# VisualCues: Visually Explaining Source Code in Computer Science Education

Benjamin Biegel, Sebastian Baltes, Bob Prevos, and Stephan Diehl
Department of Computer Science
University of Trier, Germany
Email: {biegel,s.baltes,diehl}@uni-trier.de

***Abstract*—Humans are very efficient in processing and remembering visual information. That is why metaphors and visual representations are important in education. Because of their high visual expressiveness, presentation tools like Microsoft PowerPoint are very popular for teaching in classrooms. However, representing source code with such tools is tedious and cumbersome, while alternatives like source code editors lack visual expression. Moreover, modifying prepared content, e.g. while responding to questions, is not well supported. In this paper, we introduce *VisualCues*, an approach with the goal of combining the flexibility of source code editors with the visual expressiveness of classical slide-based presentation tools. A key concept of *VisualCues* is linking visual artifacts to specific elements of source code. The main advantage is that when changing the underlying source code, the positions of linked visual artifacts are changed simultaneously. We implemented a first prototype and evaluated it in two undergraduate computer science courses.**

## I. Introduction

The phrase *"A picture is worth a thousand words."* reflects the human ability to recall complex information more easily if it is explained and represented visually [1, 2, 3, 4]. That is why professional software developers [5, 6] as well as teachers [7, 8] make use of sketches and diagrams to describe, explore, and communicate abstract concepts in a comprehensible way. Computer science education requires teachers to show concrete source code examples for demonstrating how introduced concepts can be applied. Classical presentation tools like whiteboards, blackboards, or slide-based presentation tools like Microsoft PowerPoint, are often too rigid and limited in displaying source code and eventually restrict teachers' visual expressiveness [9]. Especially, using slide-based presentation tools is controversial [10, 11]. Teachers are forced to follow a linear, pre-defined presentation [12], which makes an active interaction with the audience more complicated [13]. Moreover, such tools lack effective methods for highlighting graphical or textual content while presenting [14]. It is not surprising that teachers sometimes choose source code editors or entire integrated development environments (IDEs) as alternative teaching tools, even though these tools are not originally designed for presenting source code and are not suitable for visual explanation. However, source code editors enable a flexible and spontaneous exploration and modification of source code, which is helpful in explaining and understanding source code fragments [15]. The dynamic highlighting features in editors such as emphasizing the token at the current cursor position are also an advantage compared to PowerPoint and whiteboards.

In summary, with classical tools, computer science teachers are not able to switch seamlessly between teaching abstract software engineering concepts and concrete source code examples. Using both PowerPoint and a source code editor within a single lesson increases the complexity of the presentation for both teachers and students. In particular, teachers are highly limited in contrasting several aspects across different abstraction levels. Eventually, either they have to constantly switch between PowerPoint and the source code editor or they have to choose between a visually expressive, but static, and a flexible, but text-oriented, source code presentation. In order to fill this gap, we introduce *VisualCues*, a novel approach for displaying and annotating source code. What is special about *VisualCues* is that it allows to create visual elements similar to, for example, Microsoft PowerPoint or Adobe Illustrator. Then, these visual elements can be connected with specific source code elements. The inter-linking and anchoring mechanism ensures that when changing the underlying source code, the positions of linked visual artifacts are changed simultaneously. Since our approach combines both visual expressiveness and a flexible source code representation, it allows a seamless transition between teaching abstract software engineering concepts and demonstrating concrete source code examples.

## II. Related Work

There exist tools that enable software developers to link sketches to source code artifacts either based on source code lines [16] or based on elements of the abstract syntax tree [17]. The created links are visualized in the IDE and can be used to navigate through source code. The first tools adding annotation support to IDEs were CodeAnnotator [18], the Rich Code Annotation Tool [19], and vsInk [20]. A more specialized source code presentation tool is Explorable Code Slides [15], where code fragments are displayed in boxes and can be freely positioned on a two-dimensional canvas. By following source code links, the presenter is able to interactively explore the code base. Some IDEs like IntelliJ IDEA offer a special presentation mode in which the font size is larger than usual, the source code editor runs in fullscreen, and additional views, toolbars, and other components are hidden. Not specifically designed for source code but worth to mention is the document reader XLibris [21] that anchors sketches to single words. Prezi [22] and MindXPres [23] use infinite zoomable canvases. PaperPoint [24] anchors printed slides to the original digital slides. Classroom Presenter [25] is a tool for collaborative presentations and shareable free-form annotations.

