

# Nice Class Diagrams Admit Good Design?

Holger Eichelberger\*  
University of Würzburg  
Am Hubland  
97074 Würzburg, Germany

## Abstract

Analysis and design of programs by using tools has emerged to a standard technique in object-oriented software engineering. Many of these tools claim to implement methods according to the UML standard and some of the tools provide automatic layout of the diagrams drawn by the user or generated automatically from source code. In this paper we propose a set of aesthetic criteria for UML class diagrams and discuss the relation between these criteria, HCI and design aspects of object-oriented software.

First we describe critics from the viewpoint of HCI to the UML notation and restrict ourself to changes which do not require non-standard modifications to the UML notation guide, then we list quality relations between class diagrams and object-oriented software models. After that our set of aesthetic criteria, that reflect the highly sophisticated structural and semantic features of UML class diagrams, is explained. Finally, we show that an implementation and measurement of this proposal is realizable using a prototypical graph drawing framework.

**CR Categories:** D.2.6 [Software Engineering]: General—Standards—UML; D.2.2 [Software Engineering]: Design Tools and Techniques—Computer-aided software engineering (CASE); D.2.4 [Software Engineering]: Software/Program Verification—Model checking; D.2.8 [Software Engineering]: Metrics; H.5.2 [Information Interfaces and Presentation]: User Interfaces—Theory and methods;

**Keywords:** UML class diagrams, aesthetics, metrics, HCI, software engineering

## 1 Introduction

Class diagrams according to the standardized software development language UML (Unified Modeling Language) [OMG 2001] have been frequently implemented in different tools in the last years. Even if these tools claim to be conformant to the UML, most tools support only a subset of the standardized features [Eichelberger 2002b]. Model exchange between different tools is currently extremely difficult, because most tools implement a vendor-specific version of the standard model interchange format XMI [OMG 1999] in order to store layout information like coordinates or font descriptions. Therefore automatic layout algorithms are useful

when pure model information is exchanged, diagrams are generated from source code and the analysis/design documents are validated against the implementation.

As a modeling standard the UML does not say anything on how to produce readable diagrams. Especially when larger diagrams are shared, an agreement on aesthetics has to be made in order to reduce the costs of communication and to minimize misunderstandings which result from drawing the same diagram in many different ways. Unfortunately, tools of different vendors calculate the layout of class diagrams according to different aesthetic principles. Therefore we propose a set of aesthetic criteria for UML class diagrams that reflect all features of the current class diagram specification. From these criteria a set of metrics can be derived, that then can be used to automatically classify and calculate a quality judgement.

After a brief description of related work we review the critics on the UML notation from the viewpoint of HCI, human perception and cognitive psychology and conclude valid layout methods without introducing non-standard modifications to the UML notation guide. Since UML class diagrams are used to model static aspects of object-oriented software, we describe facts which relate to design quality and list relations between design and layout quality. In section 5 we present our set of aesthetic criteria. Finally a discussion on realizing these criteria by a concrete layout algorithm, several conclusions and future work is given.

## 2 Related Work

In [Eichelberger 2002b] we showed that current UML tools, which are usually CASE (Computer Aiding Software Engineering) tools, do not implement state-of-the-art layout algorithms. Most of the 43 tools regarded in this evaluation do not implement all features specified by the UML and some of them produce horrible layout results. Based on this evaluation we proposed in [Eichelberger 2002a] a set of aesthetic criteria and gave some positive and negative examples produced by different commercial tools. In [Eichelberger 2002a] we shortly described an algorithm to realize our proposal and a prototypical implementation of this algorithm. A more precise introduction to our algorithm can be found in [Eichelberger and Wolff von Gudenberg 2002].

Even if the layout of class diagrams appears as a NP-hard problem, there are several other promising approaches to this problem: Caesar [Gutwenger et al. 2002], yFiles [Wiese et al. 2002] or DiaGen [Köth and Minas 2002].

Caesar is a C++-library which tries to find a balanced layout respecting aesthetic criteria like crossing minimization, bend minimization, uniform direction, orthogonal layout and edge labeling but any UML-based aspects. yFiles is a Java-based library which implements different data structures and algorithms for graph drawing according to orthogonalization and compaction. The DiaGen architecture is based on a hypergraph transformer and gathers information with influence on the semantic representation of the drawing.

A set of general graph drawing aesthetics is described in [Battista et al. 1999; Batini et al. 1985; Wetherell and Shannon 1979; Tamassia et al. 1988], basic measurements on general graphs are given in

\*e-mail: eichelberger@informatik.uni-wuerzburg.de

[Bridgeman and Tamassia 2000; Gansner et al. 1988; Kosak et al. 1994; Herman et al. 1998], an aesthetic driven approach based on simulated annealing is described in [Davidson and Harel 1996]. Previous work on UML diagrams can be found in [Purchase et al. 2000; Purchase et al. 2001]. Unfortunately the main focus of these two papers is on graph drawing aesthetics (minimization of bends, minimization of edge crossings, orthogonality and a nice aspect-ratio) without considering the underlying semantics of the diagrams or the needs of software engineers. In [Purchase et al. 2000; Purchase et al. 2001] the class diagrams are treated like usual graphs without considering advanced features like packages, hyperedges, association classes and their semantic relations to other model elements.

### 3 UML Critics

After long years of discussion (Booch, OOSE, Coad-Yourdon, Fusion, Shlaer-Mellor) the notion of a commonly agreed notation for object-oriented software design has become obvious. The software engineering community agreed to a unified notation which emerged to an OMG standard. We assume that the reader is familiar with the most frequently used parts of this notation. Despite of the critics which rely on the software engineering aspects, other disciplines started to criticize the UML from their viewpoints.

The UML notation lacks in different graphical codes (see [Ware 2000]) which are common for node-link-diagrams. Neither color, which refers to the entity type, nor the size of the elements, which refers to the magnitude of the entity are specified in the UML. Partitions within enclosed regions, which obviously refer to entity partitions in packages e.g. are ignored. Shapes enclosed by contour introduce multiple confusingly meanings: packages in packages or classes in packages are contained, classes in classes are composed and inner or nested classes which are usually structurally enclosed are attached by the so called anchor notation (which is defined for packages as well). Additionally the thickness of links usually describes the strength of connections and neither the meaning of proximity nor spatial ordering of model elements are defined in the UML. An enhanced notation based on these critics and the principles of Geon diagrams (proximity, similarity, closure) is described in [Irani and Ware 1999].

