Skeleton: Visual Authoring of Non-visual Data Experiences

Frank Elavsky, Chieri Nnadozie, Lucas Nadolskis, Patrick Carrington and Dominik Moritz

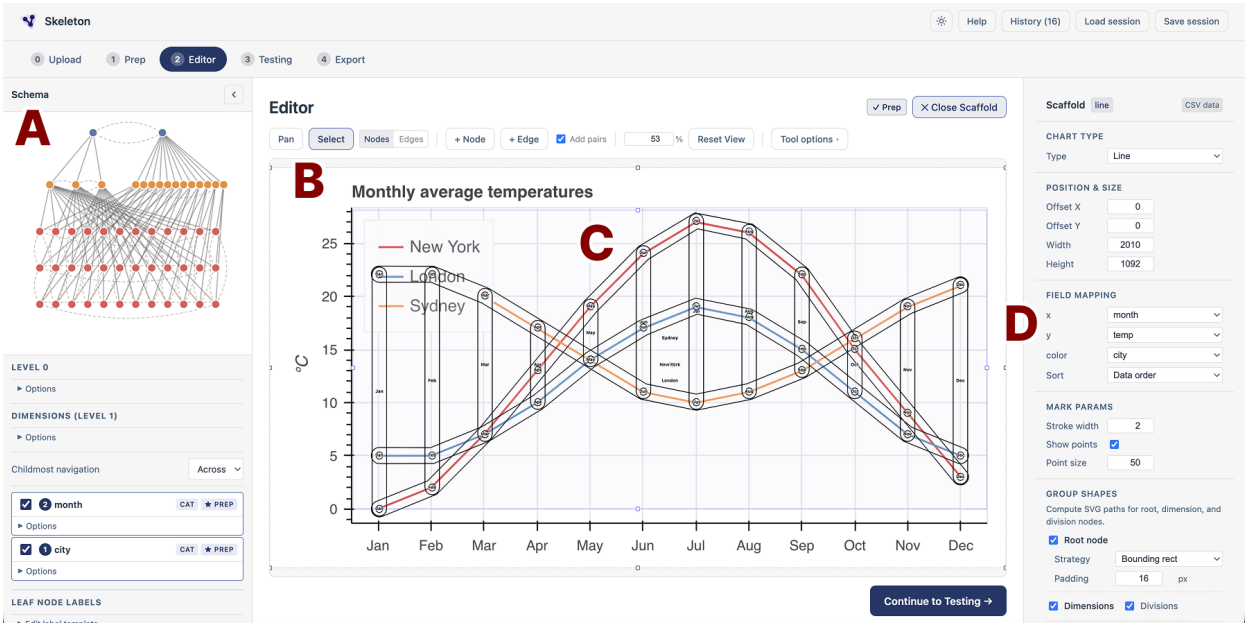

Fig. 1: *Skeleton* being used to build a navigation structure over a line chart. **A**: a dual tree visualization of nodes and edges built for a line chart with 3 lines, spanning over 12 months. **B**: a direct-manipulation canvas that resides over a static png image of a line chart. **C**: a skeleton-like structure of nodes, edges, and group outlines that correspond to the schema shown in (A), overlaid on the visual space of the chart in (B). **D**: controls for rapidly scaffolding the nodes, edges, and their shapes shown in (C), using Vega's visualization engine for automatically generating spatial properties.

Abstract—When sighted practitioners author accessible data visualizations, they build navigation structures (the nodes, edges, and input bindings that govern how assistive technologies traverse an interface) entirely in code, with no visual representation. This invisibility makes navigation structures difficult to inspect, debug, and iterate on. To sighted practitioners, every other aspect of a visualization is iterated on because it is visible; navigation structure ships as a first draft, if at all, because it is not. Without a representation to react to, practitioners cannot develop judgment about what makes navigation good or bad, and the quality ceiling of non-visual experiences is set by the absence of a feedback loop. We address this problem through longitudinal co-design with practitioners across cartography, design systems, and open-source visualization, and make three contributions. First, we introduce technical advancements for making the properties of accessible navigation structure visible and directly manipulable during authoring, grounded in two foundational pieces of infrastructure produced by our co-design work: an *Inspector* that renders navigation graphs as interactive node-link diagrams, and a *Dimensions API* that expresses navigation in terms of data dimensions rather than explicit graph construction. Second, building on these, we present *Skeleton*, a direct-manipulation authoring environment in which the properties of an accessible navigation structure are translated into visual representations authors can observe and manipulate. Key techniques include a dual-view editor that simultaneously shows the system's navigation model and the end user's spatial experience, a scaffolding engine that automates spatial node placement by repurposing a visualization rendering pipeline, a live label-template editor with real-time screen-reader-output preview, and a testing mode that makes traversal sequence visually trackable. Third, we evaluate *Skeleton* through an in-situ study with 8 practitioners across visualization design, engineering, and research. Making navigation structure visible changed how practitioners engaged with accessible design: they reconsidered the architecture of their own visualizations, attended to a broader range of input modalities, and shifted from treating accessibility as a compliance task to treating it as a design problem.
A free copy of this paper and all supplemental materials are available at https://osf.io/7afw4.

Index Terms—visualization, accessibility, non-visual design

1 Introduction

We start this work with a provocation: How might the discipline of visualization help the discipline of accessibility?

Visualization has spent decades developing techniques for a specific class of problem: representing and interacting with information visually. Grammars of visual encoding [43,51,68,69,74], direct-interaction interfaces [15,27,60], and iterative feedback loops between representation and understanding [19,23,37] are all methods that enable abstraction

• *Elavsky, Nnadozie, Carrington, and Moritz are with Carnegie Mellon University. E-mail: { fje | pcarrington | domoritz }@cmu.edu, cnnadozi@andrew.cmu.edu*

• *Nadolskis is with UC Santa Barbara. E-mail: lgilnadolskis@ucsb.edu*

Manuscript received xx xxx. 202x; accepted xx xxx. 202x. Date of Publication xx xxx. 202x; date of current version xx xxx. 202x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org.

Digital Object Identifier: xx.xxxx/TVCG.202x.xxxxxxx

and manipulation of the information that underlies the visual representation. We argue these methods have a direct application within the discipline's own accessibility challenges.

Sighted practitioners who build accessible data visualizations face an unusual authoring problem. The non-visual navigation structures they construct (the nodes, edges, focus states, input bindings, and semantics that govern how assistive technologies traverse a chart) exist only as code. A practitioner can write a navigation hierarchy, but cannot see it, click on a node to inspect what will be announced, observe the spatial relationship between a navigation path and the chart it overlays, and then manipulate its properties through direct interaction. Every other aspect of a visualization has a visible, inspectable representation during authoring: the visual encodings are visible, the layout is visible, the interaction states are visible. And yet, navigation structure is not.

This invisibility has practical consequences. Without a way to see what they are building, sighted practitioners cannot easily catch structural errors, compare design alternatives, and iterate. Accessibility becomes downstream of every other design choice because the authoring conditions do not support anything else [30, 57]. The floor and ceiling of non-visual data experiences are constrained by what sighted authors can perceive of their own work.

We engage this space with the following research questions:

R1 (Qualitative, Exploratory): What challenges do sighted practitioners face when designing and engineering navigation structures for accessible visualizations?
R2 (Qualitative, Exploratory): How do sighted authors reason about the non-visual experiences that accompany their visualizations?
R3 (System, Design): How can we make the properties of accessible navigation structure visible and directly manipulable during authoring?
R4 (Qualitative): In what ways does a directly manipulable visual representation of navigation structure change how practitioners find errors and improve upon their designs?

We address these questions through longitudinal co-design with practitioners across cartography, design systems, and open-source visualization, following an action research orientation [22] in which the research team was embedded in each community's active development work. This paper makes three contributions:

First, **we introduce technical advancements** for making the properties of accessible navigation structure visible and directly manipulable during authoring. These techniques are grounded in our co-design collaborations, which produced two foundational pieces of infrastructure: an *Inspector* that renders any navigation graph as an interactive node-link diagram, and a *Dimensions API* that formalizes a declarative grammar for expressing navigation in terms of data dimensions rather than explicit graph construction (**R1**, **R2**, **R3**).

Second, building on this infrastructure, **we present *Skeleton*, a direct-manipulation authoring environment** in which the topology, spatial mapping, semantics, and input logic of an accessible navigation structure are made visible and directly manipulable. *Skeleton* is built on Data Navigator [14], a code-based library for constructing interactive data navigation structures (**R3**). Our intention with *Skeleton* is to continue to develop it towards a usable, practical system beyond the scope of this research.

Third, **we contribute findings from an in-situ interview study** with 8 visualization practitioners. We qualitatively evaluate Skeleton as a design probe [20, 28], focusing on how it influences practitioner engagement. We find that making navigation structure visible shifted how participants engaged with accessible design: they reconsidered the architecture of their own visualizations, attended to a broader range of input modalities, and shifted from treating accessibility as a compliance task to treating it as a design problem (**R1**, **R2**, **R4**).

2 Related Work

2.1 Visualizing Non-visual Structure

Long before accessibility, computing established a practice of taking structures that are not inherently visual and giving them a visual form that can be seen and acted upon. Sutherland's Sketchpad was one of the earliest; it turned the constraints and topology of a drawing into a manipulable graphical object [63]. Later, visual programming environments and syntax-directed editors rendered computation and structure as editable visual objects [61, 64]. Myers later named the distinction this lineage had been drawing, separating *visual programming* from *program visualization*, depicting otherwise non-visual programs graphically [46]. As a consequence, many structures we now treat as inherently visual are abstractions whose visual rendering has become so dominant that the representation now stands in for the structure.

Direct manipulation interfaces, in turn, provide a continuous representation of the objects of interest and give rapid, incremental feedback, letting people work directly with a structure they would otherwise only manipulate indirectly [27, 59]; grammars of interactive graphics extend the same move to authoring [5, 51]. Yet across this tradition, one structure has remained conspicuously without a visual form to manipulate: the UI structure that a screen-reader traverses. This lack of visual representation is especially surprising because screen readers evolved in parallel with the emergence of graphical interfaces [65].

2.2 Non-visual Data Experiences

Blind people who rely on assistive technologies interact with data in fundamentally different ways than sighted users who use a direct pointer, like a mouse [36, 42, 55]. A substantial body of research has documented what these experiences look like across modalities, and what it takes to make them meaningful. In the context of "data experiences," this paper focuses specifically on interactive navigation structures, but we briefly survey adjacent modalities to situate our contribution.

Alternative text and natural language. A dominant strand of this work concerns the generation and evaluation of textual descriptions of visualizations [31, 32, 35, 39]. More recent LLM-driven systems and Q/A approaches can caption charts with varying degrees of semantic depth, some at a risk of producing bias [12, 18, 34]. While alt text makes a visualization's message available without sight, it is by nature static: a description conveys what a visualization says, but not how a user might explore or interact with it.

Sonification, haptics, and tactile rendering. Non-visual data experiences extend well beyond text. Sonification encodes data as sound [7, 24, 40], with declarative grammars emerging for authoring these experiences [33]. Haptic and tactile representations offer another channel through refreshable displays, 3D-printed graphics, and multimodal touchscreen interactions [8, 25, 41, 50]. Recent systems integrate multiple non-visual representations of the same data [9, 26, 54].

Interactive navigation structures. The primary focus of our work centers on the state of research related to structured navigation: the traversal of data points, groupings, and interface elements through assistive technologies and keyboard input [14, 62, 67, 72]. Existing systems and interfaces have demonstrated that going beyond static descriptions to support hierarchical, traversable data structures meaningfully improves how blind users can explore and reason about charts [66, 72]. Giving users control over the textual tokens surfaced during navigation improves comprehension and agency [29]. And more recent work has found that perceptually congruent navigation structures for charts and diagrams can improve goal-driven exploration [44].

2.3 Authoring Non-visual Data Experiences

Accessible visualization has historically centered the experiences of disabled users, but a parallel body of work examines the experiences of visualization practitioners working on accessibility.