TABLE I. REQUIREMENTS FOR VISUALCUES

| |
|---|
| **Preparation:** Teachers are able to entirely create the teaching material in advance and use it in class without further preparation. Furthermore, prepared content can be re-used multiple times. |
| **Presentation:** The tool is ideally suited for presenting the learning material within a classroom. By providing methods for highlighting graphical as well as textual elements, the audience can follow the presentation easily. |
| **Follow-up:** To enable an effective follow-up for the students after the lecture, the teaching material can be made available immediately as presented. |
| **Flexibility:** Presented teaching material can be modified or extended with custom content promptly and flexibly during the presentation. |
| **Expressiveness:** Teachers are able to create the teaching material in a creative process, which enables them to design and arrange the learning content freely with as little limitations as possible. |

## III. CONCEPT AND PROTOTYPE

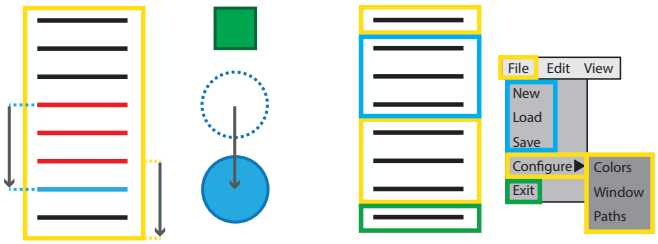
Based on literature review and own observations in teaching, we developed a set of requirements (see Table I), which, in our opinion, should be considered when designing tools for flexible and visually expressive presentations. The requirements *Preparation*, *Presentation* and *Follow-up* each represent one phase of a usual teaching lesson. The main goal of our approach is to support an iterative, interactive and creative presentation style. Thus, motivated by the Cognitive Dimensions of Notations framework [26], we added the two additional requirements *Flexibility* and *Expressiveness*.

### A. State of the Art

In order to get a better impression of the state-of-the-art tools, we use the five requirements as evaluation criteria. We consider two teaching scenarios separately: teaching abstract concepts and in particular the presentation of source code examples. We compare three different tools: slide-based tools like PowerPoint, white- and blackboards, and source code editors (IDEs).

*1) Abstract Concepts:* Tools like PowerPoint are appropriate for presenting abstract programming concepts and software designs. Since teachers are forced to use a linear presentation style and support for highlighting and adding graphical and textual content during the lecture is weak, *Presentation* and *Flexibility* are only covered partially by those tools. Boards support active sketching and thus foster a spontaneous and visually expressive presentation style. Therefore, *Expressiveness* and *Flexibility* are largely covered. Nevertheless, boards are limited in space and thus have to be cleaned periodically, making the discussion of previously presented content sometimes impossible. Their main shortcoming is that every graphical and textual content has to be created during the lecture, which is very time-consuming and, in the sense of our requirements, makes preparing and providing of teaching material hard. It is obvious that a conventional IDE is not suitable for presenting abstract concepts.

*2) Source Code Examples:* For demonstrating source code examples, slide-based tools perform worse than in the previous teaching scenario. In contrast to boards, slide-based tools put teachers in the role of a passive narrator, who strictly follows a predefined narrative thread. It is difficult for teachers to depart from the prepared presentation and spontaneously address issues raised by students, e.g., showing the source code of a method called in a prepared example, or extending and explaining a prepared example. Furthermore, before and during the lecture, it is tedious to add source code to slides or modify it. Therefore, slide-based tools only cover *Follow-up* completely and the other criteria partially. Teachers have great freedom when using boards to create visually expressive presentations, but boards lack support for presenting source code. On the one hand, existing text can be easily annotated, but on the other hand, modifying existing text is a cumbersome task. Further, dynamic exploration of linked elements, e.g., following method call paths, is missing and thus, fast switching between different source code artifacts is impossible. In summary, only *Expressiveness* and *Flexibility* are partially covered by boards. With IDEs, visual expressiveness, stepwise revealing of source code fragments, and fully preparing linear presentations in advance are not supported. Nevertheless, the ability to dynamically explore, navigate, highlight, and modify source code during the lecture seems to outweigh the disadvantages sometimes.