Regarding notational and structural aspects, in [Diskin et al. 2000; Diskin 2002] a more precise notation based on the arrow-diagram logic framework is proposed.

Additionally the UML and the tools implemented to support working with the UML could be analyzed from the viewpoint of HCI and cognitive psychology by regarding the framework of cognitive dimensions in [Green and Blackwell 1998]. *Viscosity* (resistance to local change) and *premature commitment* (based on early decisions in hand-crafted layouts) directly map to CASE tools and can be reduced by good automatic layout algorithms. The *abstraction level* (grouping of elements), *consistency* and *role-expressiveness* map to the UML itself, *error-proneness* and *hard mental operations* in interpreting diagrams can be reduced by further standardization and improved versions of the UML. Higher international acceptance can be reached by incorporating results from other disciplines like those described in this section. The experience of the designer in using the UML and the viewpoint to be described by the diagrams affect *hidden dependencies* (as well to be managed by tools). *Closeness of mapping* and *diffuseness/terseness* are subject to the individual design as well as subject of discussions about the notation itself.

We do neither want to introduce another modeling framework nor a different kind of notation to influence the relations between diagrams and design, because introducing changed notations into a standardized and widely accepted method increases *error-proneness*, the number of *mental operations* and influences *consistency*.

Therefore *secondary notation and escape from formalism* like introducing colors or respecting extensions in order to realize the common graphical codes for node-link-diagrams may lead to confusion. Additionally colors introduce further problems in the case of color-blind users or different cultural interpretation.

Hence we decided to use the current version of UML without modifications and to focus on

**P1: visibility and juxtaposability** by regarding and enforcing spatial relationships.

**P2: the size of elements** to reflect the magnitude of the entity (see [Ware 2000]).

### 4 Class Diagrams and Design

Since class diagrams describe the static class structure of object-oriented models, further aspects of object-oriented software engineering should be taken into account. The main question is: Is the beauty of a diagram measured in dimensions of aesthetic criteria somehow related to the quality the object-oriented design modeled by that diagram?

One direction is obvious: If a well designed model is laid out by one of the layout algorithms described in [Eichelberger 2002b] the result is usually a horrible layout. Therefore a horrible layout usually does not always imply a poor quality of the model.

In order to inspect the other direction, we have to find out what criteria influence the quality of an object-oriented model and how these criteria can be measured. Many publications (development processes [Jacobson et al. 1999], common software engineering [Summerville 1996], analysis and design patterns [Gamma et al. 2000] e.g.) dealing with object-oriented design, describe commonly agreed methods how to find classes and relations between them and especially how to focus on the relevant aspects of the software system to be modeled. In [Genero et al. 2000] an overview of current measurements on design aspects is given. Usually source code related measurements are not taken into account for layout algorithms because in the early iterations design documents and not source code are produced. On the other side, coding-style independent source code metrics [Lorenz and Kidd 1994; Zuse 1998] might be respected in round-trip engineering when code is synchronized with diagrams and vv.

The design criteria D1 to D8 introduced below as well as the HCI criteria P1 and P2 are then used in section 5 to emphasize the validity of our aesthetic criteria for UML class diagrams.

**D1: forests:** The depth of inheritance trees introduced in [Chidamber and Kemerer 1994] partially limits the physical dimensions of the drawing. On the other side, well-designed OO systems are those structured as forests of classes, rather than as one very large inheritance lattice [Basili et al. 1996; Marchesi 1998]. Therefore these trees should be clearly visible and spatially separated from each other according to P1.

**D2: kind of inheritance:** The in-degree restricted to inheritance relations should be minimized like described in [Lorenz and Kidd 1994] (restricted use of multiple-inheritance, e.g.). Therefore the inheritance trees/implementation hierarchies are simplified.

**D3: number of children:** The number of children introduced in [Chidamber and Kemerer 1994] relates to the difficulty to modify the implementation and usually requires more testing because the class potentially affects all of its children. Furthermore, a class with numerous children may have to provide services in a larger number of contexts and must be flexible

[Basili et al. 1996]. Therefore the out-degree restricted to inheritance relations should be limited. Hence, the inheritance trees/implementation hierarchies are simplified. Child nodes should be positioned in a close vicinity to the ancestor nodes and be spatially separated to the other nodes of the diagram (see P1 and D1).

**D4: class size metrics:** Despite of obvious countings of class members, different other approaches have been proposed in [Brito e Abreu and Melo 1996; Marchesi 1998; Lorenz and Kidd 1994; Zuse 1998]. One obvious point is that empty classes, that do neither implement any methods nor define any attributes, are a lack in design quality [Lorenz and Kidd 1994]. They do not add any functionality except of usually inheriting from superclasses. While design takes progress there might be some classes which have not been fully specified so far. This could be marked by ellipses instead of method or attribute signatures in order to show, that these classes will not remain empty. Note that empty interfaces are widely used as marker interfaces (like `Serializable` in Java) and should not be treated like empty classes.

In [Genero et al. 2000] further measurements on class complexity are mentioned: number of associations, height of a class within the aggregation hierarchy, number of multiple aggregations, number of in/out-dependencies and the number of (direct) parts/wholes respecting compositions. In [Marchesi 1998] additionally the weighted number of responsibilities as well as the weighted number of dependencies are described. Differences between key, abstract and concrete classes [Lorenz and Kidd 1994] can simply be flagged by (decorative) stereotypes by the modeler. Class reuse [Zuse 1998] is reflected by all UML-relations which can be attached to a class.

