the same reasons as the two previously discussed diagrams.

Overall, there are opportunities for improving the image recognition used. However, our image processing deals fairly well with a large portion of the images. Out of the set of 645 images that we started with (i.e., qualified our criteria for suitable UML class diagrams), it turned out that our image processing could not extract enough features from 45 diagrams. This is partially due to the diagrams using unusual features (such as e.g. curved and dotted-lines). This reduction should not have a great impact on the performance of the regressor (becasue we have good number of 609 diagrams left). Although the image processing could possibly impoved to deal with more (exotic) diagrams, there are little returns for our research for this.

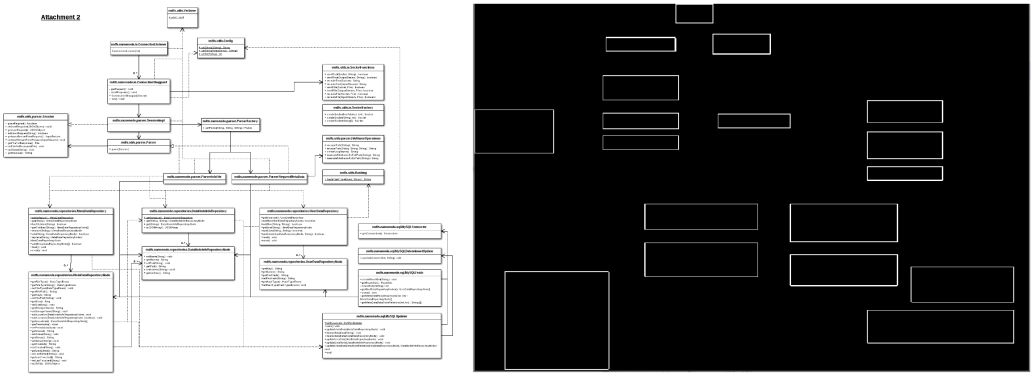


Figure 8: The diagram with the third highest prediction error.

A common denominator for the three diagrams with the highest prediction errors is that they all are labeled with a low layout quality but predicted to have a medium quality. As discussed, the most likely reason for this is the lack of detected lines. Many of the features regarding lines, including *Longest line*, *Line length*, *Line length variation*, *Line bends* and *Line crossings* are negatively correlated with layout quality, which is seen in Table 4. This makes the machine learning algorithms falsely think that it is beneficial

to have no lines at all or a few short lines. In addition, Table 4, shows that some of those features are among the most important ones when it comes to predicting layout quality. This explains that the omission of detecting these features affects the prediction error greatly.

6.3. Essential aesthetics

To find the most essential aesthetics for indicating layout quality, the feature selection results are used. As seen in Table 4 in Section 5.2, many of the features that were found among the most important ones in this study, including *longest line*, *line length* and *line length variation* are related to aesthetic A4, *line lengths*. This is a strong indication that line lengths are important when it comes to layout quality.

Table 5 shows a summary of the presence in literature of evaluation of layout aesthetics that are related to the features. Studies [23], [24], [17] and [18] investigate whether certain aesthetics have a significant effect on the quality of the layout. Significant effects are denoted with a y in those columns in the table, effects that are approaching significance are denoted with an a, and nonsignificant effects are denoted with an n. Study [24] measures both correctness and time for the tasks in their experiment, which is why this column has two entries. The first one represents significance for the correctness and the second one represents significance for time. Study [13] measures people's preference of diagrams with a high degree of certain aesthetics over diagrams with a low degree of the aesthetics. The preference is represented by a percentage. Study [19] ranks 14 groups of aesthetics by their importance. *Line bends* occurs in the groups on both sixth and fourteenth place, which is why this row has two entries.

The rightmost column R in Table 5 shows the rank of a feature's importance according to the correlation coefficients computed from our own dataset. Below, we discuss our findings regarding the importance of aesthetics and image features in comparison to the findings in literature.

