

ReConstructor: A Scalable Constructive Visualization Tool

Gonzalo Gabriel Méndez*
Escuela Superior Politécnica del Litoral
University of Calgary

Jagoda Walny†
University of Calgary

Søren Knudsen‡
University of Calgary

Charles Perin§
University of Victoria

Samuel Huron¶
Télécom Paristech, Université Paris-Saclay

Jo Vermeulen||
Aarhus University

Richard Pusch**
University of Calgary

Sheelagh Carpendale††
Simon Fraser University

ABSTRACT

Constructive approaches to visualization authoring have been shown to offer advantages such as providing options for flexible outputs, scaffolding and ideation of new data mappings, personalized exploration of data, as well as supporting data understanding and literacy. However, visualization authoring tools based on a constructive approach do not scale well to larger datasets. As construction often involves manipulating small pieces of data and visuals, it requires a significant amount of time, effort, and repetitive steps. We present ReConstructor, an authoring tool in which a visualization is constructed by instantiating its structural and functional components through four interaction elements (objects, modifiers, activators, and tools). This design offers a new balance between preserving the benefits of a constructive process and incorporating a new approach to scalability issues. It allows designers to propagate individual mapping steps to all the elements of a visualization.

Index Terms: Human-centered computing—Visualization—Visualization systems and tools; Human-centered computing—Human computer interaction (HCI)—Interactive systems and tools

1 INTRODUCTION

Visualization authoring tools give people varying degrees of control over how their data is visually represented even when they do not have enough time, resources, or skills to make custom visualizations programmatically. There are a growing number of these tools and, correspondingly, a growing number of approaches to designing them. Popular tools, such as Excel [9] and Tableau Desktop [27], are primarily based on chart templates, automated mappings, and recommendation systems. These features are key for achieving a speedy authoring process and are beneficial for accessibility and ease of use. However, they can also impose barriers. Even for those with formal training in visualization, some types of automated approaches (such as templates) can be a hurdle to flexibility, get in the way of ideation processes, and may even take over the design lead. Recent research suggests that a constructive approach to visualization [12] has the potential to avoid some of these problems.

Constructive visualization promotes the idea of creating visual representations of data by assembling building blocks that are mapped to specific aspects of a dataset. This authoring strategy

has been shown to empower people in their use of visualization without the need of formal training or other specialized skills (e.g., programming). Studies suggest that digital constructive visualization tools can also promote a mindful design process in which people are encouraged to actively reflect on their design decisions [15, 16].

Despite these advantages, the authoring process of existing constructive tools—both tangible [11, 13] and digital [17]—does not scale well to larger datasets. Even for moderately simple visualizations, construction requires a significant amount of time, effort, and repetitive interactions [15, 29]. The problem is exacerbated as the amount of data to be represented (i.e., total records and attributes) grows. Conventional visualization tools avoid the scalability issues of their constructive counterparts by automating certain steps of the visualization process. While automation may reduce the effort required from the designer, it tends to also interfere with personal agency. Automation can make the authoring process less incremental as the tasks delegated to the tool are often executed quickly and over large portions of data (e.g., entire attributes rather than individual values). This, in turn, reduces the transparency of the design process, as the tool’s actions can be hard to follow, which can interfere with comprehension.

We join the growing body of work that investigates how to expand visualization authoring options. In particular, we present ReConstructor, a new point of exploration within the design space of digital constructive visualization tools. In ReConstructor, visualizations are constructed by instantiating their structural and functional components through the use of four interaction elements (*objects*, *modifiers*, *activators*, and *tools*). This design allows people create visualizations via a user-driven constructive approach that also eases the difficulty of working with larger datasets. That is, ReConstructor supports a more scalable authoring process while keeping the agency of this process on the user’s side.

More specifically, our work contributes: (1) a construction strategy for the design of scalable visualization tools based on four reusable interaction elements: objects, modifiers, activators and tools; (2) an explanation of how these interaction elements can be incorporated in the design of visualization authoring tools; and (3) the design and implementation of ReConstructor, a tool that supports the construction of visualizations through these elements.

2 RELATED WORK

In this section, we discuss how the concept of construction has been used in the design of computer interfaces and, in particular, of recent visualization authoring tools.