Based on a combined measurement reflecting the different approaches, additional tool-specific tag-values or a magnitude-

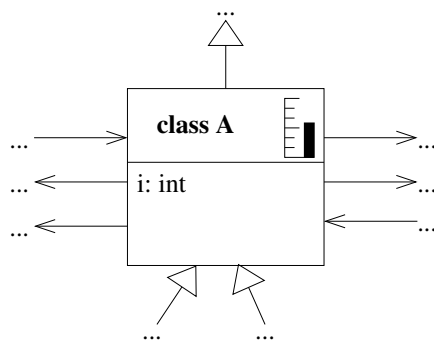


Figure 1: A class scaled to the magnitude of its coupling and a magnitude-related decorative stereotype.

related decorative stereotype (see figure 1) can be displayed within the class rectangle. Since changes of the model by a layout algorithm are usually not tolerable, the area occupied by classes (P2) might be adjusted according a weighted combination of metrics. Note, that in early analysis and design phases, classes are not always fully specified and class size metrics may lead to misinterpretations. Even if alternative line styles would be appropriate to reflect the (hidden) complexity, *error-proneness* (active classes), *hard mental operations* (increasing number of different line styles) and multiple substandards of the standard would be the consequence.

**D5: package metrics:** In [Marchesi 1998] package coupling, in [Genero et al. 2000] the number of intra/inter-package associ-

tions, aggregations and in/out-dependencies define the complexity of a package. In [Marchesi 1998] additionally the weighted number of responsibilities as well as the weighted number of dependencies are mentioned. Similar to D4 tag-values, stereotypes (see figure 1) or the magnitude of the package according to P2 might be used to reflect the complexity.

**D6: coupling:** Measuring the coupling of classes and packages is a difficult task due to different viewpoints and definitions [Bansiya et al. 1999; Chidamber and Kemerer 1994; Li et al. 1995]. An overview is given in [Briand et al. 1999]. Inheritance coupling is implicitly respected in D1, D2, D3 and obviously present in UML class diagrams by generalization edges. Component coupling is respected in D4 and D5. Interaction coupling is usually calculated from code (hidden method calls and scattered coupling) by static type analysis, but by introducing (hidden) dependencies from methods or attributes to the parameter, return or attribute types, respectively, class-attribute-interaction and class-method-interaction can be respected in a diagram. This information increases class and package complexity (see D4 and D5). Within packages, intra- and inter-package coupled classes can be visualized by spatial distribution of classes (P1, see figure 2). Shading or colouring of inter-package or intra-package coupling seems to be appropriate but this collides with the results in section 3.

**D7: design patterns:** The use of design patterns might be taken into account to measure the design quality (number of class clusters in [Zuse 1998]). The design pattern symbol and the related classes should be positioned in a close vicinity, other connected classes should be spatially separated from the design pattern collaboration according to P1. The size of the collaboration relates to P2.

**D8: relations:** Complexity of associations [Zuse 1998] and other relations could easily be displayed by applying the node-link-diagram grammar rules in [Ware 2000] but require changes to the standardized UML notation. Magnitude (P2) does not seem to be an appropriate representation.

Even if the whole complexity and history of a project is visible to a layout tool through XMI [OMG 1999], project metrics (see [Lorenz and Kidd 1994; Zuse 1998]) like application size, staffing size and scheduling do not have influence on the relation between layout and design quality. Global class diagram complexity mentioned in [Marchesi 1998] could be taken into account, if the layout result can be influenced by these measurements (genetic algorithms, simulated annealing [Davidson and Harel 1996]).

Of course many of the complexity influences can simply be visualized by introducing alternative notations like mentioned above and in section 3. According to our top-level goal, using the current version of UML without modification, we have to calculate visualizations with restricted instruments.

The combination of these metrics (see [Zuse 1998] for the mathematical background) lead to a formalized judgement of UML class diagrams with respect to object-oriented aspects. Incorporating this information into a set of aesthetic criteria for the layout of UML class diagram, a design which reaches a high judgement in design metrics will also have a high judgement respecting UML class diagram layout metrics.

## 5 Aesthetics

A class diagram can be described in terms of graph theory. It is obvious that classes, packages, rhombs representing n-ary associations and ovals in pattern notations map to nodes. Associations,



used to display the name of a package should not be crossed neither by edges nor by nodes. A general font size should be used for class interior elements belonging to the same group (class names, stereotypes e.g.) or compartment type (operations, attributes, user-defined e.g.), font attributes should be used according to the UML notation guide.

**A5: Nodes should be clustered according to semantic reasons.** Package membership, composition notation (looks like nested classes) and natural clusters like n-ary associations or patterns lead to an obvious clustering. Members of a cluster should be located in a close vicinity, especially if packages should be visualized as individual model elements. A2 and therefore P1 have to be respected.

In figure 3 C1 to C7 are members of the package sub, C9 to C11 and the rhomb are members of a ternary association. These classes are members of separate clusters.

**A6: Avoid crossings and overlappings on edges.** Different edges should not overlap, this means that every edge should be visible and readable as an individual [Fruchterman and Reingold 1991]. Of course edges should not overlap nodes. Since class diagrams usually admit a non-planar embedding, edge crossings cannot always be avoided but they should be circumvented whenever possible. In the case of edge crossings the symbol defined in the UML should be used.

**A7: General constraints on edges.** Edges should not be too short as well as edges should not be too long [Coleman and Parker 1996; Davidson and Harel 1996]. Edges should have not too much bends [Battista et al. 1999]. The angle between (horizontal) incident edges should not be too small [Coleman and Parker 1996]. Nodes should not be located too close to edges [Davidson and Harel 1996] except if they are connected to these edges or one of the other principles forces a close vicinity.

**A8: Centered position of selected nodes.** Nodes like the rhomb representing an n-ary association, the package notation node or the junction point of multiple dependencies should be centered relatively to the connected nodes. The same is true for comments but comments have a lower priority. This maps especially to P1.

In figure 3 the rhomb is centered to its attached classes C9 and C11.

**A9: Vicinity and position of association classes.** An association class is a class based specification of features of an association. An association class is attached by a dashed line to an association. Therefore the association class should be located in a close vicinity in order to simplify the reading of the diagram. Additionally the UML specification says: The attachment point should not be near enough to either end of the path that it appears to be attached to, the end of the path, or to any of the association end adornments.

In figure 3 C2 is attached to the composition between C3 and C5. C6 is attached to the reflective edge from C5 to itself.

