On Evaluating the Layout of UML Class Diagrams for Program Comprehension

Dabo Sun and Kenny Wong
Department of Computing Science
University of Alberta, Canada
{dabo,kenw}@cs.ualberta.ca

Abstract

UML class diagrams are helpful for understanding the structure of a software system. Algorithms and tools have been developed to generate UML class diagrams automatically for program understanding purposes. However, many tools often ignore perceptual factors in the layout of these diagrams. Therefore, users still have to spend much time and effort rearranging boxes and lines to make the diagram understandable. This paper presents key criteria and guidelines for the effective layout of UML class diagrams from the perspective of perceptual theories. Two UML tools have been analyzed and evaluated to illustrate how the criteria can be applied to improve the readability of class diagrams.

Keywords: *UML class diagrams, perceptual theory, aesthetics, graph layout, UML modeling tools.*

1 Introduction

The Unified Modeling Language (UML) [4] provides a graphical representation of a software system which can be used for both forward engineering and reverse engineering. UML class diagrams describe the classes and their various relationships such as generalization (inheritance), association (aggregation and composition), and other dependencies. A class diagram illustrates the static structure of a software system. Therefore, it is helpful for both modeling and understanding a system.

According to Tilley [30], UML diagrams, as a form of graphical documentation, can help software engineers to understand large-scale systems, but their efficacy depends on three main factors: syntax and semantics of UML, spatial layout of the diagrams, and domain knowledge. In particular, the spatial layout of UML diagrams plays a crucial role in fostering program understanding. UML tools have been developed to generate class diagrams automatically from source code. However, few tools can generate "nice" layouts for the diagrams [12].

Much research has been done for layouts of class diagrams. Early work explored graph drawing algorithms and aesthetics [7, 17]. Some new approaches have been proposed for graph layout specifically in the UML class diagram domain. Eiglsperger et al. proposed an algorithm based on the topology-shape-metrics approach for automatic layout of class diagrams, and it deals well with class diagrams with little or no structural information [13]. Eichelberger introduced a layout algorithm according to a large number of aesthetic criteria of UML class diagrams [11]. Gutwenger et al. introduced an approach for visualizing UML class diagrams conforming to a balanced mixture of aesthetic criteria [16]. Dwyer presented a three-dimensional UML class diagram representation using the Force Directed algorithm [10].

Purchase and her colleagues analyzed graph layout aesthetics in UML diagrams, focusing on user preferences, and conducted empirical studies of human comprehension to validate those aesthetic criteria and rank their effect [23, 24, 25, 27]. They also compared various UML notations and suggested which notations are more understandable [26].

However, since there are so many criteria and some of them conflict with each other, software engineers and tool designers are often overwhelmed and confused on choosing the appropriate criteria. Also, we observed that little research has been done on evaluating the layout functionality in existing UML modeling tools. This paper analyzes and classifies key criteria and guidelines for effective layout of UML class diagrams from the perspective of perceptual theories. We illustrate how the criteria can be applied by analyzing and evaluating the class diagram layout in two commercial tools: Borland Together [1] and Rational Rose [2].

The rest of this paper is organized as follows. Section 2 reviews cognitive theory to help understand the class diagram layout criteria. Section 3 introduces and classifies those criteria. Section 4 evaluates two UML tools, and section 5 summarizes this paper.

2 Perceptual theory

In this section, we outline several fields in cognitive science, such as the theory of perception, perceptual organization, and perceptual segregation. The laws of perception explain how our visual system identifies objects and how we put together the basic features to observe a coherent, organized world of things and surfaces. From the software visualization perspective, the principles of perceptual organization provide the basic design rules to organize multiple artifacts so that users can group related information and segregate useful information easily and without ambiguity. Perceptual theory can also be applied to evaluate the effectiveness of software visualization tools [29, 32]. In particular, the perceptual principles will help us to discover what kind of layout of UML diagrams would be more readable and understandable.

2.1 Theories of perception

It is widely accepted that visualization helps people to understand information, but how the brain processes, transforms, and interprets visual stimuli is still unclear. Various theories of perception have been proposed [22].

Marr's theory [20]: Cognitive functions are filters that operate on raw visual stimuli and turn them into information.

Gibson's theory [14]: We shift our attention based on the structure of the environment and create a cognitive map to interact with the world.

Gestalt theory [8, 21]: We restructure our perception in ways that make it unified and coherent. This theory formulates the principles of organization which explain why some displays are better than others. Perceptual organization will be discussed in more detail in the next subsection.

Theory of notation [22]: Many researchers try to find good notations for visualization which include symbol systems to create graphs that convey design semantics.

2.2 Perceptual organization

Perceptual organization refers to how objects in the world that we perceive are located and relate with one another [9]. Research in perceptual organization studies how small elements are grouped into larger objects [15], which is important for tool designers to devise an understandable visualization of a software system. The following are several important principles of perceptual organization, most of which are from the Gestalt Laws [9, 15].

Prägnanz (Good Figure): Prägnanz is a German word which means "suggestive figure". Therefore, the law of prägnanz is also called the law of *good figure* or the law of *simplicity*. That is, the images are perceived in such a way that their structures are as simple as possible. The figural "goodness" is sensitive to the amount of information necessary to describe a figure. Usually, simpler and more stable figure is considered a "good" figure [28].

Similarity: Similar elements (e.g., common shape or color) appear to be grouped together.

Continuation: Points tend to belong together if they result in straight or smoothly curved lines when connected, and lines are grouped together in such a way as to follow the smoothest path.

Proximity (Nearness): Elements that are close to each other are grouped together.