2.1 Constructive Theories

Educational and learning theories such as Piaget’s constructivism [2, 22], Papert’s constructionism [21], and Froebel’s *gifts* [28] suggest that one way that humans discover the world is by manipulating simple objects and that we can construct knowledge and meaning from these experiences. These theories focus on personal experience,

*e-mail: gmendez@espol.edu.ec

†e-mail: jkwalny@ucalgary.ca

‡e-mail: sknudsen@ucalgary.ca

§e-mail: cperin@uvic.ca

¶e-mail: samuel.huron@telecom-paristech.fr

||e-mail: jo.vermeulen@cs.au.dk

**e-mail: rapusch@ucalgary.ca

††e-mail: sheelagh@sfu.ca

“where the learner is consciously engaged in constructing a public entity” [21, p. 1], as the gateway to understanding and reflection. Computational tools that implement constructive principles generally support processes based on an incremental, bottom-up strategy. This is related to the concept of *emergence*, by which a complex entity arises as the result of interactions among smaller or simpler entities [1]. Consequently, a constructive process is beneficial to show how a complex structure is built from the ground up, as the result of many small steps or sub-processes.

Constructive theories have been widely explored in environments that support the development of computational thinking skills such as Scratch [8, 14, 23, 24] and Mindstorms [20] and, more recently, Google’s Blockly library [10]. In these tools, construction takes place with building blocks that animate interactive visuals.

2.2 Construction in InfoVis

Constructive Visualization [12], a paradigm for visualization authoring grounded in Papert’s, Piaget’s and Froebel’s theories, imports from them the idea of using physical *tokens* (e.g., Lego blocks) that can be mapped to data and manipulated to compose tangible representations. iVoLVER [17] implements constructive principles in a digital visual programming environment. Both Huron et al.’s tangible tokens ([11, 13]) and iVoLVER’s marks support work with atomic data elements (e.g., individual values as opposed to entire data attributes). This leads to a bottom-up construction strategy in which the final design emerges as the result of several small-scale decisions and manipulations of the visualization elements.

Constructing visual representations of data from their atomic building blocks is in line with Bertin’s semiotic views [4, 5]. To convey messages visually, Bertin’s marks—graphical primitives such as points or lines—are configured in particular ways by mapping data attributes to their visual properties [6, 7].

ReConstructor’s constructive approach also makes use of building blocks to provide access to a visualization’s component (e.g., marks and visual properties) and to represent the operations that take place in different types of visual encodings (e.g., sorting, spacing).

3 ENABLING CONSTRUCTION

For a visualization to be *constructed*, the first step is to *deconstruct* it into its modular components. We take this step to make use of the incremental nature of constructive visualization [12]. To support the construction of a given visualization, the design of ReConstructor requires to identify the objects involved (e.g., visual marks, axes), their attributes (e.g., visual properties, labels), and their associated functionality (i.e., the processes associated to these objects such as *sorting* or *distributing*). We also have to pay attention to how these components interact. Having the visualization’s objects, attributes, and functionality as modular components enables a constructive approach to visualization authoring.

3.1 Interaction Elements

ReConstructor’s design is based on four fundamental interaction elements: objects, modifiers, activators, and tools. When combined, these elements enable reconstruction of a visualization from its modular components.

An **object** has a graphical representation, which supports visibility [18, 19] and allows for direct manipulation [26]. They have various visible attributes (e.g., fill and stroke colors) and are by default inert—they do not cause any effect or interact with each other.

A **modifier** can take action on an object, such as applying the value of the object’s attribute. It has the ability to change an object. For example, a fill modifier can change the fill color of an object and

a stroke modifier changes the color of the stroke that a pen tool produces. Modifiers can be of two types:

- *Transient* modifiers change an object’s attribute when placed on the object. Because the change in the object is visually conveyed, these modifiers are not graphically represented on the canvas. Transient modifiers include the stroke and fill color modifiers, as well as the shape modifier that turns strokes into regular shapes.
- *Persistent* modifiers do not immediately change an attribute when placed on an object. Instead, they remain visible, attached to the object they modify. That is, they “objectify” [30] attributes (e.g., width or height). Dropping a data dimension from a dataset onto the visual representation of a persistent modifier establishes a mapping between the corresponding data dimension and attribute. Persistent modifiers can add properties that are not inherent to strokes. For example, the label modifier adds text to an object.