**A10: Vicinity of comment nodes.** Comments, which are connected to other model elements, should be located as close as possible to the connected nodes. Especially if comments are connected to multiple model elements, the comment should be centered between the connected model elements if possible. Because the main information is located in all other model elements and comments represent additional descriptive information, this principle has a lower priority than A3 and A9. Comment nodes should be respected in package containment as well.

In figure 3 the comments `note1` and `note2` are located as close as possible to the connected model elements.

**A11: Hyperedges should be as short as possible.** The length of edges between edges (hyperedges) like xor-constraints at associations or generalizations of associations should be as short as possible. The connected model elements should be located in a close vicinity because of semantic reasons and a better readability.

Hyperedges are not shown in figure 3. As an alternative, generalizations of associations might be written as generalizations of empty association classes.

**A12: Adornments should be clearly assigned to their model elements.** Generally, constraints and tagged values may be attached to every model element. Additionally, associations in UML may be specified by different graphical and textual adornments like association names, multiplicities, qualifiers, navigability or other adornments and rolenames. These additional specifications as well as discriminators at inheritance edges should be clearly assigned to their model elements and should neither overlap other adornments nor other model elements. It is desirable that the text is oriented to one general direction. A general font size should be used for edge adornments belonging to the same group (constraints e.g.), font attributes should be used according to the UML notation guide. In figure 3 a directed association at C6 and an aggregation at C3 are shown.

**A13: Special requirements for reflective associations.** Adornments to reflective associations (associations which have the same class as start and endpoint) should be clearly visible. This is especially true for multiple reflective associations at one class, which should not overlap. Relations between reflective associations and other model elements like comments or association classes should not cross other model elements (especially reflective associations).

In figure 3 two reflective associations are connected to C5.

**A14: Join edges if possible.** Inheritance relations, aggregations and compositions should be joined like described in [OMG 2001] wherever possible. This admits a kind of orthogonal layout for hierarchical relations. If orthogonal layout is mainly used for the layout of non-hierarchical edges, this criterion may be omitted in order to emphasise the hierarchical relations.

**A15: Respect graph drawing constraints.** Rectangular aspect-ratio (see also P1 and P2), compact drawing (minimizing the area of the drawing, see P1 and P2), minimization of bends (respecting the number of bends on an edge or the variance of the number of bends on all edges), angular resolution and symmetry (as far as possible for UML class diagrams) are desirable. All facts of this criterion are typical for graph drawing issues and explained in [Battista et al. 1999]. But first of all it is relevant that an algorithm covers all of the sophisticated features specified in the UML and respects the vicinities implicitly defined by the UML semantics.

## 6 Layout and Design

If a layout algorithm respects all the criteria listed in the last section, it should produce a readable diagram. According to section 4 the following (incomplete) list of indicators can be seen as design problems warnings:

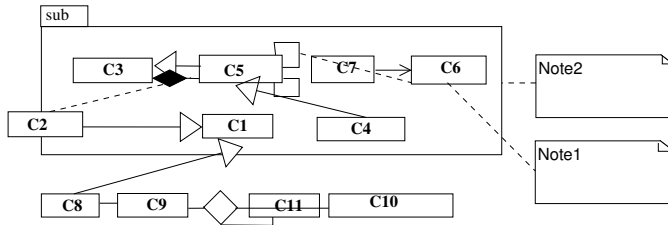


Figure 4: A bad example for a layout of the diagram in figure 3.

- 11: Huge inheritance/aggregation hierarchies (D1)** can simply be identified in a class diagram. As an example, the current Java library (version 1.4.1) consists out of more than 4000 classes, the height of the inheritance tree is 9 which fits to [Chidamber and Kemerer 1994].
- 12: Many classes at the borders of a package, few classes in the center** imply coupling problems.
- 13: Many children in hierarchy relations** signal a lack in class structuring - on the other side the inheritance hierarchy may grow while restructuring these classes.
- 14: A high percentage of the classes occupy relatively large areas** according to P2: There might be a problem with the assignment of responsibilities to that classes and a problem of class complexity.
- 15: Many inter-package-relations, fewer intra-package-relations:** The classes of such a package provide more services to classes outside that package and the members of that package use only few of that services - a problem of coupling.
- 16: Class with a high number of outgoing relations** indicate, that that classes depend on too much other classes. Again a problem of assigning responsibilities to a set of classes.
- 17: Cross-relations between independent trees respecting hierarchy relations** indicate a low use of common services or attributes in top level classes of that hierarchy.
- 18: Empty classes** are usually a design problem and should not occur (see D4).

If only layout issues like mentioned in section 5 are respected to measure the quality of a diagram, the design influences the layout, but a bad design does not automatically admit a bad drawing. Hence, a bad design does not automatically lead to lower metric values in the layout metrics. If there would be commonly agreed limits for the design metrics discussed in section 4, the indicators discussed in this section might be used to judge the design quality of a diagram. Implicitly, all these indicators affect the area occupied by the whole drawing. Therefore a better design would probably result in a lower drawing area and in a better layout metric value. Hence, there is a relation between quality and layout which is expressed in certain layout situations, the size of the drawing compared with alternative designs and the values of a composed design metric or a combined layout-design metric. Using a non-UML marking technique like shading or coloring to highlight parts of the drawing according to related design indicators in a tool, this would spot model elements and relations to be redesigned. Alternatively, classes could be flagged by additional (decorative) stereotypes like described for D4.

## 7 Realization

In [Eichelberger 2002a] we described the realization of most of the aesthetic principles listed in section 5 by a layout algorithm implemented in the prototypical graph-drawing framework *SugiBib* [Eichelberger 2002c; Eichelberger and Wolff von Gudenberg 2002]. *SugiBib* is a pure Java framework which implements preparation steps, the ranking, different edge crossing minimizations and the coordinates calculation on two types of edges based on the Sugiyama-Algorithm [Sugiyama et al. 1981]. The layout algorithm for class diagram is a multi-step extension, specialization and instantiation of the basic framework. The framework approach enables reuse and exchange of different components to easily experiment with alternative algorithms and implementations.

