End-Users Publishing Structured Information on the Web: An Observational Study of What, Why, and How

Edward Benson
MIT CSAIL
32 Vassar St., Cambridge, Massachusetts
eob@csail.mit.edu

David R. Karger
MIT CSAIL
32 Vassar St., Cambridge, Massachusetts
karger@mit.edu

ABSTRACT
End-users are accustomed to filtering and browsing styled collections of data on professional web sites, but they have few ways to create and publish such information architectures for themselves. This paper presents a full-lifecycle analysis of the Exhibit framework—an end-user tool which provides such functionality—to understand the needs, capabilities, and practices of this class of users. We include interviews, as well as analysis of over 1,800 visualizations and 200,000 web interactions with these visualizations. Our analysis reveals important findings about this user population which generalize to the task of providing better end-user structured content publication tools.

Author Keywords
Web design; Web content editing; Information architectures; Faceted browsing

ACM Classification Keywords
H.5.4. Information Interfaces and Presentation: Hypertext/Hypermedia - User Issues

INTRODUCTION

Structured data access and navigation is pervasive on the web. Whenever we visit Epicurious.com to find a recipe, browse Amazon to make a purchase, or visit Facebook to look at photos, we are presented with a collection of objects (recipes, products, photos) that are uniformly rendered based on their structure. We can sort these items based on various properties, filter them using faceted browsing, and often visualize them using different templates offered. The structure of this data makes it easier for us to understand it and find what we seek.

At present, the tools for creating such structure-based web pages remain directed almost entirely at highly trained database engineers and web developers who use SQL databases, custom code, and powerful templating engines. Ordinarily-skilled authors are largely relegated to posting unstructured text on blogs and wikis, or to giving their data to companies like Epicurious and Facebook which provide rich browsing interfaces as long as you fit their schema.

During the "Web 2.0" era, a flurry of research anticipated the idea that, given better tools for structured data, end-users would become prolific data publishers, just as they were prolific hypertext publishers on the early web. But many of the technologies that resulted—like microformats, semantic wikis [23], and RDFa [12]—were not widely adopted into the standard end-user toolset. Given the power and value that structured data sites provide professional developers, it is natural to ask why they have not been adopted by end-users. Was the anticipation of data publishing premature or even incorrect? Do these authors lack the technical skills—or the need—for such publication methods?

This paper studies the use of a web-based data publishing framework called Exhibit, which offers a balance between novice accessibility and power-user extensibility. Introduced in 2008, Exhibit allows anyone to publish an interactive structured data visualization by posting a spreadsheet (describing the data) and a single HTML document (describing the presentation and interaction) to any web site. Exhibit extends HTML with tags such as `<map>`, `<list>`, `<facet>`, and `<template>` that turn into interactive data elements. Over 1,800 Exhibit have been created in the wild.

Our study examines Exhibit using a mixed-method approach at four levels of detail. We interview 12 authors of some of the most popular exhibits. We analyze statistics about the design choices and data used by the full set of 1,800 Exhibits in use in the wild. We take the 100 most used Exhibits according to access logs and further examine and taxonomize them. Finally, we assess logged activity of 200,000 web-interactions by Exhibit site visitors. We also provide our dataset and a design gallery of the Exhibits at `csail.mit.edu/haystack/exhibit-study`.

We conducted this study with three goals in mind. First, we validate that there is a community of authors who seek to publish interactive data visualizations and whose needs are not met by the tools available to them. Our interviews explore what these needs are. Second, we seek to understand whether Exhibit's authoring model, which we call the **star model** for reasons explained in the following section, meets the needs of this community. The star model is a constrained way of exposing and integrating components from data visualization and navigation frameworks. Prior work hypothesized that this programming model supports the needs of a

CHI 2014, April 26–May 1, 2014, Toronto, Ontario, Canada.

ACM 978-1-4503-2473-1/14/04..$15.00.
http://dx.doi.org/10.1145/2556288/2557036

wide spectrum of expertise levels in a manner that facilitates code reuse and rapid construction [14]. Studying real-world use of this model provides design guidelines that can be applied to other visualization frameworks. Finally, we seek to synthesize the habits of this community of long tail data publishers and understand how their creations are published and received by web viewers.

We organize our findings by method, which can present a challenge in tying together takeaways which gather support across methods, so we highlight primary ones here and return for a summary at the end of the paper:

- The star model appears to be meeting the needs of a variety of use-cases, from simple to complex, and has provided a platform for many to learn to program along the way.
- Authoring non-tabular data (e.g., multi-valued tables and graphs) is the norm, but authors are "abusing" tabular editors (like spreadsheets) to do so. These authors have no problems with complex data models, but struggle with the text syntaxes we ask them to use to describe these models.
- Authors will choose unnecessarily complex and labor intensive solutions over equivalent simple ones if the copyable design examples address the complicated case.
- Authors are often dependent on, but "boxed in" and limited by, Content Management Systems, and need ways to take control over the entire web page and the structure of the data managed.
- Authors who think their readers want data interaction are **right**: the data browsing features authors create get used, and the kind of browsing widgets offered impacts the way readers browse.

RELATED WORK

This work aims to make contributions in the vein of other studies about the creation and management of web content. Voida's work with "homebrew databases" identified key challenges for small organizations tasked with managing their own data [22]. Landay's study of design workflow provides an understanding of how web interfaces are created from scratch [20]. Klemmer and others provide additional information about the practice of borrowing and tweaking the designs of others [11, 19, 13]. Our work has parallels to all three: we seek to describe the common practices of authors who create "homebrew" data-backed web pages and identify their needs so that toolbuilders can better serve this community[1].

In-field evaluation of data visualization toolkits has been relatively sparse. Neither Exhibit nor more numerically-focused libraries such as Protovis [4, 6] and D3 [5] have been the subject of follow-up studies which track their usage patterns in the community. Protovis, for example, instead applies a Cognitive Dimensions of Notation analysis and provides reactions to the framework [4, 6], and D3 [5] focuses on performance and developer reactions. This paper provides important insights about how such tools are actually used in the wild, and by end-users rather than programmers.