6.3.1. High importance: line lengths (A4), descriptive features, and orthogonality (A3)

Line lengths (A4). Purchase et al. [13] did not find any significance of having short, but not too short, lines with regards to the understandability of graphs. Eichelberger [19] ranked general line constraints (including line lengths) in the sixth place out of 14 aesthetics. Ware et al. [18] did not find significant

Table 5: Aesthetics and image features, evidence of their importance in literature, and our own computed ranks.

Aesthetic		**Image feature**		**[23]** sig	**[24]** sig	**[13]** pref	**[17]** sig	**[18]** sig	**[19]** rank	**R**† rank
A1	Line crossings	F1	Line crossings	y	y/y	93%	-	y‡	5	12
		F2	Crossing angles	-	-	-	-	n	-	7
A2	Line bends	F3	Line bends	y	y/a	91%	y	y	6, 14	8
A3	Orthogonality	F4	Line angles	-	n/n	61%	n	-	-	11
		F5	Line orthogonality	-	n/n	61%	n	-	-	17
		F6	Rectangle orthogonality	-	n/n	61%	n	-	2	5
A4	Line lengths	F7	Average line length	-	-	-	n	n	6	4
		F8	Line length variation	-	-	-	n	-	-	6
		F9	Longest line	-	-	-	n	-	6	1
		F10	Shortest line	-	-	-	n	-	6	18
A5	Diagram drawing size	F11	Rectangle coverage	-	-	-	n	-	-	15
		F12	Aspect ratio	-	-	73%	-	-	14	16
A6	Symmetry		*(none computed)*							
A7	Linear angular distances		*(none computed)*							
A8	Class placement	F13	Rectangle distribution	-	-	-	n	-	-	14
		F14	Rectangle proximity	-	-	-	-	-	3	9
A9	Overlapping		*(none computed)*							
A10	Node sizes	F15	Rectangle size	-	-	-	-	-	3	10
		F16	Rectangle size variation	-	-	-	-	-	-	13
Descriptive features		F17	Number of rectangles							3
		F18	Number of lines							2

† Feature importance rank according to our study.

‡ Line crossings on shortest path, but not total number of line crossings in diagram.

effects for average line length and total line length on the shortest path between two nodes in a graph with regards to the time needed to perceive the shortest path. However, we found that features related to line lengths have a significant impact to the quality of diagram layout. Specifically, features longest line (F9), average line length (F7), and line length variation (F8) rank 1st, 4th, and 6th, respectively, in our dataset. In practice, this means that lines in a diagram should not be long, and their lengths should not vary too much. One feature in this aesthetic group, shortest line (F10, ranked 18th) does not reflect a diagram's layout quality.

Descriptive features. Beyond our expectation, descriptive features that we used for machine learning have considerable correlations with the layout quality. Number of lines (F18) and rectangles (F17) rank 2nd and 3rd, respectively, in our list. Figure 9 plots the diagrams based on their quality label compared to the number of rectangles and lines. Note that in this figure,

the quality labels are treated as discrete categories or bins, i.e., the y-axis positional variances within a bin do not convey differences in quality, but rather to illustrate the number of diagrams that belong to the bin. Both plots in the figure are noticeably sparse in the bottom-right triangle, showing that it is hard to achieve good layout quality with many rectangles and lines. However, the opposite is not true—having less rectangles and/or lines does not necessarily mean the layout is good. This is indicated by the denser upper-left triangles of the plots.

