Introducing Aesthetics to Software Visualization

David Baum
University of Leipzig
Grimmaische Strasse 12
04109, Leipzig, Germany
david.baum@uni-leipzig.de

Abstract

In software visualization, but also in information visualization in general, there is a great need for evaluation of visualization metaphors. To reduce the amount of empirical studies a computational approach has been applied successfully, e.g., to graph visualization. It is based on measurable aesthetic heuristics that are used to estimate the human perception and the processing of visualizations. This paper lays a foundation for adopting this approach to any field of information visualization by providing a method, the repertory grid technique, to identify aesthetics that are measurable, metaphor-specific, and relevant to the user in a structured and repeatable way. We identified 25 unique aesthetics and revealed that the visual appearance of the investigated visualizations is mainly influenced by the package structure whereby methods are underrepresented. These findings were used to improve existing visualizations.

Keywords
Aesthetics, Repertory Grid Technique, Software Visualization

1 INTRODUCTION

The benefit of a specific visualization depends upon many factors, such as addressed end-user (e.g., developer, project manager, or client), method of representation and its use case. Therefore it is not possible to prove the superiority of one kind of visualization over another one in general, regardless of dimensionality, method of representation and other influencing factors. Every method of representation, usually called metaphor, has to be empirically evaluated on its own [Kos03]. Unfortunately, only a minority of approaches in software visualization has been empirically validated yet [WLR11a, TC08]. This disproportion is caused by the time effort as well as the expense that is necessary for the implementation of empirical studies such as controlled experiments [LBI$^+$12]. Due to many influencing factors a single experiment is often insufficient and therefore a series of experiments is required [IW03]. In practice the resulting overall costs are a problem, because the empirical studies have to be repeated with every slight modification of the metaphor or for example the layout algorithm. Further a controlled experiment can only show if the use of a metaphor is more effective or more efficient than the used object of comparison. Unfortunately, such experiments provide no explanation for measured values and thus no indication on how to improve it.

For these reasons we want to take a different approach for evaluating software visualizations that has the potential to reduce the required number of empirical studies. It is based on a cognitive model which explains the perception and processing of visualizations [Nor04]. This process is influenced by aesthetic heuristics and they in turn depend on the underlying data [BRSG07]. Once we know the relations between these elements we are able to evaluate visualizations by computation instead of empiricism. However this goal can only be achieved in several stages, and this paper is the first one. This is to identify user-relevant aesthetics. Therefore this paper focuses on the following contributions:

(1) We show, how the repertory grid technique can be applied to software visualization for identifying metaphor-specific aesthetics systematically.

(2) We provide a classification scheme to evaluate the influence of the depicted software entities and to draw comparisons between different metaphors.

(3) We present a study that identified aesthetics for the recursive disk metaphor and revealed undesired side effects.

2 RELATED WORK

The concept of aesthetics is widely used in the field of graph visualization especially for evaluating graph layout algorithms [Hua14, BRSG07]. Basic research was done by Purchase [Pur97, PMCC01, PAC02, PCA02] which led to increased research activity of several authors [BRSG07, WPCM02]. All in all at least 18 graph aesthetics were proposed by different researchers although only a few of them were evaluated empirically [BRSG07]. Due to its successful use the concept of aesthetics has already been transferred to other domains such as UML class and use case diagrams [PMCC01]. However none of these approaches includes a method for identifying aesthetics that are relevant to the user. All of the proposed aesthetics, e.g., in graph visualization have been found without user involvement. This might be a reason why only a minority of the proposed aesthetics has a significant effect with respect to the user performance.

We are not aware of any application in the field of software visualization. In addition no other approach, which has the goal to evaluate visualizations by computation does exist so far.

3 SOFTWARE VISUALIZATION

In software visualization the structure, behavior and/or evolution of software systems are visualized to facilitate a better understanding of abstract and complex software artifacts. Over the last years several visualization metaphors were proposed such as the recursive disk metaphor [MZ15] and the city metaphor [WLR11b]. They differ among other properties in the depicted information and in the used glyphs for representing software entities. *“A glyph is a small visual object that can be used independently and constructively to depict attributes of a data record or the composition of a set of data records“* [BKC+13]. Glyphs use *visual channels* such as shape, color and size to depict information [BKC+13] and they can contain other glyphs as well [War02].