Practitioner challenges. Research consistently finds that sighted visualization practitioners struggle with accessibility [17, 30, 57]. Most visualizations in the wild are inaccessible and designers themselves report lacking guidance, especially for complex and interactive graphics [30, 57]. And screen reader users experience the downstream effects of these gaps: inconsistent structure, poor keyboard support, and information that is present visually but absent in the accessibility tree [17, 55]. The pattern across this work is consistent: the practitioners who build visualizations lack the tools and feedback mechanisms to make non-visual experiences effective, useful, and good.

Across practitioner-centered literature, a recurring finding is that non-visual experiences are treated as downstream of visual design choices, added after the visual representation is finished rather than designed in parallel [30, 38, 57, 73]. This sequencing has consequences: what is navigable and how it is structured is constrained by whatever visual decisions came first.

Authoring-oriented systems. A distinct line of work has focused on building authoring tools and libraries that give practitioners more tractable paths to accessible output. Few visualization tools support the kinds of interactive navigation structures that assistive technology users most benefit from [35]. Of those that do, most rely on code-based approaches [3, 14, 56, 58]. *Umwelt* [73] takes a different and notable approach: it is a structured editing environment where authors specify representations *across* modalities (sonification and visualization) in an integrated interface, where navigation is made available over the visualization using *Olli* [3]. The latest work in this space is *Arboretum*, a tool that provides automatic conversion of diagrams to a tabular structure, navigable structure, and tactile representation [70].

Communicating visually, authoring invisibly. There is a revealing irony across this body of related work: research about navigation structures almost invariably communicates those structures visually. Papers such as "rich screen reader experiences" [72], *ChartReader* [66], *Benthic* [44], and *Data Navigator* [14] each use visual node-link diagrams and hierarchical schematics to explain navigation paths to their readers. The same pattern holds in adjacent domains that involve structuring data for navigation, such as PDF remediation [45].

Yet, most other non-visual modalities are not only communicated but *authored* in visual and non-visual modalities. Sonification is designed in editors that are both visual and auditory [33]. Tactile graphics are composed by transcribers on a graphical canvas before being physically embossed or printed [4].

Inspecting accessibility structure. In authoring-oriented systems, none provide a visual interface through which practitioners can interactively inspect and manipulate navigation structures as a first-class design material. Structure output is either downstream of code or static visuals. Structure is always *derived* or *specified*; indirectly manipulated. Additionally, verification of the structure across all of these systems requires developers to manually navigate using a screen reader after the structure has been authored and rendered, before returning to the upstream visual design space or code. Navigation structure is thus understood and communicated visually by sighted researchers and designers, yet built entirely without visual feedback by developers; this is the gap we address.

3 Co-design Foundations

Our research follows an *action research* orientation [22], in which knowledge is generated by engaging with a community to solve a real problem in-situ, alongside them rather than studying it from the outside. These collaborations started from a shared motivation: practitioners needed to make their existing systems accessible to navigational assistive technologies, using Data Navigator [14] as a foundation. Across three projects, we worked with 12 individuals outside our research team. Three blind co-designers shaped the work throughout: CD1 and CD2 (anonymous) and a co-author on this paper, Nadolskis. CD1's contributions are noted in Section 3.1, CD2's are noted in Section 4.3. Nadolskis contributed to early ideation, problem formation, and framing for the project as a whole, helping define the authoring challenges that *Skeleton* addresses, in addition to feedback on study design.

The co-design literature on accessibility has centered people with disabilities as design partners [10, 11, 48, 66, 71, 72]. Our co-design takes also sighted practitioners as partners because the authoring challenges we address happen on their side of the process: it is sighted authors who cannot see what they are building. This work was exempted by our IRB (2024_00000040).

Two themes converged across all three engagements, and we present them here before describing the individual projects that surfaced them. First, **practitioners communicated and reasoned about accessible navigation using visual representations**: while designing for language, sound, and structure, our collaborators drew on paper, built wireframes of nodes and edges, and reflected on the design space using visual artifacts. Even when collaborating with blind co-designers, a visual medium was the first language of sighted authors. Second, **development that followed visual design work faced severe iteration barriers**: verifying a navigation experience required building a working code prototype and manually navigating it with a screen reader. The gap between visual design and code-based scaffolding with manual testing produced repeated mistakes and misinterpretations.

These themes did not arrive as abstractions; each project below surfaced them as concrete friction, and together the three engagements produced five design goals that *Skeleton* was built to satisfy. We state them here and, in the subsections that follow, mark where each emerged:

DG1 Make structure visible and inspectable. Render the navigation structure as a manipulable object during authoring, not only as code, so that practitioners can perceive and react to it.

DG2 Specify navigation at the level of data. Provide a higher-level, data-driven abstraction so practitioners describe traversal in terms of data dimensions rather than hand-wiring every node and edge.

DG3 Shorten the iteration loop. Let practitioners verify a design without a full code build or a manual screen-reader pass.

DG4 Support image-only and bespoke visualizations. Do not require a single underlying dataset, format, or recognizable chart type, since the cases that most need accessible tooling often have neither.

DG5 Reach beyond keyboard input. Accommodate input modalities other than assuming keyboard navigation, including touch and text.

3.1 Geologic Map

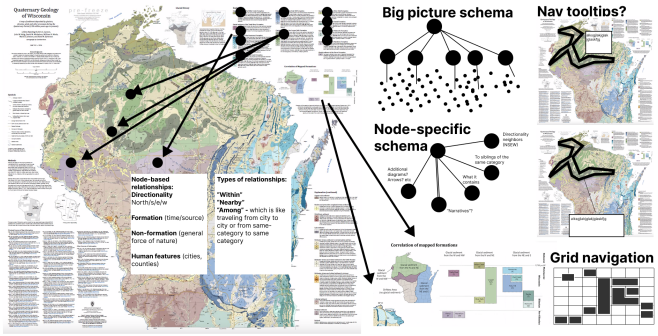

Fig. 2: Our visual design work in Figma over a static geologic infographic map of Wisconsin. We use visual forms and illustrations over and beside the map to communicate flows, structure, navigation styles, and interaction patterns.

Our longest engagement spanned just over two years, with a cartographer, accessibility consultant, and blind co-designer (CD1) building an interactive version of a quaternary geologic map of Wisconsin [49]. The map combined dozens of irregular geographic regions organized categorically by a legend.

Early design sessions took place in Figma (Figure 2), where we laid out node-link diagrams of how a screen reader user might traverse the map, legend, and peripheral information. We annotated each node with text a screen reader would announce and connected them to relevant regions. Drawing the structure made it possible to discuss design choices, catch dead ends, and debate traversal strategies. We were informally doing what *Skeleton* later formalized.

The friction began when we moved from design to implementation. We had no way to verify that our designed structure would be good or ideal, and scaffolding the project into Data Navigator was arduous: the translation from even simple navigational designs to functional code was too complex to hand off or meaningfully iterate on. The collaboration also surfaced a design-space boundary: for within-map spatial navigation, an egocentric audio-game approach [2] fit better than a node-link graph, a distinction CD1 helped surface. Reaching this boundary was only possible because we could see the structure we were trying to build (**DG1**). And the map's lack of a single tabular dataset

made clear that a useful navigation tool should not assume structured data or a grammar of graphics (**DG4**).

3.2 Design System Library

Our second engagement, spanning 7 months, was with 6 engineers and designers on Adobe's React Spectrum Charts library, an open-source chart component system. We worked in their codebase and in Miro on navigation design for bar charts, clustered and stacked bars, line charts, and related types. Because we needed generalized, reusable patterns, our design problems differed from the geologic map: we needed to account for use, re-use, and edge cases across chart types.

Our Miro sessions produced two kinds of artifacts: diagrams of navigation structure for specific chart *instances* and *schema* diagrams capturing generalized patterns in dimensional terms. The concept of a "dimension" emerged naturally: common transformations on Data Navigator's graph structure corresponded to properties within a dataset, such as a "categorical" dimension or a "numerical" dimension, where traversal took place within collections of grouped siblings. This dimensional thinking directly foreshadowed the Dimensions API (Section 3.4).

The dominant friction was iteration speed. We could sketch and converge on a schema in Miro in an hour, but verifying the design in a functional example required embedding Data Navigator into a large codebase, implementing changes, rebuilding, and manually testing with a screen reader. The collaboration also surfaced a limitation: mobile screen reader navigation uses swipe gestures rather than keyboard input, and our keyboard interaction model did not account for it. Each of these frictions pointed somewhere: the dimension concept toward a higher-level abstraction (**DG2**), the cost of verifying each change toward a faster loop that does not require a full build or screen reader (**DG3**), and the mobile-gesture gap beyond keyboard input (**DG5**).

3.3 Open Source Visualization Library

Our third collaboration, spanning nearly two years, was with Quansight Labs and contributors to Bokeh, a Python-based open-source visualization library. We performed an accessibility audit [16] based on Chartability [13] and identified that Bokeh visualizations with interactive chart elements needed to be navigable by assistive technologies.

Unlike Adobe, Bokeh has no native chart types. Its API operates at the level of glyphs, renderers, and data sources, which users assemble freely. There was no standard unit around which to anchor a navigation pattern, and any grammar or tooling would need to accommodate an open-ended range of encoding combinations. The iteration gap from Adobe was present in a more severe form: making library-wide contributions to a fully open-source project required incremental tooling for the most common cases. For Bokeh, we needed to test functional, data-driven navigation abstractions without fully embedding Data Navigator into the library. A library with no native chart types required the abstraction to be data-driven and chart-type-agnostic rather than tied to fixed templates, sharpening **DG2** into a concrete constraint.

3.4 Infrastructure from Practice

The three collaborations converged on the design goals above. Two of them demanded new infrastructure before *Skeleton* could exist: a way to visually render and inspect navigation structures (**DG1**), and a higher-level, data-driven abstraction for specifying navigation without hand-wiring every node and edge (**DG2**). We built two pieces of infrastructure to address these, described next; the remaining goals (**DG3**, **DG4**, **DG5**) are met by *Skeleton*'s authoring environment (Section 4). These infrastructural improvements as well as *Skeleton* are open source on our GitHub repo.

3.4.1 An Inspector Gadget

We built an *Inspector* (`@data-navigator/inspector`) to render any Data Navigator structure as an interactive node-link graph using D3, with an accompanying console for debugging. Hierarchical structures are colored by level; edges are drawn as directed links; the entry point is visually marked. The *Inspector*'s graph can itself be navigated using Data Navigator, with visual focus tracking during navigation, allowing practitioners to manually verify structure and reachability.

This made structural verification immediate: a practitioner could generate a structure and check at a glance whether the hierarchy had the right levels, whether circular extents produced expected wrap-around edges, or whether a particular path was reachable. The interactive console logs API information and underlying data when nodes are activated, and hovering or focusing logged information highlights the corresponding node in the graph.

The *Inspector* remains a developer tool, however. It requires code familiarity to attach and renders structure as an abstract graph with no connection to the spatial layout of the underlying visualization. It shows topology but not instantiated geometry: a practitioner can see that two nodes are connected but not where their focus indicators will appear on-screen. This gap between navigable structure and its spatial instantiation over a rendered chart motivated *Skeleton*.

3.4.2 Alternative Dimensions

The original Data Navigator library requires explicit graph construction: practitioners specify nodes, edges, and navigation rules by hand. This is general but scales poorly. Our *Dimensions API* addresses this.

Our new vocabulary mirrors how practitioners already think about their data, much as a grammar of graphics lets a designer write a visualization at the level of data fields rather than primitive marks. Rather than specifying a graph directly, a practitioner describes the *dimensions* of their data, the meaningful axes along which a user might want to navigate, and a single build step constructs the full node-link structure automatically. Notably, this enables systems to programmatically construct navigable structures as long as dimensions are configurable. Each dimension declares a `type` (`categorical` or `numerical`) and a `dimensionKey` naming the field it traverses, together with a `behavior` block that governs traversal (how navigation behaves at group boundaries, and whether leaf nodes are reachable laterally across divisions) and an `operations` block that governs how the data is shaped into the hierarchy (sorting, filtering, and binning numerical ranges). Section A gives a deeper look into the API's parameters.