Exhibit and the Star Model

[1]We consider this a separate group than professional-focused information management tools like Greenstone and Kepler [25, 1]

Exhibit is a Javascript web framework that enables end-users to author (not program) faceted interactive visualizations like the ones shown in Figure 1, just as they author HTML documents. While number-centric visualizations, such as scatter plots, are possible, the framework is heavily targeted at browsing collections of structured items. Authors create visualizations (called "Exhibits") by writing a simple HTML-based configuration and then linking to data the same way they would link to a CSS file. Many data types are supported: CSV, JSON, MS Excel, BibTeX, and the Google Spreadsheets API.

Exhibit implements an interaction design that we call the **star model** and seek to validate as a useful pattern for enabling end-user data display creation. In this model, a global collection of data items serves as the center of the star, and declaratively specified UI widgets exist as the spokes—the points of the star—that either display, filter, or modify this collection. These widgets are connected only implicitly to the global collection at the center, and they are otherwise completely independent of each other. Unlike D3 or Storytelling Alice [17, 10] everything is specified declaratively, though authors can create their own plugins easily. And unlike D3, Alice, and Yahoo Pipes! [15], there is no notion of dataflow, significantly simplifying construction, copying, and pasting, and reuse. This also means this model lacks the power that raw SQL queries or programming languages afford, but our interviews and usage analysis show this extra power is not necessary for many users. The independence between spokes and implied connection to the data in the center further simplifies identifying and copying individual fragments of a visualization as well.

The star model results in visualization configurations that can easily be scraped and analyzed programmaticaly, unlike procedural frameworks. An Exhibit configuration has three main components that we analyze. The first is one or more **views**—maps, timelines, tables, lists, or scatter plots—that can be shown modally or stacked together. The second is zero or more **facets** and text search widgets. Facets are UI widgets that display the range of some field of data, providing an affordance for viewers to filter that data to only a subset of that range. For example, in a course catalog, a viewer might choose to view only courses from the Computer Science department. Finally, the Exhibit configuration contains **lenses**, small HTML templates that describe styling for items in the dataset.

The star model is not an alternative to frameworks like D3, but a set of constraints that result in a component packaging system that can be layered on top of such systems. With this packaging system, authors do not worry about data flow or widget implementation; they simply say "put this widget here, with these options." We hypothesize that this model strikes a balance between simplicity and power that satisfies the needs of a wide range of authors, and thus deserves wider consideration. We validate this hypothesis in our author interviews.

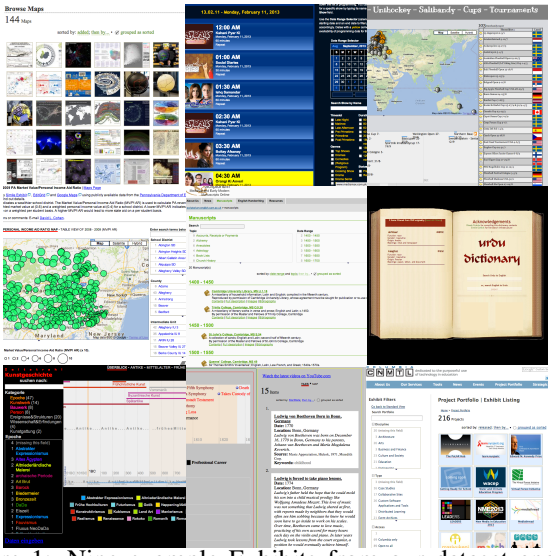

Figure 1: Nine example Exhibits from our dataset and the resulting design gallery.

DATASET

We curated three data sets for the use in this paper:

VIZ1800 Dataset. We generated a list of 1,897 Exhibits by examining the Apache server logs on the machine that hosts the Exhibit Javascript library. This dataset contains the complete set of assets used to create each visualization: the dataset, the HTML page surrounding the visualization, and the visualization configuration (views, facets, and lenses).

TOP100 Dataset. For analyses in the paper that required human coding, we used the subset of the data corresponding to the top 100 most visited exhibits according the Apache logs for a period of months in 2012. Our interview subjects were among the authors of Exhibits in this dataset.

USAGE Dataset. Several Exhibit authors we contacted agreed to use a special version of Exhibit with anonymous analytic hooks that recorded usage events such as facet operations, text searches, and view selection (e.g., switching from a Map to a Timeline). In total, we collected 266,747 distinct user events for 32,978 distinct IP addresses (hashed for anonymity) across 57 distinct Exhibits.

AUTHORS

We interviewed twelve authors to better understand how and why people use Exhibit. The interviewees responded to a request for participation sent out to the authors of the TOP100 dataset. They ranged from school administrators and teachers to business owners and urban planners. Interviews were semi-structured, consisting of a standard set of prompts about visualization creation and maintenance, with follow-up questions based on the answers received. Though this study is centered upon Exhibit, we attempt to make constructive observations that generalize to other tools.

Skills and Expertise

None of the authors we interviewed were formally trained or employed in web development. Author 1's description is representative of the group at large: "I'm fairly comfortable with HTML. I've been doing HTML stuff for a long time. CSS is doable, but I don't feel as comfortable. I don't do too many fancy things, I guess." Subtle misuse of words from the technical perspective (such as describing HTML editing as "adding brackets") was common in over half the participants, indicating that most of this population is self taught.

Five of the participants had prior experience with data visualization. Two used tools such as Tableau regularly, two were very familiar with cartography, and one runs a consumer product website. Three-fourths also had prior experience editing HTML. Three self-described themselves as having become advocates of structured data publishing, and described transitioning into local "gardeners," to apply Gantt and Nardi's *Gardeners and Gurus* metaphor [9]. In addition to creating visualizations themselves, they educated and provided support to their respective communities.

Intent

Four common themes were present when authors were asked why they chose to use Exhibit to visualize their data:

(1) Data navigation. Eight of the interviewees mentioned wanting some way to let people navigate through the information they were publishing. For some, it was the primary reason they said they chose the framework. As an example, Author 3 saw the ability to dynamically filter and explore datasets as a competitive business advantage, both for customer-facing pages and his employees' own internal use. Author 11 and Author 9 both created their visualizations as a way to help themselves and coworkers interactively filter through sizable library catalogues that they managed for their departments.