Within this paper we want to focus on the recursive disk metaphor. It uses a glyph-based approach to visualize the structure of software systems. Each software entity, i.e., attributes, methods, classes, and packages is mapped to a different glyph [MZ15]. An attribute is represented by a yellow ring segment. Method glyphs are blue ring segments and their size corresponds roughly to the methods number of statements (NOS). Classes and packages are depicted by purple and gray rings respectively any they may contain several attribute, method and class glyphs. With these basic glyphs even large structures can be visualized. Figure 1 shows a package that contains some exemplary classes. For a more complex real world example see Figure 2. It visualizes the structure of *JUnit 4*,

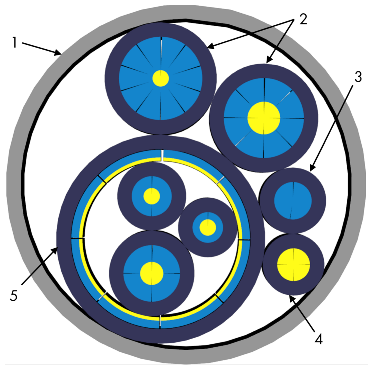

Figure 1: Basic glyphs and relations with the recursive disk metaphor: 1 - Package with five classes, 2 - General classes with altogether eighteen methods and five attributes, 3 - Method class with two methods, 4 - Data class with four attributes, 5 - Class with eight methods, eight attributes, and three inner classes [MZ15].

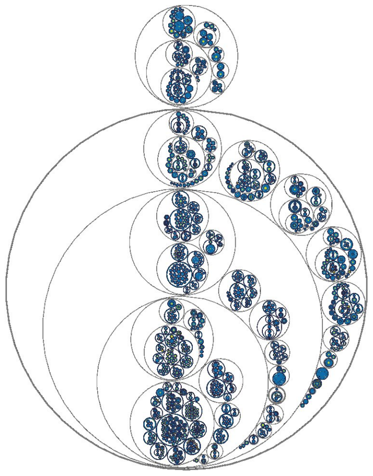

Figure 2: The structure of JUnit 4.12 visualized with the recursive disk metaphor.

one of the most frequent used test frameworks for Java [JUn15].

4 AESTHETICS

Since aesthetics originates in graph visualization we want to introduce them using a corresponding example (c.f. Figure 3). All three node-link diagrams visualize the same graph and only differ in their layout algorithm. Readability as well as understandability is decreased from left to right. This is caused by their different appearance and not only detectable through empirical experiments but also directly measurable. For example the diagrams differ among other things in their number of edge crossings and in their degree of symmetry. which are both basic aesthetics. Studies haven shown, that reducing edge crossings and increasing symmetry will lead to a faster perception and a better understanding of graphs. If a further layout algorithm will be evaluated with respect to understandability and readability no further empirical studies are necessary. By taking the number of edge crossings and the degree of symmetry into account we can predict the mentioned properties of the graph.

Since a unified definition for aesthetics is missing in the literature, we define them as follows: *Aesthetics are visual properties of a visualization that are observable for human readers as well as directly measurable.* By this definition abstract attributes such as complexity are not considered as aesthetics. Indeed the complexity of a visualization can be measured as well, but only after it was operationalized.

Aesthetics can be used in two different ways: On the one hand certain aesthetics can be optimized, so that the resulting graph is easier to understand for human readers [BRSG07]. This can be measured by the time needed by a user to solve different tasks and the number of errors he made while doing so [Hua14]. On the other hand due to the different appearance of the three node-link diagrams a user would expect the graph on the right side to be more complex than the graph on the left side [PAC02]. It is a basic purpose of a model that assumptions about its original are made. If the visualizations of two systems differ clearly in one point the beholder assumes that this is caused by the underlying data. But in some cases these differences are merely undesired side effects without any reasonable meaning. The resulting assumptions would be at best useless or even wrong as in the mentioned example since all graphs share the same degree of complexity.

However, the use of aesthetics in software visualization, especially when the third dimension is used, is hindered by the lack of methods to identify new aesthetics in a structured and repeatable way. If 18 aesthetics would be proposed for every visualization metaphor, the need for empirical studies is not reduced but increased. This is the key problem that has to be solved in order to apply aesthetics efficiently to a wide field of applications. Therefore we used the repertory grid technique to identify metaphor-specific aesthetics as described in the following section.