(a) Linking visuals with source code. (b) Linking pictures with source code.

Fig. 1. Inter-linking and anchoring mechanism of VisualCues.

### B. Concept

The concept of our approach is based on the five requirements mentioned above and includes solutions that try to fulfill those requirements. To this end, we introduce our ideas for each requirement separately.

*1) Preparation, Presentation, and Follow-up:* To prepare a lecture, the teacher defines beforehand in which order the source code should be revealed. During the lecture, teachers can then reveal the code step by step. In the preparation phase, the teacher can create and edit visual elements similar to the drawing features of PowerPoint (see requirement *Expressiveness* below). The visual elements can then be linked to the corresponding source code elements (see requirement *Flexibility* below). All drawing features are also available during the lecture such that the teacher can decide to create content live, e.g. to respond to questions. Graphical as well as textual elements can be highlighted by hovering over them; source code can be highlighted similar to the highlighting mechanism in IDEs. The presentation can be saved at any time to share it with the students. After the lecture, the source code and the linked visuals are made available. Students can then branch the documents to get their own copy for further annotations.

*2) Flexibility:* The main idea of *VisualCues* is inspired by the inter-linking and anchoring mechanism of PowerPoint, which enables, for example, visually connecting two boxes with a line. When moving one box around, the position and

the length of the line are updated simultaneously. This ensures that the visual connection remains. Applying this basic idea to source code presentation leads to the concept illustrated in Fig. 1a. The source code (lines on the left) is framed by a yellow rectangle. Same colors indicate a connection between elements and consequently the blue line and the blue circle are related and have been anchored. If new lines (red) are added, the blue circle moves down simultaneously with the blue line. Then, in order to still frame the complete source code, the yellow rectangle increases its height. The position of the green rectangle remains unchanged because it is not anchored to any element.

*3) Expressiveness:* An integrated source code editor allows displaying and modifying source code as known from IDEs. Graphical artifacts can be added and arranged freely using predefined shapes, color palettes, and freehand drawings. Visual anchor points help to easily connect several elements. Beside manually creating and arranging visual artifacts, importing pictures is also supported. Fig. 1b depicts how a source code example creating a menu could be explained by using a picture showing such a menu. Again, same colors indicate a connection between graphical elements and source code. More precisely, the colors indicate which source code fragments are responsible for creating a specific menu entry. In this example, anchoring elements is not necessary.

### C. Prototype Implementation

Based on our concept, we created a first prototype implementing the most important ideas. The prototype was built using web technologies (i.e., HTML5, JavaScript, SVG, and PHP) such that it can be executed on different devices (including tablets).

At the bottom of the graphical user interface (see Fig. 2), we placed a toolbar with buttons for creating predefined shapes (left side) and a color palette, highlighting modes, and buttons for undo and redo (right side). Currently, the prototype has only rudimentary support for importing source code examples: The teacher pastes his or her code examples into a converter that generates XML-files that are automatically imported into a database. By editing these files, one can define at what point which part of the code is revealed. Then, one can use a generated URL to open the examples in *VisualCues* for preparing the visual content. The source code is tokenized such that each word can be used for anchoring visual elements. Thus, basically every programming language is supported. We used Java and pseudo code in our evaluation (see Section IV). Two different anchors can be created: (1) By setting only one anchor, the position of the graphical element is bound to the position of the anchored source code element; (2) by setting two anchors, the size of the graphical element is bound to the distance between both anchored source code elements. In this case not only the position of a visual element is adjusted, but also its size.

Textual and graphical elements can be highlighted. Selecting a graphical element reveals resizing handles and a context menu. This menu includes functions for changing the z-index of an element and for setting anchors. Furthermore, it is possible to load and save the current state of *VisualCues*. Additional information can be found as supplementary material [27].

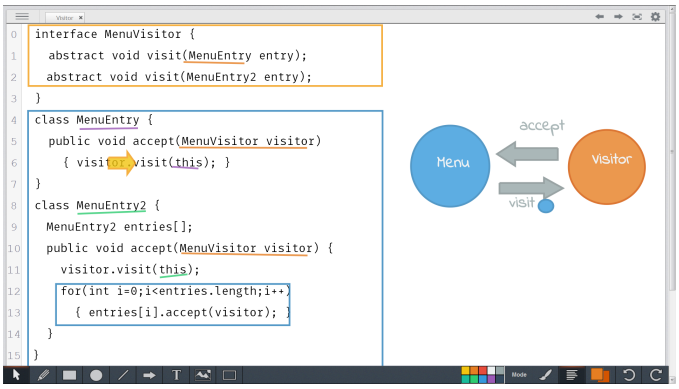



Fig. 2. Explanation of the visitor design pattern (first course).