This data-dimensions framing both aligns with and departs from *Olli* [3], which similarly lets authors express a navigable structure over a chart from a higher-level specification rather than hand-built nodes. Both share the premise that fields, not graph primitives, are the right authoring unit. The difference is what the specification produces and where it lives: *Olli* compiles a chart specification into a traversable tree that an author does not subsequently manipulate as a graph, whereas the *Dimensions API* generates an explicit Data Navigator node-link structure whose every node, edge, and rule remains addressable, so that the declarative output is itself the object a practitioner can then inspect in the *Inspector* and edit directly in *Skeleton*.

4 *Skeleton*: System Design

With a visual structure renderer (the *Inspector*) and a declarative abstraction for producing navigation structures (the *Dimensions API*), the remaining problem was to bring these capabilities into an integrated, direct-manipulation authoring environment where practitioners could not only see structure but manipulate, test, and iterate on it with immediate feedback. *Skeleton* is that environment. It integrates both and extends them with a guided preparation phase, a spatial rendering canvas, and a live testing mode, reversing the *Inspector*'s code-first direction: practitioners design a navigation structure visually and inspect the code representation as a consequence of that design. Each of its authoring techniques makes visible a specific property of non-visual interaction that sighted practitioners otherwise cannot see during authoring: the topology of what is navigable, the spatial mapping of where navigation lives over the chart, the semantics of what is announced, and the temporal sequence in which a user encounters nodes.

A principle runs through the environment: everything *Skeleton* generates is an editable default, not a fixed output. The *Dimensions API* proposes a topology, the scaffold proposes spatial positions and group outlines, and the preparation wizard proposes labels and rules, but each is materialized as concrete nodes, edges, positions, and text that a practitioner can then select and override by direct manipulation, or bypass entirely by constructing nodes and edges manually over an image. The

declarative and automated layers lower the cost of a reasonable starting point; they do not constrain what the final structure can be.

4.1 Staging: Input and Preparation for Authoring

The authoring workflow proceeds through four stages: upload, prepare, edit, and test. Making a visualization accessible involves decisions about what is navigable, how navigation is triggered, and where focus indicators appear in space. These decisions are related but distinct, and collapsing them into a single undifferentiated interface, as code-only workflows effectively do, makes each one harder to reason about. The stage architecture surfaces them as separate concerns. It also preserves a direct correspondence between what practitioners see in the interface and the structure of the Data Navigator API: each stage maps onto a distinct layer of the API, lowering the barrier to moving from visual authoring into code when production deployment requires it.

Upload. The upload phase is deliberately permissive. Practitioners can bring a dataset, a chart image, both, or neither. When a dataset is present, *Skeleton* parses its fields and infers dimension types automatically, producing a default, starting configuration for the *Prepare* step. When only an image is present, practitioners proceed directly to editing and construct nodes and edges manually over the image. Because dimensions are derived from whatever fields a dataset actually carries rather than from a fixed, pre-registered schema, the approach does not require the data to be known in advance: a different dataset simply yields a different set of inferred dimensions, and a visualization with no tabular dataset at all is handled through the image-only path. This was motivated by our geologic map co-design work: many bespoke visualizations do not have a single underlying dataset, and any tool that requires structured data as a precondition excludes the cases that need it most. *Skeleton* can be applied to any 2D image surface, not only to visualizations in the conventional sense.

Prepare. The prepare stage addresses a hard authoring bottleneck: not placing nodes, but deciding what structure to build at all. A practitioner who has never designed a navigation structure faces an open configuration space with no obvious entry point. The prep stage presents a four-chapter Q&A wizard that moves through authoring decisions sequentially: (1) whether the chart should have a root node and what it should announce, (2) which data fields should be navigable dimensions and how those dimensions should behave at their boundaries, (3) which keyboard interactions each dimension should be assigned to, and (4) what text labels each level of the hierarchy should produce. Each chapter is accompanied by an illustrative schematic diagram of which part of the hierarchy is being edited as well as a diagram showing examples of what these decisions look like when showing on a chart. The wizard's output populates a configuration in the editor that practitioners can then inspect, refine, and revise.

4.2 Edit: Interacting with Topology, Layout, and Semantics

Seeing the system, seeing the experience. The editor is *Skeleton*'s primary authoring environment (Figure 1) and presents two interlinked representations of the same navigation structure. A schema panel shows the structure as an abstract hierarchical tree layout that makes levels and parent-child relationships immediately readable. A graph canvas shows the same structure rendered as geometric elements positioned over the uploaded chart image, representing what an end user would encounter spatially. These two representations are bidirectionally linked: selecting a node in either view propagates the selection to the other, so practitioners can simultaneously hold in mind both the abstract topology of what is navigable and the spatial instantiation of where that navigation will live. The dual-view design makes visible a real conceptual divide between the system's model of navigation and a user's experience of it, one that practitioners recognize once they can see it, even without prior vocabulary for it.

Leveraging visualization as a scaffolding engine. Manually positioning leaf nodes over each data mark is the most mechanical step in the authoring workflow, and it scales poorly with dataset size. In our early pilot sessions, actually placing nodes in the canvas space was the slowest and most tedious part of the process. To address this, *Skeleton* includes a scaffold tool (Figure 3) that automates spatial placement by repurposing Vega [51] as a layout engine.

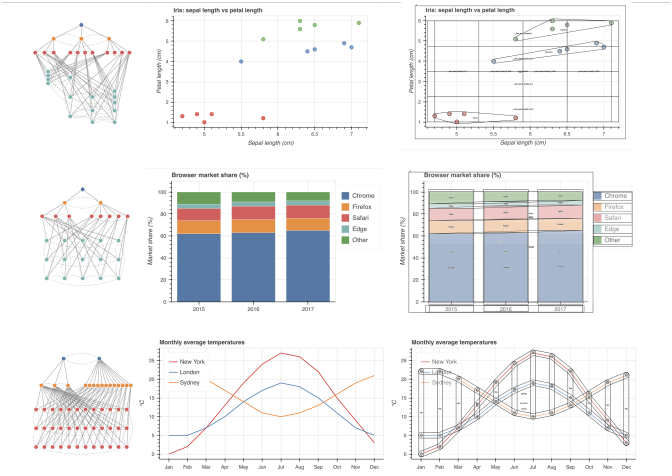

Fig. 3: Input data transformed into a navigable structure using the *Dimensions API* and visualized with our *Inspector* gadget (left). The input chart (middle). The navigable structure is transformed and drawn over the chart using the *Scaffolding Engine* (right).

The scaffold renders a Vega chart specification to a hidden, off-screen container and extracts node positions via the library's internal scale and view APIs, using the rendering engine purely as a coordinate computation step: no actual chart is ever shown to the practitioner. Coincidentally, none of our co-designers were crafting visualizations using Vega or Vega-Lite (or derivatives), yet the Vega rendering engine could faithfully reconstruct every necessary mark position over the underlying, inaccessible data visualization provided to *Skeleton*.

Additionally, positions and outline strategies for category-level group nodes are computed from the leaf positions using geometric algorithms: a union of child node paths, a convex hull, a grid over numerical bounds, or a bounding rectangle. These designs were consistently produced as ideal treatments during co-design, so we chose them for our starting set of outline strategies.

The scaffold is optional and works by generating synthetic placeholder positions that practitioners can adjust manually. It dramatically improved authoring speed: in light pilot tests, a research team member completed a three-dimension structure without scaffolding in 8 minutes 22 seconds and with scaffolding in 56 seconds. A co-designer completed the task incorrectly (failing to account for one dimension's division nodes) in 13 minutes 7 seconds without scaffolding, and correctly in 2 minutes 44 seconds with it.

Specifying token patterns, editing instances. Selecting any node populates a properties panel with spatial properties (position, size, shape) and semantic properties (ARIA role, description, and a label template editor; Figure 6). The label template allows practitioners to assemble the text a screen reader will announce at that node from tokens drawn from data fields, including aggregate statistics at group-level nodes and precise data values at leaf nodes [29]. Editing labels can apply to all nodes of the same type or to a single instance.

At the bottom of the semantic section, a live preview displays the full assembled announcement string in the exact form a screen reader would produce: role, semantics, group membership, and label combined into a single rendered output that updates in real time. Prior to this preview, understanding what would be announced at a given node required running a screen reader and navigating to it sequentially. In deeply nested structures, this meant spending several seconds listening to labels and drilling in. This was consistently the main point where bugs were produced and missed during our co-design work.

The preview makes text announcements inspectable as visible, editable objects. This surfaces a class of highly specific, low-level problems that code-only workflows leave invisible: redundancies in an-

nounced text, missing contextual framing, and label ordering and punctuation that affects comprehension and reading speed. Of all the authoring decisions *Skeleton* exposes, label templates involve the most degrees of freedom and have the most direct bearing on the quality of the non-visual experience. Getting the structure right ensures navigability; getting the labels right determines whether navigation communicates anything meaningful.

4.3 Test: Debugging Interaction Interactively

The testing stage allows practitioners to navigate the structure they have built using the same keyboard input and navigation rules that assistive technologies would use, without leaving the tool. When a practitioner enters testing, the three Data Navigator modules are instantiated in sequence: the structure module rebuilds the navigation graph from the current configuration, the rendering module creates an HTML layer positioned over the chart image at each node's spatial coordinates, and the input module registers keyboard listeners for all navigation rules. The result is a live, keyboard-navigable structure. An event log records navigation events in order, letting practitioners verify that all nodes are reachable and that label sequences make sense when encountered serially rather than read simultaneously.

The abstract graph continues to display during testing. As the practitioner navigates, the focused node is highlighted simultaneously in the canvas (showing its spatial position over the chart image) and in the abstract graph (showing its structural position in the hierarchy). Practitioners can verify at a glance both where focus is and what role it occupies. A mobile-friendly text-chat mode is also available, in which practitioners navigate by typing natural language commands, motivated by the Adobe collaboration (Section 3.2).

Practice-based validation. The testing stage also served, during development, as the primary debugging interface for *Skeleton*'s own data pipeline. Additionally, CD2, a subject matter expert who professionally evaluates interfaces for screen reader access and is familiar with other visualization navigation systems, used *Skeleton*'s testing stage to evaluate navigation output across several chart types and dimension configurations: line charts (3 configurations), bar charts (2), stacked bar charts (3), and scatterplots (4). This evaluation followed a manual, systematic approach combining standards-based criteria with expert screen reader testing. Scatterplots required the most iteration, surfacing bugs in Data Navigator's core library that were then fixed. CD2 also recommended that we use list-based navigation while in the editor (not in testing) in case users build themselves into a keyboard trap.

5 User Study

Our co-design work was motivated by the problems of the communities the research team was embedded in, and our co-designers were deeply familiar with the problem space, having spent months or years working on accessible navigation. We still needed to understand what happens when practitioners who are *not* embedded in this process design, author, and debug navigation structure visually: whether the representations we built are legible to them, whether the techniques change how they reason about accessible design, and what new questions or problems emerge when navigation structure becomes visible. These are empirical questions that required a study.

To evaluate how *Skeleton* influences the way practitioners engage with accessible design, we conducted an in-situ interview study with 8 participants across visualization design, engineering, and research. The study was conducted remotely over video call, took approximately 45–60 minutes per session, and was approved by our IRB (2025_00000285). Participants gave informed, written consent before our sessions and provided verbal consent at the start of each session. Video and audio were not recorded, however some participants consented to share the data/image they brought to the study as well as screenshots of their work; data collection was note-based throughout.

5.1 Participants

Participants were recruited through snowball sampling within the visualization and accessibility community, and through referrals from co-designers involved in our earlier collaborations. We asked each participant to self-report their primary work role (engineering, research, design, or student) and their existing level of accessibility expertise on a 1–5 Likert scale. We also asked whether the visualizations they build are ever bespoke, that is, custom rather than instances of a recognizable, standard chart type. This distinction mattered because bespoke visualizations represent an especially underserved case in accessible design tooling: no library pattern applies, and every navigation structure must be designed from scratch. Participants were not compensated.