5 RESEARCH DESIGN

For the identification of user-relevant and metaphor-specific aesthetics empirical studies are still required. Therefore we investigated the perception of different

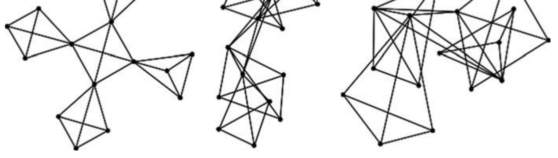
Figure 3: Three node-link diagrams of the same graph [Hua14]

software systems focusing on the following research questions:

(1) Which aesthetics affect the perception of a software visualization based on the recursive disk metaphor?

(2) How strong is the influence of different software entities on the perception of a software visualization?

Hence the study is based on an exploratory research design. Since the method for identifying relevant aesthetics is the main contribution of this publication the methodology of the applied repertory grid method is described in general as well as our implementation below in detail.

5.1 The Repertory Grid Technique

The repertory grid technique is an empirical research method that originates in psychology. It was already adopted to other research fields such as marketing and software engineering, e.g., to analyze soft skills of software engineers [KSB06], to identify aspects in requirements engineering and risks in software development projects [EMM09] and to evaluate the user experience [VLR$^+$10]. As you will see below neither the method in general nor our implementation are specific to software visualization but also applicable to any kind of information visualization.

The theoretical foundation of the repertory grid technique is the “theory of personal constructs“ developed by G. Kelly [Kel55]. Its basic assumption is that everybody constructs its own reality through his individual perception of the world. To describe and evaluate elements of this world as well as to distinguish between them, people use *bipolar constructs*. A construct is defined as “a way in which two or more things are alike and thereby different from a third or more things“ [Kel55, p. 61]. These construct consist of a construct pole (e.g. “reliable“), a contrast pole (e.g. “unreliable”) and a construct continuum in between, i.e., different degrees of reliability. For example, regarding user experience a user categorizes software *A* as fast, reliable and ugly and software *B* as slow, unreliable and handsome. Thereby he used three bipolar constructs, “fast – slow“, “reliable – unreliable“ and “ugly – handsome“ which means these are the relevant attributes for him in which the two systems differ. The repertory grid method is an approach to make these constructs explicit and visible. Within the terminology of the repertory grid technique software *A* and *B* are referred to as *elements*, that are described through *constructs* [Fra04, p. 15].

A repertory grid interview consists of several steps. For each step multiple design decisions have to be made, e.g., about how elements and constructs are selected. In the following we do not discuss all possibilities, but the research design that corresponds to our research questions, hence variants such as constructs provided by the researcher are not reasonable for exploratory research. In the first step the investigated elements have to be chosen through the participant or the researcher. In our case visualizations of software systems were used as elements. Second, these elements are combined into triads. This can be done randomly or by creating combinations of particular interest. Then the triads are presented successively to the participant, i.e., the participant sees exactly three elements at the same time without access to the other elements.

For each triad presented, the participant has to answer the following question: “How are any two of these alike in some way?“, complemented by “What is the opposite of that?“ [Fra04, p. 29]. The answer to the first question is the construct pole and the answer to the second is the contrast pole. It might happen that these abstract constructs lead to further constructs if they are investigated in depth. It is not uncommon that a construct implies another construct. They only vary in their level of abstraction. The process of using a construct to attain a construct on a different level of abstraction is called *laddering* and is a common part of the repertory grid interview [Fra04, p. 39]. This can be done by asking “Why appears this software more reliable to you?“. The whole procedure is repeated by using other randomly selected triads as long as the participant creates new constructs to distinguish between the elements. During the interview the participants have no access to any constructs they used before. It is crucial that the interviewer understands what exactly the participant describes with a construct. For this reason informal communication between both persons is a regular and intended part of the repertory grid technique.

Once no new constructs can be elicited, the participant has to assess all elements in consideration of all mentioned constructs. For twelve elements and, e.g., 15 constructs the participant has to make 180 decisions. Since a repertory grid interview is already exhausting for the participant often a 5-point rating scale is used to simplify the process [EMM09]. During this phase the participant has access to all 12 visualizations at any time.