Familiarity (Meaningfulness): Elements are more likely to be grouped together if the groups seem familiar or meaningful.

Element Connectedness: Elements that are physically connected are perceived as a unit. In Figure 1, we perceive three dumbbells rather than some pairs of dots. It is interesting to notice that the dots that are next to each other in adjoint dumbbells are actually closer and according to the law of proximity, they should be grouped together. However, in this case, the law of connectedness overpowers the law of proximity.

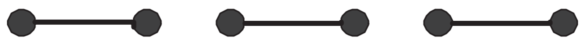

Figure 1. Element connectedness

2.3 Perceptual segregation

Compared to perceptual organization, research in perceptual segregation basically studies the problem of "figure-ground segregation" to indicate when objects are separated [15]. Objects are usually referred to as figures and the background is called the ground. The fundamental figure-ground theory was proposed by Gestalt psychologists who believed that the figure is more like what we are familiar with and it is more memorable than the ground. They also stated that the figure is perceived as being in front of the ground, and the contour of the figure and the ground more likely belongs to the figure. Experiments show that the following factors make an object more like a figure:

Symmetry: Symmetric areas are usually seen as a figure.

Orientation: Horizontal or vertical orientations have higher probabilities to be seen as a figure than other orientations.

Contours: Modern theories about figure-ground segregation discovered that contours (i.e. edges) play a very important role in figure-ground perception [19].

The perceptual theories unveil how humans perceive objects and interact with the environment, which may help us to understand why some kinds of UML diagram layouts are more readable than the others, so that we can better evaluate UML tool support.

3 UML class diagram layout criteria

According to Purchase et al. [24], not much research has been performed on the usability and understandability of graph drawing. Eichelberger also points out that the UML standard lacks a specification for the readability of the diagrams [12]. As a result, there are no common agreements on the criteria of readable and understandable layout of class diagrams. Our hypothesis is that criteria closely related to the laws of perception cannot be ignored, otherwise, our perceptual system might not be able to easily create correct "images" in our minds. Although semantics plays an important role in understanding UML diagrams, effective layout can substantially improve the quality of the diagram by increasing readability. In this section, we select key criteria from research results of graph drawing aesthetics [17, 23, 25, 33], various notation preferences for UML [26], UML layout aesthetics [11, 27], and UML style guidelines [5, 6] which can be directly related to perceptual principles, and we classify the criteria according to these principles. Figure 2 shows the classification of the selected criteria.

3.1 Law of good figure

C1: Join inheritance arcs. In [24], Purchase et al. state that inheritance arcs should be joined rather than separated lines. In Figure 3, (b) is preferable to (a), because notation (b) is more suggestive of a hierarchy than (a).

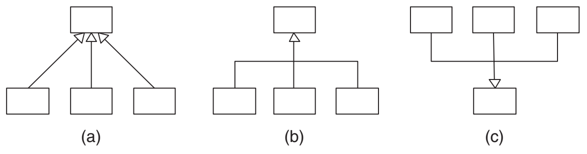

Figure 3. Notational variations of inheritance

C2: Represent association. The association name should be set beside the line, rather than in another association class linked to the line. In Figure 4, people prefer (a) to (b), because (b) has unnecessary complexity. Also, the association label should be set around the middle of the line [26].

C3: Be selective. Showing everything would not be helpful for understanding, especially for large systems.

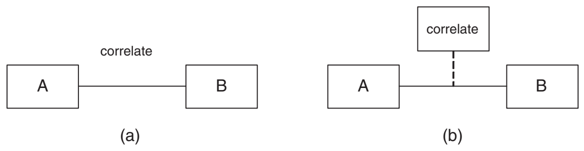

Figure 4. Notational variations of association

The large volume and dense information makes diagrams difficult to read. Selective information is simpler and easier to group [6].

3.2 Law of similarity

C4: Use colors. Most people are sensitive to different colors. Applying colors to different groups of entities and relationships will help the user to distinguish things because objects in the same color are more similar and easier to be distinguished from objects in other colors [5].

3.3 Law of continuation

C5: Minimize edge crossings and bends. The number of edge crossings and bends should be minimized to make edges more continuous and easier to follow [33].

3.4 Law of proximity

C6: Center parents or children. A parent node and its child nodes should be placed as close as possible because they are closely related. This is supported by the law of proximity [11].

C7: Reduce length of edges. Edges should not be too long or too short because they make grouping and separation hard. This is supported by the law of proximity [27].

3.5 Law of familiarity

C8: Position superclasses above subclasses. A superclass should be above its subclasses and the inheritance arrows should be upwards. In Figure 3, (a) and (b) are preferable to (c) because people usually are familiar with putting superior objects on top of other objects [26].

3.6 Law of connectedness

C9: Avoid overlapping. Overlapping should be avoided, i.e., nodes should not overlap other nodes or edges. There might be two kinds of overlapping: edges overlapping nodes, and nodes overlapping nodes. We consider edges overlapping edges as *edge crossings* as described in C5. In Figure 5, (a) is confusing because it is