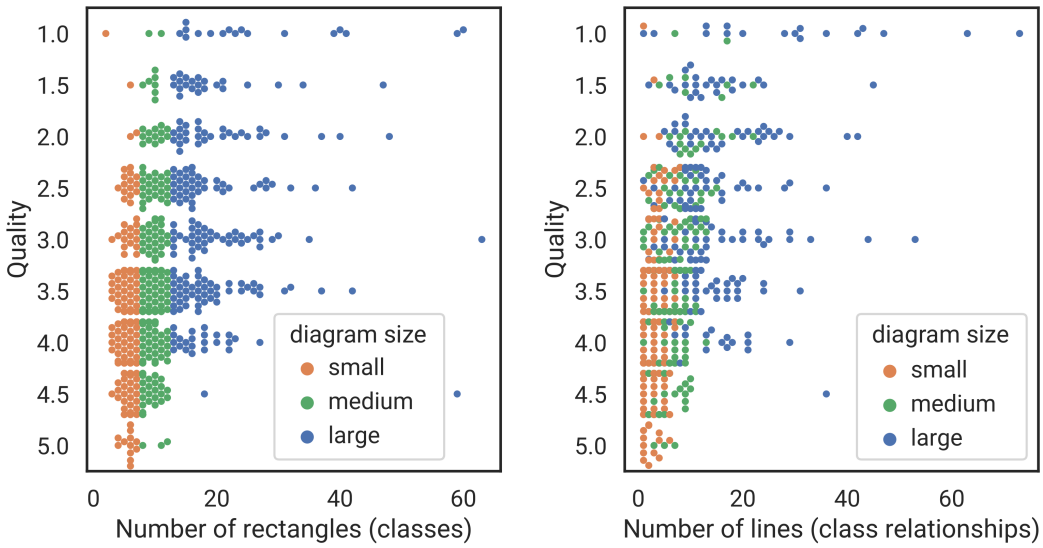


Figure 9: The number of classes and relationships between classes are important for layout quality. The legend boxes are placed carefully so that no plots are covered by the boxes.

Orthogonaltiy (A3). In this aesthetic, we found one important feature: rectangle orthogonality (F6). The positive correlation coefficient for this feature (ranked 5$^{\text{th}}$ in our list) means that rectangle placements should be centered against one another in one of the axes. This is consistent with Eichelberger's study [19] which puts centered neighboring rectangles in the 2$^{\text{nd}}$ place in their list. Purchase et al. [13] similarly found a 61% preference for more orthogonal diagrams. On the other hand, Purchase [24] and Purchase et al. [17] did not find a significant effect of orthogonality on the understandability of graphs. In our dataset, line angles (F4) only shows a slight correlation and line orthogonality (F5) do not show significant correlation with layout quality. The features are ranked 11$^{\text{th}}$ and 17$^{\text{th}}$, respectively.

6.3.2. Somewhat important: line crossings (A1), line bends (A2), class placement (A8), and node sizes (A10)

Line crossings (A1). Ware et al. [18] did not find a significant effect for line crossing angles (F2) on the shortest path between two nodes in a graph with regards to the time needed to perceive the shortest path. However, in our dataset, this feature shows a slight importance and ranks 7^{th} among all features we considered. The positive correlaction coefficient means that crossing angles should aproach orthogonality (90°) for a better layout. In contrast, while most studied literatures show a strong preference of minimizing line crossings (F1) to help with the time needed and the correctness when carrying out graph understandability tasks [23, 24, 13, 18, 19], this feature is only placed in the 12^{th} rank in our study.

Line bends (A2). Earlier studies [23, 17, 24] found a significant effect of minimizing line bends to the task of understanding graphs. The effect also approaches significance with regards to the time needed to solve the task. Purchase et al. [13] found a 91% preference for fewer line bends, and Ware et al. [18] found a significant effect for increasing the continuity of the shortest path between two nodes in a graph with regards to the time needed to perceive the shortest path. Eichelberger [19] ranked general line constraints, including line bends, in the sixth place out of 14 aesthetics. He also ranked graph drawing constraints, including line bends, in the fourteenth place. In our study, line bends have a small effect on layout quality, ranking 7^{th} out of 18.

Class placement (A8). Our study puts rectangle proximity (F14) in the 9^{th} place, while Eichelberger [19] bestows more importance to general node constraints, including distances between nodes, in the third place out of 14 aesthetics. Purchase et al. [17] did not find a significant effect for node distribution (F13) with regards to the understandability of a graph. Similarly, we found little contribution of this feature to layout quality.

Node sizes (A10). Eichelberger [19] puts rectangle size (F15) in the same category of general node constraints menitoned in the previous paragraph, and endorses smaller rectangles in a diagram. We have a slight tendency to agree with this observation, reflected by the small negative correlation in our dataset, putting this feature in the 10^{th} place. We found no previous work related to the importance of rectangle size variation. Our study shows

a modest preference of having similarly sized rectangles, ranking 13th out of 18.