To sum up, the participants assess specified elements on constructs they create. As the repertory grid technique demands high standards of the interviewer and of the research design, a pretest is advisable to detect unexpected difficulties.

Once the interviews are conducted successfully the resulting grids can be analyzed in multiple ways. For a qualitative interpretation the results are visualized as a two-dimensional space based on a principal component analysis (PCA) [Fra04, p. 86]. The distances between elements in this space represents their similarity. Small distances represent a high degree of similarity, whereas large distances indicate a low degree of similarity.

Given that most constructs are mentioned only by one or two participants a quantitative analysis of the constructs is not directly possible. Therefore it is necessary to abstract from concrete constructs. This is done by assigning all constructs to given categories [Fra04, p. 49], e.g., “system behavior“ and “appearance“ in the case of the example mentioned before. This way it is possible to draw comparisons between grids of different participants, even if these grids are based on different elements and the participants used different constructs.

5.2 Study Design

The repertory grid technique is a very flexible method, thus it can be applied in many ways. In the following we describe our conducted research design and explain the design decisions we made.

Twelve visualizations with the recursive disk metaphor were used as elements. They were preselected based on static code metrics such as NOS and their number of packages, classes, methods and attributes. Because the recursive disk metaphor visualizes the structure of a system the metrics were limited to structural aspects as well. Except for NOS these structural entities are directly represented in the visualization by glyphs [MZ15]. The NOS were included since they are used to ensure that small, medium and large systems are used in the study. Table 1 gives an overview over the chosen elements. In addition the visualizations can be accessed online [1]. The triads were created randomly and individually for each interview.

The InstantPlayer which is part of the InstantReality platform [fCGRI15] was used to show the visualizations. The participants could freely navigate within the model, i.e., changing the viewpoint along the x-, y- and z-axis which includes zooming and changing the field of view. Each visualization was presented on a separate but similar screen.

We conducted eight repertory grid interviews with four male and four female participants. Their age varies between 19 and 52 years. None of them has seen a recursive disk visualization before. To ensure the participants focus on the visual differences and not on the underlying structural differences of the software, we have not explained the meaning of the shown visualizations to them. The visualizations were only referred to as “model 1“ etc., so they did not even know that the study was about software visualization. We just determined how the participant should call the glyphs to guarantee interviewer and participant talk about the same. This was necessary because in the pretest we detected

[1] `https://github.com/naraesk/aesthetics`

that every participant uses different terms for naming the same glyph. We chose easy-to-understand names, therefore the wording differs from the regular vocabulary of the recursive disk metaphor: The glyphs were called “gray rings“, “purple disks”, “blue segments“ and “yellow segments“. Furthermore the terms “outer gray rings“ and “inner gray rings“ were introduced. The first one describes the root package disks and the second term names the remaining package disks. In addition “element“ was defined to describe any glyph regardless of its shape or color. Besides these small adjustments, we could apply repertory grid, as described in section 5.1, without modification.

The next step after conducting the interviews is to categorize the identified constructs. The means of construct categorization were proposed by Landfield [Lan71] and especially for psychological grids there exist different categorization schemes. Since we applied the repertory grid technique to a new field, existing categorization schemes are insufficient. Therefore we used a simple scheme for object-oriented software systems based on the entities system, package, class, method, and attribute. Every construct is now mapped to exactly one category depending on which entity is the object of comparison. We explain the categorization process using the construct “many purple disks – few purple disks“ which refers to the number of classes. Two packages can differ in their number of classes, for a class itself this is not possible. Therefore the category of the construct is *package*. Of course systems can differ in their number of classes as well, but only the least entity is used. Otherwise *system* would be used for every single construct.

6 RESULTS AND DISCUSSION

The goal of this study was to reveal aesthetics for the recursive disk metaphor. Therefore we refrain from doing a large qualitative statistical interpretation of the individual constructs, because once the aesthetics were extracted it is more meaningful to examine them in a separate study. In this spirit the described study can be seen as a data collection to prepare a more comprehensive quantitative study. Still, we present an overview of the identified constructs. The complete raw data is provided online as well [1].