5.2 Procedure

Each 45-minute session proceeded in four phases. Before the session, all participants were asked to prepare a chart image they were currently working on or had recently built, for use in the third phase.

Phase 1: Introduction and demographics (5 minutes). After obtaining verbal consent and recording a pseudonym, we collected self-reported role, accessibility expertise level, and whether the participant regularly builds bespoke visualizations.

Phase 2: Generic chart think-aloud [1] (10 minutes). Participants used *Skeleton* on a provided bar chart and dataset of fruit counts (Apples, Pears, Nectarines, Plums, Grapes), asked to design an accessible navigation experience for a screen reader user with no instructions on how the tool worked. The tool loaded with a default structure having both a categorical dimension (`fruit`) and a numerical dimension (`count`) active. This default was intentionally problematic for two reasons: numerical navigation sorts by count value and groups data into subdivided ranges, producing a traversal order different than the visual layout and adds an additional, largely unhelpful level in the hierarchy for such a simple chart. We observed how participants reasoned about what they saw and whether and how they noticed this extra dimension.

Phase 3: Own-chart think-aloud (15 minutes). Participants loaded their own chart image and attempted the same task, except they were also asked to explain their graphic to the research team (purpose, role, data, and domain). This phase was open-ended: charts ranged from standard types to bespoke visualizations, and the goal was to observe how participants reasoned about navigation structure when the context was their own work.

Phase 4: Reflective interview (15 minutes). We conducted a semi-structured interview in which participants reflected on their decisions in Phase 2 versus Phase 3, their experience with the generic versus their own chart, and their assessment of the tool's capabilities and limitations. We asked what felt possible or impossible, what they wanted to do that they could not, and what they found themselves thinking about that they had not considered in Phase 2.

5.3 Analysis

Notes from each session were compiled into a shared document. Participant quotes reported in the results are reconstructed from these researcher notes, not verbatim transcripts. We analyzed the data using a combination of thematic analysis [6] and affinity diagramming [21], iterating across both methods to surface recurring patterns while preserving the specificity of individual participant experiences. Analysis attended particularly to differences in how participants engaged with accessible design before and after using the tool, the range of input modalities and user scenarios they considered, and moments when participants reconsidered or wished to redesign their own visualizations.

6 Results

We organize our findings into five themes that emerged from thematic analysis and affinity diagramming across all eight sessions. Each theme captures a qualitative pattern in how practitioners engaged with accessible navigation design when its structure was made visible and manipulable. We report these findings descriptively and ground them in specific participant moments; interpretation follows in the Discussion.

6.1 Seeing Navigation Made Structural Problems Legible as Design Problems

The generic bar chart in Phase 2 loaded with an intentionally problematic default: both a categorical dimension (`fruit`) and a numerical dimension (`count`) were active, producing overlapping navigation

structures with different traversal orders over the same data. This configuration is a poor design choice, arguably a design failure, but one that would be difficult to detect in code alone (**R1**, **R4**).

Participants varied widely in how quickly they recognized the problem. P1 turned off the numerical dimension within seconds of seeing the editor, without commenting on it. Most participants, however, initially struggled to understand what they were seeing, remarking on the unfamiliar structure: "what is this? what are these?" when encountering the numerical divisions for the first time. P8 spent time trying to guess what the extra divisions represented but did not remove them during Phase 2, only realizing during Phase 3 that the additional dimension was "probably bad." P2 and P5 expressed suspicion early: P5 asked, "Is this too much data? This seems like way too much to just navigate through," and P2 noted, "I feel like a lot of data points would be bad, yeah? Like, too many at once is bad?"

The testing stage (Section 4.3) proved critical for resolution. P4, P5, and P7 each removed the extra dimension only after navigating the structure with keyboard input in testing mode, where the traversal sequence made the redundancy experientially apparent. In total, five of eight participants resolved the problem during the session: P1 and P2 during editing, and P4, P5, and P7 after testing.

Beyond the intentional default, participants identified other problems through visual inspection. P8 reacted to a generated node name: "Okay dim_fruit node... that is horrible, what is that?" During Phase 3, P7 looked at the edges of their multi-line chart and asked, "are all these bad? Is it bad that I don't even really know what the takeaway of this [structure] is?" P3, seeing a full hierarchy for their own simple six-item bar chart (during Phase 3), concluded "I should just skip the root and grouping and go straight to the data. This seems like too many steps." In each case, the visual representation of their navigation structure motivated judgment about potential negative design qualities.

6.2 Practitioners Developed a Designerly Interest in What Constitutes Good Navigation

The most pervasive pattern across sessions was that participants began asking design questions about navigation quality, unprompted by any instruction or guidance from the research team (**R2**). These questions went beyond identifying errors: participants wanted to know what *good* navigation would be for their charts.

P2, working through the bar chart in Phase 2, deliberated over boundary behavior: "Loop back or stop? I don't think there is a right way. I will just pick *fruit* for now and *loop* and see what this does." P6, who brought a bespoke flower visualization, wondered how to translate the affective quality of their chart: "I think my visualization should be more about the vibes, but I don't know how to make the alt text have good vibes. What is *fun*?" P4 asked fundamental questions about the interaction model itself: "Why do screen readers and keyboards have to work this way? Do people like that?... why do we navigate?" And later after testing, P4 concluded, "I bet we should make this *faster*" before cutting out the additional numerical dimension in Phase 2.

Several participants engaged with the concept of narrative and flow. P8 articulated this as a question about the goal of the visualization: "sometimes I want a big picture, not precision. I may want to drill down a little..." and observed that "we think too much in terms of components... sometimes accuracy isn't the actual goal, it's getting a general sense of something." P4, upon discovering that *Skeleton* supports text-based input, asked, "How do you make that good, though? Like chatGPT, or do people want to, like, interact with the chart [elements]?'

During the interview, several participants explicitly requested guidance. P1, P2, P3, and P6 wanted to see examples of well-designed navigation experiences. P1, P3, P4, P5, P6, and P7 wanted embedded guidelines within the tool. P2 and P4 actually used web search to look for "chart navigation for accessibility guidelines" (P2) and "accessible viz screen reader design" (P4). P3 was interested in automation and heuristics that could suggest reasonable defaults. These requests are consistent with the pattern (**R2**): practitioners who could see the design space wanted orientation within it.

6.3 Iteration Was Substantive and Self-directed

Every participant iterated on their navigation designs, and this iteration was neither perfunctory nor prompted by the research team (**R4**). Participants revisited decisions, revised configurations, and in some cases restructured their entire approach after encountering their design in a new stage of the tool.

The most sustained iteration concerned labels. Every participant spent the largest share of their authoring time in the label template editor (Figure 6), editing the text that a screen reader would announce at each node. This editing was granular: participants wrote full sentences, rearranged the order of data tokens, debated whether to include field keys alongside values or values alone, experimented with how to name groups and individual elements, and considered the length and density of the resulting announcements. At the division and dimension levels, some participants added multiple aggregate statistics (count, sum, average, range, trend) and then returned to trim them. P2, for instance, edited data point labels, left them, and returned to revise them two additional times: "This label is way too complicated, I think."

A second, distinct pattern of iteration emerged around testing. The editor displays all navigation nodes simultaneously, showing the full structure at a glance. The testing stage (Section 4.3), by contrast, shows only the currently focused node, highlighting it in the canvas one at a time as the practitioner navigates. This difference consistently prompted participants to revise. Several returned to the editor after testing to adjust group node outlines, because outline strategies that looked distinct when displayed simultaneously (such as bounding rectangles vs. convex hulls) became harder to differentiate when encountered one at a time. Others adjusted their dimension configurations: P3 and P5 restructured their dimensions after testing, and P4, P6, and P8 experimented with different key bindings and navigation rules. P2 used the testing stage specifically to identify labels that needed revision.

The most extensive iteration came from P1, who returned to the preparation wizard after reaching the testing stage and re-took the entire wizard from scratch. Seeing the full structure in testing motivated them to re-evaluate their earliest decisions. They reduced their navigation from three dimensions to one, producing a navigation experience they described as deliberately minimal. In the reflective interview, P1 explained that they had been thinking about trimming excess, joking that they were "working on a data-to-word ratio."

The scaffolding tool shaped the pace and character of these iterations. Participants who used the scaffold (Figure 3) generally required only minor manual adjustment of node positions, spending one to two minutes refining placements in-aggregate before moving on. Iteration on spatial layout was primarily about the appearance of group-level outlines: because the scaffold computed leaf positions from the chart's encoding, the main remaining spatial decision was how division and dimension nodes should visually indicate grouping.

6.4 Seeing Navigation Prompted Practitioners to Reconsider the Architecture of Their Own Charts

An unexpected finding was that working with *Skeleton* prompted several participants to question the design of the visualization they had brought, not just the navigation structure they were building over it (**R4**). Five participants (P4, P5, P6, P7, P8) expressed a desire to simplify their own charts after seeing what the corresponding navigation structure required. Three (P1, P5, P6) considered whether a different chart type might serve the same communicative goal with a simpler navigational architecture. And in 2 cases (P1, P6), participants questioned whether a chart was the right medium at all.

P5, who brought a scatter plot with over two thousand data points, initially tried to build a navigation structure over the full dataset, recognized that the result was unmanageable, and then asked: "do I even need visualization?" They went on to wonder whether the information could be communicated as "a few sentences or like, some data [users] can prompt" (referring to using a large-language model). P6, who brought a bespoke flower visualization in which each petal encoded a different variable, initially tried to express the chart's full complexity in navigation (grouping by flower and then navigating by petal) but then reversed course: "okay, what if we actually just treat this like a bar

chart?" Using the scaffold tool, they produced a simple list-style navigation that worked well for their data, and reflected: "my visualization has a lot, but actually, this could be pretty easy to navigate, I think." P6 also wondered whether there was a non-chart way to "tell their story," and in further discussion described something closer to a scrollytelling article or guided walkthrough than a single interactive chart.

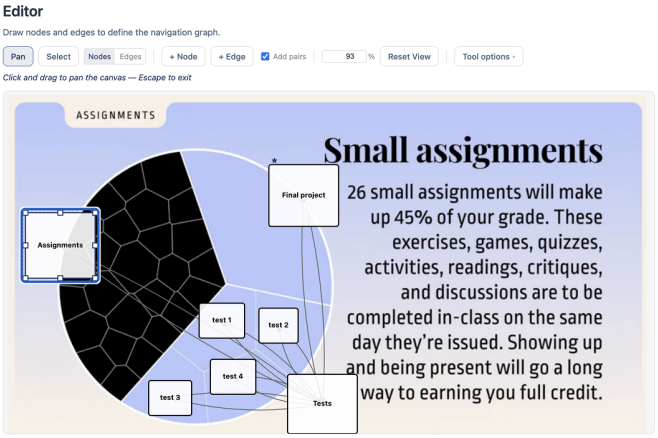

Fig. 4: Re-creation of P8's moment of realization, placing nodes manually: not every element in their voronoi pie chart *should* be navigable.

P8's case was the most striking. They brought a voronoi pie chart (Figure 4) used to communicate to students that 26 assignments make up 45% of their grade, the largest slice of the pie. Working through *Skeleton*, P8 first considered making all 26 voronoi cells navigable, then considered making only the three main slices of the pie navigable, and then questioned whether navigation was needed at all. The "point of this," as they described it, was to communicate a single ratio; students did not need to traverse individual cells. P8 concluded that well-written alternative text was sufficient for the screen reader experience and chose not to build any structure. They also reflected on the design of the visualization itself, concluding that the voronoi treatment served a visual purpose (it is visually striking) that was separable from the informational purpose (communicating a grade breakdown). They kept the visual design and simplified the non-visual experience accordingly.

6.5 Experiencing Keyboard Navigation Surfaced a Broader Range of Users and Input Technologies

The testing stage (Section 4.3) provided what was, for most participants, their first experience of keyboard navigation through a data visualization. Only three participants (P1, P2, P8) reported prior experience using a screen reader, and only two (P1, P8) had previously navigated a visualization using keyboard input alone. Yet during the study, every participant navigated using a keyboard.