6.3.3. Insignificant: diagram drawing size (A5)

Purchase et al. [13] found a 73% preference for narrower diagrams. Eichelberger [19] ranked graph drawing constraints, including aspect ratio, on the fourteenth place out of 14 aesthetics. We found that neither rectangle coverage (F11) nor diagram aspect ratio (F12) have any effect to perceived layout quality. No previous work related to the importance of rectangle coverage was found.

7. Threats to Validity

This section brings up threats to the validity of this research and what was done to mitigate those threats. Various forms of validity risks (i.e., internal, external, construct, and conclusion validity) are frequently encountered in experiment-based research [52]. We tried to mitigate and eradicate these threats as much as possible during this study; however, some of these threats are beyond our control.

7.1. Internal validity

Internal threats involve the factors that influence evaluation without our knowledge or outside of our control. One of the internal threats arises from the validity of ground truth in terms of labelling.

Validity of the Ground Truth of the layout quality of the diagrams

The labeling of the diagrams was done by asking people about their subjective perception of the layout quality of the diagrams, which means the labels are no absolute truth for layout quality. To mitigate this risk, each diagram was labeled by two different persons, and the average of those was used as the label for the diagram. The dataset is also considered large enough to balance out potential differences between labelers.

There is also a risk that the labels do not represent layout quality regarding how easy it is to understand and work with the diagrams. This risk had to be taken to label such an extensive dataset that we used. Extensive user experiments would have to be carried out on each diagram to get more

representative labels in this aspect.

In the second round of labeling validation, the introduction of layout flaws options might have biased labelers to correlate the labeled quality with particular aesthetics. For example, when a labeler sees *crossing lines* as a layout flaw, they might look for crossing lines in the diagram and rate it higher if there are few crossing lines. This might increase the correlation between crossing lines and the labels. This was still considered a good tradeoff as it greatly simplified the discussions about the labeling strategy.

7.2. Construct validity

Construct threats are linked to the application–theory relationship. The threat here is that there is no formal agreement on what denotes the quality of class diagram layouts. There are only features that indicate low or high quality. In our case, we used guidelines and assessed inter-rater agreement on labels.

Feature extraction

There is a risk that the calculated features do not accurately represent the aesthetics they are supposed to represent. As seen in Section 4.3.1, the image processing is not perfect, which affects how the features are calculated. If, for example, the longest line in a diagram is not found, the *Longest line* feature will not be accurate. Moreover, the actual calculation of the features might not perfectly represent the features. For some features, like *Longest line* and *Shortest line*, the calculation is straightforward. However, for features with a more complex calculation, like *Rectangle orthogonality*, it is possible that the calculations are not perfect representatives of the features. While these risks are indeed validity threats, it is more likely that they have a negative impact, rather than a positive, on our results: Incorrectly calculated features create noise that makes it harder for the machine learning algorithms to find correlations between the features and layout quality. Therefore, the results gained can be considered valid from this aspect.

Feature selection

There is also a possibility that there are other features than the extracted ones that better indicate layout quality. For example, it could be that some semantic issues need to be considered when constructing layouts, which is proposed by, for example, Purchase et al. in [17]. A limitation of our research

was not to consider such semantics aspects of the design. Moreover, we did not find it feasible to extract some of the aesthetic features, e.g., *Symmetry*, as described in Section 4.3.2.

7.3. External Validity

Threats to external validity are related to the generalization of the results to various forms of real-world problems.

Representativeness of the Diagram dataset

All diagrams that we used come from an earlier study that crawled open source projects for the use of UML. There is a possibility that open source projects might not represent software projects in general.

In that study, the aim was to select 'real' software development projects and avoid 'toy'- and 'student'-projects. While their selection may not be perfect, we expect the same focus in our dataset. Also, reverse engineered diagrams were filtered out of the original dataset and thus do not appear in our study.

Complementary to the original dataset, it could be interesting for an evaluator to be trained on student diagrams. However, we do know that there is no fundamental difference in the size of diagrams because 'real' projects also limit the size of their diagrams to no more than fit reasonably on one page of A4 paper, which is typically around 10 ± 2 classes (larger designs are typically broken up across multiple diagrams hierarchically). Nevertheless, overall, the criteria for a good layout should still be the same across all types of class diagrams, independent of the nature of the project.