The study revealed 53 unique constructs that were mentioned by the participants although not all of them are aesthetics since not all of them can be directly measured. Table 2 shows all mentioned constructs and their frequency, which we will discuss briefly. To determine the frequency we had to decide whether two constructs have the same meaning or not, which is not trivial. The measurable constructs tend to be precise and for the participants they were easy to explain by pointing on a concrete example on the screen. A construct such as “number of gray disks“ it is easy to understand and therefore we were able to count these constructs. In case of less precise terms this procedure is problematic, because participants often use a slightly different wording. Therefore we only counted identical constructs. Some of the rarely used constructs may have the same meaning, but still appear as two independent entries, e.g, “chaotic – logical“ (#10) and “regular – irregular“ (#17) could be considered equal. On the one hand Table 2 shows that eleven identified constructs were used by at least half of the participants. On the other hand 26 constructs were mentioned only once. Actually the aesthetics that were mentioned more often were used more often within an interview too. However, for example the construct “dynamic - static“ was mentioned only by one participant nevertheless it seems to be important to him since he used it several times. That just shows that it is not reasonable to focus only on the frequent aesthetics but rather one must take all of them into account. Due to the fact that over 50% immeasurable constructs were used by the participants it seems that in some cases the laddering was not done intensively enough. Nevertheless, this only leads to less data but not false data and does not affect the identified aesthetics.

6.1 Categorization

The categorization revealed that the recursive disk metaphor tends to emphasize packages. Three of the four most frequent aesthetics refer to package disks as objects of comparisons, 48% overall. Further the package structure affects 60% of the identified aesthetics. This means that the visual appearance as well as the assumptions about the underlying data is primarily influenced by how classes are grouped into packages. From our point of view this is neither intended nor desirable. In program understanding as well as in program analysis methods are the center of interest. For instance most pattern as well as anti-pattern are detected on method level [GHJV93, Lan06]. Whereas not a single aesthetic was identified, that refers to methods. Merely three aesthetics use classes as objects of comparisons. In case an aesthetic depends on the class structure it only focuses on the absence of methods (#51) or inner classes (#42). Although the metaphor provides more information such as the size of individual methods, the number of methods of a class and the proportions of these glyphs. But none of these aspects of the visualization were recognized by the participants.

The last column of Table 2 shows which influencing factors an aesthetic has, whereby *everything* means, that this aesthetic is influenced by the number of the different software entities, NOS and the layout algorithm. This is, e.g., the case for the density of a package. The density specifies how much of the area of a disk is filled. By a density of 100% there is no empty space inside

	Packages	Classes	Methods	Attributes	Statements
android_packages_apps_Phone	3	262	1,426	1,394	15,113
android_packages_apps_Settings	40	1,351	5,516	6,140	51,489
apache Storm	71	3,203	2,129	7,417	35,295
ChatSecureAndroid	32	620	2,257	3,334	22,371
cw-omnibus	339	1,224	2,545	7,212	36,696
disruptor	16	271	696	1,100	4,274
FreeFlow	17	73	242	467	2,014
libsvm	2	34	136	134	2,333
JUnit	66	1,163	773	3,704	10,736
Roboguice	98	1,039	2,574	7,014	30,317
Tachyon	18	913	2,153	7,255	43,808
ua-parser	5	35	58	123	741

Table 1: Code metrics of the selected elements

the disks. Changes regarding the number of classes, the NOS of methods or the placement strategy for the glyphs affects the density of the visualization. Some aesthetics are influenced by multiple factors and such a factor in turn affects multiple aesthetics at once. That is why we were not able to extract meaningful factors via a PCA based on the object of comparison or the cause. The explained variance was always insufficient – below 50% for most factors. To obtain more satisfying results about one factor per aesthetic would be necessary. All in all we can say that the relations between aesthetics and their cause in the underlying data are quite complex.

6.2 Implications for the Recursive Disk Metaphor

Based on the results of the study, i.e., identified aesthetics and applied categories we want to identify some drawbacks of the recursive disk metaphor. Further we suggest some modifications to improve the metaphor. The identified aesthetics indicate that it will not be possible to detect typical anti-pattern such as brain methods, brain classes or god classes although the metaphor was expected to be useful for this task [MZ15]. Therefore as a first direct consequence of this study we suggest to make methods more visible and distinguishable. Since the participants were able to perceive the size of class and package disks as well as their proportions methods should be visualized in the same way. Furthermore a method can be considered as the smallest unit of a software system. The presentation is more consistent if all structure units share a similar glyph shape. Figure 4 shows a possible modified version of the recursive disk metaphor. Methods are now represented as own disks just as classes and packages and thereby their importance to the appearance is increased. Although this is less space-efficient we prefer it since it leads to a better perception of the visualization.