(2) High-level data understanding. All of the authors who used maps or timelines, and some who did not, mentioned wanting to provide an understanding of their data in a different light than only text allowed. "I wanted to draw students' attention to different categories of information," Author 10, a professor, describes, "so having a marriage of these two visualizations [a map and a timeline] seemed very appealing to me." Author 2, also an educator, created curations of prior work for new projects. At first, "there were other people working on the project that said, 'Ah, this is great,' because they could have a quick look at all the different papers." Later, he discovered that the information displayed on facets themselves were a useful indicator of aggregate trends in his data (e.g., publications over time, or by school), not just a way to filter information.

(3) A programming-free way to publish data. Some interviewees were not thinking of themselves as data publishers, but rather just looking for a way to create a web page with information from some other source. Author 7 wanted to put an Excel spreadsheet of employee information into an HTML table. She had copied the Exhibit configuration from another department's web page and simply changed the data link. Unknowing that Exhibit provides faceted browsing and text search, she showed off the new employee-page they had recently paid a PHP developer to create that added facets to

the employee directory, commenting that she wished "Exhibit was capable of doing all this stuff [faceted browsing]" so that they wouldn't have needed a custom system.

(4) Frustration with traditional web development. Tech savvy interviewees also commented about their desire for respite from the complexity of web development. Author 3's company had gone through a custom PHP-based product catalogue ("that was particularly clumsy"), storefront management software ("didn't work so well"), and Google's enterprise search solutions ("didn't really work well either"), before choosing Exhibit because it provide a product catalogue with "fast, cross-referencable information" without any need to set up multiple web pages or maintain complex software.

These motivations affirm the initial needs Exhibit sought to target: navigating and visualizing sets of data without requiring programming. They also provide evidence that Exhibit's "star model" of programming provides a "pay-as-you-go" complexity that meets the needs of both novices (point 3 above) and experts (point 4). These users replaced, rather than complimented, their traditional techniques — exporting to HTML for (3), and SaaS/bespoken systems for (4).

Data Creation

All of the authors we spoke to authored data in a GUI-based tool, such as FileMaker Pro, Excel, or Google Spreadsheets. Even though Exhibit supports Excel as a data source, Excel-based authors reported converting their data to JSON for publication. After this initial export, some would later update the data in JSON format rather than its source format.

While a few authors described switching to JSON as an authoring format as they became more familiar with it, the majority had negative opinions about JSON's usability, citing it as one of the larger challenges of the visualization process. "I taught a course where we built [Exhibits] from JSON files, instead of Google Spreadsheets...and students unfamiliar with programming just found it too tedious and too difficult to chase after every missed comma and every unequal bracket...it defeated too many of them...quite a few just sort of threw up their hands in defeat," says Author 10. Author 3 has similar experience: "With JSON one of my concerns is not the coding but the missing brackets, and I've since found applications that help me find when I've got a missing bracket or an extra comma or something, but that gets to be an issue." As does Author 1: "that was the part that was less intuitive to me, setting up the JSON file."

These conversations suggest that the problem is one of syntax and tooling, not understanding. These authors (and their students) are capable of organizing and thinking about structured data, but not skilled in the practice and tools of structured text editing that programmers take for granted. Editors that support syntax highlighting and bracket matching are largely unknown outside the technical community.

Trial and Error Editing

Despite the lack of formal training, these authors were able to accomplish an impressive array of visualizations, using HTML and sometimes CSS and Javascript. The majority referenced a trial-and-error development style in which they would iteratively experiment with modifications to their code and check the results, often without firm knowledge of how to accomplish their goal.

During one interview, an author whose HTML skills were among the most basic of the group opened the Chrome debugging console to demonstrate experimenting with different CSS styles. Quick feedback is a good substitute for *a priori* knowledge, as this author had discovered. And a common request by interviewees was to be able to edit visualizations in the browser and save them somehow, as a way to shorten this feedback loop. This theme has broader implications for web development, suggesting benefits to making the web browser capable of replacing a separate program for editing and saving web documents.

Copy, Paste, and Tweak Workflow

Armed with a basic knowledge of web syntax and semantics, but a lack of formal knowledge, *examples* from which to draw become incredibly important. Nearly all of the authors reported creating visualizations by coping and then modifying an existing visualization. This is concordant with previous findings about the both the practice and utility of copying and modifying code and design examples to create new artifacts [19, 11, 13].

Author 8, a repeat user of Exhibit, describes his process as "basically taking an existing file, reusing that, editing it as need be, and then also...unless it's a real time crunch, I'm trying to learn more...add more features or add more components whether that's improved design on the CSS side or trying to do more with the data file." Author 3 describes his creation experience: "I remember back. I was on that wiki a lot...The reason that the exhibits ended up being something I was able to use was that there were some examples, and God I just wished there were more. It was one of the things I remember very clearly, going through and seeing how everybody did things. Examples were invaluable." Author 7's visualizations were copied and pasted from another department's website within her company without any modification: only the data link was changed.

After hearing what an important role design examples played in the authoring process, we decided to repurpose our visualization dataset as a design gallery. Users of the gallery can browse through Exhibits found on the web based on what features of the framework they use (we can programatically extract this). The screenshots in Figure 1 were selected from the visualizations curated in this gallery[2].

Publication

The struggle to integrate non-text forms of expression into Content Management Systems (CMSs) was a common theme. Author 1, whose office mandates that all web content be published through a CMS, reported that after she created the Exhibit as HTML, "I wasn't able to publish the Exhibit myself. I had to go to the technology people and ask them put

[2] Available at `simile-widgets.org/exhibit/gallery`

it up." Author 7 remarks about posting a new web page online, "I had to get help from a PHP developer. I'm not a PHP developer...to get help from an expert who knows PHP was a big plus." Author 10 states, "I have a love hate relationship with Content Management Systems...[they] very often have overhead that I don't want to mess with."

The crux of this problem seems to be that CMSs provide authors a very narrow canvas on which to draw: a contiguous rectangular region in the middle of the web page, with no easy access to side gutters or the `HEAD` element. Side gutters are important for design reasons—they are the canonical place for navigation—but the CMS generally does not permit page authors to modify them. And access to the `HEAD` element is sometimes necessary to include CSS and Javascript. Accessing either of these requires that the user become not only a system administrator, but also a programmer, editing low-level theme files. And authors who did this often made mistakes: our scrape of the web revealed several sites in which a theme file had been mis-configured to include Exhibit's Javascript and CSS on every page, regardless of whether there was a visualization.