We manually constructed a dataset by going over all diagrams one by one. As a result, there could be a risk that the filtering was biased. The filtering was performed systematically and carefully by defining and applying clear filtering rules to minimize this risk. Also, we looked at the distribution of the quality labels of the resulting collection of diagrams, and this shows a close to a normal distribution of the labels (see Section 4.2.1) and ranges from terrible layouts to excellent ones, with most being 'average'.

7.4. Conclusion Validity

Threats to conclusion validity concern the relationship between the results and the conclusion.

Machine learning

The Likert scale used for labeling diagrams is naturally ordinal, while regression algorithms require an interval scale. As argued for in Section 4.2.1, the used scale can be considered an interval scale since it is symmetric around a middle point, and the distance between the options is intended to be the same.

Another threat here is related to the implemented machine learning algorithms. Many different machine learning algorithms may be suitable for different use cases. For the scope of this project, algorithms provided by the machine learning software tool *WEKA* that can perform regression are used. Many machine learning algorithms have parameters that can be configured to optimize the performance of the algorithms. In this project, only machine learning algorithms provided by *WEKA* and their default parameter settings were investigated. However, there is a possibility that other machine learning approaches and optimization of parameter settings could give better-performing evaluators. However, this does not mean that the found results are invalid, only that there is a potential for improvements.

8. Conclusion and Future Work

We have developed a fully automated approach for evaluating the layout of UML class diagrams. This approach was built using regression-based machine learners, using an extensive collection of 600+ manually labeled diagrams as ground truth for training and testing.

The performance of the automatic evaluator was analysed: The scale for quality of diagrams we used ranged from 1 (denoting very bad) to 5 (denoting very good). Our evaluator produced a number on this scale: in 75.4% of the cases the value produced was not more than 0.5 away from the ground truth. On the entire dataset, the evaluator has a Relative Absolute Error (RAE) of around 0.6 on a 5-point scale.

We analyzed which features of UML class diagrams are most strongly related to the quality of their layout. For this, we presented a ranked list of features. Our study points out the following features as the most important for the quality of UML class diagrams:

- Features related to the line lengths imply that in a good layout, related classes should be placed closely together, and the lines that connect

classes should take a short/direct route between these classes. At the same time, diagrams should avoid crossing lines.

- Orthogonality of classes placement, i.e., classes should be placed as much as possible on a (virtual) grid with horizontal and vertical alignment of classes.

The evaluator can be used for various purposes: In an industrial setting, the evaluator can be embedded in Quality-Assurance tools that automatically check the quality of artifacts produced within a project. This could propel the quality management of modeling artifacts, which lags behind that of code artifacts. In an educational setting, our evaluator can be used as a component in the automated grading of UML class diagrams. This could fit well in online learning environments, which are becoming more popular. Finally, our evaluator could also be used in the field of algorithms that try to generate layouts for class diagrams automatically. For example, such algorithms are used in reverse engineering scenarios where diagrams represent source code artifacts. In this scenario, we imagine that our evaluator could be used as an oracle that would be able to provide feedback to machine-learning layouting algorithms.

As part of this study, we produced a dataset of images together with a ground truth consisting of manually produced and validated labels describing the layout quality of the respective diagram and a set of features extracted via image processing. As far as we know, this is the first dataset of this kind. We make this dataset available such that it can be used for verification of our results and further studies into layout aesthetics and layout quality of diagrams.

Future Directions

This work focuses specifically on evaluating the layout of UML class diagrams. It would be interesting to apply the same approach to other types of diagrams. As different diagram types have different kinds of elements and are structured differently, it might be challenging to find a general approach that works well for all diagram types. At the same time, there are many commonalities in guidelines for the aesthetics of diagram layouts, such as minimizing crossing lines, which apply to multiple types of diagrams. Perhaps some context-dependent tailoring of this work would make it usable for other diagram types as well.