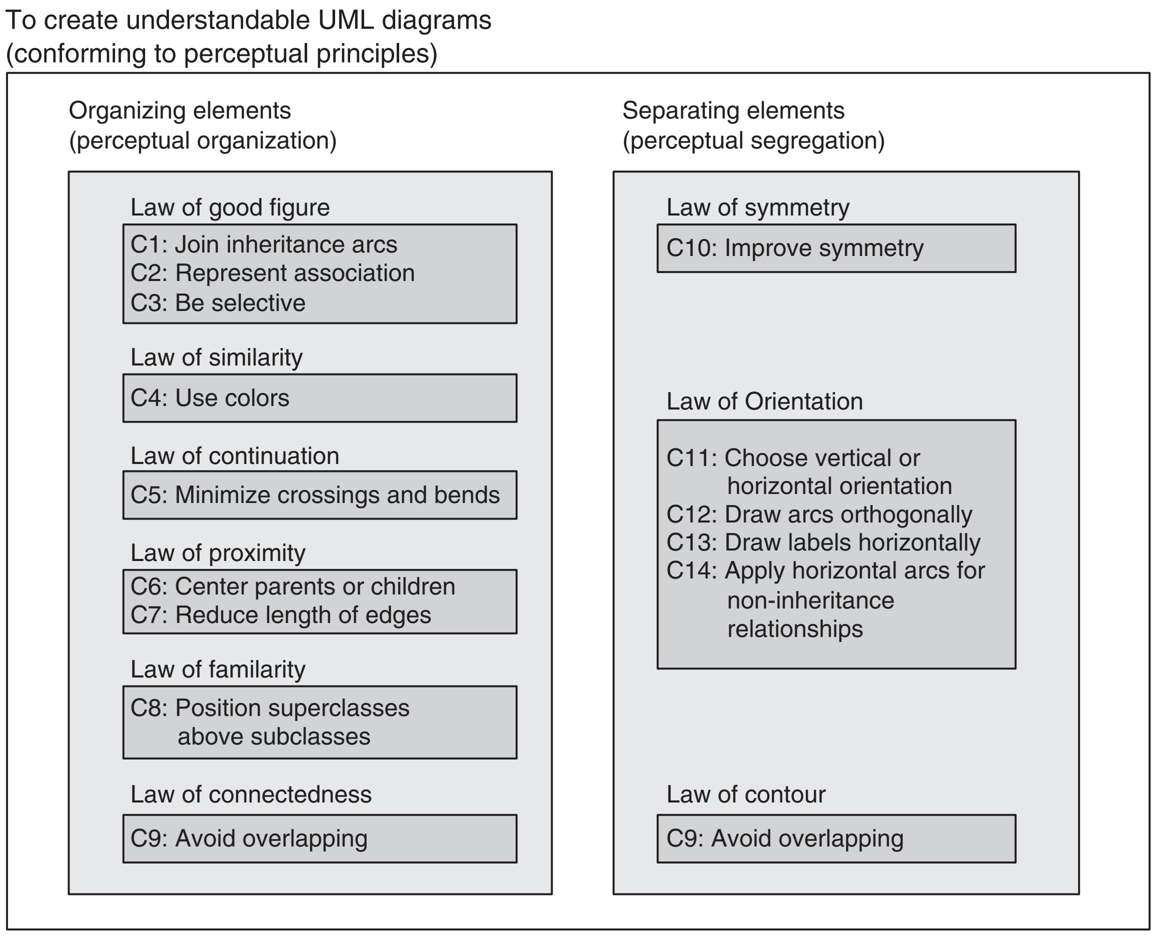
To create understandable UML diagrams
(conforming to perceptual principles)
Organizing elements
(perceptual organization)
Law of good figure
C1: Join inheritance arcs
C2: Represent association
C3: Be selective
Law of similarity
C4: Use colors
Law of continuation
C5: Minimize crossings and bends
Law of proximity
C6: Center parents or children
C7: Reduce length of edges
Law of familarity
C8: Position superclasses above subclasses
Law of connectedness
C9: Avoid overlapping
Separating elements
(perceptual segregation)
Law of symmetry
C10: Improve symmetry
Law of Orientation
C11: Choose vertical or horizontal orientation
C12: Draw arcs orthogonally
C13: Draw labels horizontally
C14: Apply horizontal arcs for non-inheritance relationships
Law of contour
C9: Avoid overlapping

Figure 2. UML Class Diagram Criteria

not clear whether box A is connected to box B or directly connected to box C. As a result, it could be perceived as either (b) following the law of connectedness, or (c) following the law of continuation [11]. Probably most people perceive (a) as (b), as the law of connectedness overpowers he law of continuation in this case.

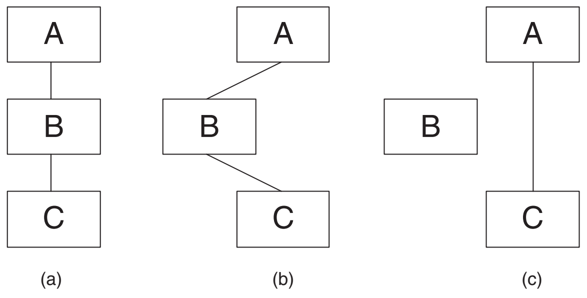

Figure 5. Effect of overlapping

3.7 Law of symmetry

C10: Improve symmetry. Symmetry of the diagram should be maximized, as symmetric areas are usually seen as a "good figure" [23].

3.8 Law of orientation

C11: Choose vertical or horizontal orientation. In general, most people prefer the orientation of left to right and top to bottom. Thus, the UML diagram should be drawn in vertical and horizonal directions [6].

C12: Draw arcs orthogonally. Nodes and edges should be arranged to an orthogonal grid, i.e., maximize the number of orthogonal edges [23].

C13: Draw labels horizontally. All the labels should be placed horizontally so that they can be read easily [24].

C14: Apply horizontal arcs for non-inheritance relationships. The common convention is to place relationships horizontally with the exception of inheritance [5].