Several participants responded to this experience with genuine engagement. P4 reacted with enthusiasm: "Woah, this is incredible. Wait, what other visualizations can do this?" and later, "Maybe you could make a visualization game, where you can move around the data?" P7 said, "I love this. I want to make charts with this." P5, who had not previously encountered keyboard-navigable charts, said, "I didn't know you could do this."

The experience also expanded participants' sense of who these structures serve (**R2**, **R4**). P8 adjusted a node's spatial dimensions and said, "Let's make the hitbox as big as possible here... for my neighbor with Parkinson's to click these," treating the navigation structure as relevant to motor accessibility, not only screen reader access. P3, observing the visual focus indicators that appeared during scaffold-based placement, remarked that "this is for more people than [someone] blind," recognizing that sighted keyboard users and users with partial vision also rely on visible focus states. During the interview, P4 asked sustained questions about what kinds of technologies different people with disabilities use, and P4 and P7 both asked why focus indication was designed to be visible rather than invisible.

P4's curiosity extended to alternative input modalities. Upon learning that *Skeleton* supports text-based navigation and (due to leveraging Data Navigator) also supports a wide array of other input modalities, they asked, "Can you talk at it, too? Is that what some people do?" and followed up with, "How cool is that? How do you make that good, though?" P8 raised a concern about discoverability in the current interaction model: "I've never used j or w to drill out, always shift arrow or option arrow..." and worried that a user might not know how to exit a nested level. These observations reflect a broad mental model of the user, from an abstract "screen reader user" to a person with multiple capabilities, preferences, and interaction patterns (**R2**).

7 Discussion

7.1 Visibility as a Precondition for Iteration

The central finding of this work is not that *Skeleton* taught sighted practitioners how to design accessible navigation, but that it gave them something to react to. When navigation structure was invisible, our co-designers would only seek to verify whether navigation followed our design documentation. Once visible, questions shifted to whether it was good, why, how it could be improved, and what else it could do. Visibility encouraged continuous design iteration.

This mechanism is consistent with Schön's account of reflective practice [53]: designers iterate by externalizing a representation, perceiving its properties, and responding to what they see. The representation talks back, and the designer adjusts. But this loop requires a representation. In our co-design work, we assumed that the gap was hand-off between design and development, and the slow process of manual verification. Instead, the gap is that development was not participating in, and encouraging, further design reflection. In a developer's code-only workflow, navigation structure has no externalization that supports this kind of perceptual engagement. A practitioner can read the code that specifies a navigation graph, but they cannot see the graph, cannot perceive its topology at a glance, cannot notice that a label is redundant or that a hierarchy is too deep by looking at it. The reflective loop is disrupted and occluded at the first step.

Skeleton encourages this loop by rendering navigation structure in a form that supports the same kind of perceptual judgment that visualization practitioners already apply to every other aspect of their work. The results show what happened when this loop was available: every participant iterated, substantively and self-directedly (Section 6.3). They revised labels repeatedly, restructured dimensions after testing, adjusted group outlines when sequential traversal revealed problems that simultaneous display had hidden. P1 restarted the entire preparation wizard after seeing their structure in testing mode. These are not the behaviors of practitioners following a specification; they are the behaviors of practitioners negotiating with a design material.

The co-design work (Section 3) provides complementary evidence: in each collaboration, sighted practitioners reached for visual representations when reasoning about navigation, through node-link diagrams in Figma, schema sketches in Miro, and annotated wireframes on paper. They were already thinking visually about non-visual structure; the development tooling simply had not caught up. The practical implication is broad: if sighted authors depend on visibility, then any authoring workflow that keeps navigation structure invisible limits design iteration. Making structure visible does not guarantee good design, but it is a precondition for the kind of sustained, judgment-driven refinement that good design requires.

7.2 From Compliance to Design

A consistent pattern across the accessibility literature is that practitioners frame accessibility as a compliance problem: a set of requirements to satisfy, a checklist to complete, a legal or institutional obligation to meet [13, 30, 57]. The framing matters because compliance and design orient practitioners toward fundamentally different activities. Compliance asks: "does this pass?" Design asks: "is this good?" Our results suggest that *Skeleton* produced a partial shift from the first orientation to the second. Participants' initial instincts clustered around compliance-oriented responses: provide alternative text, follow guidelines, ask an expert (Section 6.2). But alongside that desire for guidance, they began

doing something compliance framing does not typically produce: they encountered complexity and then *iterated* (Section 6.3). This sustained, self-directed refinement is the behavioral signature of design, not compliance. As P4 put it: "I'd love to have someone blind actually just with me while I make this, but I also understand that I should learn what makes a good experience too."

The shift extended beyond the non-visual experience itself. As reported in Section 6.4, five participants reconsidered the design of the visualization they had brought, not just the navigation structure. Making the accessibility consequences of visual design choices visible prompted practitioners to question whether a different chart type, a simpler encoding, or a non-chart medium might better serve their communicative goals. This interrelation between non-visual and visual design suggests that the widespread treatment of accessibility as a compliance activity may be partly a consequence of tooling that offers no legible, manipulable design surface. Auditing frameworks are valuable, but they are *evaluative* tools, not *authoring* tools. The field needs both.

7.3 Bespoke Visualizations as an Unaddressed Accessibility Research Problem

Diagrams, infographics, and data-driven illustrations are often one-off, custom designs with bespoke symbols, layouts, and visual languages. These representations are increasingly common in journalism, scientific communication, personal projects, art, and public-facing data work, and they represent the cases where accessibility-focused tools are needed most and available least. *Skeleton*'s image-based workflow (upload any 2D image, place nodes manually) provides a starting point, but the study made clear that bespoke visualizations need more than node placement. They need support for reasoning about what navigational structure (if any) is appropriate when no template applies, a research problem that remains largely unexplored. It is worth being precise about where generality holds and where it stops. Because the *Dimensions API* derives structure from whatever fields a dataset carries rather than from a fixed schema, the data side of the approach already extends to datasets not known in advance; what does not yet generalize is the prior, harder question of which dimensions, if any, a bespoke visual should expose when its visual form was never organized around a tabular structure to begin with. Generalizing the abstraction is therefore not primarily a matter of supporting more schemas but of supporting the design judgment that precedes choosing a schema at all.

7.4 What Visualization Owes Accessibility

Our approach, to make visual non-visual experiences, should not be limited to data visualization's own accessibility challenges. Navigation structure is a foundational component of accessible experience across domains: PDF and document reading order, web page structures, and software application layouts. In each of these areas, sighted practitioners author non-visual experiences without visual feedback, and in each, the same gap between design intent and verifiable outcome constrains quality. Testing is slow, error prone, and requires expertise in assistive technology use. Visual tooling for authoring, inspecting, and debugging non-visual structure (**R3**) is a tractable and high-value problem across application domains.

There is also a deeper question about what visibility can accomplish in principle. Work on multi-modal authoring environments has argued for de-centering visual representation and treating modalities as equal partners in the design process [73], an important commitment. *Skeleton* holds this in tension: it re-centers visual representation as the medium through which sighted practitioners engage with non-visual structure. We believe this is justified pragmatically, because sighted authors need to articulate navigation design in their own perceptual language before they can reason about it, and this paper provides evidence that they do. But articulating a design in one's own language is not the same as understanding how it will be experienced in someone else's. The risk we want to name is that making non-visual structure visible to sighted practitioners could be mistaken for making it *understood*, when in fact it makes it *designable*. The fuller design process requires collaboration with disabled users, not as an occasional supplement but as a regular practice. *Skeleton* can make that collaboration more productive by giving both parties a shared representation or space of translation between representations, but it cannot replace it.

What visualization owes accessibility, then, is not simply better output but authoring tools that better stimulate reasoning, both individually and collaborative, about design.

8 Limitations and Future Work

Skeleton surfaces design questions but does not answer them: several participants asked what constitutes good navigation, and the tool had nothing authoritative to offer. As Section 7 argues, our study evaluated whether making structure visible stimulated design consideration, not whether the resulting designs were good for the people who will use them, which is a distinct question that remains open. CD2's expert screen reader evaluation of *Skeleton*'s navigation output (Section 4.3) provided practice-based validation for several common chart types and surfaced concrete bugs, but this evaluation was neither comprehensive nor controlled: many configurations remain untested, and expert review is not a substitute for evaluation with a broader population of end users.

Our approach using *Skeleton* as a design probe only with sighted participants is a limitation and, we believe, the right sequencing: *Skeleton* improves the intentionality and iterability of what sighted practitioners produce, which is a necessary precondition for a subsequent evaluation or future collaborative design work between sighted and blind authors. Further research and evaluation should close this loop, ideally to engage how mixed-ability teams co-design multi-modal data experiences.

A further boundary concerns the kind of visualization *Skeleton* targets. Highly interactive or dynamic visualizations, where selecting an element reloads the data, where marks animate, or where the navigable structure itself changes in response to user interaction, are not yet addressed. We see this as an important future direction rather than a fundamental obstacle: the same argument for making structure visible applies with more force when there are several structures to coordinate, and an inspectable representation may be precisely what makes a dynamic navigation experience tractable to author.

Skeleton is also a prototype with substantial work remaining. Not within the scope of the paper, but our participants provided ample feedback on the functionality of the prototype itself. The most urgent gap is export functionality: practitioners can design and inspect navigation structures in the tool but cannot yet produce deployable output.

9 Conclusion

Accessible navigation structure has long occupied an awkward position in visualization practice: known to matter, difficult to design, and invisible to the people responsible for building it. Without a way to see what they were making, sighted practitioners could not catch errors, could not iterate, and could not develop the kind of considered judgment that good design requires; accessibility remained downstream of every other decision not because of any single failure, but because the authoring conditions did not support anything else. *Skeleton* demonstrates that those conditions can be changed. Making navigation structure visible and manipulable (as an interactive graph rendered over the spatial layout of a real visualization, with live label previews and testable traversal) shifted how practitioners engaged with and reasoned about accessible design, prompting them to ask whether the design of their structure was *good* rather than merely whether it passed.

The implication generalizes beyond our tool: wherever a non-visual structure or user experience is authored without a visible representation, building one the author can react to and manipulate invites them to design. What the paper leaves open is more than what it closes. We used *Skeleton* as a design probe with sighted practitioners; we did not evaluate the structures they produced with the end users those structures are meant to serve, and the broader disciplinary conversation about what visualization research owes accessibility, and what methods might transfer between them, has more questions than answers. We offer *Skeleton* not as a solution to these problems but as evidence that engaging them directly, with the full weight of visualization's methodological tradition, is both possible and fruitful.

Acknowledgments

This work was supported by a grant from Apple, Inc. Any views, opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and should not be interpreted as reflecting the views, policies or position, either expressed or implied, of Apple Inc.

The in-situ co-design and co-engineering work we performed was made possible for our partners by a CZI EOSS Cycle 6 Grant, a research gift from Adobe, Inc., and a small grant from the University of Wisconsin. Any views, opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and should not be interpreted as reflecting the views, policies or position, either expressed or implied, of these organizations.

I want to especially thank our fantastic, brilliant co-designers and collaborators, who made this work possible.