In [Eichelberger 2002a] to each step of the algorithm the aesthetic principles in section 5 guaranteed or respected by this step have been mentioned. The object-oriented measures can be delivered from outside or calculated in the UML extension of our basic drawing framework before or while running the layout algorithm. The new dimensions (P1 to P2) introduced by this paper can be realized as follows:

- **D6:** Signature-based coupling can be represented by hidden dependencies between class signatures and classes representing the types in these signatures. Therefore after introducing (weighted) hidden dependencies to the framework, the sub-algorithms implementing the algorithmic steps automatically respect this information.
- **P2:** Enlargement according to the magnitude of classes (complexity, coupling) can be realized after the minimum area sizes have been calculated. Model elements like packages, which contain other model elements, are automatically scaled according to the area occupied by the contained elements.
- **P1:** Spatial distribution has to be guaranteed by extensions to the ranking mechanism and the coordinates algorithm in order to enlarge the package nodes and to ensure the coupling related node positions or by an additional separate subalgorithm.

The figures 5 to 11 as well as the figures in the color plate depict the current state of the implementation.

## 8 Conclusion and Further Work

In this paper we described changes to the UML notation guide from non-software-engineering viewpoints. Most of these modifications would probably increase the readability of UML class diagrams but they should be respected in the layout of UML software diagrams only, if they are incorporated into the standard. Therefore we restricted ourselves to two dimensions (P1 and P2) which can be respected in UML class diagram layout without further changes to the standard. We analyzed object-oriented design metrics which are currently in discussion and related the information which can be gathered from these metrics to layout proposals. We presented a set of aesthetic criteria with respect to these results and discussed the realization of our proposal.

In order to verify our ideas, all aesthetic criteria have to be translated into mathematical formulae which then represent layout metrics for UML class diagrams. These formulae can then be encoded as a metrics calculation framework to be applied directly to the layout results of *SugiBib* as well as to modeling information exported by other tools (XMI [OMG 1999] e.g.). The integration of XMI into *SugiBib* is currently in progress. Based on the automatic calculation of metrics for object-oriented design models, an empirical validation of user preferences, which is scheduled for the future, can

be executed. On the one side, CASE tools might be respected in this evaluation and on the other side the layout calculated by current CASE tools can be measured and compared based on mathematical measurements. Additionally, further practical experience on encoding advanced software engineering information into standard UML diagrams will be gained from this study.

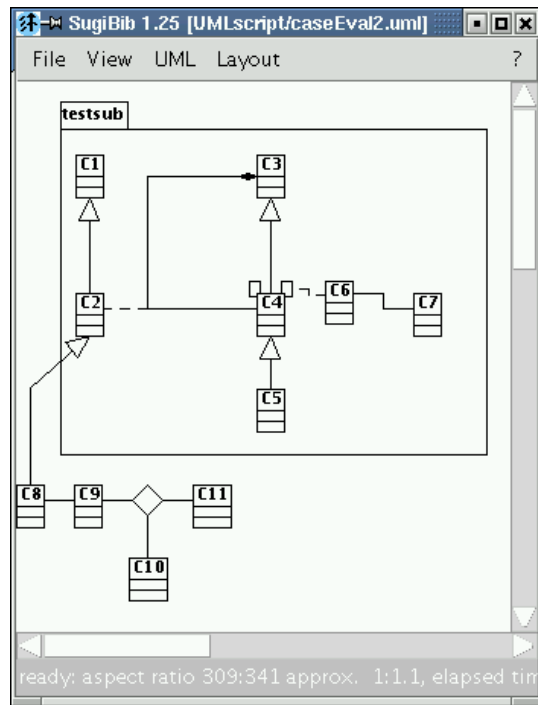


Figure 5: The example diagram in figure 3 drawn by *SugiBib*. Neither complexity nor coupling is respected. So far the layout of annotated comments is not implemented.

## References

- BANSIYA, J., ETZKORN, L., DAVIS, C., AND LI, W. 1999. A Class Cohesion Metric for Object-Oriented Designs. *Journal of Object-Oriented Programming* 11, 8 (Jan.), 47–52.
- BASIL, V. R., BRIAND, L. C., AND MELO, W. L. 1996. A Validation of Object-Oriented Design Metrics as Quality Indicators. *IEEE Transactions on Software Engineering* SE-22, 10, 751–761.
- BATINI, C., FURLANI, L., AND NARDELLI, E. 1985. What is a Good Diagram? A Pragmatic Approach. In *Entity-Relationship Approach: The Use of ER Concept in Knowledge Representation, Proceedings of the Fourth International Conference on Entity-Relationship Approach, Chicago, Illinois, USA, 29-30 October 1985*, IEEE Computer Society and North-Holland, P. P. Chen, Ed., IEEE, 312–319.
- BATTISTA, G. D., EADES, P., TAMASSIA, R., AND TOLLIS, I. 1999. *Graph Drawing*. Prentice Hall.
- BRIAND, L. C., DALY, J. W., AND WÜST, J. K. 1999. A Unified Framework for Coupling Measurement in Object-Oriented Systems. *IEEE Transactions on Software Engineering* SE-25, 1 (January/February), 91–121.
- BRIDGEMAN, S., AND TAMASSIA, R. 2000. Difference Metrics for Interactive Orthogonal Graph Drawing Algorithms. *Journal of Graph Algorithms and Applications* 4, 3, 47–74.
- BRITO E ABREU, F., AND MELO, W. 1996. Evaluating the Impact of Object-Oriented Design on Software Quality. In *Proceedings of the 3rd International Software Metrics Symposium (METRICS '96), Berlin, Germany*, IEEE, 90–99.
- CHIDAMBER, S., AND KEMERER, C. 1994. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering* SE-20, 6 (June), 476–492. Copy.
- COLEMAN, M. K., AND PARKER, D. S. 1996. Aesthetics-based graph layout for human consumption. *Software - Practice and Experience* 26, 12, 1415–1438.
- DAVIDSON, R., AND HAREL, D. 1996. Drawing Graphs Nicely Using Simulated Annealing. *ACM Transactions on Graphics* 15, 4 (Oct.), 301–331.
- DISKIN, Z., KADISH, B., PIESSENS, F., AND JOHNSON, M. 2000. Universal Arrow Foundations for Visual Modeling. In *Diagrams 2000*, Springer-Verlag, vol. 1889 of *LNAI*, Springer, 345–357. M. Anderson, P. Cheng, V. Haarslev (Eds.): *Diagrams 2000*.
- DISKIN, Z. 2002. Visualization vs. Spezifikation in Diagrammatic Notations: A Case Study With UML. In *Diagrams 2002*, Springer-Verlag, M. Hegarty, B. Meyer, and H. Narayanan, Eds., vol. 2317 of *LNAI*, Springer, 112–115.