3.9 Law of contour

C9: Avoid overlapping. As discussed before, nodes should not overlap other nodes partly because overlapping destroys the contours of objects, making them difficult to recognize.

3.10 Applying the layout criteria

Applying the layout criteria can improve the understandability of UML diagrams. However, we should notice that some criteria conflict with each other. For example, avoiding edge crossings and minimizing bends cannot both be achieved, and minimizing crossings and bends (C5) is very hard if only vertical or horizontal orientations are allowed (C11). Battista et al. state that it is very difficult to deal with all of the criteria algorithmically at the same time [7]. Thus tradeoffs should be well considered. Although the priority of implementing these criteria might differ from application to application, there is some general agreement on the relative importance of each criterion, especially in the UML diagram domain. Himsolt's work on evaluating graph layout algorithms indicated that *improving symmetry (C10), and minimizing crossings and bends (C5)* are very important [18]. Purchase investigated the prioritization of five graph drawing aesthetics, and proved that the priority (from strong to weak) is *minimizing crossings (C5), minimizing bends (C5), maximizing symmetry (C10), maximizing orthogonality (C12)*, and *maximizing the minimum angles between edges* [23]. Purchase studied the aesthetics prioritization for UML class diagrams and ranked eight aesthetics (from strong to weak): *minimizing crossings (C5), minimizing bends (C5), horizontal labels (C2), joining inheritance arcs (C1), narrower diagram, orthogonality (C12), no font variation*, and *directional indicators* [24].

Furthermore, the criteria do not take semantics of the depicted design into account. Purchase et al. point out that semantics such as domain knowledge should be considered for grouping in class diagram layout [27].

Our objective of classifying various layout criteria is to provide guidelines for designing and evaluating software visualization. In this paper, the classification of the UML layout criteria based on various perceptual principles is to provide guidelines for both UML users and tool designers. A user should try to conform to these criteria to produce clear and understandable diagrams. For UML tool designers, these criteria should be carefully considered to lay out UML diagrams (both automatically and manually).

4 Tool evaluation

In this section, we analyze and evaluate the class diagram layout functionality of two commercial UML tools, Rational Rose [2] and Borland Together [1], based on the layout criteria we discussed in the previous section. We consider two subject systems. The first one is a *Thermometer* application using a small Java framework that implements the Model/View/Controller (MVC) design pattern [31]. Figure 6 shows how Wampler draws the class diagrams of the framework and an application that uses it. The second software system is part of the JUnit testing framework [3]. We selected 28 classes from two packages: *junit.framework* and *junit.tests.framework* to check Rose and Together on a larger model.

4.1 Evaluating Wampler's diagram

Before we evaluate Rose and Together, it is interesting to evaluate the class diagram drawn by Wampler in Figure 6. Note that he separates the framework and application diagrams to better express the design semantics.

- **Good figure**: The diagrams join inheritance arcs (C1) and they use the concise association representation (C2), but it would be better if the association labels are put in the middle of the arcs, e.g. *update* between *WmvcView* and *WmvcModel*. The diagrams do not show all the detailed information of the classes, which conforms to selectivity (C3).
- **Similarity**: The diagrams do not use color to group classes (C4).
- **Continuation**: There are no line crossings or bends in the diagrams (C5).
- **Proximity**: In the diagrams, the superclass and its subclasses are placed closely (C6) and the length of the edges are very reasonable (C7).
- **Familiarity**: The superclass is placed above its subclasses (C8).
- **Connectedness**: Edges do not overlap any other nodes in the diagrams (C9).
- **Symmetry**: Both diagrams have good symmetry (C10).
- **Orientation**: All the edges are either vertical or horizontal (C11) and arcs are drawn orthogonally (C12). All the labels are horizontal (C13). However, there are quite a few non-inheritance relationships that use vertical arcs violating C14.
- **Contour**: No nodes overlap (C9).

The evaluation results show that Wampler's diagrams conform to most of the criteria, especially to those important criteria (C1, C2, C5, C9, and C10) as discussed in Subsection 3.10. This is consistent with our experience that students appreciate Wampler's diagrams when we present them in a senior software engineering course.

4.2 Tool overview

Rational Rose is a visual modeling tool for the design and analysis of object-oriented software systems. Rose

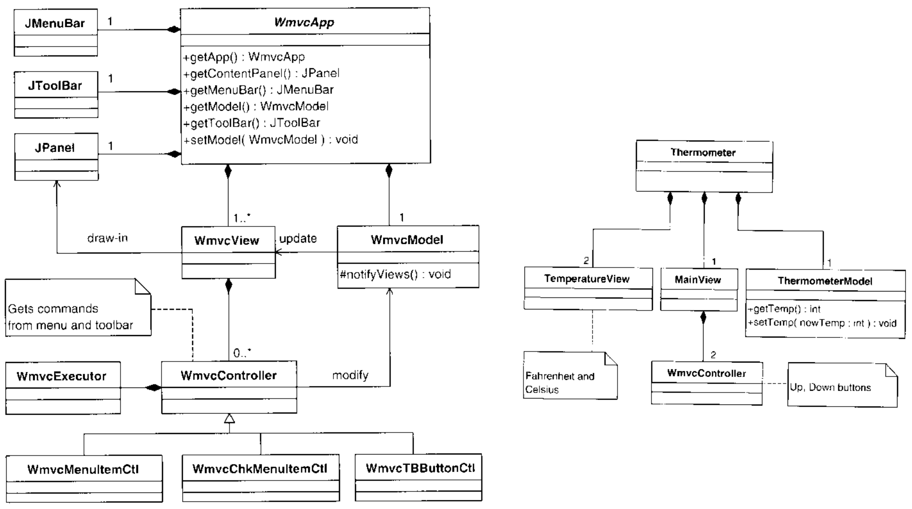

Figure 6. Class diagrams of Thermometer application, by Wampler

provides powerful functions for working with UML diagrams. It also supports reverse engineering various kinds of systems such as C++, Enterprise JavaBeans (EJBs), Java, etc. The version we evaluated is Rose Enterprise Edition, release 2003.06.13.402.000, on Windows 2000.