References

[1] O. Alhadreti and P. Mayhew. Rethinking thinking aloud: A comparison of three think-aloud protocols. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pp. 1–12. ACM, Apr. 2018. doi: 10.1145/3173574.3173618 6

[2] B. Biggs, C. Toth, T. Stockman, J. M. Coughlan, and B. N. Walker. Evaluation of a non-visual auditory choropleth and travel map viewer. In *Proceedings of the 27th International Conference on Auditory Display (ICAD 2022)*, pp. 82–90. International Community for Auditory Display, June 2022. PMCID: PMC10010675. doi: 10.21785/icad2022.027 3

[3] M. Blanco, J. Zong, and A. Satyanarayan. Olli: An Extensible Visualization Library for Screen Reader Accessibility. In *IEEE VIS Posters*, 2022. https://vis.csail.mit.edu/pubs/olli. 3, 4

[4] J. Bornschein, D. Prescher, and G. Weber. Collaborative Creation of Digital Tactile Graphics. In *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility - ASSETS '15*, ASSETS '15, pp. 117–126. Association for Computing Machinery, ACM Press, New York, NY, USA, Oct. 2015. doi: 10.1145/2700648.2809869 3

[5] M. Bostock, V. Ogievetsky, and J. Heer. D³ data-driven documents. *IEEE TVCG*, 17(12):2301–2309, Dec. 2011. doi: 10.1109/tvcg.2011.185 2

[6] V. Braun and V. Clarke. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2):77–101, Jan. 2006. doi: 10.1191/1478088706qp063oa 6

[7] S. Brewster. Visualization tools for blind people using multiple modalities. *Disability and Rehabilitation*, 24(11-12):613–621, Jan. 2002. doi: 10.1080/09638280110111388 2

[8] M. Butler, L. M. Holloway, S. Reinders, C. Goncu, and K. Marriott. Technology developments in touch-based accessible graphics: A systematic review of research 2010-2020. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, pp. 1–15. ACM, May 2021. doi: 10.1145/3411764.3445207 2

[9] P. Chundury, B. Patnaik, Y. Reyazuddin, C. Tang, J. Lazar, and N. Elmqvist. Towards understanding sensory substitution for accessible visualization: An interview study. *IEEE TVCG*, 28(1):1084–1094, Jan. 2022. doi: 10.1109/tvcg.2021.3114829 2

[10] C. Cullen and O. Metatla. Co-designing inclusive multisensory story mapping with children with mixed visual abilities. In *IDC '19*, IDC '19, pp. 361–373. ACM, June 2019. doi: 10.1145/3311927.3323146 3

[11] L. de Greef, D. Moritz, and C. Bennett. Interdependent variables: Remotely designing tactile graphics for an accessible workflow. In *Proceedings of the 23rd International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '21, art. no. 36, 6 pages. ACM, Oct. 2021. doi: 10.1145/3441852.3476468 3

[12] F. Elavsky and C. X. Bearfield. Playing telephone with generative models. In *2025 IEEE Workshop on Accessible Data Visualization (AccessViz)*, pp. 14–24. IEEE, Nov. 2025. doi: 10.1109/accessviz68666.2025.00008 2

[13] F. Elavsky, C. Bennett, and D. Moritz. How accessible is my visualization? evaluating visualization accessibility with chartability. *Computer Graphics Forum*, 41(3):57–70, June 2022. doi: 10.1111/cgf.14522 4, 8

[14] F. Elavsky, L. Nadolskis, and D. Moritz. Data navigator: An accessibility-centered data navigation toolkit. *IEEE TVCG*, 2023. doi: 10.1109/tvcg.2023.3327393 2, 3

[15] A. Endert, L. Bradel, and C. North. Beyond control panels: Direct manipulation for visual analytics. *IEEE Computer Graphics and Applications*, 33(4):6–13, July 2013. doi: 10.1109/mcg.2013.53 1

[16] P. Eswaramoorthy, F. Elavsky, T. Allard, and gabalafou. Quansight-Labs/bokeh-a11y-audit, Feb. 2025. doi: 10.5281/zenodo.14923642 4

[17] D. Fan, A. Fay Siu, H. Rao, G. S.-H. Kim, X. Vazquez, L. Greco et al. The accessibility of data visualizations on the web for screen reader users: Practices and experiences during covid-19. *ACM Transactions on Accessible Computing*, 16(1):1–29, Mar. 2023. doi: 10.1145/3557899 2

[18] A. M. Farahani, P. Adibi, M. S. Ehsani, H.-P. Hutter, and A. Darvishy. Automatic chart understanding: A review. *IEEE Access*, 11:76202–76221, 2023. doi: 10.1109/access.2023.3298050 2

[19] D. Fisher, I. Popov, S. Drucker, and m. schraefel. Trust me, i'm partially right: incremental visualization lets analysts explore large datasets faster. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pp. 1673–1682. ACM, May 2012. doi: 10.1145/2207676.2208294 1

[20] N. Hammad, F. Elavsky, S. Moharana, J. Chen, S. Lee, P. Carrington et al. Exploring the affordances of game-aware streaming to support blind and low vision viewers: A design probe study. In *The 26th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '24, pp. 1–13. ACM, Oct. 2024. doi: 10.1145/3663548.3675665 2

[21] G. Harboe and E. M. Huang. Real-world affinity diagramming practices: Bridging the paper-digital gap. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pp. 95–104. ACM, Apr. 2015. doi: 10.1145/2702123.2702561 6

[22] G. R. Hayes. The relationship of action research to human-computer interaction. *ACM Transactions on Computer-Human Interaction*, 18(3):1–20, July 2011. doi: 10.1145/1993060.1993065 2, 3

[23] J. Heer and M. Agrawala. Design considerations for collaborative visual analytics. In *2007 IEEE Symposium on Visual Analytics Science and Technology*, pp. 171–178. IEEE, 2007. doi: 10.1109/vast.2007.4389011 1

[24] T. Hermann, A. Hunt, and J. G. Neuhoff, eds. *The Sonification Handbook*. Logos Verlag Berlin, Berlin, Germany, Dec. 2011. ISBN: 978-3-8325-2819-5. 2

[25] L. Holloway, P. Cracknell, K. Stephens, M. Fanshawe, S. Reinders, K. Marriott et al. Refreshable tactile displays for accessible data visualisation, 2024. doi: 10.48550/ARXIV.2401.15836 2

[26] L. M. Holloway, C. Goncu, A. Ilsar, M. Butler, and K. Marriott. Infosonics: Accessible infographics for people who are blind using sonification and voice. In *CHI Conference on Human Factors in Computing Systems*, CHI '22, pp. 1–13. ACM, Apr. 2022. doi: 10.1145/3491102.3517465 2

[27] E. L. Hutchins, J. D. Hollan, and D. A. Norman. Direct manipulation interfaces. *Human–Computer Interaction*, 1(4):311–338, Dec. 1985. doi: 10.1207/s15327051hci0104_2 1, 2

[28] H. Hutchinson, W. Mackay, B. Westerlund, B. B. Bederson, A. Druin, C. Plaisant et al. Technology probes: inspiring design for and with families. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 17–24. ACM, Apr. 2003. doi: 10.1145/642611.642616 2

[29] S. Jones, I. Pedraza Pineros, D. Hajas, J. Zong, and A. Satyanarayan. "customization is key": Reconfigurable textual tokens for accessible data visualizations. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, pp. 1–14. ACM, May 2024. doi: 10.1145/3613904.3641970 2, 5

[30] S. C. S. Joyner, A. Riegelhuth, K. Garrity, Y.-S. Kim, and N. W. Kim. Visualization accessibility in the wild: Challenges faced by visualization designers. In *CHI Conference on Human Factors in Computing Systems*, pp. 1–19. ACM, Apr. 2022. doi: 10.1145/3491102.3517630 2, 3, 8

[31] C. Jung, S. Mehta, A. Kulkarni, Y. Zhao, and Y.-S. Kim. Communicating visualizations without visuals: Investigation of visualization alternative text for people with visual impairments. *IEEE TVCG*, 28(1):1095–1105, Jan. 2022. doi: 10.1109/tvcg.2021.3114846 2

[32] G. Kim, J. Kim, and Y.-S. Kim. "explain what a treemap is": Exploratory investigation of strategies for explaining unfamiliar chart to blind and low vision users. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, pp. 1–13. ACM, Apr. 2023. doi: 10.1145/3544548.3581139 2

[33] H. Kim, Y.-S. Kim, and J. Hullman. Erie: A declarative grammar for data sonification. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, pp. 1–19. ACM, May 2024. doi: 10.1145/3613904.3642442 2, 3

[34] J. Kim, A. Srinivasan, N. W. Kim, and Y.-S. Kim. Exploring chart question answering for blind and low vision users. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, pp. 1–15. ACM, Apr. 2023. doi: 10.1145/3544548.3581532 2

[35] N. W. Kim, G. Ataguba, S. C. Joyner, C. Zhao, and H. Im. Beyond alternative text and tables: Comparative analysis of visualization tools and accessibility methods. *Computer Graphics Forum*, 42(3):323–335, June 2023. doi: 10.1111/cgf.14833 2, 3

[36] N. W. Kim, S. C. Joyner, A. Riegelhuth, and Y. Kim. Accessible visualization: Design space, opportunities, and challenges. *Computer Graphics Forum*, 40(3):173–188, June 2021. doi: 10.1111/cgf.14298 2

[37] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE TVCG*, 20(12):2122–2131, Dec. 2014. doi: 10.1109/tvcg.2014.2346452 1

[38] A. Lundgard, C. Lee, and A. Satyanarayan. Sociotechnical considerations for accessible visualization design. In *2019 IEEE Visualization Conference (VIS)*, pp. 16–20. IEEE, Oct. 2019. doi: 10.1109/visual.2019.8933762 3

[39] A. Lundgard and A. Satyanarayan. Accessible visualization via natural language descriptions: A four-level model of semantic content. *IEEE TVCG*, 28(1):1073–1083, Jan. 2022. doi: 10.1109/tvcg.2021.3114770 2

[40] D. L. Mansur, M. M. Blattner, and K. I. Joy. Sound graphs: A numerical data analysis method for the blind. *Journal of Medical Systems*, 9(3):163–174, June 1985. doi: 10.1007/bf00996201 2

[41] K. Marriott, M. Butler, L. Holloway, W. Jolley, B. Lee, B. Maguire et al. From vision to touch: Bridging visual and tactile principles for accessible data representation. *IEEE TVCG*, 32(1):659–669, Jan. 2026. doi: 10.1109/tvcg.2025.3634254 2

[42] K. Marriott, B. Lee, M. Butler, E. Cutrell, K. Ellis, C. Goncu et al. Inclusive data visualization for people with disabilities: a call to action. *Interactions*, 28(3):47–51, Apr. 2021. doi: 10.1145/3457875 2

[43] A. M. McNutt. No grammar to rule them all: A survey of json-style dsls for visualization. *IEEE TVCG*, pp. 1–11, 2022. doi: 10.1109/tvcg.2022.3209460 1

[44] C. Mei∗, J. Pollock∗, D. Hajas, J. Zong, and A. Satyanarayan. Benthic: Perceptually congruent structures for accessible charts and diagrams. In *Proceedings of the 27th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '25, pp. 1–17. ACM, Oct. 2025. doi: 10.1145/3663547.3746342 2, 3

[45] P. Mowar, A. Steinfeld, and J. P. Bigham. iTagPDF: Towards finally automating PDF accessibility. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, pp. 1–17. ACM, Apr. 2026. Best Paper Award. doi: 10.1145/3772318.3790289 3

[46] B. A. Myers. Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*, 1(1):97–123, Mar. 1990. doi: 10.1016/S1045-926X(05)80036-9 2

[47] J. Poco and J. Heer. Reverse-engineering visualizations: Recovering visual encodings from chart images. *Computer Graphics Forum*, 36(3):353–363, 2017. doi: 10.1111/cgf.13193 14

[48] L. Race, C. Fleet, D. Montour, L. Yazzolino, M. Salsiccia, C. Ferrari et al. Designing while blind: Nonvisual tools and inclusive workflows for tactile graphic creation. In *The 25th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '23, pp. 1–8. ACM, Oct. 2023. doi: 10.1145/3597638.3614546 3

[49] J. Rawling, E. Carson, J. Attig, D. Mickelson, W. Mode, M. Johnson et al. Quaternary geology of wisconsin. Technical report, Wisconsin Geological and Natural History Survey, 2025. doi: 10.54915/xqpw9883 3

[50] S. Reinders, M. Butler, I. Zukerman, B. Lee, L. Qu, and K. Marriott. When refreshable tactile displays meet conversational agents: Investigating accessible data presentation and analysis with touch and speech. *IEEE TVCG*, 31(1):864–874, Jan. 2025. doi: 10.1109/tvcg.2024.3456358 2

[51] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE TVCG*, 23(1):341–350, Jan. 2017. doi: 10.1109/tvcg.2016.2599030 1, 2, 5, 13