The results of this work can be used for developing new automatic layout algorithms. The aesthetics that were found to be most important could possibly be given more weight in these algorithms. Also, our evaluator can be used to evaluate the diagrams that are produced by the algorithms. This could open up the possibility for reinforcement learning for layout algorithms.

Another area where our evaluator could be useful is in the overall assessment of UML models. This topic has recently gained increasing attention [53, 54, 55], but is not entirely solved. These types of assessments focus on the quality of the design by looking at the quality of the decomposition and the conformance or violation of design principles, such as coupling. Such assessments could be complemented by assessing the quality of the layout.

From the perspective of learning how to create diagrams with a good layout, it could be helpful to get specific feedback on which aspect of a diagram could be improved, rather than only a grade—which is the feedback produced by the current evaluator. Possibly, more specific feedback could be automatically created in addition to the 1-to-5 scale number currently produced by looking at which layout features of a diagram have a value that stands out compared to the average value of that feature for a collection of good diagrams.

In order to improve the quality and robustness of our evaluator, image recognition could be improved. Our image processes algorithms do not always recognize all lines in diagrams. Even though this happens most often in complicated cases, such as dotted lines, curved lines, or partially obscured lines, some improvement to this step might improve the robustness of the approach. We also implore future studies to consider features regarding text within diagrams that we have not been able to include in this study, e.g., text size and alignment.

Acknowledgments

The research of Fadhl Hujainah and Michel Chaudron is supported by VINNOVA Grant 2018-05010 (TrafCloud).

References

[1] G. Reggio, M. Leotta, F. Ricca, D. Clerissi, What are the used UML diagrams? a preliminary survey., EESSMOD@ MoDELS 1078 (2013).

[2] G. Reggio, M. Leotta, F. Ricca, Who knows/uses what of the UML: A personal opinion survey, in: International Conference on Model Driven Engineering Languages and Systems, Springer, 2014, pp. 149–165.

[3] O. Badreddin, R. Khandoker, A. Forward, T. Lethbridge, The evolution of software design practices over a decade: A long term study of practitioners., J. Object Technol. 20 (2021) 1–1.

[4] G. Scanniello, C. Gravino, M. Genero, J. A. Cruz-Lemus, G. Tortora, M. Risi, G. Dodero, Do software models based on the uml aid in source-code comprehensibility? aggregating evidence from 12 controlled experiments, Empirical Software Engineering 23 (2018) 2695–2733.

[5] A. M. Fernández-Sáez, M. Genero, M. R. V. Chaudron, D. Caivano, I. Ramos, Are forward designed or reverse-engineered uml diagrams more helpful for code maintenance?: A family of experiments, Information and Software Technology 57 (2015) 644–663.

[6] H. Störrle, On the impact of layout quality to understanding UML diagrams, in: 2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2011, pp. 135–142.

[7] T. Ziadi, M. A. A. d. Silva, L. M. Hillah, M. Ziane, A fully dynamic approach to the reverse engineering of UML sequence diagrams, in: 2011 16th IEEE International Conference on Engineering of Complex Computer Systems, 2011, pp. 107–116. doi:10.1109/ICECCS.2011.18.

[8] E. Fauzi, B. Hendradjaya, W. D. Sunindyo, Reverse engineering of source code to sequence diagram using abstract syntax tree, in: 2016 International Conference on Data and Software Engineering (ICoDSE), 2016, pp. 1–6. doi:10.1109/ICODSE.2016.7936137.

[9] M. J. Decker, K. Swartz, M. L. Collard, J. I. Maletic, A tool for efficiently reverse engineering accurate UML class diagrams, in: 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2016, pp. 607–609. doi:10.1109/ICSME.2016.37.

[10] U. Sabir, F. Azam, S. U. Haq, M. W. Anwar, W. H. Butt, A. Amjad, A model driven reverse engineering framework for generating high level UML models from java source code, IEEE Access 7 (2019) 158931–158950. doi:10.1109/ACCESS.2019.2950884.

[11] K. Singh, Transformation of source code into UML diagrams through visualization tool, International Journal of Advanced Science and Technology 29 (2020) 4861–1114.