An **activator** carries a process that can be dynamically added to an object. That is, activators bring inert objects to life by turning them into *tools*. For example, a push activator carries the process of pushing other objects around. Adding a push activator to a squiggly stroke would give it the ability to push other objects when dragged. This activating strategy is related to ActiveInk [25], where ink annotations can be activated to operate on visual representations.

A **tool** is an object that has been activated. A tool can have one or more activators associated with it. For example, adding an ink activator to a push-activated stroke results in a pen that draws while pushing other objects out of the way. Activators and persistent modifiers can also be removed from tools. For example, when a push activator is removed from an object it no longer pushes other objects. Removing one activator does not affect the functionality added by other activators (if any).

3.2 An Illustrative Example

To illustrate how we can use activators and modifiers to turn a passive object into an active tool, we describe the construction of a “pen” (Fig. 1). In this case, we do not use any data. Instead, we simply illustrate the interaction aspects of our constructive approach through construction of a drawing tool. Here modifiers and activators have graphical representations that suggest their type and function and are added to objects via drag-and-drop gestures.

We first add to a user-drawn line (1) an ink activator (2), which in this scenario carries a draw function. The added activator appears as a directly manipulable graphical element that is attached to the activated stroke (3). The ink activator of our example gives objects the ability to draw on the canvas when moved, converting the acti-

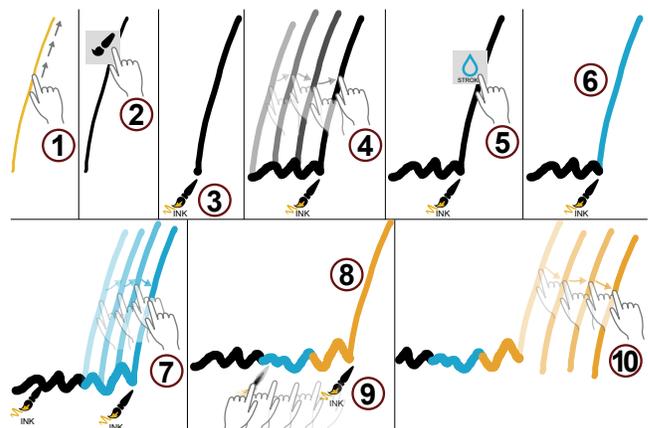


Figure 1: Constructing a “pen” tool from a user-drawn stroke (1) by attaching an ink activator (2). The pen’s (3, 4) ink is configured via color modifiers (5–9).

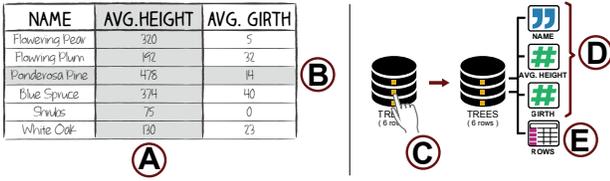


Figure 3: A tabular dataset about trees, deconstructed into attributes (A) and records (B). This dataset can have a compressed representation (C), or an expanded representation showing its attributes (D) and records (E).

vated line into a “pen”. In this example, a pen generates scribbles on the canvas with the same width and color of the activated line (4). We then use a stroke modifier (5) to change the activated line’s stroke color to blue (6). Consequently, new movements of the pen produce blue scribbles (7). Subsequent modifications of the object’s stroke color will also change the color of the pen’s ink (e.g., to orange–8). Dragging the brush icon out of our pen tool deactivates it (9). This step does not affect the line’s current visual appearance or the scribbles previously drawn with it. Once deactivated, the stroke no longer draws when moved (10).

4 RECONSTRUCTOR

We now explain how objects, modifiers, activators and tools integrate in ReConstructor to support visualization creation.