## IV. Practical Experience

To evaluate *VisualCues*, we applied our prototype in two undergraduate courses: one on software engineering, the other on programming concepts. During the first course (20 students), we introduced software design patterns [28] using *VisualCues*. After the lecture, we interviewed three students, the current teacher (T1), and the previous year's teacher (T2); both are co-authors of this paper. In the second course (63 students), T1 utilized *VisualCues* to explain a binary search tree implementation. In the beginning of that lecture, T2 explained two sorting algorithms with traditional methods (PowerPoint slides and blackboard). Thus, the students were able to compare the traditional methods to *VisualCues*. After the lecture, we handed out a questionnaire to the students and again interviewed three of them and the two teachers. The questionnaire, the students' answers as well as the slides created during the lectures are available as supplementary material [27]. The feedback is of course subjective, but nevertheless valuable for improving the tool. Before presenting results from the evaluation, we shortly describe the visual notation we used in the lectures.

### A. Visual Notation

Inspired by Gestalt theory [29], for each course, we worked out a specialized visual notation. In previous lectures, we noticed that novices had problems to distinguish between static and dynamic aspects of software. In object-oriented programming, for example, it was difficult for some students to see the difference between classes and objects. That is why we consistently used rectangles for representing classes and circles for representing objects during the first evaluation of our tool (see Fig. 2). Same colors indicated a relation between graphical and textual elements, which was also considered in the second course. For the binary search tree, we performed a manual algorithm animation: Both in the source code and a concrete example tree, we executed the algorithm step by step and updated the corresponding markers.

### B. Questionnaire

The questionnaire was anonymous and not mandatory for the students. After requesting demographic data, the students were asked to rate six statements on a 5-point Likert scale ranging from 'I do not agree' to 'I agree' (see [27]). All students filled out the questionnaire (n=63). The average

participant was 22 years old, in the fourth semester, and male (we only had 3 female participants). About half of the students were enrolled in a computer science program, the others were business informatics students.

When interpreting the answers, one has to be careful due to the setting in which the evaluation took place (see Section IV-E). However, for most statements, the tendency in the responses was pretty clear: 94% of the students stated that they were able to easily follow the subject matter presented with *VisualCues* (rating 3 or 4). Furthermore, 63% thought that it was easier to follow the *VisualCues* presentation than the traditional part (rating 3 or 4), 25% were not sure (rating 2). The last four statements aimed at finding out what method the students would like to be used in future lectures for presenting source code. The clear winners of this comparison were live coding (62% rated 3 or 4) and *VisualCues* (56% rated 3 or 4).

### C. Students' View

The interviewed students agreed that it is important to have a common and understandable visual language, which, according to them, was true for the ad-hoc notation we used in *VisualCues*. Furthermore, the students said that the separation between class structure and dynamic behavior became immediately clear to them. They appreciated that the source code was revealed step by step. A feature that the students particularly highlighted was the automatic relocation of the linked drawings when new source code was added. One student called this a "basic feature" of *VisualCues*. The interactivity of our approach, enabling the teacher to include comments from students, and generally the visual explanations, were perceived well. One student claimed that *VisualCues* prevents the lecturer from doing "PowerPoint karaoke". Some students pointed at lengthy passages when the teacher was drawing or had problems using the prototype. They said that at some points, for them it seemed like the teacher was distracted because of usability problems with the tool. Generally, students noted that more features known from common source code editors should be integrated into *VisualCues* (e.g., syntax highlighting or the possibility to fold source code regions). With respect to the device setup, several students suggested that a touch screen would enable the teacher to interact with the tool easier and faster.