[52] M. Savva, N. Kong, A. Chhajta, L. Fei-Fei, M. Agrawala, and J. Heer. Revision: automated classification, analysis and redesign of chart images. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, UIST '11, pp. 393–402. ACM, Oct. 2011. doi: 10.1145/2047196.2047247 14

[53] D. Schön and J. Bennett. Reflective conversation with materials, Apr. 1996. doi: 10.1145/229868.230044 8

[54] J. Seo, Y. Xia, B. Lee, S. Mccurry, and Y. J. Yam. Maidr: Making statistical visualizations accessible with multimodal data representation. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, pp. 1–22. ACM, May 2024. doi: 10.1145/3613904.3642730 2

[55] A. Sharif, S. S. Chintalapati, J. O. Wobbrock, and K. Reinecke. Understanding screen-reader users' experiences with online data visualizations. In *Proceedings of the 23rd International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '21, pp. 1–16. ACM, Oct. 2021. doi: 10.1145/3441852.3471202 2

[56] A. Sharif and B. Forouraghi. evographs — a jquery plugin to create web accessible graphs. In *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pp. 1–4. IEEE, Jan. 2018. doi: 10.1109/ccnc.2018.8319239 3

[57] A. Sharif, J. G. Kim, J. Z. Xu, and J. O. Wobbrock. Understanding and reducing the challenges faced by creators of accessible online data visualizations. In *The 26th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '24, pp. 1–20. ACM, Oct. 2024. doi: 10.1145/3663548.3675625 2, 3, 8

[58] A. Sharif, O. H. Wang, A. T. Muongchan, K. Reinecke, and J. O. Wobbrock. Voxlens: Making online data visualizations accessible with an interactive javascript plug-in. In *CHI Conference on Human Factors in Computing Systems*, CHI '22, art. no. 478, 19 pages. ACM, Apr. 2022. doi: 10.1145/3491102.3517431 3

[59] Shneiderman. Direct manipulation: A step beyond programming languages. *Computer*, 16(8):57–69, Aug. 1983. doi: 10.1109/mc.1983.1654471 2

[60] B. Shneiderman, C. Williamson, and C. Ahlberg. Dynamic queries: database searching by direct manipulation. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, CHI '92, pp. 669–670. ACM Press, 1992. doi: 10.1145/142750.143082 1

[61] D. C. Smith. *Pygmalion: A Computer Program to Model and Stimulate Creative Thought*. 1977. doi: 10.1007/978-3-0348-5744-4 2

[62] V. Sorge. Polyfilling accessible chemistry diagrams. In *Lecture Notes in Computer Science*, pp. 43–50. Springer International Publishing, 2016. doi: 10.1007/978-3-319-41264-1_6 2

[63] I. E. Sutherland. Sketchpad: A man-machine graphical communication system. In *Proceedings of the May 21-23, 1963, spring joint computer conference on - AFIPS '63 (Spring)*, AFIPS '63 (Spring), p. 329. ACM Press, New York, NY, USA, 1963. doi: 10.1145/1461551.1461591 2

[64] T. Teitelbaum and T. Reps. Cornell program synthesizer. *Communications of the ACM*, 24(9):563–573, 1981. doi: 10.1145/358746.358755 2

[65] J. Thatcher. Screen reader/2: access to os/2 and the graphical user interface. In *Proceedings of the first annual ACM conference on Assistive technologies - Assets '94*, Assets '94, pp. 39–46. ACM Press, 1994. doi: 10.1145/191028.191039 2

[66] J. R. Thompson, J. J. Martinez, A. Sarikaya, E. Cutrell, and B. Lee. Chart reader: Accessible visualization experiences designed with screen reader users. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, pp. 1–18. ACM, Apr. 2023. doi: 10.1145/3544548.3581186 2, 3

[67] WAI. Understanding success criterion 2.1.1: keyboard. WCAG standard, W3C, 2017. Accessed: 2026-03-20, `https://www.w3.org/WAI/WCAG21/Understanding/keyboard.html`. 2

[68] H. Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1):3–28, Jan. 2010. doi: 10.1198/jcgs.2009.07098 1

[69] L. Wilkinson. The grammar of graphics. In *Handbook of Computational Statistics*, pp. 375–414. Springer Berlin Heidelberg, Dec. 2011. doi: 10.1007/978-3-642-21551-3_13 1

[70] B. L. Wimer, R. Kanchi, K. Frierson, V. Potluri, R. Metoyer, J. Mankoff et al. Nonvisual support for understanding and reasoning about data structures, 2026. doi: 10.48550/ARXIV.2601.19168 3

[71] J. Zong. Using real names of disabled participant-contributors to practice citational justice in accessibility. In *2025 IEEE Workshop on Accessible Data Visualization (AccessViz)*, pp. 30–33. IEEE, Nov. 2025. doi: 10.1109/accessviz68666.2025.00011 3

[72] J. Zong, C. Lee, A. Lundgard, J. Jang, D. Hajas, and A. Satyanarayan. Rich screen reader experiences for accessible data visualization. *Computer Graphics Forum*, 41(3):15–27, June 2022. doi: 10.1111/cgf.14519 2, 3

[73] J. Zong, I. Pedraza Pineros, M. K. Chen, D. Hajas, and A. Satyanarayan. Umwelt: Accessible structured editing of multi-modal data representations. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, pp. 1–20. ACM, May 2024. doi: 10.1145/3613904.3641996 3, 9

[74] J. Zong, J. Pollock, D. Wootton, and A. Satyanarayan. Animated vega-lite: Unifying animation with a grammar of interactive graphics. *IEEE TVCG*, 29(1):149–159, Jan. 2023. doi: 10.1109/tvcg.2022.3209369 1

A Dimensions API Reference

Each dimension in the *Dimensions API* (Section 3.4) declares a `type` (`categorical` or `numerical`, inferred from the data when omitted) and a `dimensionKey` naming the data field it traverses. Two further groups of properties govern behavior. A `behavior` block controls traversal: `extents` sets boundary behavior (`terminal` stops at the edges of a group, `circular` wraps around, and `bridgedCousins` carries focus across to the nearest element of an adjacent group rather than stopping or wrapping), and `childmostNavigation` determines whether leaf-level nodes are reachable laterally across a dimension's divisions (`across`) or only within a single division before returning to a parent (`within`, the default). An `operations` block controls how the data is shaped into the hierarchy: a `sortFunction` orders divisions and leaves, a `filterFunction` selects which data participate, and `createNumericalSubdivisions` bins a numerical dimension into a chosen number of ranges.

The generated structure is a multi-level hierarchy: each dimension produces a root node, below which division nodes group the data, below which leaf nodes represent individual data points. Multiple dimensions over the same dataset share leaf nodes, so users navigating via different dimensions reach the same data through different paths.

To make the input and output concrete, the examples below use this small four-row dataset of average temperatures, with an explicit `id` on each row:

```
const data = [
  { id: '_0', month: 'Jan', city: 'NYC', temp: 0 },
  { id: '_1', month: 'Jul', city: 'NYC', temp: 25 },
  { id: '_2', month: 'Jan', city: 'Sydney', temp: 22 },
  { id: '_3', month: 'Jul', city: 'Sydney', temp: 8 }
];
```

With the Dimensions API, bar chart navigation that would otherwise require constructing every node and edge by hand is expressed as a single dimension declaration:

```
dimensions: {
  values: [
    {
      dimensionKey: 'month',
      type: 'categorical',
      behavior: { extents: 'circular' }
    }
  ]
}
```

From this declaration, the build step does three things automatically. It assembles the nodes and the parent-child and sibling edges that connect them, including, when multiple dimensions are declared, the cross-dimension edges that let a user move between hierarchies over shared leaves. It applies the boundary and childmost rules to decide what happens at each edge of the structure. And it assigns keyboard navigation rules to each dimension, drawing from a pool of directional key pairs; because that pool is finite, the API recommends keeping a structure to six dimensions or fewer and asks for explicit navigation rules beyond that point.

This bounds the structures the abstraction expresses by default: it generates the regular, dimensionally-organized hierarchies that recur across common chart types, and structures that do not decompose into a small set of data dimensions (or that exceed the directional-key budget) fall back to explicit graph construction or manual editing in the editor.

The abstraction is otherwise chart-type-agnostic: bar charts, scatter plots, line charts, and layered charts all use the same vocabulary, with different combinations producing different navigation topologies. This directly addressed Bokeh's problem, where no native chart types exist to anchor a pattern: the API mirrors data fields and encoding choices as a set of dimensions, so a thin wrapper can map an arbitrary glyph assembly onto a handful of recurring dimensional shapes (Section 3.3).

The vocabulary composes to richer topologies simply by adding dimensions. Declaring two categorical dimensions over the same dataset, for instance a temperature series recorded by month and by city, produces a dual hierarchy in which left and right traverse one axis and up and down traverse the other, with both sharing the same leaf nodes; here the month dimension wraps at its boundaries while the city dimension stops:

```
dimensions: {
  values: [
    {
      dimensionKey: 'month',
      type: 'categorical',
      behavior: {
        extents: 'circular',
        childmostNavigation: 'across'
      }
    },
    {
      dimensionKey: 'city',
      type: 'categorical',
      behavior: {
        extents: 'terminal',
        childmostNavigation: 'across'
      }
    }
  ]
}
```

This two-dimension declaration assigns a keyboard control to each axis. Table 1 lists the controls Data Navigator generates for it; the arrow keys move between siblings along each dimension, `Enter` drills toward the data, and each dimension receives its own drill-out key from the default key pool (here `W` and `J`).

Table 1: Keyboard controls generated for the two-dimension (`month`, `city`) structure over the sample dataset. Movement along `month` wraps (`circular`); movement along `city` stops at the ends (`terminal`).

Key	Action
← / →	Move to previous / next `month` (wraps Dec–Jan)
↑ / ↓	Move to previous / next `city` (stops at ends)
`Enter`	Drill in (dimension → division → data)
`W`	Drill out along the `month` grouping
`J`	Drill out along the `city` grouping
`Esc`	Exit the structure

Finally, to show what the build step emits, the two-dimension declaration above, applied to the four-row dataset, expands into the nodes and edges below. Each dimension produces a dimension node carrying its `dimensionKey`, a set of division nodes (one per distinct value, each tagged with the field it was `derived` from), and the shared leaf nodes that carry the original rows. A node's `edges` list the edges it participates in; each edge names a `source`, a `target`, and the `navigationRules` that traverse it. Crucially, a single edge is bidirectional: the `left`/`right` pair on one edge means pressing `left` moves toward its `source` and `right` toward its `target`, so the `month` and `city` axes at the leaf level each collapse to one edge with two rules (abbreviated here for space):

```
// nodes (excerpt: one division + one leaf per axis)
{
  month: {
    id: 'month',
    data: { dimensionKey: 'month' },
    edges: ['month-Jan', 'any-exit']
  },
  city: {
    id: 'city',
    data: { dimensionKey: 'city' },
```

```
    edges: ['city-NYC', 'any-exit']
  },
  'Jan': {
    id: 'Jan',
    derivedNode: 'month',
    data: { month: 'Jan' },
    edges: [
      'Jan-Jul',
      'month-Jan',
      'Jan-_0'
    ]
  },
  'NYC':  {
    id: 'NYC',
    derivedNode: 'city',
    data: { city: 'NYC' },
    edges: [
      'NYC-Sydney',
      'city-NYC',
      'NYC-_0'
    ]
  },
  _0: {
    id: '_0',
    data: { month: 'Jan', city: 'NYC', temp: 0 },
    edges: [
      'leaf-_0-_1',    // month axis (l/r)
      'leaf-_0-_2',    // city axis  (u/d)
      'Jan-_0', 'NYC-_0',
      'any-exit'
    ]
  }
  // Jul, Sydney, and _1/_2/_3
  // follow the same pattern
}

// edges
{
  'month-Jan': {
    source: 'month',
    target: 'Jan',
    navigationRules: ['drill-out_month', 'drill-in']
  },
  'city-NYC': {
    source: 'city',
    target: 'NYC',
    navigationRules: ['drill-out_city', 'drill-in']
  },
  'Jan-Jul': {
    source: 'Jan',
    target: 'Jul',
    navigationRules: ['left','right']
  },
  'NYC-Sydney': {
    source: 'NYC',
    target: 'Sydney',
    navigationRules: ['up','down'] },
  'leaf-_0-_1': {
    source: '_0',
    target: '_1',
    navigationRules: ['left','right']
  },
  'leaf-_0-_2': {
    source: '_0',
    target: '_2',
    navigationRules: ['up','down']
  },
  // ...etc
}

// navigationRules
{
  left: { key: 'ArrowLeft',  direction: 'source' },
  right: { key: 'ArrowRight', direction: 'target' },
  up: { key: 'ArrowUp',    direction: 'source' },
  down: { key: 'ArrowDown',  direction: 'target' },
  'drill-in': { key: 'Enter', direction: 'target' },
  'drill-out_month': {
    key: 'KeyW',
    direction: 'source'
  },
  'drill-out_city': {
    key: 'KeyJ',
    direction: 'source'
  }
}
```

Every node, edge, and rule in this output remains individually addressable: this is the structure the *Inspector* renders and that practitioners select and edit directly in *Skeleton*, rather than a compiled artifact they cannot reopen.