4.1 The Interactive Environment

In ReConstructor, visualizations are constructed within a canvas where people draw and move strokes via touch, pen or mouse. These strokes are the *objects* of our domain. We use *modifiers* to change objects’ visual appearance. *Activators* add functionality to objects, which enables the construction of tools to operate on other objects (e.g., a tool to spatially arrange a set of rectangles). As encouraged by the idea of Instrumental Interaction [3], ReConstructor pays attention to visibility and enables direct manipulation [26] of its interaction elements. Thus, both activators and modifiers are available as icons in a palette and are attached to (or removed from) objects via drag-and-drop—as in the illustrative example. When added to an object, activators are represented graphically, with an icon that suggests their function [18]. The same applies for persistent modifiers. Transient modifiers produce an immediate change on an object’s visual appearance.

4.2 Working with Data

ReConstructor currently supports tabular datasets. We deconstruct tabular datasets into their structural components: data dimensions (i.e. columns) and records (i.e. rows). A data dimension is a collection of all the values of a given column and is named after the column (Fig. 3.A). A record is a set of attribute-value pairs (e.g., the record

{name: ‘Ponderosa Pine’, avg.height: 478, avg.girth: 14} shown in Fig. 3.B) In ReConstructor, a dataset is represented on the canvas either in compressed (Fig. 3.C) or expanded form (Fig. 3.D). When expanded, the dataset provides access to representations of its data dimensions (Fig. 3.D) and records (Fig. 3.E). These representations are draggable objects whose icon and color suggest the underlying data type: blue half-quotes for categorical attributes and green pound sign for quantitative ones. ReConstructor organizes the dataset’s components in an ordered sequence. For example, the first value of the Name dimension is Name[0] and the first record of the dataset is Rows[0]. This order is relevant for some activators such as the tuple and replicate activators, as explained later.

5 CONSTRUCTING A BAR CHART

We now use a running example of constructing a sorted bar chart made of tree shapes to show the use of our interaction elements.

A bar chart typically consists of two or more bars which: *a*) have a height that represents an aspect of the data, *b*) are aligned to a common base, *c*) are optionally spaced for readability, and *d*) ideally, are labeled according to the data they present.

Step 1: Creating a visual mark with visual properties. Marks can be created from any object drawn on the canvas. These objects have inherent attributes such as width, height, and stroke and fill colors. Fig. 2 (1–3) shows how to construct a visual mark from a hand-drawn stroke using the fill transient modifier to set the fill color property of the tree to green. Other transient modifiers such as stroke color and shape beautification can be applied similarly.

Step 2: Mapping data dimensions to visual properties. We now show the label associated to the data record our tree mark represents, as well as map its height value to the mark’s height attribute. Fig. 2 (4–9) shows the steps involved in establishing these data mappings. We first add two persistent modifiers to the mark: label (4) and height (5). Because no data is mapped to either modifier, there is no effect on the mark itself but icons representing these modifiers are attached underneath the mark (5, 6). These icons can be used to map different data dimensions to visual properties. To establish a data mapping, we drag a data dimension from the dataset and drop it onto the icon of a persistent modifier. When mapping the Name attribute to the Label modifier of the mark (7), the name of the first record in the dataset (Name[0]) is attached to the tree (8). We then map the Avg.Height data dimension to the height modifier (8). This changes the height of the mark using the value of the first record in the dataset (Avg.Height[0]).

Step 3: Replicating the defined mappings. At this stage, we have created a single mark that represents the first record of our dataset.