### D. Teachers' View

According to T1, roughly half of the drawings in the first lecture were sketched out and practiced upfront, the rest was created ad-hoc during the lecture. He noted that after having used *VisualCues* several times, less preparation would be needed. Another part of the preparation phase was planning which source code fragments to show at what point in time. When referring to parts of the source code, he often framed the code with a colored rectangle and then used the same color for the corresponding drawings. Furthermore, he took care to use geometrical shapes consistently and to introduce the notation step by step. The tool allowed him to respond to questions with unplanned ad-hoc visualizations. What he did not like was the fixation on the monitor while drawing and some bugs in the prototype implementation. T1 proposed that if the lecturer would use a tablet for drawing and presenting, he or she would be more flexible and could move around during the lecture. After the lecture, T1 made screenshots with the final state of the drawings available to the students. He remarked that a future version of the tool should provide a more advanced export functionality. In the future, he would like the tool to support structuring the lecture and providing notes for the teacher such that he or she does not miss to present important points (as it happened at one point during the second lecture). We interviewed the teacher who gave the traditional part of the lectures (T2) and asked him to compare the new approach to the previous lectures. He said that the live drawing worked "astoundingly good" and that there was not much overhead introduced. He did not like the strong focus on source code while presenting the design patterns in the first lecture. T2 said that using our prototype, it was possible for the audience to see to what part of the source code the presenter is currently referring, which is sometimes hard to see in traditional lectures.

### E. Threats to Validity

The generalizability of our evaluation results is limited, because the evaluation took place at a single university, with a single programming language (Java), and only in undergraduate courses. Furthermore, the two different teachers may have affected the outcome. However, for presenting source code, students clearly preferred *VisualCues* and live coding over traditional methods like blackboards and PowerPoint slides. We plan to validate these results in a larger evaluation with an improved version of *VisualCues*, where the feedback from our interviews with students and teachers is considered.

## V. Conclusion and Future Work

In this paper, we introduced *VisualCues*, a novel approach for visually explaining source code in computer science education. The key features are annotating source code with visual artifacts, stepwise revealing of source code, and a spatially stable inter-linking and anchoring mechanism. First, we determined a set of five requirements, which led to a general concept and a prototype implementation. Then, we applied the prototype in two undergraduate software engineering courses. During the evaluation of the *VisualCues* prototype, we gained important insights for further developing our tool. We got positive feedback and valuable hints for improvements from the students as well as the teachers. Most of the negative points were related to the usability of the prototype implementation, which we want to improve in future versions. The results from the questionnaire indicate that students would like to see *VisualCues* and live coding being used for presenting source code. Furthermore, the interviewed students said that more features known from source code editors should be integrated into *VisualCues*. We take this as a strong motivation to integrate a full-fledged source code editor as well as the possibility to compile and execute the written source code. We think that this, combined with the possibility to add prepared source code during the lecture and the visual annotation we already support, is an approach that supports the students needs better than the methods currently being used for presenting source code.

## Acknowledgment

The authors thank all students who participated in the evaluation of *VisualCues*.

## References


[1] S. M. Kosslyn, "Graphics and human information processing," *Journal of the American Statistical Association*, vol. 80, no. 391, pp. 499–512, 1985.

[2] E. R. Tufte and E. Weise Moeller, *Visual explanations: Images and quantities, evidence and narrative*. Graphics Press Cheshire, 1997, vol. 36.

[3] P. Goolkasian, "Pictures, words, and sounds: From which format are we best able to reason?" *The Journal of General Psychology*, vol. 127, no. 4, pp. 439–459, 2000.

[4] D. L. Moody, "The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering," *IEEE Transactions Software Engineering*, vol. 35, no. 6, pp. 756–779, 2009.

[5] S. Baltes and S. Diehl, "Sketches and diagrams in practice," in *22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'14)*, 2014, pp. 530–541.

[6] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko, "Let's go to the whiteboard: how and why software developers use drawings," in *Proceedings of the 2007 Conference on Human Factors in Computing Systems (CHI'07)*, 2007, pp. 557–566.

[7] A. F. Blackwell, "The reification of metaphor as a design tool," *ACM Trans. Comput.-Hum. Interact.*, vol. 13, no. 4, pp. 490–530, 2006.

[8] T. L. Naps, G. Rößling, V. L. Almstrum, W. Dann, R. Fleischer, C. D. Hundhausen, A. Korhonen, L. Malmi, M. F. McNally, S. H. Rodger, and J. Á. Velázquez-Iturbide, "Exploring the role of visualization and engagement in computer science education," *SIGCSE Bulletin*, vol. 35, no. 2, pp. 131–152, 2003.