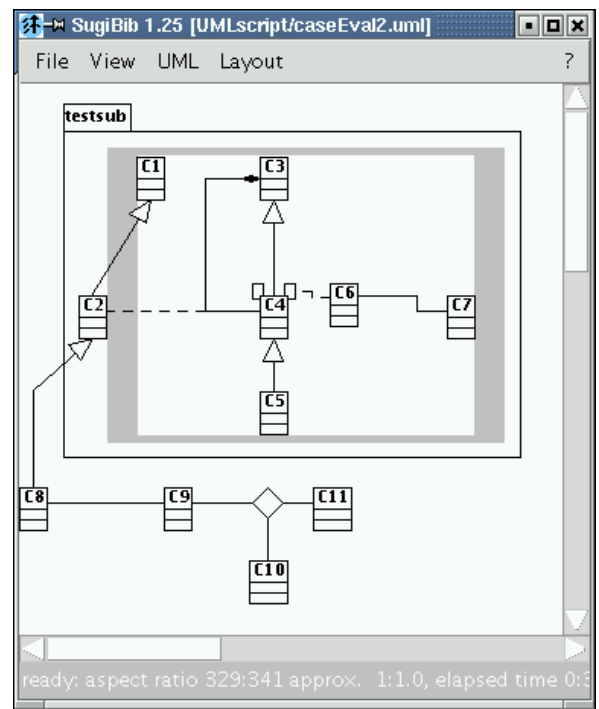


Figure 6: Activated spatial distribution on the example diagram. The shaded area emphasizes the borderline between inter-package coupled classes and intra-package coupled classes. This area is a non-UML feature and shown only for demonstration purpose.

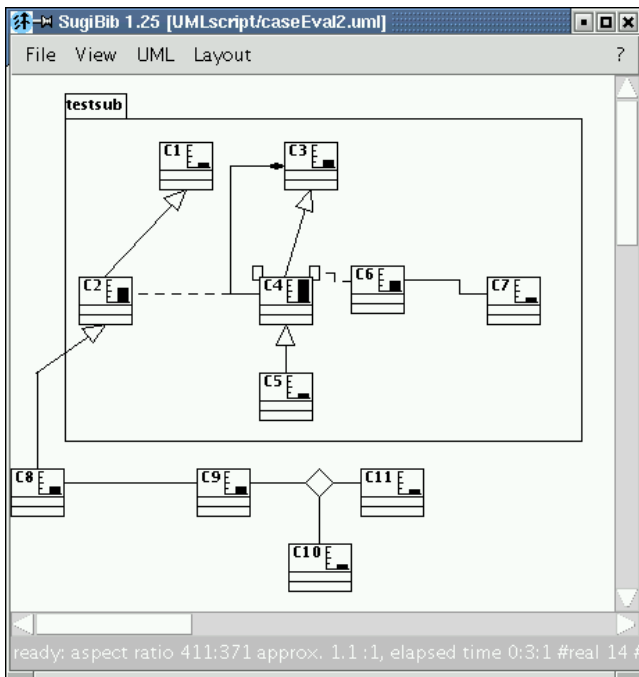


Figure 7: The example diagram with annotated complexity stereotypes and activated spatial distribution. Currently a simple plug-in complexity metric was implemented for demonstration purpose only.

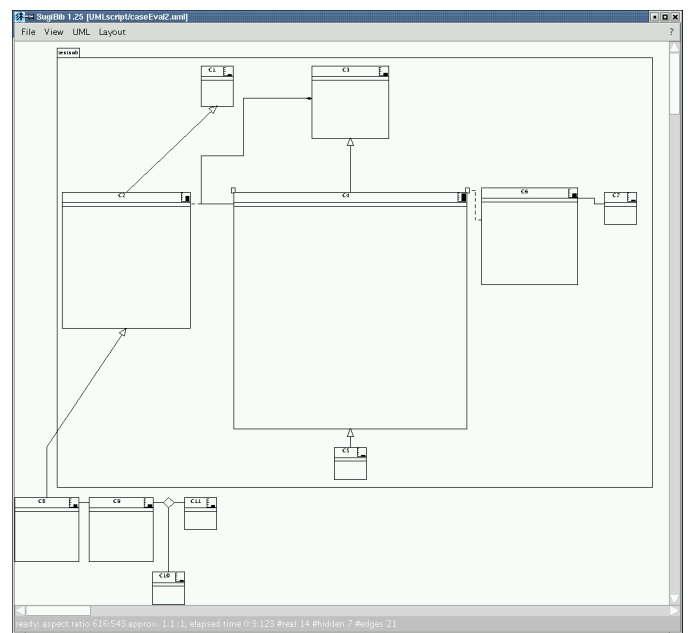


Figure 8: The example diagram with annotated complexity stereotypes, activated spatial distribution and scaling of nodes according to the complexity of the classes.

EICHELBERGER, H., AND WOLFF VON GUDENBERG, J. 2002. On the Visualization of Java Programs. In *Software Visualization, State-of-the-Art Survey*, Springer, S. Diehl, Ed., vol. 2269 of *Lecture Notes in Computer Science*, Springer, 295–306.

EICHELBERGER, H. 2002. Aesthetics of Class Diagrams. In *Proceedings of the First IEEE International Workshop on Visualizing Software for Understanding and Analysis, Paris, France*, IEEE, IEEE, 23–31.

EICHELBERGER, H. 2002. Evaluation-Report on the Layout Facilities of UML Tools. TR 298, Institut für Informatik, Universität Würzburg, jul.