Multiple participants chose the thickness of the border of package disks as a construct (#5). Currently the width of a border is fixed, but when packages exist, that only contain exactly one other package the two gray borders looks like one thick one as shown in the left of Figure 5. That structure can be considered as common for java source code and it is not apparent how this information can be used to make meaningful conclusions about the software. For example if three nested packages *org*, *apache* and *common* exist it appears as one package disk with a border three times as thick (given that the packages *org* and *apache* do not contain any other elements). To eliminate this undesired visual difference the three borders should be merged into one disk representing the three packages at once as shown in the right of Figure 5.

Another notable construct is the number of outer gray rings (#3), i.e., package disks. The variance is caused by the number of root packages in the source code. Once again this information is not suitable to make

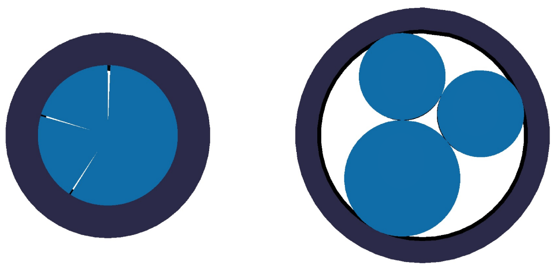

Figure 4: Old (l.) and new (r.) representation of a class with three methods

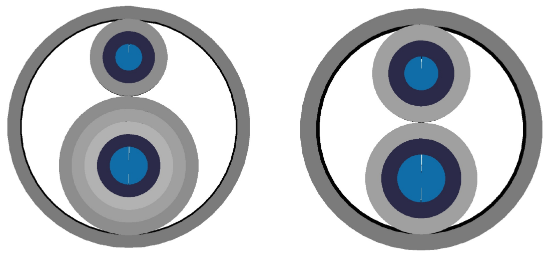

Figure 5: Old (l.) and new (r.) representation of nested packages

statements about the software. If a program has multiple root packages it is always possible to create a new package containing the former root packages. This changes the visual appearance significantly although nearly all code metrics will be unchanged. Therefore we suggest that the visualization should always contain one all-encompassing package disk to unify the appearance regardless of the existence of a corresponding package in the source code.

Furthermore it is conspicuous, that only two participants consider size (#22) as a suitable construct although it seems like a very basic attribute of a visualization. The area of the largest visualization is about 146 as big as the smallest one. This is a sign for a missing scale which makes it nearly impossible to estimate size of a visualization. The user interface did not contain any possibility to compare the size of the displayed objects. However this can not be seen as a disadvantage of the visualization metaphor, because the user interface has to provide these features. Hence it should be modified to provide a visible scale, coordination system or something similar to empower the user to compare the size of different visualizations.

7 FUTURE WORK

The insights provided in this paper lay a foundation for additional avenues for future work on computational evaluation in software visualization. First, a quantified study is in preparation to investigate the cause and the effect of the identified aesthetics. This will help to understand the relations between the underlying data and these aesthetics as well as the extracted factors of the PCAs of the repertory grid interviews. Second, the suggested modifications to the recursive disk metaphor have to be evaluated empirically. Furthermore the repertory grid technique will be applied to more visualization metaphors such as the city metaphor. Thereby we will be able to compare the impact of the depicted software entities between the different metaphors and investigate how this influences the drawn conclusions from a visualization.

8 CONCLUSION

In this paper, we introduced aesthetics to pave the way towards a computational approach of evaluating software visualizations. Although the presented work has only made a first step towards this goal it already provides new means to improve existing visualization metaphors. With the mean of the repertory grid technique we showed how aesthetics can be identified methodically. Through categorizing the found constructs the impact of the different software entities was made comparable. The recursive disk metaphor overemphasizes the package structure and hinders comparisons between methods due to their similar appearance. We suggested some modifications to the glyph shape and placement strategy to counteract.