Despite the challenges of using external Javascript frameworks alongside a CMS, Author 5, who worked for a newspaper, believes that this is a better way to extend CMS functionality than native CMS plugins. "Media organizations can't work with something new unless that something was explicitly designed to work with [the CMS] they already have. There is an obvious workaround that every single modern CMS will allow you to embed custom Javascript into the published page." Javascript libraries, from his perspective, are the one truly platform independent way to extend web publishing systems. To avoid the problems we observed in our interviews, Javascript library authors should ensure their libraries can be included from within the authoring view of a CMS.

THE DATA AUTHORS PUBLISH

This section looks at what kinds of data people visualize and how they create and publish that data to the web. We use both the TOP100 and the VIZ1800 dataset to understand the domain, scale, model, and format of data authored by the Exhibit community. Taken together with the interviews, the two chief conclusions supported by this section are: (1) authors understand complex data models, but are stymied by lack of support for them in mainstream editors (like spreadsheets), and (2) authors sers will choose unnecessarily complex and labor intensive data solutions over equivalent simple ones if the copyable design examples address the complicated case.

The scale and topic of hand-crafted visualizations

Put in spreadsheet terms, the average visualization created with Exhibit is 14 columns wide and 142 rows tall (with medians 12 and 67, respectively). The distribution of sizes is shown in Figure 2. The two are shown as a pair of histograms instead of a scatter plot because there is little correlation between schema size and item count ($r = 0.16$).

We visited each site in our TOP100 dataset and performed three rounds of open coding to characterize the topic about

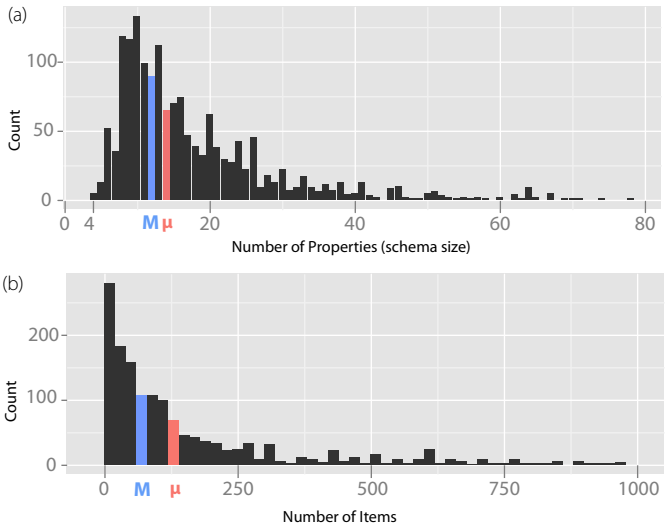

Figure 2: **(a)** The average dataset used with Exhibit has a schema of 14 properties (median 12). This figure and calculation omits two outliers with more than 200 properties. **(b)** The average Exhibit has 142 distinct items (median 67). This graph and calculation is performed after clipping datasets greater than 1000, which hand-sampling shows are more likely to be generated by a dynamic database query.

which the data was created. This process resulted in the two-level breakdown of topics shown in Figure 3. As seen in the diagram, nearly half of the visualization in this dataset were devoted to the topic we called Information Resources, which consisted of abstract facts about the world (such a faceted list of chemical compounds) as well as references to other pieces of information (such as a list of research publications).

Interestingly, only ten of the twenty-three visualizations in the Events cluster made use of Exhibit's timeline visualization, despite "Events" being inherently temporal in nature. The other thirteen used only faceted text-based tables and lists could have easily been created with plain HTML except for the faceted data navigation. This supports the common refrain in interviews and prior work [16] that data *navigation*—rather than visualization—is more important for some authors. HTML alone does not suffice this need, as it provides only the ability to create tables and lists, but not to sort, facet, or filter them.

(Ab)using Spreadsheets to author non-Tabular Data

A *data model* is an abstract set of rules that dictates how data can be structured. A spreadsheet, for example, uses a *tabular* model which represents a list of items (rows), each with the same set of scalar properties (columns). *Multi-valued tables* are like tables, but allow table cells to contain multiple values (e.g., a list of countries). And a *graph* represents a collection of nodes with relationships drawn between them. These abstract models have different breadths of expressiveness:

$$\text{Tables} \subset \text{M-V Tables} \subset \text{Acyc. Graphs} \subset \text{Cyclic Graphs}$$

Tools (and specialized languages) for authoring data are typically targeted at a particular type of abstract data model, such

42	Information resources	18	Research Papers
		8	Facts (e.g., Chemistry, Municiple)
		4	Library Resources
		3	Reference Materials
		3	Commentary (e.g., Blog)
		3	Other
		2	Projects
		2	Datasets
27	People	18	Group Members
		4	Historical Figures
		2	Groups
		2	Other
23	Events	7	Historical
		7	Classes
		5	Other
		4	Entertainment
5	Objects	3	Maps, Books
		2	Products
3	Places	2	Brick and Mortar
		1	Locales

Figure 3: Item types visualized in the hundred most popular Exhibits in our dataset. Types were and sub-types were refined using a three-round open coding process.

as spreadsheets (and CSV) for tabular data. But it is often possible to author data of any model using any tool as long as the writer and reader agree upon the rules to marshal and de-marshal it. Someone might author a multi-valued table inside a spreadsheet by placing comma-separated lists of items inside cells, for example, or they may author graphical data by agreeing upon the columns SUBJECT, PREDICATE, and OBJECT to record the graph.

We programatically analyzed each dataset in the VIZ1800 collection to determine what what underlying data model it used. To our surprise, less than half (41%) of the data sets were strictly tabular. 32% were multi-valued tables, 21% were acyclic graphs, and 6% were cyclic graphs.

Spreadsheets are the overwhelmingly predominant data authoring tool for end-users, and Exhibit authors are no different: all but one interviewee reported having authored their data in a spreadsheet application. Assuming the population of the VIZ1800 dataset continues this trend, this breakdown of data model is alarming: most people author non-tabular data, but they are stuck using table-centric tools to do so.