EICHELBERGER, H., 2002. Layout in UML and CASE, homepage of SugiBib. via <http://www.sugibib.de/english.html>.

FRUCHTERMAN, T. M. J., AND REINGOLD, E. M. 1991. Graph Drawing by Force-directed Placement. *Software – Practice and Experience* 21, 11 (Nov.), 1129–1164.

GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. 2000. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Massachusetts.

GANSNER, E. R., NORTH, S. C., AND VO, K. P. 1988. DAG: A program that draws directed graphs. *Software – Practice and Experience* 18, 11 (Nov.), 1047–1062.

GENERO, M., PIATTINI, M., AND CALERO, C. 2000. Early measures for UML class Diagrams. *L'Objet* 6, 4.

GREEN, T., AND BLACKWELL, A., 1998. Cognitive Dimensions of Information Artefacts: a tutorial. Version 1.2, October 1998, via

<http://www.ndirect.co.uk/~thomas.green/workStuff/Papers/>.

GUTWENGER, C., JÜNGER, M., KLEIN, K., KUPKE, J., LEIPERT, S., AND MUTZEL, P. 2002. Caesar Automatic Layout of UML Class Diagrams. In *Proc. Graph Drawing, 9th International Symposium, GD '02*, Springer, P. Mutzel, M. Jünger, and S. Leipert, Eds., vol. 2265 of *Lecture Notes in Computer Science*, Springer, 461–462.

HERMAN, I., DELEST, M., AND MELANÇON, G. 1998. Tree Visualization and Navigation Clues for Information Visualization. *Computer Graphics Forum* 17, 2, 153–167.

IRANI, P., AND WARE, C. 1999. The Geon Diagram. In *Graphics Interface '99, Kingston, Ontario*, IRIS/PRECARN. Poster Abstracts.

JACOBSON, I., RUMBAUGH, J., AND BOOCH, G. 1999. *The Unified Software Development Process*. Object Technology Series. Addison-Wesley, Reading, MA.

KOSAK, C., MARKS, J., AND SHIEBER, S. 1994. Automating the layout of network diagrams with specified visual organization. *IEEE Transactions on Systems, Man, and Cybernetics SMC-24*, 3, 440–454.

KÖTH, O., AND MINAS, M. 2002. Structure, Abstraction and Direct Manipulation in Diagram Editors. In *Diagrams 2002*, Springer-Verlag, M. Hegarty, B. Meyer, and H. Narayanan, Eds., vol. 2317 of *LNAI*, Springer, 290–304.

LI, W., HENRY, S., KAFURA, D., AND SCHULMAN, R. 1995. Measuring Object-Oriented Design. *Journal of Object-Oriented Programming* (July-August), 48–55.

LORENZ, M., AND KIDD, J. 1994. *Object-Oriented Software Metrics*. Prentice Hall.

MARCHESI, M. 1998. OOA Metrics for the Unified Modeling Language. In *Proceedings of the 2<sup>nd</sup> Euromicro Conference on Software maintenance and reengineering*, JSA, 67–73.

OMG, 1999. XML Metadata Interchange (XMI). Version 1.1, via <http://cgi.omg.org/docs/ad/99-10-02.pdf>.

OMG, 2001. Unified Modeling Language Specification. Version 1.4, February 2001 via <http://www.omg.org>.

PURCHASE, H., ALLDER, J.-A., AND CARRINGTON, D. 2000. User Preference of Graph Layout Aesthetics: A UML Study. In *Graph Drawing - 8th International Symposium*, Springer, J. Marks, Ed., vol. 1984 of *Lecture Notes in Computer Science*, Springer, 5–18.

PURCHASE, H., MCGILL, M., COLPOYS, L., AND CARRINGTON, D. 2001. Graph drawing aesthetics and the comprehension of UML class diagrams: an empirical study. In *Proceedings of the Australian Symposium on Information Visualisation*, Australian Computer Society Inc., P. Eades and T. Pattison, Eds., vol. 9, Australian Computer Society Inc.

SUGIYAMA, K., TAGAWA, S., AND TODA, M. 1981. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics SMC-11*, 2 (Feb.), 109–125.

SUMMERVILLE, I. 1996. *Software Engineering*. Prentice Hall.

TAMASSIA, R., DI BATTISTA, G., AND BATINI, C. 1988. Automatic Graph Drawing and Readability of Diagrams. *IEEE Transactions on Systems, Man, and Cybernetics SMC-18*, 1 (Jan./Feb.), 61–79.

WARE, C. 2000. *Information visualization: design for perception*. Academic Press.

WETHERELL, C., AND SHANNON, A. 1979. Tidy drawings of trees. *IEEE Transactions on Software Engineering SE-5*, 5, 514–520.

WIESE, R., EIGELSPERGER, M., AND KAUFMANN, M. 2002. yFiles: Visualization and Automatic Layout of Graphs. In *Proc. Graph Drawing, 9th International Symposium, GD '02*, Springer, P. Mutzel, M. Jünger, and S. Leipert, Eds., vol. 2265 of *Lecture Notes in Computer Science*, Springer, 453–454.

ZUSE, H. 1998. *A framework of Software Measurement*. deGruyter.

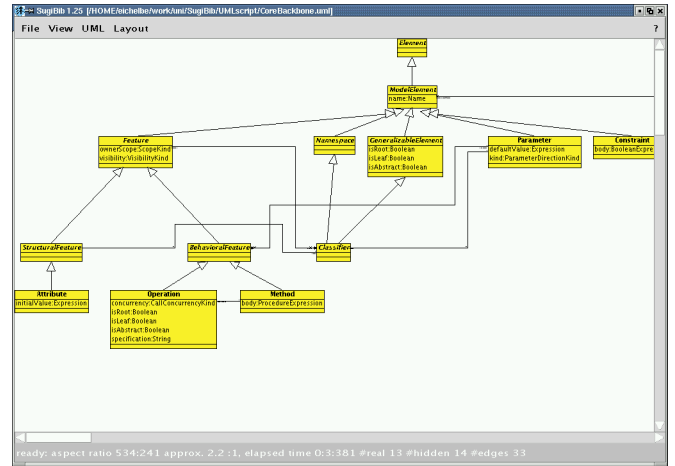


Figure 9: Another example: The UML core backbone class diagram drawn by *SugiBib*. The classes are coloured by a non-UML feature.