Together ControlCenter is an integrated environment for UML modeling and source code management. It also provides reverse engineering functions, which can synchronize class diagrams with the source code. The version we evaluated is Together ControlCenter, version 6.2, 2997, on Solaris 9.

Both Rose and Together can generate and lay out UML class diagrams automatically from source code.

4.3 Evaluating Rose

Figure 7 and 8 show the UML class diagrams of the MVC and JUnit frameworks generated by Rose. We try to evaluate whether the automatic layout of the class diagram generated by Rose conforms to the layout criteria. Also, we explore the support for manual adjustment of the diagrams.

• **Good figure**: Rose does not support joining multiple inheritance arcs (C1). It uses a clear and concise association representation, but labels are usually not placed at the middle of the line (C2). To support selectivity, Rose can render a subset of the classes, show, hide and delete elements, and filter relationships (C3).

• **Similarity**: Rose has a useful color filling function which allows the user to change the color of selected nodes or arcs (C4)

• **Continuation**: The generated diagram effectively reduces line crossings and bends. There is only one crossing and one bend in Figure 7 (C5).

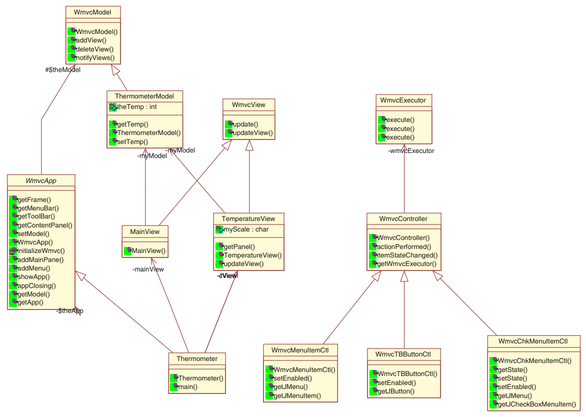

Figure 7. Class diagram of Thermometer application, by Rose

• **Proximity**: Some subclasses are a little too far from their superclasses. For example, class *Thermometer* should be closer to its superclass *WmvcApp* (C6). In general the lengths of edges are reasonable in Rose (C7).

• **Familiarity**: Rose has a rectilinear line style where inheritance arrows are mostly upwards, conforming to people's preference (C8).

• **Connectedness**: Labels often overlap with the lines and many lines overlap with nodes in large diagrams which makes the diagram harder to read, see Figure 8 (C9).

• **Symmetry**: Rose diagrams are laid out in balance (C10).

• **Orientation**: In the rectilinear line style (not shown), lines are either vertical or horizontal (C11) and edges are

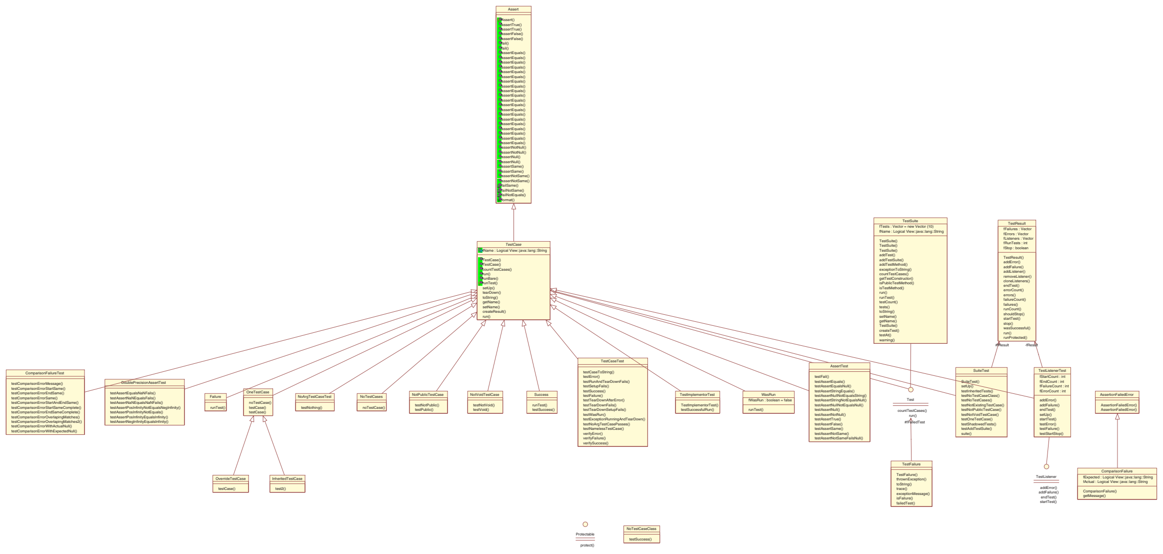

Figure 8. Class diagram of JUnit framework, by Rose

orthogonal (C12). In Rose, all labels are placed horizontally (C13), but many non-inheritance relationships are also vertical (C14).

• **Contour**: Nodes overlap in very large diagrams (examples not shown) (C9).