While Exhibit provides workarounds for encoding nontabular data inside a tabular model (semicolon-separated lists for multi-valued tables and reference-able node IDs for graphs), spreadsheet UIs are not designed to provide any particular assistance for this kind of data. Spreadsheets do not allow authors to navigate or process lists encoded as semicolon-separated strings, for instance, and they force them into narrow cell layouts designed for numbers. Figure 4 shows that mismatches like this create enough friction to drive some graphical data authors to text formats (JSON) instead. A chi-squared test shows the distribution of publishing medium (JSON, RDF, or Google Spreadsheets) is significantly different for graphical data authors ($p < 2.2 \times 10^{-16}, df = 2$). Recall from our interviews that JSON was not a pleasant format among these authors, but they are driven to it by the lack of good graphical data support in spreadsheets.

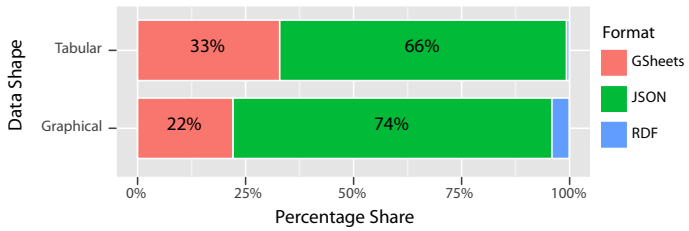

Figure 4: Distribution of data authoring format used conditioned on the shape of data being authored.

This is a quantified gap between the observed habits of data authoring end-users and the tools the community has provided them. Either there has been a failure of industry to understand the sophistication of end-user data modeling, or a failure to properly integrate and educate authors about options available to them. The Related Worksheets line of work provides a promising way to enhance existing spreadsheet UIs to accommodate multi-valued tables and hierarchical data [3]. A similar way to enable better graphical data editing within spreadsheets would also be welcomed by the community.

"Linked Data" is actually used

Why not just encourage authors to stick to strictly tabular data, which tools like Excel already expertly support? One reason is the ability of the graphical data model to "link" entities together based on their relationships. For example, a list of Bob's grandchildren may be expressed as `Bob.children.children`. This is a powerful expressive capability not possible with the tabular model, and our data shows that authors are taking advantage of this power.

We measured the number of relationship "hops" used in Exhibit configurations in the VIZ1800 dataset. Tabular data is fixed at 1 hop, e.g., BOB.BIRTHDAY, whereas graphical data can contain more than one. A T-Test ($p < 3.2 \times 10^{-6}, df = 591$) shows that visualizations backed by graphical data have configurations that utilize more relationship hops than non-graphical data. In other words, authors who use more flexible data models are also visualizing more complex relationships in their data. Supporting graphical data editing in spreadsheets would therefore directly support end-user visualization needs, not just spurious data editing habits.

Authors follow examples, not self interest

Another phenomenon we observed was that authors will choose between alternative approaches to a problem based on the availability of design examples rather than personal utility. This validates Weiss' study of API selection in mashups [24]. In our case, the result can be seen in the format authors selected to publish their data.

Exhibit accepts data in a number of formats, including Excel, Google Spreadsheets, CSV, and JSON. Table 1 shows that JSON dominates over the others, and Google Spreadsheets is the clear alternative. CSV is remarkably scarce given that it

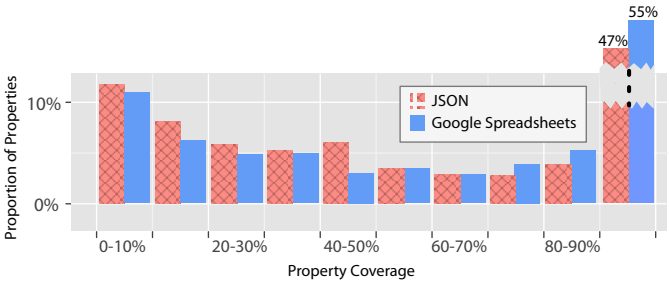

Figure 5: Normalized histogram of property coverage for properties of JSON datasets versus Google Spreadsheet dataset. JSON data is more likely to contain low-coverage properties, while Google Spreadsheets are more likely to contain high-coverage properties.

Data Format	Count	Percent
JSON	1246	69%
Google Spreadsheets	580	32%
Bibtex	38	2%
RDF	25	1.4%
MS Excel	2	0.12%
Freebase	1	0.06%
CSV	1	0.06%

Table 1: Data formats used by Exhibit authors, the total count of visualizations using each type, and the percentage of visualizations using each type. Note that because some visualizations incorporate data from multiples sources of different types, this table adds up to more then 100%.

is the standard text serialization of spreadsheets, and Excel is virtually absent as well. Are Exhibit authors simply more comfortable with JSON and Google Spreadsheets?

If the interview population is representative of the VIZ1800 population, then a different explanation is more likely. Our interviewees reported authoring data in spreadsheets and then converting it to JSON for publication. When asked why they took this unnecessary last step (publishing the Excel file to the web would have sufficed), they cited following the examples they found online. Our *post hoc* review of the Exhibit wiki confirmed that virtually all the design examples used JSON and the documentation tended to focus on JSON and Google Spreadsheets. The easier paths to publication were less documented, and as a result virtually unused.

Self-describing data formats are used more flexibly

Separate from the topic of data model is that of data format—the rules which govern the way data is written down. Data formats can largely be divided into schema-based (such as CSV and Spreadsheets) and self-describing (such as XML, JSON, and RDF). Schema-based formats are more compact: they list the structure of the data once, like the header row of a CSV file, and it remains implied for the rest of the file. Self-describing formats are more verbose: they always specify the structure of the data in full, so that a fragment can be understood independent of its file.

One of the primary advantages claimed of self-describing formats is that they permit adding "one-off" properties that are relevant to only a small set of items. Adding a PETNAME

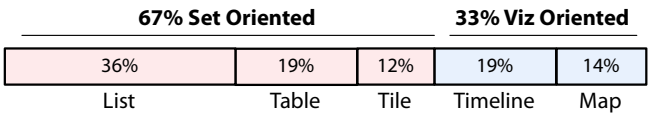

Figure 6: View types used in single-view exhibits.

column on behalf of just a single customer in a spreadsheet might result in thousands of empty cells on the other rows, but with a self-describing format, this change would only affect the one customer record who actually used the property.