[9] E. R. Tufte, *The cognitive style of PowerPoint*. Graphics Press Cheshire, 2003, vol. 2006.

[10] R. J. Craig and J. H. Amernic, "PowerPoint presentation technology and the dynamics of teaching," *Innovative Higher Education*, vol. 31, no. 3, pp. 147–160, 2006.

[11] S. M. Kosslyn, R. A. Kievit, A. G. Russell, and J. M. Shephard, "PowerPoint presentation flaws and failures: A psychological analysis," *Frontiers in psychology*, vol. 3, 2012.

[12] E. Reuss, B. Signer, and M. C. Norrie, "Powerpoint multimedia presentations in computer science education: What do users need?" in *Proceedings of the 4th Symposium of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society (USAB'08)*, 2008, pp. 281–298.

[13] L. A. M. Abdelrahman, M. Attaran, and C. Hai-Leng, "What does PowerPoint mean to you? a phenomenological study," *Procedia – Social and Behavioral Sciences*, vol. 103, pp. 1319–1326, 2013.

[14] C. Ware, *Visual Thinking for Design*. Morgan Kaufmann Publishers, 2008.

[15] M. Fritz, B. Biegel, and S. Diehl, "Explorable code slides," in *26th International Conference on Software Engineering Education and Training (CSEE&T'13)*, 2013, pp. 199–208.

[16] L. Lichtschlag, L. Spychalski, and J. Bochers, "Codegraffiti: Using hand-drawn sketches connected to code bases in navigation tasks," in *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'14)*. IEEE, 2014, pp. 65–68.

[17] S. Baltes, P. Schmitz, and S. Diehl, "Linking sketches and diagrams to source code artifacts," in *22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE'14)*, 2014, pp. 743–746.

[18] X. Chen and B. Plimmer, "CodeAnnotator: digital ink annotation within eclipse," in *Proceedings of the 19th Australasian conference on Computer-Human Interaction (AUIC'07)*. ACM, 2007, pp. 211–214.

[19] R. Priest and B. Plimmer, "RCA: Experiences with an IDE annotation tool," in *Proceedings of the 7th ACM SIGCHI New Zealand Chapter's International Conference on Computer-Human Interaction (CHINZ'06)*. ACM, 2006, pp. 53–60.

[20] C. J. Sutherland and B. Plimmer, "vsInk: Integrating digital ink with program code in Visual Studio," in *Proceedings of the 14th Australasian User Interface Conference (AUIC'13)*. Australian Computer Society, 2013, pp. 13–22.

[21] G. Golovchinsky and L. Denoue, "Moving markup: repositioning freeform annotations," in *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology (UIST'02)*, 2002, pp. 21–30.

[22] L. Laufer, P. Halácsy, and A. Somlai-Fischer, "Prezi meeting: Collaboration in a zoomable canvas based environment," in *Proceedings of the International Conference on Human Factors in Computing Systems (CHI'11)*, 2011, pp. 749–752.

[23] R. Roels and B. Signer, "MindXpres: An extensible content-driven cross-media presentation platform," in *Proceedings of the 15th International Conference on Web Information Systems Engineering (WISE'14)*, 2014, pp. 215–230.

[24] B. Signer and M. C. Norrie, "PaperPoint: A paper-based presentation and interactive paper prototyping tool," in *1st International Conference on Tangible and Embedded Interaction (TEI'07)*, 2007, pp. 57–64.

[25] R. J. Anderson, R. E. Anderson, B. Simon, S. A. Wolfman, T. VanDeGrift, and K. Yasuhara, "Experiences with a tablet PC based lecture presentation system in computer science courses," in *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE'04)*, 2004, pp. 56–60.

[26] T. Green, "Cognitive dimensions of notations," *People and Computers V*, pp. 443–460, 1989.

[27] B. Biegel, S. Baltes, B. Prevos, and S. Diehl. VisualCues – Supplementary material. [Online]. Available: http://st.uni-trier.de/visualcues

[28] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: Elements of reusable object-oriented software*. Pearson Education, 1994.

[29] M. Wertheimer, "Laws of organization in perceptual forms," *A source book of Gestalt psychology*, 1938.