In general, Rose has good diagram layout and edge layout such as normal, rectilinear, oblique, and toggle. However, overlapping happens, especially in large diagrams. Rose only supports one UML presentation style. It would be helpful if it could support standard UML presentation style (e.g., use + and - to represent public and private attributes or methods) rather than using Rose's own icons and symbols. There is not much support for the user to manually adjust diagrams. Some problems are not hard to adjust; e.g., the user can drag and drop association labels to the middle of edges and remove overlapping. But the inability to join inheritance arcs is time consuming to fix. Also, it is tedious to arrange the non-inheritance relationships horizontally.

4.4 Evaluating Together

Together provides various diagram options and there are two main styles of link representation: direct and rectilinear as shown in Figure 9 and 10. The figures represent the UML class diagrams of the Thermometer application and JUnit framework generated by Together. Similarly, we examine Together's class diagram layout by the layout criteria.

• **Good figure**: Together nicely joins multiple inheritance arcs in the rectilinear link representation (C1). Together does not generate association labels (C2), but it has a pow-

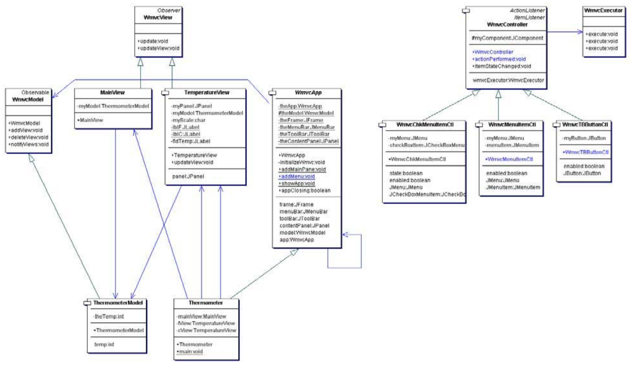

Figure 9. Class diagram of Thermometer application (direct links), by Together

erful "show/hide" function by using regular expressions to support selectivity (C3).

• **Similarity**: Together supports coloring selected nodes and arcs (C4).

• **Continuation**: Together avoids crossings and bends effectively (C5).

• **Proximity** Some subclasses are a bit too far from their superclasses. In Figure 9, class *ThermometerModel* is far from *WmvcModel* (C6). The lengths of edges are reasonable (C7)

• **Familiarity**: The inheritance direction is same as Rose, which is consistent with user preferences (C8).

• **Connectedness**: Together avoids overlapping between arcs and nodes, even in large diagrams (C9).