[12] R. G. Alsarraj, A. M. Altaie, A. A. Fadhil, Designing and implementing a tool to transform source code to UML diagrams, Periodicals Of Engineering And Natural Sciences 9 (2021) 430–440.

[13] H. C. Purchase, J.-A. Allder, D. Carrington, User preference of graph layout aesthetics: A UML study, in: International Symposium on Graph Drawing, Springer, 2000, pp. 5–18.

[14] H. Eichelberger, Aesthetics and automatic layout of UML class diagrams, doctoral thesis, Universität Würzburg, 2005.

[15] C. F. J. Lange, Assessing and improving the quality of modeling : a series of empirical studies about the UML, Ph.D. thesis, Mathematics and Computer Science, 2007. doi:10.6100/IR629604, proefschrift.

[16] H. C. Purchase, Metrics for graph drawing aesthetics, Journal of Visual Languages & Computing 13 (2002) 501–516.

[17] H. C. Purchase, M. McGill, L. Colpoys, D. Carrington, Graph drawing aesthetics and the comprehension of UML class diagrams: An empirical study, in: Proceedings of the 2001 Asia-Pacific Symposium on Information Visualisation - Volume 9, APVis '01, 2001, pp. 129–137.

[18] C. Ware, H. Purchase, L. Colpoys, M. McGill, Cognitive measurements of graph aesthetics, Information Visualization 1 (2002) 103–110.

[19] H. Eichelberger, Aesthetics of class diagrams, in: Proceedings First International Workshop on Visualizing Software for Understanding and Analysis, 2002, pp. 23–31.

[20] H. Eichelberger, K. Schmid, Guidelines on the aesthetic quality of UML class diagrams, Information and Software Technology 51 (2009) 1686 – 1698. Quality of UML Models.

[21] D. Sun, K. Wong, On evaluating the layout of UML class diagrams for program comprehension, in: 13th International Workshop on Program Comprehension (IWPC'05), 2005, pp. 317–326.

[22] M. K. Coleman, D. S. Parker, Aesthetics-based graph layout for human consumption, Software: Practice and Experience 26 (1996) 1415–1438.

[23] H. C. Purchase, R. F. Cohen, M. James, Validating graph drawing aesthetics, in: F. J. Brandenburg (Ed.), Graph Drawing, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996, pp. 435–446.

[24] H. Purchase, Which aesthetic has the greatest effect on human understanding?, in: G. DiBattista (Ed.), Graph Drawing, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 248–261.

[25] P. Effinger, N. Jogsch, S. Seiz, On a study of layout aesthetics for business process models using bpmn, in: International Workshop on Business Process Modeling Notation, Springer, 2010, pp. 31–45.

[26] A. Dikici, O. Turetken, O. Demirors, Factors influencing the understandability of process models: A systematic literature review, Information and Software Technology 93 (2018) 112–129.

[27] B. Karasneh, M. R. V. Chaudron, Extracting UML models from images, in: 2013 5th International Conference on Computer Science and Information Technology, 2013, pp. 169–178.

[28] B. Karasneh, M. R. V. Chaudron, Img2uml: A system for extracting UML models from images, in: 2013 39th Euromicro Conference on Software Engineering and Advanced Applications, 2013, pp. 134–137.

[29] T. Ho-Quang, M. R. V. Chaudron, I. Samúelsson, J. Hjaltason, B. Karasneh, H. Osman, Automatic classification of UML class diagrams from images, in: 2014 21st Asia-Pacific Software Engineering Conference, volume 1, 2014, pp. 399–406.

[30] O. Ņikiforova, D. Ahiļčenoka, D. Ungurs, K. Gusarovs, L. Kozačenko, Several issues on the layout of the UML sequence and class diagram, publication. editionName (2014) 40–47.

[31] R. Hebig, T. H. Quang, M. R. V. Chaudron, G. Robles, M. A. Fernandez, The quest for open source projects that use UML: Mining github, in: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, MODELS '16, 2016, pp. 173–183.

[32] R. Likert, A technique for the measurement of attitudes, Archives of Psychology 22 (1932) 5 – 55.