These kinds of "one-off" properties are an inevitability in many real-world scenarios. We can measure them by looking at a statistic called *property coverage*: the number of items with a property divided by the total number of items in the dataset. A coverage of 1 means every item has the property, and a coverage of 0.1 means that only 10% do.

Figure 5 shows the distribution of property coverage of visualizations backed by JSON (self-describing) and Google Spreadsheets (schema-based) in the VIS1800 dataset. JSON-based data is more likely to have low coverage ($> 50\%$) properties, and Google Spreadsheet-based data is more likely to have high-coverage ($> 70\%$) properties. This shows that authors are in fact using self-describing formats differently than schema-based ones in exactly the manner that the abstract argument for self-describing formats would suggest.

VISUALIZATIONS

This section examines how authors designed their visualizations. Recall that Exhibit visualizations are divided into three primary components: views, such as a map, timeline, or table; facets, which provide searching and filtering functionality; and lenses, which are miniature web templates for data items. These three components are woven into the rest of the web page with HTML.

Exhibits can have multiple views, displayed either modally or side-by-side. The mean view count is 1.4 and the median is one. 70% of Exhibits have one view, and 20% have two. The highest view count observed is 6. Exhibit visualizations have a mean of 3.1 facets and a median of 3. The most facets of any Exhibit in our dataset is 26. Only 17.5% of all Exhibit have no facets at all, which means that the vast majority of Exhibit authors both understand the concept of faceted navigation and consider the ability to filter sets of data a useful enough feature to include in their visualization.

This section uses the VIZ1800 dataset to find three results. First, we identify three distinct data publishing needs: navigation, visualization, and "just plain publishing". We show correlation between heavy use of a component and the complexity with which it is used. The most important and final result is that Exhibit authors tend to publish visualizations separately from text articles and need CMSs to allow them to take over the entire web page.

Navigation, Visualization, and Just Plain Publishing

The motivation of the Exhibit paper hypothesized that there were communities of authors who wanted to (1) *provide navigation through* and (2) *visualize the relationships between*

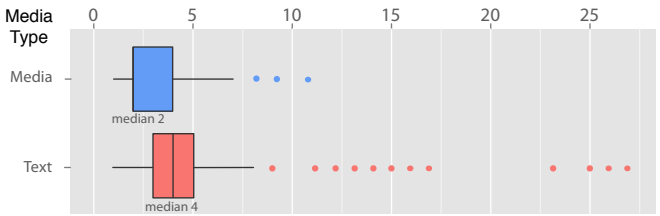

Figure 7: Of Exhibits that have only one view, text-only views have significantly more facets than rich-media views.

sets of items, but they were unable to do so because they lacked the proper tools. This claim was left unevaluated, and in this section we use data from the past eight years of Exhibit use in the wild to supplement our user interviews in demonstrating that this community, and these two needs do exist. We further show a third use evidenced in the data and interviews: using visualization frameworks as an on-ramp to web publishing for individuals not familiar with HTML.

To show this, we simplify the dataset by filtering it to only those Exhibits with a single view, 70% of all Exhibits total. This represents the set of visualizations with a focused purpose (i.e., just showing a map, or just showing a table), the breakdown of which is shown in Figure 6. 80% of these single-view Exhibits have facets.

Maps and timelines make up 33% of these Exhibits, which we claim are primarily used to visualize the relationship between data. Text-only Exhibits (lists, tables, and tiles) make up the other 67% of this collection. These Exhibits could have easily been reproduced in HTML alone *except for* Exhibit's faceting, sorting, and searching functionality. This extra functionality must have been the reason these authors spent time learning and deploying Exhibit on their site, demonstrating that data navigation is at least as important as visualization among Exhibit's user population.

By examining the distribution of facets for these two groups we can strengthen this distinction. We can infer that if an author's goal is to provide data navigation, she will have an incentive to provide more facets than if her goal was visualization of the relationship between items. Figure 7 shows a boxplot of facet count for the groups. Sure enough, text-only Exhibits (lists, tables, and tiles) contain significantly more facets than media-based Exhibits ($t(n = 1005) = -7.75, p < 2.2 \times 10^{-14}$).

Use of Exhibit as an on-ramp to HTML publishing is evidenced by the existence of text-based Exhibits with no facets at all. These tables or lists could easily have been ordinary HTML, but copying and modifying an example Exhibit configuration, which provides separation between the HTML design and data storage, must have been easier for these users than authoring the data as HTML. Author 7, who used Exhibit to publish a solitary HTML table, falls into this group.

Those who author more, author more complexly

Do big data sets beget complex visualizations? Exhibit certainly permits complicated visualizations: the most views we observed stacked into a single Exhibit were 6 tables and 3 timelines, and the most facets we observed in a single Exhibit

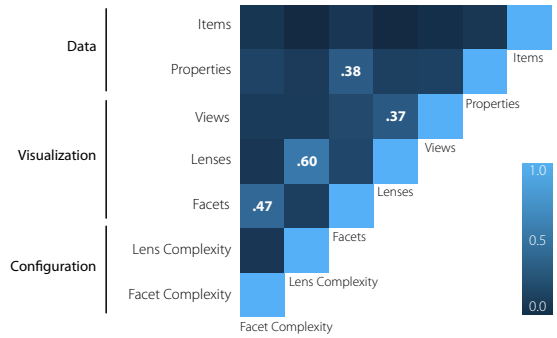

Figure 8: Correlation matrix between several data, visualization, and configuration-centric attributes. Notable is the lack of any strong correlation between aspects of the data and the properties of the visualizations created for that data.

was 28! But, as Figure 8 shows, the cross correlation between several attributes related to data, visualization, and configuration complexity is weak. Big datasets do not necessarily mean big visualizations.

The strongest relationship observed was between lenses (HTML templates used to style an item) and the complexity of configuration expressions within those lenses ($r = 0.6$), indicating that people who override the default design styles of a visualization more often are also likely to be modifying them more heavily. A similar, but weaker relationship was found for facets and facet expression complexity ($r = 0.47$).

Page Layout

The most important observation in this section is about page layout and what visualization authors need from their CMS software (recall that the CMS publishing experience was a point of complaint in interviews). Here we ask two questions about visualization design to help guide CMS improvements for the visualization community:

- How do authors lay out visualization components?
- How do authors integrate visualizations with the text content on their sites?