• **Symmetry**: Together generates reasonably symmetric

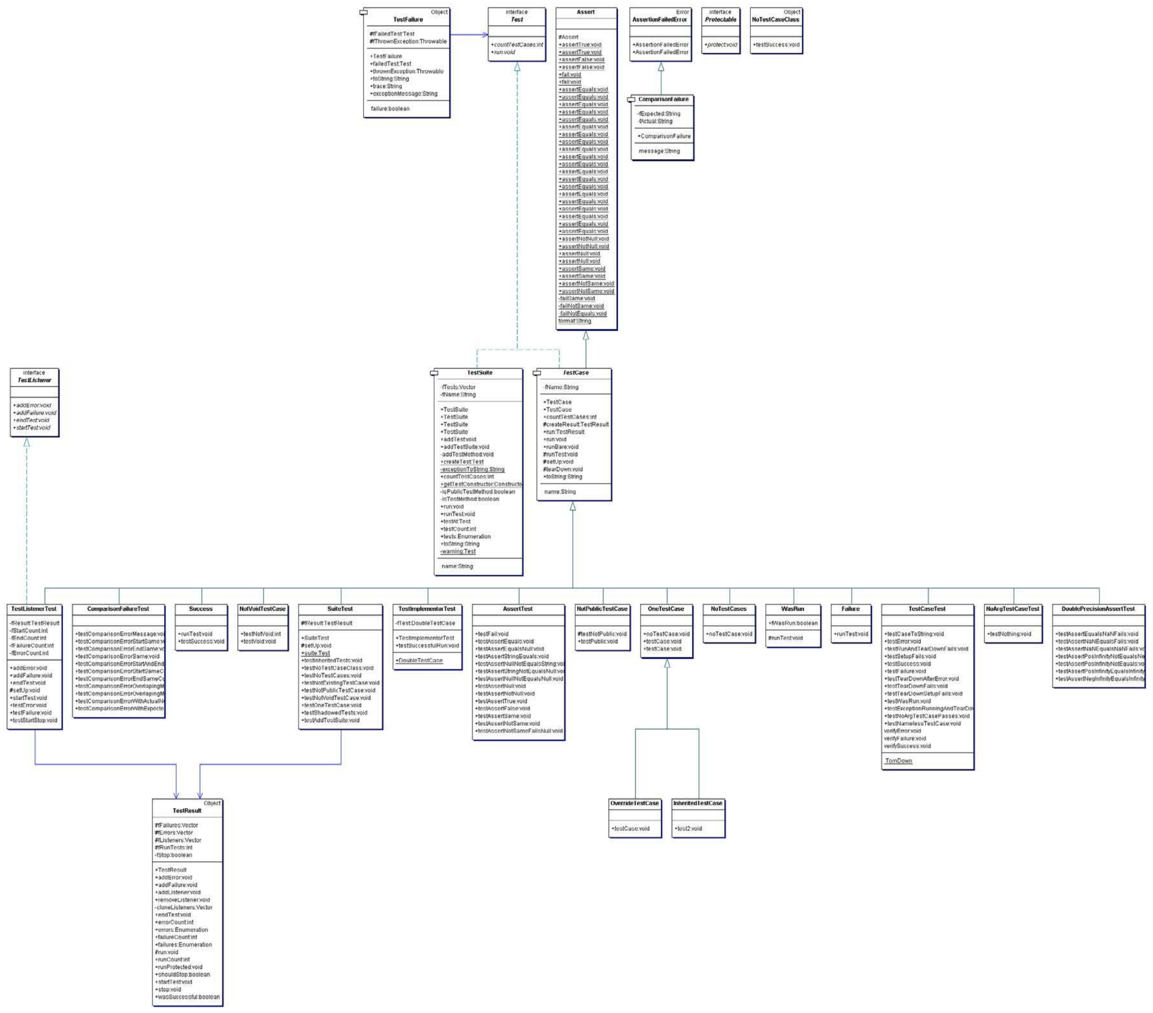

Figure 10. Class diagram of JUnit framework, by Together

subgraphs (C10).

• **Orientation**: Together orients nodes well (arcs are laid out either horizontally/vertically or nearly so) (C11). Together allows the user to choose between straight line and orthogonal representations (C12). Together does not generate association labels (C13). Together depicts some non-inheritance relationships horizontally, but others vertically (C14).

• **Contour** Nodes do not overlap, even in very large diagrams (more than 100 classes) (C8).

As listed above, Together provides various functions to adjust the layout of class diagrams both automatically and manually, such as supporting line styles (straight or orthogonal), providing regular expression match on what to show/hide, etc. Another useful feature is that Together keeps consistency between the source code and the class diagrams. Together separates different relationships by using different kinds and colors of arcs: by default, black edges represent inheritance, blue lines indicate association, and green dashed lines depict interface implementation. There is one defect found in the Thermometer class diagram: in Figure 9, we can find two association lines between class *Thermometer* and *TemperatureView*. This violates the law of good figure and will decrease the readability of the diagram if there are too many redundant arcs. It would be better to use multiplicity labels on associations to avoid multiple association arcs.

The layout defects found in Together are usually easy to fix manually: the subclasses can be moved nearer to their superclasses by dragging. The symmetry can also be increased by moving some nodes and arcs.

Table 1 summarizes the evaluation of Rose and Together.

4.5 Discussions

By evaluating the layout of class diagrams in Rose and Together, we demonstrate that criteria based on perceptual

Table 1. Evaluation summary

Criteria	Wampler	Rose	Together
C1: Joining inheritance arcs	Yes	Manually	Automatically
C2: Association representation	Good	Labels not centered	No label generated
C3: Selectivity	Good	Selective generation & filtering	Powerful show/hide
C4: Using colors	N/A	Support	Support
C5: Minimizing crossings and bends	Yes	Yes	Yes
C6: Centering parents or children	Yes	Not all	Not all
C7: Reducing length of edges	Yes	Reasonable	Some lines could be shorter
C8: Inheritance direction	Appropriate	Appropriate	Appropriate
C9: Avoiding overlapping	Yes	Happens in large diagrams	No overlapping
C10: Symmetry	Good	Well arranged	Reasonable
C11: Orientation	Good	Various choices	Conforms to user preference
C12: Orthogonality	Yes	Support	Support
C13: Horizontal labels	Yes	Yes	Draw manually
C14: Horizontal relationships	Partially	Rarely	Partially

principles provide valuable guidelines to aid users and tool designers to produce clear and readable class diagrams. Rose and Together conform to most of the criteria that we categorized, but there are still features to be improved. Since the overall results show that Together can generate a better layout in general, we discuss how layout in Together can be further improved. Our suggestions should also be applicable to Rose.

- Since it is a hard task to take all the layout criteria into account, it would be helpful to allow the user to adjust the parameters of some layout algorithms so that the layout can be rearranged automatically to the user's satisfaction. For example, the user should be able to balance crossings, bends, orthogonality, and vertical and horizontal arcs.
- Certain constraints should be applied in manual adjustment. For example, suppose the user needs to move the position of a subclass to satisfy some criteria, it would be useful if other sibling subclasses would move automatically to open up needed space or close up any gaps accordingly.
- The Gestalt theory states that the whole is more than the sum of its parts [9, 28]. Therefore, it is useful to add a separate overview window showing the whole diagram, which will help users to locate themselves within the graph and reduce perceptual overhead.
- The tool should support the ability to repeat past manual layout adjustments to save the user from tedious work and to reinforce past perceptions.

As UML (tool) users, it is also important to apply the UML layout criteria as guidelines to produce more understandable diagrams for both design, maintenance, and communication purposes. Learning how to use UML tools effectively, especially making the best use of automatic, semi-automatic, and manual layout support will save time and effort.

5 Conclusions

UML class diagrams are useful for both forward engineering and reverse engineering as they describe the static structure of a software system. However, how to evaluate the effectiveness of the layout of class diagrams has been a long term issue. Generally speaking, software visualization tools need more reliable evaluation criteria such as perceptual principles rather than only following intuitions of the tool designers.

In this paper, we present a classification of graph layout criteria, especially for UML class diagrams according to the laws of perceptual theories. These selected criteria are close to human perceptual principles, so they cannot be ignored. The evaluation of two commercial UML modeling tools, Rose and Together, not only reveal the advantages and limitation of the tools, but also convince us that perceptual factors are important for devising diagram design guidelines and evaluating software visualization tools in general. Tradeoffs of applying layout criteria and suggestions of tool improvement are also presented.

For future work, we would like to apply perceptual theories to other kinds of UML diagrams such as the sequence diagram. Meanwhile, we would like to investigate how the progress in modern human perception can be applied in program comprehension research. Our objective is to apply these theories to building a set of tests for effective software visualization.