B Deterministic Scaffolding, Group Outline Geometries, and Padding Estimation

This appendix details the three technical components behind *Skeleton*'s spatial authoring (Section 4): how the scaffold repurposes a visualization rendering engine to approximate mark positions, how group outlines are computed from those positions, and how a lightweight, non-ML computer vision pass aligns the scaffold to an uploaded chart image. We intentionally chose deterministic math approaches at any opportunity, for speed and inspectability benefits.

B.1 Vega as a Deterministic Layout Engine

The scaffold (Section 4, "Leveraging visualization as a scaffolding engine") treats a visualization grammar as a coordinate oracle rather than a rendering target. From the active configuration, which carries a chart type, field-to-channel mappings (`xField`, `yField`, and an optional `colorField`), mark parameters, and an outer plot box, *Skeleton* assembles a *Vega-Lite* [51] specification and renders it through `vega-embed` into a hidden, off-screen container using the SVG renderer. The container is never attached to the visible document; it exists only so that *Vega* will compile a full scenegraph and, with it, the scale functions that map data values to pixels.

Positions are then read directly from the compiled view rather than parsed from the rendered output. For each leaf node, *Skeleton* evaluates the view's `x` and `y` scales on the corresponding data values: a band scale supplies a bar's horizontal position and, through its `bandwidth`, its width, while a linear scale maps a quantitative value to a pixel height with the baseline taken at the scale's zero. Stacked bars accumulate per-segment offsets in data space before scaling; clustered bars add the sub-band offset; scatter and line marks evaluate both scales at each point. Because the scale functions are pure, this path needs no DOM measurement and is fully deterministic: identical configurations yield identical coordinates. A secondary path, used when scale evaluation does not apply, parses the hidden SVG and reads mark bounding boxes instead.

The configuration distinguishes the inner mark area from the outer box. *Vega-Lite*'s `width` and `height` describe the inner area in which marks are drawn, while a `padding` object reserves space for axes and labels; *Skeleton* treats `plotWidth` and `plotHeight` as the outer, border-box dimensions and derives the inner area by subtracting padding. A mark at normalized position f within the inner area therefore lands at image coordinate $o + p + f\,w_{\text{in}}$, where o is the plot offset, p the leading padding, and w_{in} the inner extent. This relation is what later makes padding recoverable.

A practical subtlety motivates the rest of this appendix. The engine that produced the uploaded image is generally unknown and is rarely *Vega*; our co-designers authored charts in other tools entirely (Section 3).

What transfers across engines is the *relative* geometry of the marks, the spacing of bars or the placement of points within the data rectangle, not the absolute pixel offsets, which depend on how much room each particular chart reserved for its axes, ticks, labels, and title. The scaffold reconstructs the relative geometry faithfully; reconciling it with the absolute placement in the image is the task taken up by the padding estimation described below.

B.2 Computing Group Outlines

Once leaf positions exist, the group nodes above them (divisions, dimensions, and the root) receive a visible outline computed purely from the geometry of their descendants. Each leaf contributes a rectangle (for bars) or a circle (for points), and a chosen strategy turns the collection into a single SVG path string. These functions live in the Data Navigator core (`geometry.ts`) and are side-effect free, taking arrays of rectangles or circles and an optional padding term and returning a path `d` attribute, so they can be reused outside *Skeleton* and packaged for export.

Four strategies cover the cases that recurred during co-design (Section 3). A *bounding rectangle* encloses all descendants in the single axis-aligned box given by their combined extent. A *union* emits one rectangular subpath per mark and relies on the SVG fill rule to render disconnected regions, which suits groups whose marks are spatially separated. A *convex hull* collects the corner points of every rectangle (or sixteen sampled points around every circle) and runs a Graham scan, optionally pushing each hull vertex radially outward from the centroid to add padding; this gives a tight, convex boundary appropriate to scatter clusters and bar groups. For line series, an *offset path* traces the polyline of points at a fixed perpendicular distance on each side, using per-vertex averaged normals for miter joins and semicircular arc caps at the ends, producing a closed band that follows the line. Numerical dimensions additionally support a grid drawn over the dimension's value bounds.

Because every outline derives from the same leaf coordinates the scaffold computed, changing a mark position, a chart type, or the padding propagates automatically to the group shapes, and authors edit outline strategy and padding as ordinary node properties rather than drawing shapes by hand.

B.3 Estimating Plot Padding from the Image

In this appendix subsection, we specifically outline the final work we did before publication of this project to improve it. In our scaffolding tool, we reduced the time it took for nodes to be placed from eight minutes down to one or less. But human error was still a factor. We wanted to make this process more accurate and *even* faster.

Up to this point, the scaffold's layout is internally consistent but free floating: it knows where marks sit relative to one another, not how the uploaded image split its canvas between the data rectangle and the axis space, which authors originally recovered by manually dragging the four padding values until the scaffold marks settled onto the real marks. We now automate this with a small, fast, dependency-free computer vision pass that runs entirely in the browser, in the spirit of chart reverse-engineering that recovers marks from bitmap images [47, 52].

Detection proceeds on a downscaled copy of the plot region. The image is drawn into an off-screen canvas and a binary foreground mask is built by thresholding the pixel data. The key observation is that data marks are almost always rendered in color while chart chrome (axis lines, gridlines, tick labels, titles, and legend text) is desaturated. A saturation threshold therefore isolates the marks and prevents an axis line from bridging adjacent bars into a single blob; for the rare monochrome chart the pass falls back to a luminance contrast mask. Connected-component labeling groups the mask into candidate marks, each carrying a bounding box, centroid, area, and mean color. Two filters clean the result: components far smaller than the median are discarded as legend swatches, and for line and area charts, where the mark is one long stroke rather than discrete glyphs, the endpoints of the dominant component are used in place of separate marks. The pass returns the extreme detected marks along each axis (leftmost and rightmost, topmost and bottommost) as points in image space.

Padding then follows. A scaffold mark at normalized inner position f sits at $o + p + f\,w_{\text{in}}$, and that f is invariant to padding because it is fixed by the data and scale. Taking the two extreme scaffold marks on an axis, with inner positions f_{lo} and f_{hi}, and the two detected image positions d_{lo} and d_{hi} they should align to, the inner extent and the two padding values follow while the outer box (o, W) is fixed:

$$w'_{\text{in}} = \frac{d_{\text{hi}} - d_{\text{lo}}}{f_{\text{hi}} - f_{\text{lo}}}, \quad p'_{\text{lo}} = d_{\text{lo}} - o - f_{\text{lo}}\,w'_{\text{in}}, \quad p'_{\text{hi}} = W - w'_{\text{in}} - p'_{\text{lo}}.$$

The two axes are solved independently. Several guards keep the result trustworthy: an axis whose two marks are nearly coincident in f is ill-conditioned and is skipped, negative padding is clamped, and a solved inner extent larger than the outer box signals that the box itself is too small and is reported rather than forced. The pass runs once automatically when an image, a chart type, field mappings, and at least two scaffold marks are all present, and on demand thereafter from a control in the scaffold panel; after applying the padding it re-measures the residual between detected and scaffold marks as a one-shot verification.

In practice this recovers padding consistent with the original chart across the four canonical types we tested (bar, stacked bar, line, and scatter), shown in Figure 5. Rarely is manual adjustment needed, and when it is, it is on the magnitude of one or two pixels of nudging. On the bar example, for instance, the solver recovers a left padding close to the width of the chart's y-axis label gutter and a small right padding, with only bar inner padding required for adjustment. The overall time spent by the user is likely on the magnitude of seconds at most, down from a minute. We regard this as promising but early, and a fuller evaluation across more diverse chart images remains future work.

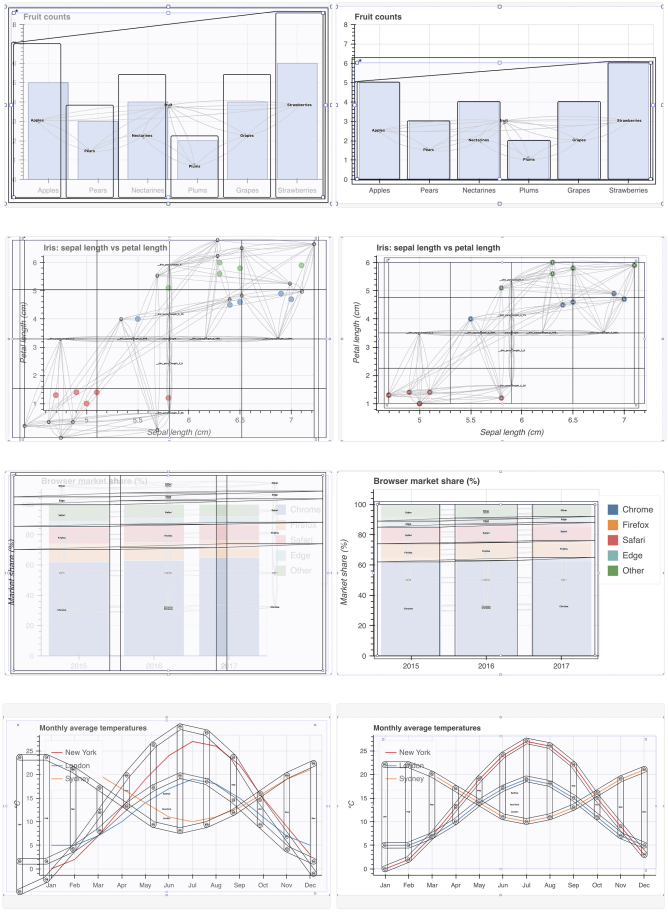

Fig. 5: Computed layout for marks via Vega's visualization engine plus group outlines (left) and with padding estimation using our non-ML computer vision approach (right) for various types of common visualizations.

C GROUP LABEL BUILDER UI

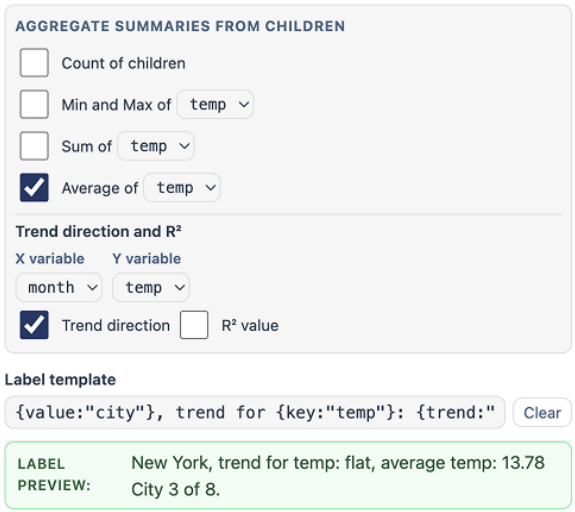

Fig. 6: Group label pattern builder, including an array of aggregate summary options, template formatter field, and preview.