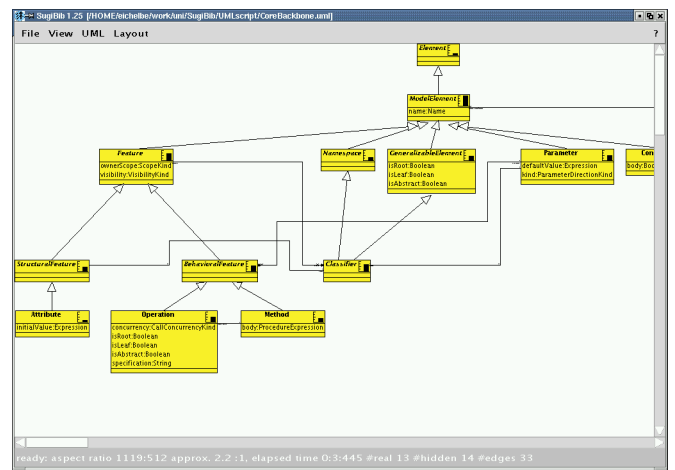


Figure 10: The UML core backbone class diagram with annotated complexity stereotypes.

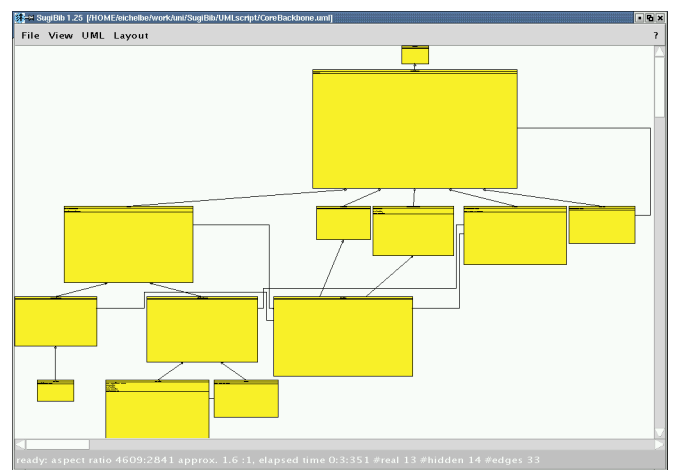


Figure 11: The nodes in the UML core backbone class diagram scaled according to the complexity of the classes.

In these figures, classes are coloured according to package containment and the shaded area within the package denotes the borderline between inter- and intra-package coupled classes. Both colouring options are non-UML features and used for demonstration purpose only. The metric to calculate the complexity of classes is a simple plug-in demonstration implementation based on the weighted number of edges.

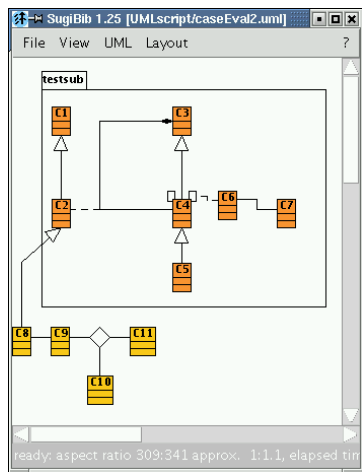


Figure 1: The example diagram drawn by *SugiBib*. Neither complexity nor coupling are respected. So far layout of annotated comments is not implemented.

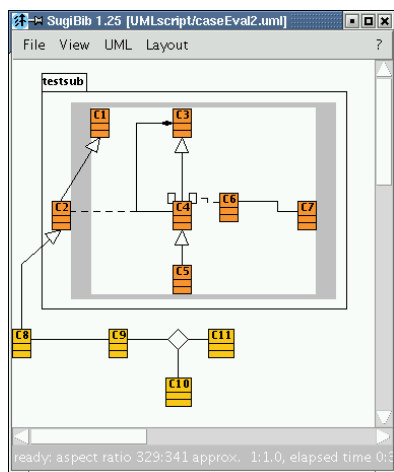


Figure 2: Coupling mode in order to visualize intra- and inner-package coupling by spatial distribution.

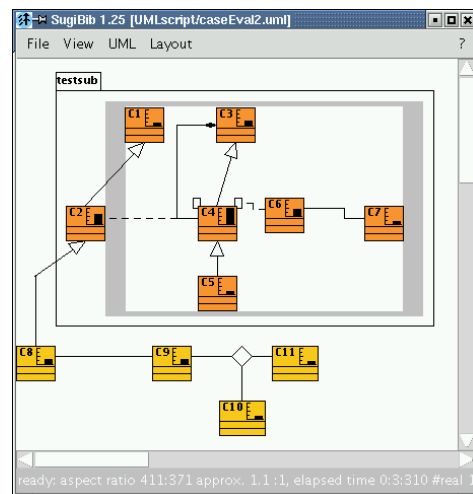


Figure 3: The example diagram with annotated complexity stereotypes and activated spatial distribution.

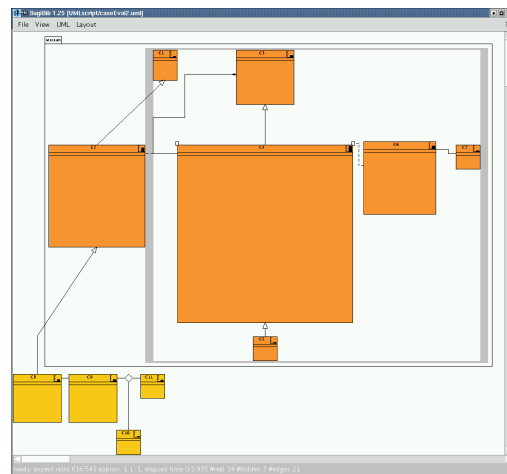


Figure 4: The example diagram with annotated complexity stereotypes, nodes scaled according to the complexity of the classes and activated spatial distribution.