References

[1] Borland Together. http://www.borland.com/together/.

[2] IBM Rational Software. http://www-306.ibm.com/software/rational/.

[3] JUnit web site. http://www.junit.org/index.htm.

[4] UML resource page. http://www.uml.org/.

[5] S. W. Ambler. Modeling Style Guidelines. http://www.agilemodeling.com/style/.

[6] S. W. Ambler. *The Elements of UML Style*. Cambridge University Press, 2003.

[7] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[8] J. G. Benjafield. *Cognition*. Prentice Hall, 1992.

[9] K. R. Boff, L. Kaufman, and J. P. Thomas. *Handbook of Perception and Human Performance. Volume II. Cognitive Processes and Performance*. Wiley-Interscience Publication, 1986.

[10] T. Dwyer. Three dimensional UML using force directed layout. In *CRPITS '01: Australian Symposium on Information Visualisation*, pages 77–85. Australian Computer Society, Inc., 2001.

[11] H. Eichelberger. Aesthetics of class diagrams. In *VISSOFT '02: Proceedings of the 1st International Workshop on Visualizing Software for Understanding and Analysis*, page 23. IEEE Computer Society, 2002.

[12] H. Eichelberger. Nice class diagrams admit good design? In *SoftVis '03: Proceedings of the 2003 ACM Symposium on Software Visualization*, pages 159–168. ACM Press, 2003.

[13] M. Eiglsperger, M. Kaufmann, and M. Siebenhaller. A topology-shape-metrics approach for the automatic layout of UML class diagrams. In *SoftVis '03: Proceedings of the 2003 ACM Symposium on Software Visualization*, pages 189–198. ACM Press, 2003.

[14] J. J. Gibson. *The Ecological Approach to Visual Perception*. Boston: Houghton Mifflin, 1979.

[15] B. E. Goldstein. *Sensation and Perception*. Wadsworth-Thomson Learning, 6th edition, 2002.

[16] C. Gutwenger, M. Jünger, K. Klein, J. Kupke, S. Leipert, and P. Mutzel. A new approach for visualizing UML class diagrams. In *SoftVis '03: Proceedings of the 2003 ACM Symposium on Software Visualization*, pages 179–188. ACM Press, 2003.

[17] I. Herman, G. Melanon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[18] M. Himsolt. Comparing and evaluating layout algorithms within GraphEd. *Journal of Visual Language and Computing*, 6(3):255–273, 1995.

[19] R. Kimchi, M. Behrmann, and C. R. Olson. *Perceptual Organization in Vision: Behavioral and Neural Perspectives*. Lawrence Erlbaum, 2003.

[20] D. Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W H Freeman, 1982.

[21] P. Moore and C. Fitz. Gestalt theory and instructional design. *Journal of Technical Writing and Communication*, 23(2):137–157, 1993.

[22] M. Petre, A. Blackwell, and T. Green. Cognitive questions in software visualisation. *Invited chapter in: Stasko, J., Domingue, J., Brown, M., and Price, B. (Eds.), Software Visualization: Programming as a Multimedia Experience. MIT Press.*, pages 453–480, 1998.

[23] H. C. Purchase. Which aesthetic has the greatest effect on human understanding? In *GD '97: Proceedings of the 5th International Symposium on Graph Drawing*, pages 248–261. Springer-Verlag, 1997.

[24] H. C. Purchase, J.-A. Allder, and D. A. Carrington. User preference of graph layout aesthetics: A UML study. In *GD '00: Proceedings of the 8th International Symposium on Graph Drawing*, pages 5–18. Springer-Verlag, 2001.

[25] H. C. Purchase, R. F. Cohen, and M. James. Validating graph drawing aesthetics. In *GD '95: Proceedings of the Symposium on Graph Drawing*, pages 435–446. Springer-Verlag, 1996.

[26] H. C. Purchase, L. Colpoys, M. McGill, D. Carrington, and C. Britton. UML class diagram syntax: an empirical study of comprehension. In *CRPITS '01: Australian Symposium on Information Visualisation*, pages 113–120. Australian Computer Society, Inc., 2001.

[27] H. C. Purchase, M. McGill, L. Colpoys, and D. Carrington. Graph drawing aesthetics and the comprehension of UML class diagrams: an empirical study. In *CRPITS '01: Australian Symposium on Information Visualisation*, pages 129–137. Australian Computer Society, Inc., 2001.

[28] H. R. Schiffman. *Sensation and Perception: An Integrated Approach*. John Wiley & Sons, 5th edition, 2001.

[29] D. Sun and K. Wong. On understanding software tool adoption using perceptual theories. In *ACSE '04: Proceedings of the Fourth International Workshop on Adoption-Centric Software Engineering*, pages 51–55. Institution of Electrical Engineers, 2004.

[30] S. Tilley and S. Huang. A qualitative assessment of the efficacy of UML diagrams as a form of graphical documentation in aiding program understanding. In *SIGDOC '03: Proceedings of the 21st Annual International Conference on Documentation*, pages 184–191. ACM Press, 2003.

[31] B. E. Wampler. *The Essence of Object-Oriented Programming with Java and UML*. Addison Wesley, 2002. http://www.objectcentral.com/oobook/book-code.zip.

[32] C. Ware. *Information Visualization: Perception for Design*. Morgan Kaufmann Publishers, 2000.

[33] C. Ware, H. Purchase, L. Colpoys, and M. McGill. Cognitive measurements of graph aesthetics. *Information Visualization*, 1(2):103–110, 2002.