9 ACKNOWLEDGMENTS

We are grateful to the volunteers who participated in this study.

10 REFERENCES

[BKC+13] R. Borgo, J. Kehrer, D. H. S. Chung, E. Maguire, R. S. Laramee, H. Hauser, M. O. Ward, and M. Chen. Glyph-based Visualization: Foundations, Design Guidelines, Techniques and Applications. 2013.

[BRSG07] Chris Bennett, Jody Ryall, Leo Spalteholz, and Amy Gooch. The aesthetics of graph visualization. In *Proc. of the Third Eurographics Conf. on Comp. Aesthetics in Graphics, Vis. and Imaging*, Computational Aesthetics'07, pages 57–64, Aire-la-Ville, Switzerland, Switzerland, 2007.

[EMM09] Helen M. Edwards, Sharon McDonald, and S. Michelle Young. The repertory grid technique: Its place in empirical software engineering research. *Inf. Softw. Technol.*, 51(4):785–798, April 2009.

[fCGRI15] The Fraunhofer Institute for Computer Graphics Research IGD. InstantReality. http://www.instantreality.org/, 2015.

[Fra04] Fay Fransella. *A manual for repertory grid technique* . Wiley, Chichester, 2. edition, 2004.

[GHJV93] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Abstraction and Reuse of Object-Oriented Design. *Lect. Notes Comput. Sci.*, 707:406–431, 1993.

[Hua14] Weidong Huang. Evaluating Overall Quality of Graph Visualizations Indirectly and Directly. In Weidong Huang, editor, *Handb. Hum. Centric Vis.* Springer, New York, 2014.

[IW03] Pourang Irani and Colin Ware. Diagramming information structures using 3D perceptual primitives. *ACM Trans. Comput. Interact.*, 10(1):1–19, 2003.

[JUn15] JUnit 4. JUnit 4, May 2015. `https://github.com/junit-team/junit`. Accessed: 2015-05-20.

[Kel55] George A. Kelly. *The psychology of personal constructs. Volume I.* Norton, New York, 1955.

[Kos03] Rainer Koschke. Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey. *J. Softw. Maint. Evol.*, 15(October 2002):87–109, 2003.

[KSB06] V.A. Khamisani, M.S. Siddiqui, and M.Y. Bawany. Analyzing soft skills of software engineers using repertory grid. In *Multitopic Conf., 2006. INMIC '06. IEEE*, pages 259–264, Dec 2006.

[Lan71] A. W. Landfield. *Personal construct systems in psychotherapy*. Rand McNally, Chicago, 1971.

[Lan06] Michele Lanza. *Object oriented metrics in practice : Using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer, Berlin, 2006.

[LBI$^+$12] H. Lam, E. Bertini, P. Isenberg, C. Plaisant, and S. Carpendale. Empirical studies in information visualization: Seven scenarios. *Vis. and Comp. Graphics, IEEE Transactions on*, 18(9):1520–1536, Sept 2012.

[MZ15] Richard Müller and Dirk Zeckzer. The Recursive Disc Metaphor: A Glyph-based Approach for Software Visualization. *6th Int. Conf. on Inf. Vis. Theory and Applications*, 2015.

[Nor04] Donald Norman. *Emotional Design. Why We Love (or Hate) Everyday Things*. Basic Books, 2004.

[PAC02] Helen C. Purchase, Jo-Anne Allder, and David Carrington. Graph Layout Aesthetics in UML Diagrams: User Preferences. *J. Graph Algorithms Appl.*, 6(3):255–279, 2002.

[PCA02] Helen C. Purchase, David Carrington, and Jo-anne Allder. Empirical Evaluation of Aesthetics-based Graph Layout. *Empir. Softw. Eng.*, 7:233–255, 2002.

[PMCC01] Helen C. Purchase, Matthew McGill, Linda Colpoys, and David Carrington. Graph drawing aesthetics and the comprehension of uml class diagrams: An empirical study. In *Proc. of the 2001 Asia-Pacific Symp. on Inf. Vis. - Volume 9*, APVis '01, pages 129–137, Darlinghurst, Australia, Australia, 2001. Australian Computer Society, Inc.

[Pur97] Helen C. Purchase. Which Aesthetic Has the Greatest Effect on Human Understanding? In *Proc. 5th Int. Symp. Graph Draw.*, 1997.