[33] P. E. Shrout, J. L. Fleiss, Intraclass correlations: uses in assessing rater reliability., Psychological bulletin 86 (1979) 420.

[34] D. V. Cicchetti, Guidelines, criteria, and rules of thumb for evaluating normed and standardized assessment instruments in psychology, Psychological Assessment 6 (1994) 284 – 290.

[35] T. K. Koo, M. Y. Li, A guideline of selecting and reporting intraclass correlation coefficients for reliability research, Journal of Chiropractic Medicine 15 (2016) 155 – 163.

[36] A. Leguina, A primer on partial least squares structural equation modeling (pls-sem), 2015.

[37] M. G. Bulmer, Principles of statistics, Courier Corporation, 1979.

[38] J. Canny, A computational approach to edge detection, IEEE Transactions on pattern analysis and machine intelligence (1986) 679–698.

[39] P. J. Burt, Fast filter transform for image processing, Computer graphics and image processing 16 (1981) 20–51.

[40] R. O. Duda, P. E. Hart, Use of the hough transformation to detect lines and curves in pictures, Commun. ACM 15 (1972) 11–15.

[41] S. Suzuki, et al., Topological structural analysis of digitized binary images by border following, Computer vision, graphics, and image processing 30 (1985) 32–46.

[42] U. Ramer, An iterative procedure for the polygonal approximation of plane curves, Computer graphics and image processing 1 (1972) 244–256.

[43] D. J. Mackay, Introduction to gaussian processes, 1998.

[44] S. Shevade, S. Keerthi, C. Bhattacharyya, K. Murthy, Improvements to the smo algorithm for svm regression, in: IEEE Transactions on Neural Networks, 1999.

[45] A. Smola, B. Schoelkopf, A tutorial on support vector regression, Technical Report, 1998. NeuroCOLT2 Technical Report NC2-TR-1998-030.

[46] R. Kohavi, The power of decision tables, in: 8th European Conference on Machine Learning, Springer, 1995, pp. 174–189.

[47] G. Holmes, M. Hall, E. Frank, Generating rule sets from model trees, in: Twelfth Australian Joint Conference on Artificial Intelligence, Springer, 1999, pp. 1–12.

[48] R. J. Quinlan, Learning with continuous classes, in: 5th Australian Joint Conference on Artificial Intelligence, World Scientific, Singapore, 1992, pp. 343–348.

[49] Y. Wang, I. H. Witten, Induction of model trees for predicting continuous classes, in: Poster papers of the 9th European Conference on Machine Learning, Springer, 1997.

[50] L. Breiman, Random forests, Machine Learning 45 (2001) 5–32.

[51] J. D. Evans, Straightforward statistics for the behavioral sciences., Thomson Brooks/Cole Publishing Co, 1996.

[52] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer Science & Business Media, 2012.

[53] D. R. Stikkolorum, P. van der Putten, C. Sperandio, M. R. V. Chaudron, Towards automated grading of UML class diagrams with machine learning, in: K. Beuls, B. Bogaerts, G. Bontempi, P. Geurts, N. Harley, B. Lebichot, T. Lenaerts, G. Louppe, P. V. Eecke (Eds.), Proceedings of the 31st Benelux Conference on Artificial Intelligence (BNAIC 2019) and the 28th Belgian Dutch Conference on Machine Learning (Benelearn 2019), Brussels, Belgium, November 6-8, 2019, volume 2491 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2019.

[54] Y. Boubekeur, G. Mussbacher, S. McIntosh, Automatic assessment of students' software models using a simple heuristic and machine learning, in: E. Guerra, L. Iovino (Eds.), MODELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and

Systems, Virtual Event, Canada, 18-23 October, 2020, Companion Proceedings, ACM, 2020, pp. 20:1–20:10.

[55] W. Bian, O. Alam, J. Kienzle, Is automated grading of models effective?: assessing automated grading of class diagrams, in: E. Syriani, H. A. Sahraoui, J. de Lara, S. Abrahão (Eds.), MODELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems, Virtual Event, Canada, 18-23 October, 2020, ACM, 2020, pp. 365–376.