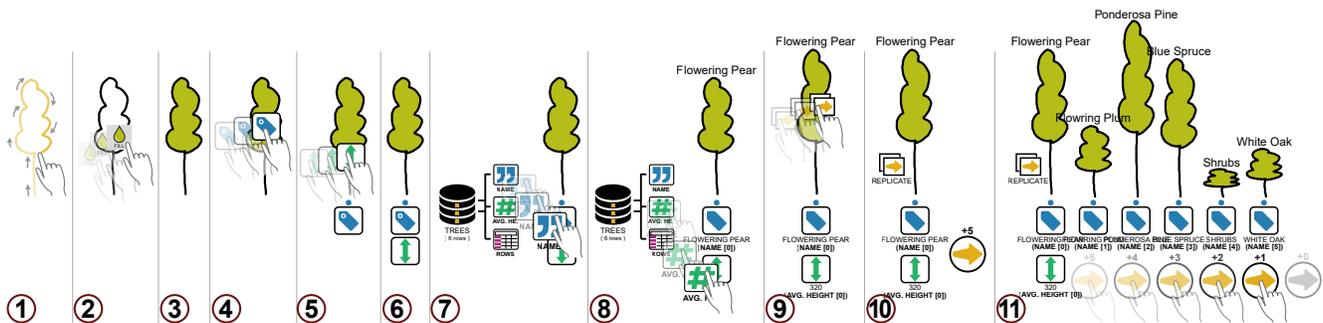


Figure 2: Constructing a bar chart using marks in the shape of trees. After drawing the outline of a tree (1), dropping the transient fill color modifier (2) changes the fill color of the mark (3). Adding a label (4) and a height (5) modifier attach persistent representations of these modifiers underneath the mark (6), without any effect on the mark itself. Mapping the name attribute to the Label modifier (7) adds the name of the first tree in the dataset (8). Mapping the avg. height attribute to the height modifier (8) sets the height of the mark to the average height of the first tree in the dataset (9). Adding a replicate activator to the tree and creates an interactive handle (10) that can be dragged right to create new visual mark that replicate the mappings of the original mark.

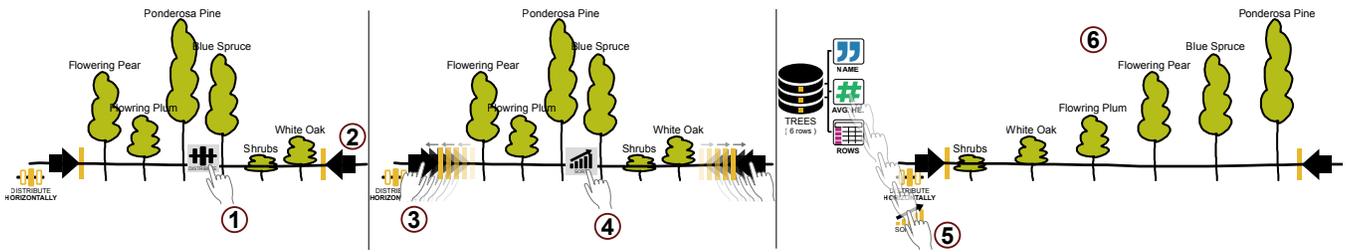


Figure 4: Distributing and sorting the marks of a chart. Drawing a horizontal stroke on top of the trees and adding a distribute activator to that stroke (1) turns the stroke into a tool. This tool shows two handles (2) that can be dragged outward (3) to distribute and increase the horizontal space between marks. Adding a sort activator (4) to the tool and mapping the avg. height data attribute to it (5) sorts the marks based on their avg. height value, in ascending order (6).

Fig. 2 (9–11) shows how we use a replicate activator to avoid having to manually define mappings for each record of the dataset. We first add the replicate activator to the mark (9). This adds a handler to the right of the mark’s visual properties (10, yellow encircled arrow). This step turns the newly activated object into an interactive tool that can replicate itself. A numeric value on top of the yellow handler indicates the number of additional data records that can be replicated. We then drag the handler from left to right to create new marks (11). The more we drag, the more visual marks are created. Each newly created mark replicates the mappings of the original one using subsequent data records (e.g., the first replicated mark uses the values Name[1] and Avg. Height[1]). As new marks are replicated the number of available records to replicate shown on top of the handler decreases. When this value is zero the handler is grayed out and cannot be dragged further.

Step 4: Distributing and sorting the trees. Now that we have constructed a bar chart, we want to rearrange the trees to evenly add some space between them, and to sort them in ascending order. We achieve these actions by creating another interactive tool, as shown in Fig. 4. We draw a horizontal stroke on top of the trees and add a distribute activator to it (1). Upon adding the distribute activator, two interactive handles appear at the start and end points of the space spanned by the marks intersected by the stroke (2). Activating the stroke turns it into a distributing *tool*. Moving its handles (3) increases (or decreases) the horizontal space between the trees. The last step is to sort the trees in ascending order. We add a sort activator (4) to the horizontal stroke and drag the Avg. Height data dimension of the dataset onto it (5). This sorts the trees in ascending order by default (6). Tapping the sort activator’s icon changes the sorting direction (to descending) if needed.