[TC08] Alfredo R Teyseyre and Marcelo R Campo. An overview of 3D software visualization. *IEEE Trans. Vis. Comput. Graph.*, 15(1):87–105, 2008.

[VLR$^+$10] Arnold P O S Vermeeren, Effie Lai-chong Law, Virpi Roto, Marianna Obrist, Jettie Hoonhout, and Kaisa Väänänen-Vainio-Mattila. User experience evaluation methods. *Proc. 6th Nord. Conf. Human-Computer Interact. Extending Boundaries - Nord. '10*, page 521, 2010.

[War02] Matthew O. Ward. A taxonomy of glyph placement strategies for multidimensional data visualization. *Information Visualization*, 1(3/4):194–210, December 2002.

[WLR11a] Richard Wettel, Michele Lanza, and Romain Robbes. Software systems as cities: a controlled experiment. *2011 33rd Int. Conf. Softw. Eng.*, pages 551–560, 2011.

[WLR11b] Richard Wettel, Michele Lanza, and Romain Robbes. Software systems as cities: A controlled experiment. In *Proc. of the 33rd Int. Conf. on Soft. Eng.*, ICSE '11, pages 551–560, New York, USA, 2011. ACM.

[WPCM02] Colin Ware, Helen C. Purchase, Linda Colpoys, and Matthew McGill. Cognitive measurements of graph aesthetics. *Inf. Vis.*, 1(2):103–110, June 2002.

#	Construct	Freq	Object of Comparison	Cause
1	low density – high density	8	Package	*everything*
2	heavily nested – little nested	8	Package	package structure
3	many outer gray rings – few outer gray rings	8	System	no. of packages package structure
4	simple – complex	7		
5	thin edge of outer ring – thick edge of outer ring	5	Package	package structure
6	high yellow share – less yellow share	5	Class	no. of attributes
7	centered – off-center	5		
8	elements distributed equally/leaning to the right	5	System	*everything*
9	short strings of purple disks – long strings of purple disks	4	Package	package structure no. of classes
10	chaotic – logical	4		
11	many purple disks – few purple disks	4	Package	no. of classes
12	few/many inner outer rings with the same size	3	System	package structure package size
13	open strings – closed strings	3		
14	many/few outer rings with the same size	3	System	package structure package size
15	many/few purple disks with the same size	3	Package	class size
16	detailed – not detailed	2		
17	many/few gray rings	2	System	package structure
18	harmonic – inharmonic	2		
19	many/few inner gray rings	2	Package	package structure
20	circular/semicircular positioning of elements	2		
21	symmetrical – asymmetric	2	System	*everything*
22	small – large	2		
23	nested gray rings – no nested gray rings	2	System	package structure
24	boring – interesting	2		
25	one layer of purple disks – multiple layers of purple disks	2	Package	no. of classes
26	unstructured - structured	2		
27	predominant simple/complex purple disks	2	Package	class structure
28	manageable – overloaded	1		
29	balanced – not balanced	1		
30	many/few purple disks with a similar structure	1	Package	class structure
31	dynamic – static	1		
32	(no) constantly increasing element size within a gray ring	1		
33	few/many elements in the outer rings	1	System	package structure no. of classes
34	compact – not compact	1		
35	regular – irregular	1		
36	few/many inner gray rings with the same size	1	System	package structure package size
37	beautiful – less beautiful	1		
38	isolated purple disks – no isolated purple disks	1		
39	centered y-axis – shifted y-axis	1	System	*everything*
40	closed – open	1		
41	area with dominant color – no area with dominant color	1		
42	yellow rings – yellow disks	1	Class	class structure
43	orderly – disordered	1		
44	simple spiral shape – complex spiral shape	1		
45	small purple disks – large purple disks	1	Class	class size
46	complicated – uncomplicated	1		
47	loose structure – compact structure	1		
48	yellow evenly distributed/concentrated on one area	1	Package	class structure
49	coarse structure – fine structure	1		
50	flat – deep	1		
51	few/many purple disks without blue segments	1	Package	class structure
52	homogeneous/heterogeneous inner gray rings	1		
53	artificial – natural	1		

Table 2: List of identified constructs. Aesthetics are highlighted gray.