To answer this first question, we coded the layout patterns of Exhibits found in the TOP100 dataset. The results are shown in Figure 9, with the view area depicted in white and the facet/search ares depicted in red. The results show that left and right sides of a visualization are overwhelmingly the preferred places for navigational elements. Over 80% of all facets we coded were placed to the side of a visualization, and over 50% of visualizations used the sides exclusively.

CMS software typically constrains authoring to a center column of text flanked on the sides by one or two columns reserved for automated content (links, navigation, etc). But access to this center column alone does not provide authors the required width to display both a visualization and filtering widgets to manipulate the visualization. To support the visualization community, it is therefore important that CMS makers either allow authors to override the side columns with custom, page-specific content, or to take over the page entirely, stripping it of the CMS-provided UI chrome.

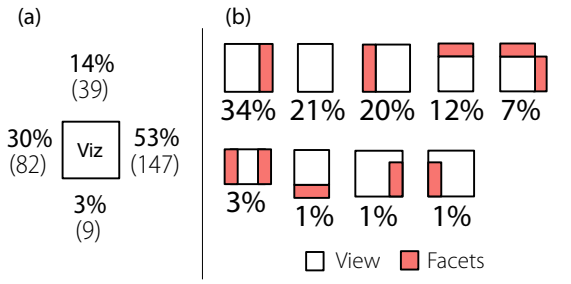

Figure 9: **(a)** Raw counts of facet location in the TOP100 Dataset. Most facets occur to the side of the visualization. **(b)** Counts of global visualization layout in TOP100 Dataset. Over half utilize a two-column format.

To examine how authors integrate visualizations with text content on their sites, we programatically analyzed the HTML structure of the pages which contain each visualization in the VIZ1800 dataset. For each page, we recorded two statistics: the total amount of text on the page in characters and the size of the largest single text node (e.g., what was the longest paragraph (<p>) element?). We also recorded these statistics for several well known websites. We then interpreted these measurements with the following heuristic: if a visualization is embedded in an article as supplementary information, we should expect the text statistics of the web page to appear like other pages known to contain articles. If the visualization stands alone, on a page by it self, we should expect much less text than on an article-containing page.

Figure 10 shows a scatterplot of the data. The horizontal axis is the amount of total text on each page, and the vertical axis is the largest block of text. Web pages containing Exhibits are shown in black, and reference points are shown in red: the 2014 CHI CFP, the front page of Slashdot.org and Reddit.com, and randomly selected articles from the New York Times, Washington Post, and Chronicle of Higher Education.

These results show that Exhibit visualizations are overwhelmingly published as stand-alone content that occupies its own page, not supplementary figures woven into articles. 85% of the points are strictly bound by the point representing the New York Times article, itself the smallest of our reference points. The remaining reference points are at the extremes of the collection. The line visible at $y = x$ represents web pages that contain only a single unit of text outside the visualization.

CMS builders can use this information by prioritizing support for page-centric visualizations over interactive features embedded in prose. It may be that we simply have not yet developed the journalistic culture of mixing the two forms yet, though recent publications like the Snowfall feature by the New York Times [7] and several interactive web adaptations of papers from Berkeley's AMP lab [18, 2] show that there is high payoff when text and interactivity are integrated well.

BROWSING BEHAVIOR OF EXHIBIT SITE VISITORS

We tracked the behavior of 33 thousand distinct site visitors accessing 57 different Exhibits whose authors agreed to participate in our study. We limited our analysis to only the first 30 minutes of interaction for each IP address, the session timeout suggested by prior work [8]. While we do not have the space for a detailed discussion, we preview here a few observations about browsing behavior:

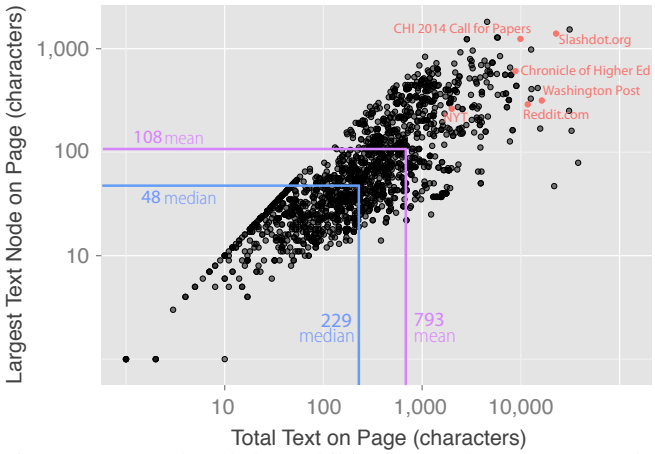

Figure 10: Nearly all the Exhibits in our dataset are stand-alone visualizations, not inline supplements to an article. Reference points (red) of articles on the web contain far more text (horizontal axis) and far larger paragraphs (vertical axis) than the pages in our dataset.

If possible, visitors favor interaction. 93% of visitors who could interact with an Exhibit visualization did. Interaction was either switching between multiple views or filtering the data in some way. The duration of interaction nicely fits an inverse exponential curve with respect to time, validating prior results in faceted browsing systems [21].

Visitors browse in contract-and-expand cycles. Browsing behavior follows a pattern of contracting the displayed dataset (via text search or facet filtering) and then expanding the dataset—a clear indication of browsing rather than searching for a particular item. The number of contraction-expansion cycles follows an inverse exponential curve, depicted in Figure 11. The average contraction phase consists of 1.7 filter actions (stddev 0.8) and the average expansion phase is 1.2 filter actions (stddev 0.8).

Design choice affects browsing habits. The maximum filtering achieved by viewers is significantly different when visualization authors provide just text search, just facets, or both. For facet-only visualizations 41% of viewers filter the dataset to under 5% of its initial dataset size. That number is 84% for text-search only visualizations and 94% if both filtering mechanisms are available.

One advantage of the **star model** for visualization construction is that its constrained processing model (data at the center, filters and views on the spokes) and declarative invocation allow easy collection and processing of usage statistics across wildly different visualizations: actions such as filtering, searching, aggregating, and switching views are invariant to widget implementation details. We plan to follow up on these initial results with a more complete study of *in situ* visualization browsing behavior.