6 DISCUSSION

We chose to explore a variation of a digital constructive approach to creating visualizations. A constructive approach requires components from which the visualization can be built. Previous constructive approaches [12, 17] have decomposed the data into individual data entities which can be represented as tokens. Construction can then proceed with these tokens. While this has shown to empower people and promote data and visualization understanding [13, 15, 29], people also object to the tedium of moving individual tokens [29].

In ReConstructor we consider leveraging the tabular data structure in our deconstruction. This approach to deconstruction, allows the user to make use of this structure through our replicator activator to pull related data entities unto the canvas. This helps alleviate some of tedium of placing the single data items, and points to a possible direction for combining the advantages of constructive visualization with a more scalable interactive approach.

6.1 Flexibility and Sequencing

One of the key points of construction is the potential to achieve similar outcomes by combining atomic elements in multiple ways. In contrast to tangible tools, this is particularly true for digital construction, as software-supported processes can be notoriously mutable. Our ongoing observations show that people expect a more relaxed workflow regarding the order of operations they executed in ReConstructor. This is in line with studies that have shown that, when the freedom is available, humans do not follow particular sequences during design but rather opt for personal variations that change with the problem at hand [19]. Developing ReConstructor, however, made evident that full flexibility can be hard to achieve in a software solution. It appears that some order is required to achieve a construction. In ReConstructor we make use of the order used to store the tabular data.

6.2 Granularity, Scalability and Agency

ReConstructor operates on data dimensions—rather than individual values like other constructive tools. In combination with our replication strategy, working at this coarser granularity level provides some scalability. We also circumvent scalability issues by enabling people to create tools to operate on the tokens they use. One can construct, for example, a “sorting tool” to sort the marks of a visualization according to a specific data dimension. This is a unique aspect of ReConstructor that places more responsibility (hence, agency) in the hands of the designer. It also differs from most conventional visualization tools in that although certain operations can be speeded up, they are not fully delegated to the tool but rather constructed by the designer.

7 CONCLUSION

ReConstructor provides a new variation on constructive visualization that supports the creation of visualizations based on four reusable interaction elements—objects, modifiers, activators and tools. In creating a visualization with ReConstructor visual marks are constructed by attaching modifiers to hand-drawn canvas *objects*. Persistent *modifiers* provide mechanisms for establishing data bindings, achieving data-driven modifications of an object’s visual appearance. *Activators* can also be added to the visual marks to enable the construction of *tools* that can operate on the visualization’s components. In combination, these four interaction elements enable the creation of visualizations through a constructive authoring process.

ACKNOWLEDGMENTS

This project was funded in part by the National Sciences and Engineering Research Council of Canada, Alberta Innovates Technology Futures, SMART Technologies, and the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 753816.