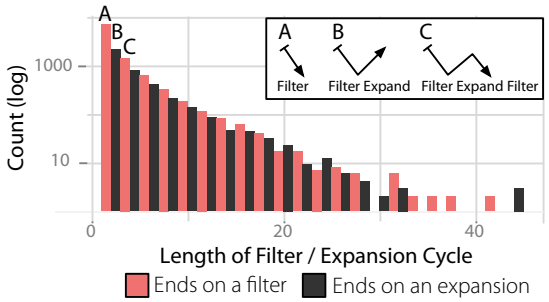

Figure 11: Histogram of the length of contraction-expansion cycles when visitors filter and unfilter data.

CONCLUSION AND TAKEAWAYS

This work provided a mixed-method, full lifecycle analysis of set-based information visualization on the web using the Exhibit framework. Among our chief takeaways are:

- A wide range of users find that the star programming model is both simple enough (novices could use it) and powerful enough (professionals prefer it to alternatives) to meet their needs. We recommend visualization framework authors consider exposing functionality in this style.
- End-user data modeling habits are more sophisticated than their authoring tools. Most Exhibit users author supratabular data, but spreadsheets lack native support for it, so they often (unhappily) turn to text formats as a result. We recommend that spreadsheet makers consider adding support for data models more complex than a simple table.
- Exhibit authors follow the design examples available to them, even if it results in unnecessary labor. Good documentation remains a critical but undersupplied resource, and copy-able design examples are particularly important.
- Visualization and data interface authors have layout needs that exceed the options CMSs provide, and they struggle with their CMSs as a result. Toolkit builders should ensure their Javascript can be loaded from the `<body>` of a page, and CMS builders should enable user control of either the full page (discarding the theme) or the sidebar gutters (which are typically reserved space).
- The star model offers bright possibility for studying how web readers interact with data. Its constraints allow for a wide variety of interfaces, but provide a standardized data bus that enables unified behavior tracking across varied end-user creations.

We additionally provide an interview-based sketch of the motivations and practices of the Exhibit community and a dataset and design gallery of the visualizations this community has created. This community has sophisticated intentions for their data but often no formal training in computer science. The star programming model Exhibit utilizes has enabled them to translate these intentions into working interfaces, and in many cases has provided an environment within which to learn to build more complex displays.

REFERENCES

1. Altinas, I., Berkley, C., Jaegar, E., Jones, M., Ludaescher, B., and Mock, S. Kepler: An extensible system for design and execution of scientific workflows. In *Proc. Future of Grid Data Environments, Global Grid Forum* (2004).
2. Bailis, P., Venkataraman, S., Franklin, M. J., Hellerstein, J. M., and Stoica, I. Probabilistically bounded staleness for practical partial quorums. In *Proc. VLDB* (2012).
3. Bakke, E., Karger, D. R., and Miller, R. C. A spreadsheet-based user interface for managing plural relationships in structured data. In *Proc. CHI* (2011).
4. Bostock, M., and Heer, J. A graphical toolkit for visualization. In *IEEE Trans Vis and Comp Graphics* (2009).
5. Bostock, M., Ogievetsky, V., and Heer, J. D3: Data driven documents. In *IEEE Trans Vis and Comp Graphics* (2011).
6. Bostock, Michael and Heer, Jeffrey. Declarative language design for interactive visualization. In *IEEE Trans Vis and Comp Graphics* (2010).
7. Branch, J. Snow fall: The avalanche at tunnel creek. In *The New York Times* (2013).
8. Cooley, R and Mobasher, B. and Srivastava, J. "Data Preparation for Mining World Wide Web Browsing Patterns. *Journal of Knowledge and Information System* (1999).
9. Gantt, M., and Nardi, B. A. Gardeners and gurus: Patterns of cooperation among cad users. In *Proc. CHI* (1992).
10. Gross, P., Herstand, M., Hodges, J., and Kelleher, C. A code reuse interface for non-programmer middle school students. In *Proc. IUI* (2010).
11. Hartmann, B., Doorley, S., and Klemmer, S. R. Understanding opportunistic design. In *IEEE Pervasive Computing* (2008).
12. Herman, I., Adida, B., Sporny, M., and Birbeck, M. Rdfa 1.1 primer: Rich structured data markup for web documents. In *W3C Working Group Note* (2013).
13. Herring, S. R., Chang, C.-C., Krantzler, J., and Bailey, B. P. Getting inspired! understanding how and why examples are used in creative design practice. In *Proc. CHI* (2009).
14. Huynh, D. F., Karger, D. R., and Miller, R. C. Exhibit: Lightweight structured data publishing. In *Proc. WWW* (2007).
15. Jones, M. C., and Churchill, E. F. Conversations in developer communities: A preliminary analysis of the yahoo! pipes community. In *Proc. C&T* (2009).
16. Karger, D. Standards opportunities around data-bearing web pages. In *Proc. HCIR* (2012).
17. Kelleher, C., Pausch, R., and Kiesler, S. Storytelling alice motivates middle school girls to learn computer programming. In *Proc. CHI* (2007).
18. Kraska, T., Pang, G., Franklin, M. J., Madden, S., and Feketey, A. Mdcc: Multi-data center consistency. In *Proc. Eurosys* (2013).
19. Lee, B., Srivastava, S., Kuar, R., Brafman, R., and Klemmer, S. R. Designing with interactive example galleries. In *Proc. CHI* (2010).
20. Newman, M., and Landay, J. Sitemaps, storyboards, and specifications: a sketch of web site design practice. In *Proc. DIS* (2000).
21. Niu, X., and Hemminger, B. Beyond text querying and ranking list: How people are searching through faceted catalogs in two library environments. In *Proc. ASIS&T* (2010).
22. Voida, A., Harmon, E., and Al-Ani, B. Homebrew databases: Complexities of everyday information management in nonprofit organizations. In *Proc. CHI* (2011).
23. Völkel, M., Krötzsch, M., Vrandecic, D., Haller, H., and Studer, R. Semantic wikipedia. In *Proc. WWW* (2006).
24. Weiss, M., and Sari, S. Evolution of the mashup ecosystem by copying. In *International Workshop on Web APIs and Services Mashups* (2010).
25. Witten, I. H., Bainbridge, D., and Boddie, S. J. Greenstone open source digital library software. In *D-Lib Magazine* (2001).