REFERENCES

- [1] R. Abbott. Emergence, entities, entropy, and binding forces. In *The Agent 2004 Conference on: Social Dynamics: Interaction, Reflexivity, and Emergence*, p. 17, 2004.
- [2] E. Ackermann. Piaget's constructivism, papert's constructionism: What's the difference. *Future of learning group publication*, 5(3):438, 2001.
- [3] M. Beaudouin-Lafon. Instrumental interaction: An interaction model for designing post-wimp user interfaces. In *CHI '00*, pp. 446–453. ACM, New York, NY, USA, 2000. doi: 10.1145/332040.332473
- [4] J. Bertin. *Graphics and Graphic Information Processing*. De Gruyter, Berlin, 1981. Translation: William J. Berg, Paul Scott.
- [5] J. Bertin. *Semiology of Graphics: Diagrams, Networks, Maps*. ESRI Press, Redlands, Calif, 1 edition ed., Oct. 2010.
- [6] W. S. Cleveland and R. McGill. Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984. doi: 10.2307/2288400
- [7] W. S. Cleveland and R. McGill. Graphical perception: The visual decoding of quantitative information on graphical displays of data. *Journal of the Royal Statistical Society. Series A (General)*, 150(3):pp. 192–229, 1987.
- [8] S. Dasgupta, S. M. Clements, A. Y. Idlbi, C. Willis-Ford, and M. Resnick. Extending Scratch: New pathways into programming. In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, vol. 2015-Decem, pp. 165–169, 2015. doi: 10.1109/VLHCC.2015.7357212
- [9] M. Excel. Create a chart with recommended charts - excel. <https://support.office.com/en-gb/article/Create-a-chart-with-recommended-charts-cd131b77-79c7-4537-a438-8db20cea84c0> [Online; accessed 16-September-2018], 2018.
- [10] Google. Blockly — Google Developers, 2017.
- [11] S. Huron, S. Carpendale, J. Boy, and J.-D. Fekete. Using VisKit: A Manual for Running a Constructive Visualization Workshop. In *Pedagogy of Data Visualization Workshop at IEEE VIS 2016*. Baltimore, MD, United States, Oct. 2016.
- [12] S. Huron, S. Carpendale, A. Thudt, A. Tang, and M. Mauerer. Constructive visualization. In *Proceedings of the 2014 Conference on Designing Interactive Systems, DIS '14*, pp. 433–442. ACM, New York, NY, USA, 2014. doi: 10.1145/2598510.2598566
- [13] S. Huron, Y. Jansen, and S. Carpendale. Constructing visual representations: Investigating the use of tangible tokens. *Visualization and Computer Graphics, IEEE Transactions on*, 20(12):2102–2111, Dec 2014. doi: 10.1109/TVCG.2014.2346292
- [14] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond. The scratch programming language and environment. *Trans. Comput. Educ.*, 10(4):16:1–16:15, Nov. 2010. doi: 10.1145/1868358.1868363
- [15] G. G. Méndez, U. Hinrichs, and M. A. Nacenta. Bottom-up vs. top-down: trade-offs in efficiency, understanding, freedom and creativity with infovis tools. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, CHI '17*. ACM, New York, NY, USA, 2017. doi: 10.1145/3025453.3025942
- [16] G. G. Méndez, M. A. Nacenta, and U. Hinrichs. Considering agency and data granularity in the design of visualization tools. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI '18*. ACM, New York, NY, USA, 2018. doi: 10.1145/3173574.3174
- [17] G. G. Méndez, M. A. Nacenta, and S. Vandenheste. ivolver: Interactive visual language for visualization extraction and reconstruction. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, CHI '16*, pp. 4073–4085. ACM, New York, NY, USA, 2016. doi: 10.1145/2858036.2858435
- [18] D. A. Norman. *The Psychology Of Everyday Things*. Basic Books, 1988.
- [19] D. A. Norman and J. Nielsen. Gestural interfaces: A step backward in usability. *interactions*, 17(5):46–49, Sept. 2010. doi: 10.1145/1836216.1836228
- [20] S. Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, New York, July 1993.
- [21] S. Papert and I. Harel. Situating constructionism. In *Constructionism*. Ablex Publishing Corporation, 1991.
- [22] J. Piaget. *Child's Conception of Space: Selected Works*, vol. 4. Routledge, 2013.
- [23] M. Resnick. All i really need to know (about creative thinking) i learned (by studying how children learn) in kindergarten. In *Proceedings of the 6th ACM SIGCHI Conference on Creativity & Cognition, C&C '07*, pp. 1–6. ACM, New York, NY, USA, 2007. doi: 10.1145/1254960.1254961
- [24] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. Scratch: Programming for All. *Commun. ACM*, 52(11):60–67, Nov. 2009. doi: 10.1145/1592761.1592779
- [25] H. Romat, N. Riche, K. Hinckley, B. Lee, C. Appert, E. Pietriga, and C. Collins. Activeink:(th) inking with data. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, 2019.
- [26] B. Shneiderman. Direct manipulation: A step beyond programming languages. *Computer*, 16(8):57–69, Aug 1983. doi: 10.1109/MC.1983.1654471
- [27] T. Software. Business intelligence and analytics, 2018.
- [28] G. Stiny. Kindergarten grammars: designing with froebel's building gifts. *Environment and Planning B: Planning and Design*, 7(4):409–462, 1980. doi: 10.1068/b070409
- [29] T. Wun, J. Payne, S. Huron, and S. Carpendale. Comparing Bar Chart Authoring with Microsoft Excel and Tangible Tiles. *Computer Graphics Forum*, 2016. doi: 10.1111/cgf.12887
- [30] H. Xia, B. Araujo, T. Grossman, and D. Wigdor. Object-oriented drawing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, CHI '16*, pp. 4610–4621. ACM, New York, NY, USA, 2016. doi: 10.1145/2858036.2858075